Bulk Rename Utility Operations Manual

Volume 2

Expert's Corner

By Timothy R. Mongeon

Revised Edition

TABLE Of CONTENTS

Welcome to 2nd Volume ... 32
Common Errors 35
About the Format 45

Expert's Corner - Bulk Rename Utility

Changing Parentheses to Brackets 47
Add Alpha Order Name (Alphabet A-Z Serial Numbering) 48
Remove .com, All Brackets, Hyphens and Dashes and Maintain <space> before Year 49
Completely Renaming Files Sequentially (1,2,3...etc) 51
Replace Filename with Folder Name and Append Sequential Number 52
Remove Underscore and Add Brackets around Last Word in String 53
Renaming Folders During Move 54
Rename Duplicate Filenames W/Original Filename and Delete Them 55
Rename Files Based on Folder Name 56
Cropping File Extension 58
Remove First Word 59
Adding Space between Hyphens Anywhere in String 60
Add Creation Date as Suffix 61
Renaming with Odd or Even Numerals using Autonumber 62
Use 'Folder' to Reset Autonumber for each Subdirectory's Sequential Numbering of Files 63

TABLE Of CONTENTS

Expert's Corner - Bulk Rename Utility

How to Add Title from Mp3 files to File Name Using ID3 Data Extraction 66
Autonumber Files based on When Text in String Changes 67
Remove Extraneous Length of Text at End of Filename when Length is Known 69
Rename Same Number Sequence Alpha 70
Appending Parent Folders Names to All SubFolders Names 71
Import Rename-Pairs using Unicode (UTF-16) Files 73
Convert Case to Uppercase for Significant Words Only 74
Pad with Alpha not Numeric 80
Rename Photo Files with Prefix and date of Photo 81

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

A Note On Replacement Strings 88

Railroad Diagrams – Visualizations 90
Regex Buddy 96
BRU and String Position 97
Character Position vs String Position 99
The RegEx Sections 102
Tim Additions 103
Additional Notes 104
My Philosophy 105
Testing the Waters 108

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Date Manipulation

Move Date M.D.Y to front 114
Convert Date (M.D.Y) to (Y.M.D) 115
Convert M-D-Y to Y-M-D 121
Move Date to Front 122
Convert DMY to Y-M-D 124
Convert MDY into YMD so the Files will List Chronologically ... 126
Convert Date from mm-dd-yy to yyyy-mm-dd 128
Insert Current 'Modified Date' and Move Date within Filename .. 132
Rename DDMMYYYY to YYYYMMDD 136
dd_mmm_yyyy to yymmdd Convert Abbreviated Written Date to Numeric 140
Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD And Rearrange to Front 146
Using Custom date with BRC 156
Use Character Translation to Convert Written Month to Numeric Format 163

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

String Manipulation (Parsing)

Tim Notes: Parsing 194
Re-arrange String Parts 195
Moving Part of a Filename to Another Location 197
Rename Camera Image Files 198
Switch Artist - Title to Title - Artist 201
Change around String but Add Zeros to Single Digits 202
Change Format to both Folders and the Files Within 205
Place Parentheses Around Year in String 207
Within string - Title first , Date Second 211
Separate Run on Names, Remove Extraneous Hyphens 213
Move Underscored Date to Beginning of String 237
Simple Rearrange 239
Simple Rearrange – More difficult 241
Another Switch around 243
Not as Simple a Rearrange 245
Ignore files with parentheses, otherwise append (Original Mix) 260
Move 'The' to End of Artist up to Hyphen 261
Move 'The, 'A', 'An' (insignificant words) to End of Filename 263
Convert <first name> <space> <surname> ~to~
<first Initial> <surname> 266
Move End to Beginning (Numeric Digits) 267
Move 4-digit year (within Filename) to Beginning of File 269

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

String Manipulation (Parsing) cont.

Moving a Date (within Filename) to the Front of the File 271
Rearrange Removing First Name and Parentheses 273
Moving around Digits in date for Filename 275
Rearrange Word and Remove first Underscore 277
Change Second Hyphen to Underscore 279
Parsing at its Simplest – Reformat items with Comma 281
Another Real Simple Example of Re-Arrangement 284
Moving the last 2 Words of any Phrase 286
Rearrange Text in Parentheses, Remove Parentheses, Move Hyphen 289
Parse and Rearrange Complex String 292
Rearrange Name in Brackets at End of Filename 297
Change 'Date Taken' attribute of the Photo (YYYYMMDD_HH.MM.SS) to More Legible 298
Rearrange MP3 Title 301
Simple Rearrange #1 305
Simple Rearrange #2 306
Rearrange Names to LastNameFirstInitial 309
Replace all the Spaces and Brackets in a Filename by Underscores 311
Simple Rearrange #3 317
Re-arranging Year Placement w Variable Lengths 319
Exercise in Parsing String Containing Numeric Digits – Restrictive to Universal 325

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

String Manipulation (Parsing) cont.

A Parsing Lesson 338
Rearrange – Take First Initial, Append to End and Lowercase All 344
Demonstration of 3 Methods to Rearrange a not-so-simple String 346
Rearrange Names and Remove Extraneous 351
Change around Names at End of Filename 352
Simple Rearrange #4 361
Rearrange Word following Comma to Beginning of String 363
Example of Parsing a Group of Numbers with no Delimiters 365
Switch Portions of Filename strings 371
Insertion or Deletion from End of Filename 373
More Run on Sentences 383

Move all Text after "keyword" to Front of File Name..... 409
The Case for Case Insensitivity 410

Suffix Year and Add Parentheses 412

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Find Occurrences

Find any First Occurrence of a Single Digit Number in a Filename 449
Pad with Zero at Beginning, Middle and End of String 450
Find any Single Digit Number anywhere in a Filename and Add a Preceding Zero #1 454
Find any Single Digit Number anywhere in a Filename and Add a Preceding Zero #2 461
Insert Text at Second Comma in String – i.e., 'brief' 486
Replace last Hyphen (in a 3 Hyphen String) with <space> 488
Move Everything prior to ' - ' to the End of Filename 498
Move Person's Name to end, Eliminate the Underscore after Initial and Add Underscore after # 501
Replace Third Underscore with a Space 503
Add a Space to Certain Files Before/After a Certain Word 505
Move the Word 'The' in first Instance Only to Front of Title and Remove Comma 507
Remove all Characters after x Occurrences of a Character (Remove All after 2nd Occurrence of Underscore) 509
Insert 'Hyphen' after Second Word in String 513
Copy Characters to Start of Name (Good Explanation of \w) 517
Remove all Text after the 'Last' Occurrence of T# when followed by Hyphen 519

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Find Occurrences cont.

Add Text after First or Last Space in String 521
Replace the "." between Letters with a <space>, Ignore "." between Numbers 524

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Filter RegEx (Section #12: Filter)

Ending with 3 but Not 4 Digits 529
Displaying Filenames with One or More Numeric Digits Anywhere in String 531
How to Select Files with No Extension (to Rename) 534

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Remove

Delete Everything after SxxExx i.e., S07E05 536
Remove all Extraneous information after First group of Digits ... 538
Remove Everything after First Occurrence of ')' 540
Remove all Extraneous junk after Extension 542
Remove 'The' from Beginning to End of Title (Not End of Filename) 544
Remove Second + and Replace with Underscore (Capture parts of Filename and Put back Together) 546
Remove First 3 Characters if ##_ 549
Remove the Space Before the Number while Ignoring the Files that do Not Include a Number 550
Remove all text before and after format -$-$- 552
A Word on Simple RegEx 557
Strip Away all Prefix of Numbers and Space Character 559
Remove Everything after Second Hyphen. No Change if Only One Hyphen 561
Remove Everything Between the First _ and last _ 563
Switch Around String and Remove Digits 565
Keep Last part of Filename (Preceded with Underscore) with Extension 571
Remove Everything within First set of Square Brackets 579
Change First Name to Initials on Folders 581

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Remove cont.

To Delete the First Word & <space> in String Following the '-' 583

Remove 'Devo* <space>' from Title 587
Remove all Words Before Last Word in String Retaining Characters At Beginning up to Hyphen 590
Another Method of Parsing the String 595

Remove Underscore from Title, Replace with <space> and Remove Brackets 598
Remove Numeric Digits for any Files that Have Three or More .. 601
Remove All Except the 'pattern in bold' part of Filename 605
Get rid of the URL part of the Name 608
Remove Extraneous Junk at End of Filename 610
Remove Single Numeric Digit (and Preceding Underscore) from Files that End in Single Digit 612
Remove Trailing Dots 614
Get last x Characters 617
Remove All after Year and Convert Dots to Spaces and Add Parentheses Where Noted 620
How to find 12 Digit Number and just Remove it 631
Remove the Timestamp and Change Dot to Hyphen 635
Remove Everything after (USA) 640

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Remove cont.

Remove All before last Bracket 646
Remove Numeric Dollar Amount from String 649
Convert TV Series Season Information from s(##)e(##) to just Numeric 651
Delete Everything in Brackets without Numbers 654
Remove Hyphen as First Character in String 661
How to Keep just a 4 Digit or 5 Digit Number 663
Remove Front end of Filename using Pattern <space> <hyphen> <space> 666
Similar but Not the Same – Remove Front end of Filename using Pattern..... 668
Remove Numeric Digits, Spaces and Hyphen at the Beginning of Filename 669
Remove Everything after Brackets at End of Filename and Replace Brackets with Parentheses 671
Remove Space from End of Filename (to Demonstrate the $ Anchor) 676
Remove Text between Two Dots 677
Remove Duplicate Title from Multiple Music Filenames 685
Removing the Numeric string '_#####' from the Middle of File Names 689
Simple Removes – Keep Last 5 Characters of String, Keep First 6 Characters of String 693
Remove All Except Last Numeric Digits Greater than Zero 696
Remove All Text Between { and } 726

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Remove cont.

Remove 'The' only at the Start of a Filename 729
Removing Multiple Instances of the Same Character 731
Remove parts 'a' and 'b' from Filename (Demonstrate the \w) 734
Remove Hyphens from String (Demonstrate Greedy vs Lazy) 738
Remove Email Addresses from String 742
Renaming Files to only Initials 768
A Simple Lesson on RegEx Remove 774
Rename to Remove UTC (Coordinated Universal Time) Date/Time/Zone 789
Remove All Text after the Dot and keep the Second Underscore 793
From End ... Remove Last Underscore with any Following Text at End of Filename 795
To Beginning ... Remove Blank Space in Front of Filename 797
Remove Text Between a Word and a Date 801
Remove All Text prior to Song Title at Beginning of String 802
Remove Everything after Last Space 808
Remove All Except Preserve Filename and Date in Parentheses 810
Filename should have One <space> Preceding 813
Moving Articles (a, an, the) from the End of the Filename 820
Remove hyphens between date at Beginning of String 824
Remove Text in Parentheses or Brackets 831

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Add

Add Comma after First Item, i.e., Last Name 840
Add Leading Zero after # 842
Padding with Zeros 844
Add the Century to YY in String and Move to End 848
Add Additional 'Year' as Second Item, Remove Hyphen 850
Pad Single Digit Day with Zero 858
Add a Hyphen before First Numeric Digit in String 860
Pad Random Number Prefix with Zeros 862
Pad Digits at End of Filename 865
Add Text before Numeric Digits in String (Parses Text Portion from Numeric) 872
Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename 884
Add '-' AFTER all Upper Case (Artist Name) Or BEFORE Uppercase/Lowercase (title)..... 918
Pad 0 Before Variable Numbers (Represented by #) 938
Add Space Before Numeric at End of Filename, Ignore Those That Already have 947
Add 0 to Single Numeric Digit in String (Position Does not Matter) 953
Prefix 0 if the Number is Between 1-9 955
Add Quotation Signs to String 956
How to Add '-' Between Letters and Digits 961

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Add cont.

Pad Zeros to Single Numeric Digit MM, DD in Dates 963
Prefix Final Digit with Zero if Preceded by Word-Separator (or null)
(Pad in SINGLE Pass!) 969
Add Autonumber in Middle of String (Variable Placement) 988
Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) 991
Add the Word DVD to Files (Demonstrate Negated Character Class) 1000
Add the Text in Parentheses (not Move it) to Beginning of Filename 1002
Renaming All Files with no Extension to .txt 1005

Pad Double Digit at End of String (in one Pass!) 1007
For Single Numeric Digits 1009
For Triple Numeric Digits 1010
Match Against the Last Match 1012
Match Against the First Match 1014
Match Against First Two Numeric Groups 1016
Here is a three numeric digit sample with <dot> characters 1019
Here is a three numeric digit sample With or Without <dot> characters 1021
Multiple Digits within the Same String 1023
Mixed 1023
Mix of Three 1026
Padding/Appending More than One Zero 1028
Pad/Append Only Single Numeric Digits in Mixed 1028
Remove the Two Numeric Digit Numbers from String 1029
Finally Here is A Mixture of 4..... 1031

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Add cont.

Pad Zeros to Single Numeric (In One Pass!) 1037
Single Numeric Digits 1038
Single Numeric Digit as First Numeric group in String 1039
Double Numeric Digits 1042
Prefix First two Numeric Groups in ANY File Example #1 1045
Prefix Numeric Digits in Strings that have Only 2 Numeric Groups .. 1049
Prefix First two Numeric Groups in ANY File Example #2 1053
Prefix First three Numeric Groups in ANY File Example #1 1057
Prefix First three Numeric Groups in ANY File Example #2 1061
Prefix First four Numeric Groups in ANY File 1067
Prefix First three Numeric Groups in ANY File Example #3 1074
Prefix First two Numeric Groups in String with Three or More 1078
Prefix First two Numeric Groups in ANY File Example #3 1080
Prefix Last two Numeric Groups in a String 1084
Prefix five Numeric Groups in a String 1089
Using a Negative Lookahead and a Negative Lookbehind to Determine A Single Numeric Digit at End of String 1094

Changing YYMMDD to YYYYMMDD and Rearrange (Demonstrate \s Whitespace for <space> Character) 1096
Add Spaces Between First Hyphen 1098
Simple Pad of (Zero) to Suffix 1101
Renaming based on the Contents of the Original File Name 1102

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Add cont.

Autonumber and Add Brackets to Suffix Number 1104
A Word on the 'Section #10: Numbering' - Insert At - Option ... 1107
Append Zeros to Prefix of Numeric Numbers of varying Lengths in String 1108
Insert Spaces between First Occurrence of Hyphen in String 1119

Add prefix Based on name 1122
Take uppercase letter of the first uppercase word and prefix it to front 1122
Take the last lowercase or uppercase letter of string and prefix it to front 1129
Take the last Uppercase character of string and prefix it to front 1132
Take the last alphanumeric character of string and prefix it to front .. 1135
Take the initial of the last word (upper or lowercase) and prefix it to the front 1138
Take the first letter (upper or lowercase) and prefix it to the front 1143
Take the initial of the first word (upper or lowercase) and prefix it to the front 1146
Take last character in string and prefix it to front 1160
Take second character of string and prefix it to front 1163
Take second character from end of string and prefix to front 1168
Take second letter at beginning of string and prefix to front 1172
Take second character at beginning of string and prefix to front 1174
Take third character from end of string and prefix to front 1176

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Element Substitution

Convert a Numeric Identification Preceded with Underscore to (#) 1181
Replace 19 with 2019 within Filename 1183
Move String to Another Location within and Replace Dot with Hyphen 1184
Substitute Dot for First Hyphen in String 1187
Replace a Space Between Numbers with a Dot 1188
Substitute 'QB' for First two Digits 1194
Replace Parentheses ( ) with Brackets [ ] 1195
Simple Substitution – Substitute 'M' for 4th Character in String 1196

TABLE Of CONTENTS

Expert's Corner - Regular Expressions

Uncategorized

Downloaded TV Shows *1199*
Isolate non-ASCII Characters *1204*
More Downloaded TV Shows *1227*
Separate Run-On Sentences with Spacing #1 ..... *1236*
Even More Downloaded TV Shows *1257*
Example of a Very Specific Rename of a Run-On Word Group *1261*
Renaming Long Filenames *1263*
Identify Every 'Nth' File from a Folder Containing Hundreds of Files *1273*
Rename TV Shows from 1x01 to s01e01 ..... *1277*
Filtering Out Filenames/Foldernames Containing Spaces *1279*
Move Files based on Creation Date to another Directory *1280*
Add Left, Right Suffix for Even/Odd Numbered Files *1284*
Specifying a Filename Containing a <space> in a BRC /Pattern Switch *1352*
Match Groupings *1354*

TABLE Of CONTENTS

Expert's Corner - JavaScript

Convert Written date to YMD 1371
Convert YMD to Written date 1373
Change Letter in any Position in String from Lowercase to Uppercase 1374
Remove Hyphens Between Date 1375
Remove Redundant Information 1376
Convert High ASCII and Unicode to ASCII 1378
Removing Repeated Words in File Names 1384
Separate Run-On Sentences with Spacing #2 (refer to Uncategorized section for #1) 1385
Replace a Series of Hyphens with Spaces, but Leave the last Hyphen 1386
Uppercase First Letter of Name 1387
Example of Using string.repeat(number) of JavaScript 1388
Change Date Format 1389
Remove Multiple Hyphens Leaving One 1392
Move Everything after Name to Beginning of Filename and Add Underscore 1394
Replace Specific Characters after a Specific Position (Replace Hyphens with Dots at Section Following date) 1397
Add Today's Date and Time to Filename 1399
Rename Filename to Match the Folder Name 1400
Rename Part of File Name from: dd-mm-yyyy to: yyyy-mm-dd ... 1401

TABLE Of CONTENTS

Expert's Corner - JavaScript cont.

Capitalize First Letter of each Word 1406
Add Random String of Letters and Numbers to
End of a Filename 1407
JavaScript Condition Filter All Files that Contain name, 'walk' 1408
Rename Filenames with Random String of Letters and Numbers 1409
Append Folder Name Levels and Rearrange Them 1411
Remove if the Prefix or Suffix Character of a Filename is
Either a Space or a Dot 1417
Shorten Music Directory Names by Removing Artist and
Leaving only Album Names 1434
JavaScript Condition Filter Files with Modified date not Matching
Exif Date Taken Timestamp 1435
Replace Prefix '18' with '20' in String 1436
Removing . (dot) from Beginning of Files 1440
Move only Renamed Files to a New Specified Folder 1442

Replace Filenames with Values based on Nested Loops 1446
A Word about ~~ 1457

Remove the Second Repetition of Filename from the End 1460
Change to Title case from 40th character 1465
Renumber in Groups of Five 1472
Remove (e##) from File Name 1473

TABLE Of CONTENTS

Expert's Corner - JavaScript cont.

Add Incrementing Counter within String *1475*
A discussion using .toLowerCase() and .toUpperCase() with arrays *1476*

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> *1487*

TABLE Of CONTENTS

Appendix

A Word on Order of Evaluation 1517
Documenting Tools 1520
Methods of Creating File Lists for Import Rename-Pairs 1523

Methods of Creating File List Structure 1524
Excel Technique (Refer to Volume 1)
You can use BRU itself to Create file list for Import into Rename Pair 1524
Ztree (or other File Manager with Batch File Creation) 1525
Utilize DOS to Create a List of Files 1525

Methods of Creating the Complete File Lists 1527
Utilize Powershell 1527
The AutoHotkey Script 1530

Accessing a Network Drive 1533
BRU Does not Support UNC Paths 1534
Mapping a Network Drive 1535
Freeze or Slow Response when Accessing Mapped Network Drive 1536
Mapped Drives not Showing in BRU 1537

Renaming with Dates Prior to January 1, 1971 1538
Working with Folders (Directories) 1543

TABLE Of CONTENTS

Appendix

Long File Paths 1546
Unicode and UNC (Universal Naming Convention) 1548
Using Substituted Drives 1549
Symbolic Links (Hard and Soft Links) 1551

Privileges - Administrator/User Access 1555
Error 740 with 'Bulk Rename Here' from Windows Explorer 1556
Remove 'Bulk Rename Here' from Windows Explorer Context Menu 1559
Copy/Move Files to the C:\ Root Directory 1560

A Word on Pad Alpha 1561
Selection Issue in Content Pane using BRU 1567

TABLE Of CONTENTS

Appendix

Regular Expressions (RegEx) Manual - Volume II)

Literal Text Match 1574
The <dot> Metacharacter (aka Wildcard) 1574
A Word on Tokens 1576

Additional RegEx Information on Greedy and Lazy 1577
A Short Discussion on Capture Groups
'Giving Up' Characters – Greedy 1589
A Short Discussion on Capture Groups
'Expanding' Characters – Lazy 1590

Additional RegEx Information on Range Quantifiers 1592

Understanding Character Classes 1599
Negated Classes 1602
Compound Character Classes 1606
Using Range Quantifiers with Classes 1611
Character Class Summary 1613

Understanding Anchors 1614
Using both Anchors in a RegEx 1617

TABLE Of CONTENTS

Appendix

Regular Expressions (RegEx) Manual - Volume II) cont.

Understanding Lookarounds 1618
Examining the Lookarounds 1618
Examining Positive Lookaheads 1626
Examining Negative Lookaheads 1636
Examining Positive Lookbehinds 1641
Examining Negative Lookbehinds 1649
Examining Combination Lookarounds 1655
Using Lookarounds with Anchors 1672
Lookaround Conditional (with Alternates) 1686

Understanding Referenced Conditional Groups 1704
Using a Referenced Capturing Group for the 'If' 1704

Understanding Relative Conditionals 1715
Understanding Subroutine Calls 1725
Understanding Branch Reset Group 1735

Understanding Atomic Groups with BRU 1746
Without Quantifier but with Alternation 1749
With Quantifier 1750
Changing the Atomic Group to a Possessive Quantifier 1753

TABLE Of CONTENTS

Appendix

Regular Expressions (RegEx) Manual - Volume II) cont.

Examining the RegEx Engine Behavior on Quantifiers 1756
Examining Possessive Quantifiers 1757
The Differences Between Greedy, Lazy and Possessive Affect on a Test String 1763

TABLE Of CONTENTS

Appendix

BRU JavaScript Manual 1766
- Understanding the JavaScript Code Entry Form Color Coding 1768
- Unicode 1769
- Unicode UTF-8 Chart (For JavaScript) 1770
- BRU Specific JavaScript Features 1778

ASCII Chart & ISO 1252 Latin-1 Char Set 1806
A Word: The Regular Expressions Engine: PCRE 1815
Software Used In Testing for this Volume 1816
Author Proclamation to Contributors 1817

Last Word 1818

Index .....

[This Page Intentionally Blank]

Welcome to Volume 2 of Bulk Rename Utility Operations Manual

This is a compendium of some of the vast knowledge found in the Forums. Where Volume I illustrated in depth each feature and function of BRU, think of Volume 2 as a BRU by Example. It contains three main sections detailing Renaming Requests - Bulk Rename Utility, RegEx and JavaScript.

It should be noted that these posts are not verbatim but have in many instances been re-edited and formatted for the purposes of this volume.

This book is meant to be a teaching guide, and where possible, I have added additional material that was not part of the original forum posts. I have, however, credited each example back to their original contributer(s). The contributor is defined as the person(s) providing the solution, not the original requester.

Some examples were found not to work. These have been included as well for your own self analysis, although corrections have been added for much of these.

The JavaScript section is included to be informative to those who understand the language. I, at this current date, do not. But I have tested all of the examples and they all work. They are more universal than RegEx and you can use them simply by copying and pasting into the JavaScript Coding Form found in 'Section #14: JavaScript'. JavaScript requires an inexpensive commercial license available through the BRU website.

Why get a commercial license? You say you are not in business nor do you plan on selling anything? However, to really appreciate the full benefit of BRU's power is to have the JavaScript capability. Look at the examples in the JavaScript section and see for yourself.

I hope you enjoy, and learn.

Timothy R. Mongeon

This book is dedicated to the hard working Admin of the BRU Forums and those Contributors who found the time to assist others
With special mention, thanks and appreciation to RegexNinja
And of course, my wife Patty of 39 years

Note: Modification of this book is not permitted without express written permission of the author.

Disclaimer – Sources

To the best of my knowledge, any sourced text that appears here, whether verbatim or highly edited, are taken from public domain sources or under Fair Use clauses. Much of the text is original, but sources are noted where appropriate. They are a reminder that information appearing within a certain section may be attributable to these sources both prior and after the appearance of the notation.

All photographic images are also, to the best of my knowledge, taken from the public domain. Many of them have been personally edited and brought up to high resolution.

Disclaimer – General Use

Always back up any datasets you wish to test. I take no responsibility for loss or damage of data. Use at your own risk. This book is a Teaching Tool, nothing more. Check your work before committing to a Renaming of files and remember the Undo Feature (refer to Volume 1 for more information).

Typographical Errors

In a book this size especially, there are probably undiscovered typographical errors. In addition to the normal text errors, there also may be more irritating ones that crept into the Analysis like less obvious <space> characters or other literals and metacharacters that may throw off some individual values (but not the overall conclusion). Look at any Regex Buddy sample runs that may shine a light on these.

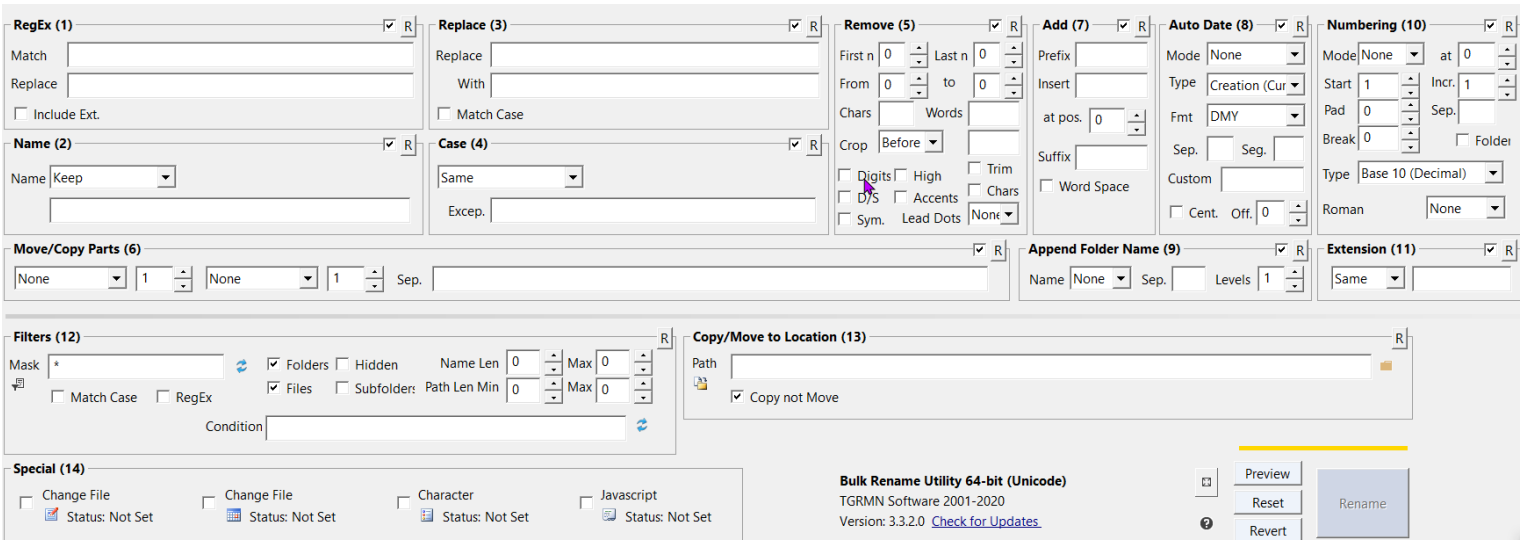
Timothy R. Mongeon

Common Errors

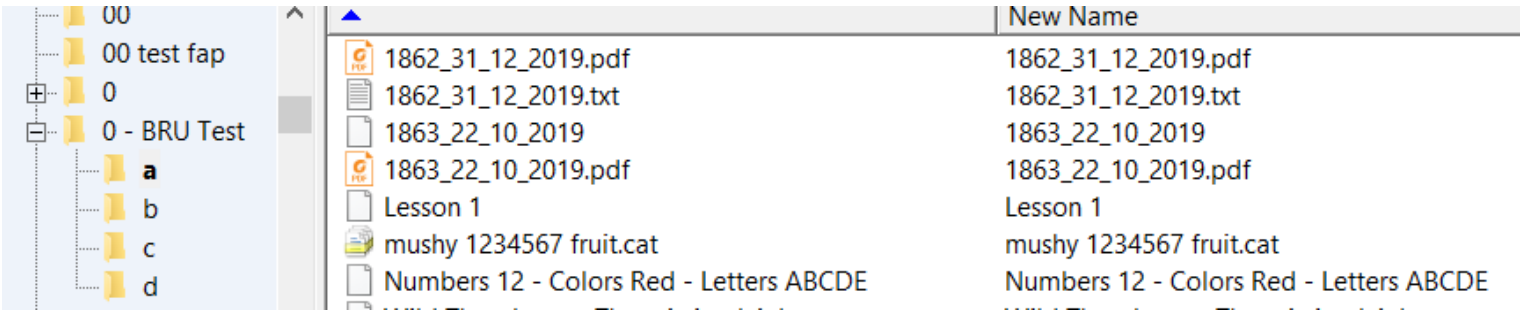
If something doesn't work as it should, here is a checklist.

BRU

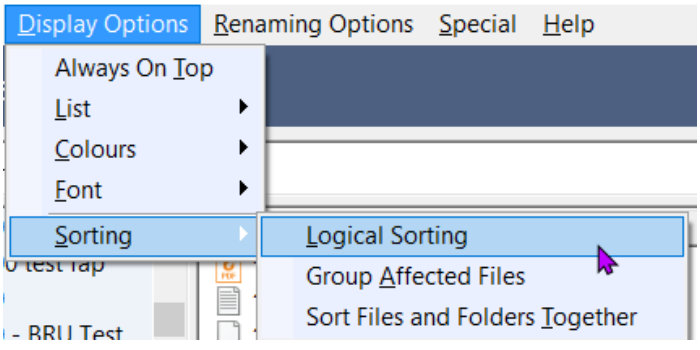
1. Make sure that you haven't inadvertently set a criteria or left one active from previous. Use the 'Reset' button located to the left of the 'Rename' Button if in doubt.



2. Check that you are in the proper directory in the Navigation Pane, and that the Content Pane is updated to reflect the directory's contents. If in doubt Press F5 to refresh the Content Pane.



3. If your sort results are unexpected, check the Sorting Options:

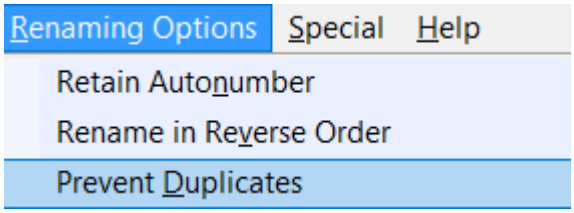


Common Errors

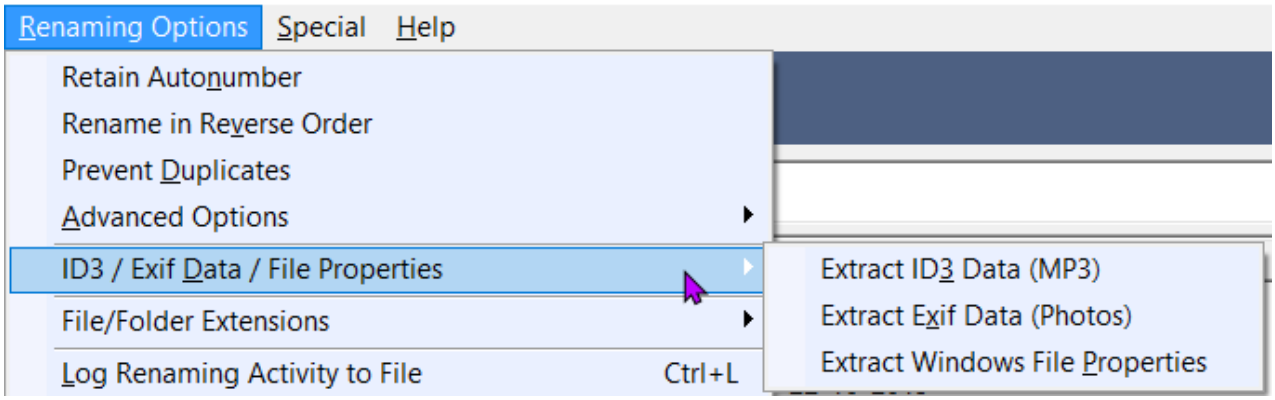
If something doesn't work as it should, here is a checklist.

BRU cont.

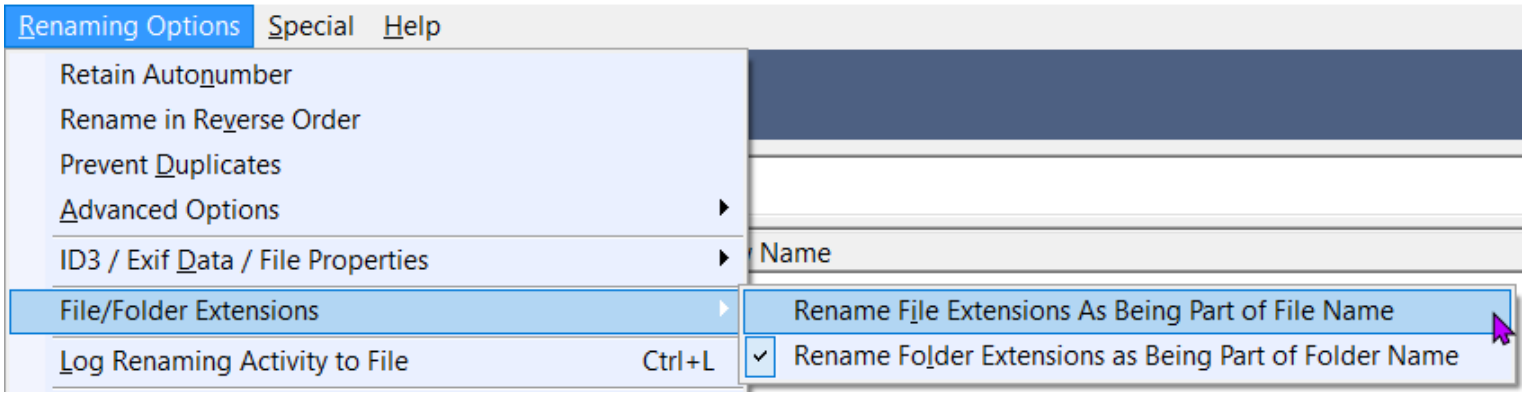
4. If your files are getting overwritten unexpectedly, enable 'Prevent Duplicates':



5. If you need to access ID3, EXIF (and for Date Taken (Original) column) or Windows File Properties, refer to these options:



6. If your samples have <dot> characters as part of the file name, enable, 'Rename File Extensions As Being Part of File Name'. This is also known as 'Ignore File Extension' taken from an older version of the BRU program.



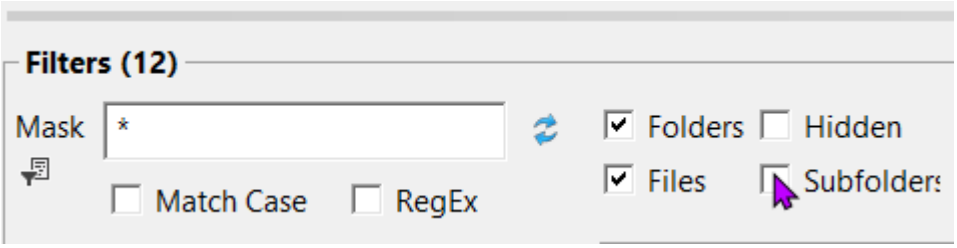
It should be noted though, that some examples do not require this.

Common Errors

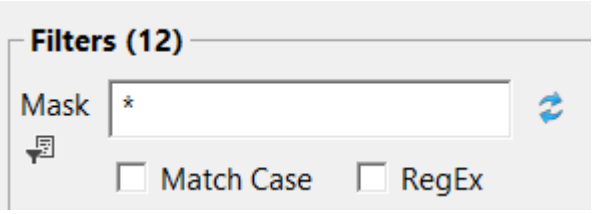
If something doesn't work as it should, here is a checklist. cont.

BRU cont.

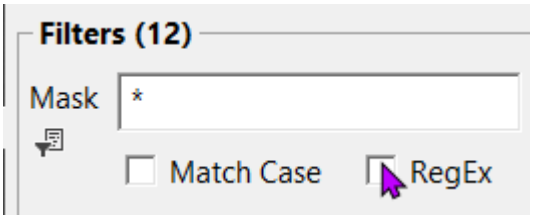
7. If you wish to include files located in sub-folders (subdirectories) in the Renaming process, then you need to turn on 'Recursion'. This is done by selecting, 'Subfolders' from 'Section #12: Filters':



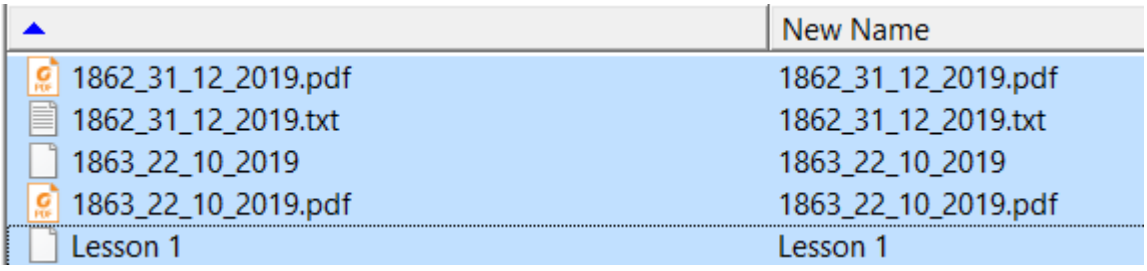
8. If you add a filter, the Content Pane must be refreshed to reflect the change. You can either press F5 or use the Blue Filter Refresh button to the right of the Mask field.



9. If you use RegEx in your Mask, you must enable 'RegEx'.



10. Files to be included in the Renaming process must be selected (highlighted) in the Content Pane.



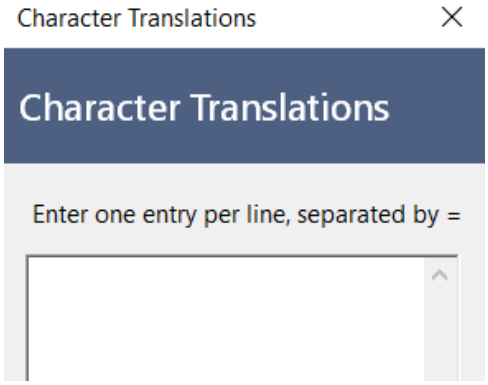
Common Errors

If something doesn't work as it should, here is a checklist. cont.

BRU cont.

11. When using Drag and Drop function to transfer files from another source into BRU, you now have the F7 key that will clear all of the entries in the Content Pane before the transfer.

12. If using Character Translation, only one entry per line and make sure there are no trailing <spaces>. Using Ctrl + Enter at the end of each line will assist in this regard.



Common Errors

If something doesn't work as it should, here is a checklist. cont.

RegEx

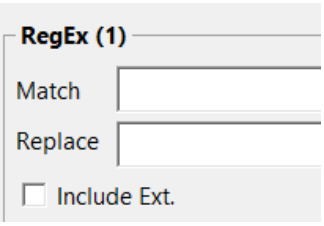
1. Check your Replacement String. Many times there are <space> characters and other literals that may easily be missed.

   Replace: \1 \2\3

   Here there is a <space> character between \1 and \2 for example.

2. Same with the RegEx itself. There can be easily missed characters that will throw off the entire evaluation. Also don't forget to check for any trailing or leading <spaces>.

3. If the samples contain <dot> characters, then you may need to enable 'Ignore File Extension' as previously discussed. In addition there is a 'Include Ext.' that may be enabled as well.



4. Along the same lines, you may have left these inadvertently enabled and this may cause problems if they are not required.

5. If you are taking the RegEx from this book, I recommend Copy and Paste. I have left the ability for you to use Copy and Paste from any textual part of the book (won't work with photos). You are still, though, not allowed to modify the book, however, high-res printing is allowed.

6. Use F8 in the Match data entry field to bring up a zoomed window to allow you more room to work if needed. The Zoom feature works with 'most' of the data fields in BRU.

7. If a RegEx is not working with certain strings, make sure that the hyphen (-) is not being confused with the dash (–). The 'hyphen' is that common character found on the keyboard with the underscore located next to the '0' key. It is erroneously called a 'dash' by most people. A 'dash' is actually a rare character that is High ASCII (0150) or –. Many editors, though, may change the 'hyphen' into a 'dash' - MS Word for one, or vice versa – if for example copying a title from an Example in the book that does use a 'hyphen' into MS Word, it may be converted into a 'dash' under certain conditions; one being if you press <ENTER> after the character. On the other hand, copying a 'dash' into a plain text editor may be converted into a 'hyphen'.

Common Errors

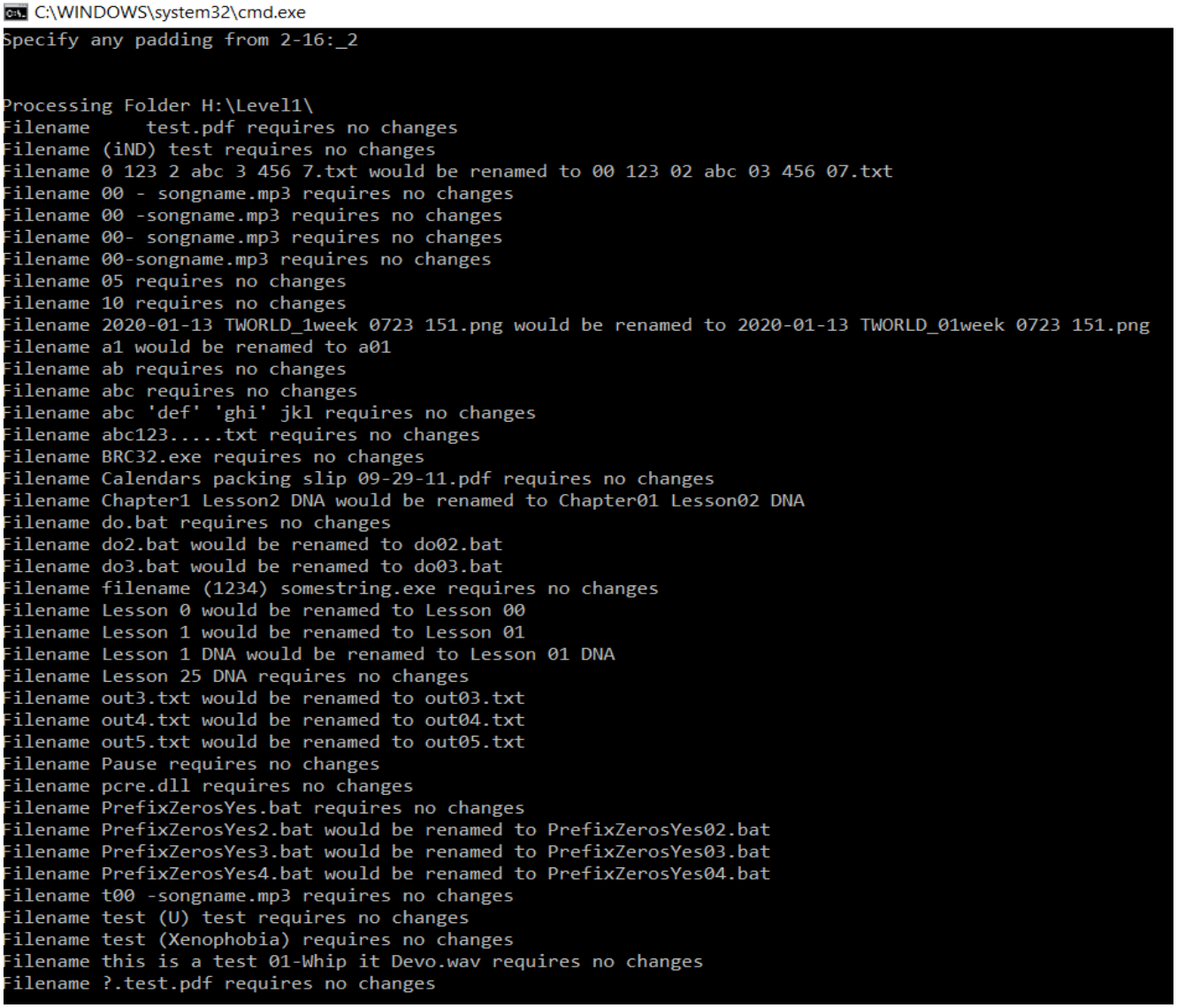
If something doesn't work as it should, here is a checklist. cont.

BRC

BRC is a separate program, a command line version of BRU that has its own advantages and disadvantages. It is a bit complex because it can run multiple RegEx and like RegEx has a very distinct syntax. I have many BRC examples scattered about, but there is not a section dedicated to the program.

1. Many examples of BRC in this book display the code in 'Preview' mode. Meaning, when the code is run, it will display an output stating – this will be renamed or this won't be renamed, but without renaming. This is done intentionally, because unlike BRU, there is no going back – no second chance – no Undo. To perform the actual renaming, you have to add /Execute at the end of the line that calls the BRC program.

Preview Mode:

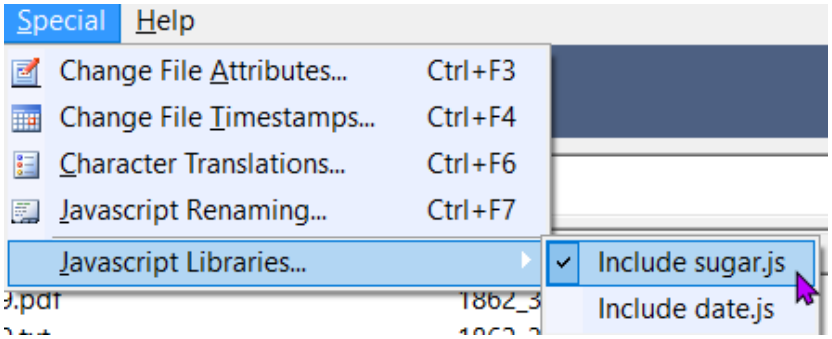


Common Errors

If something doesn't work as it should, here is a checklist. cont.

JavaScript

1. If an example is using any of the functions from either of the sugar.js or date.js built-in libraries, these must be enabled.



2. This book contains a brief JavaScript language reference in the Appendix. I advise you to look online and also obtain some books on the subject.

3. JavaScript is a markup language meant for programming web pages in supported browsers. Much of the language has to do with this. It is also a dedicated language that can lend itself to the development of desktop applications as well.

   The people at TGRMN, I think, have found a unique way to employ the language in renaming files. It is much faster and in many cases, if you understand the language, more efficient and of course more powerful than using Regular Expressions. It also supports a Global switch meaning that it can perform multiple matches within its code; something currently both BRU RegEx and BRC do not support. TGRMN have also added some of their own constants and functions to the language specific to BRU.

   It is hoped that you will consider supporting future development of this and the freeware versions of both BRU and BRC by purchasing a license. Other freeware programs encourage a paid registration. This is no different.

4. The JavaScript syntax is very strict. It is Case Sensitive and much of the language is in lowercase or mixed case. If a line is not working and the other syntax is correct, it may be related to this.

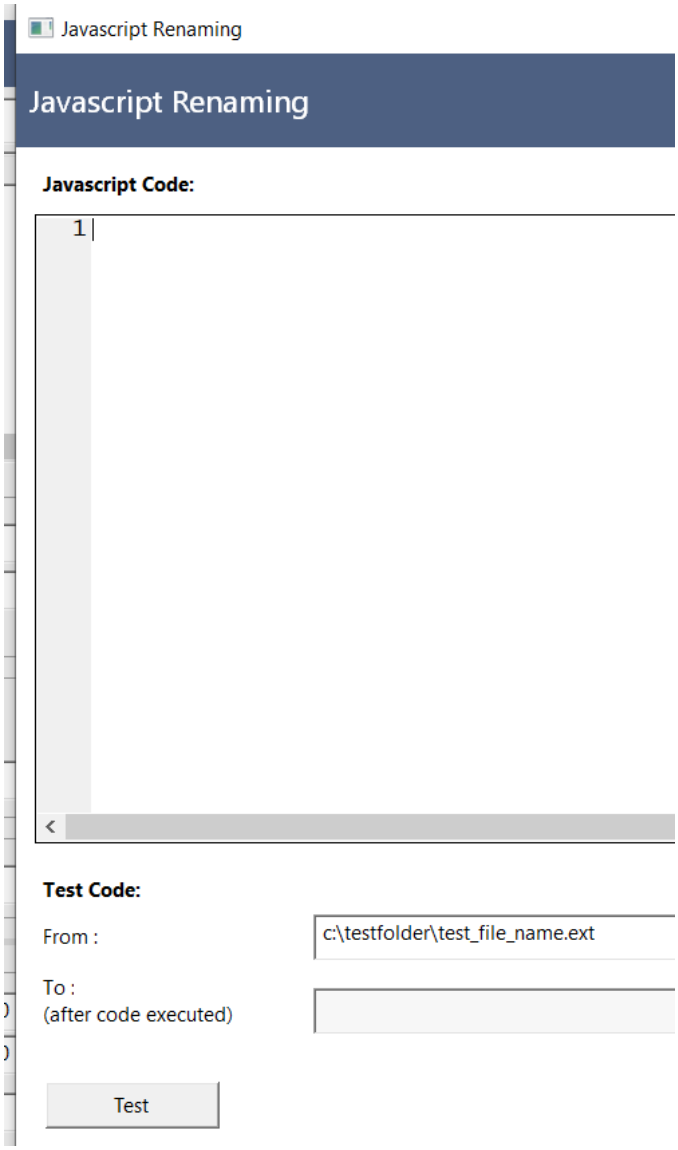
5. BRU JavaScript is written to support ECMAScript 5. If your JavaScript uses some of the newer code from ECMAScript 6, this may not be supported. Also BRU JavaScript may not support some Methods, Functions and Properties that would have no application in BRU's functionality.

Common Errors

If something doesn't work as it should, here is a checklist. cont.

JavaScript cont.

6. There is a TEST button in the JavaScript code entry form. Use this to test your syntax.



One of the clues of a syntax error is that the word in error is in black and white and should be blue or another color that indicates the syntax is correct..

Common Errors

If something doesn't work as it should, here is a checklist. cont.

JavaScript cont.

The color coding system used in the JavaScript code entry form is to help with syntax and errors.

Within function

Item	Color	Correct	Bad	Neutral
Keywords	blue	*	Black & White	
strings	Reddish-Brown	*	"	
Variables	Black & White			*
Numerals	Black & White			*
Function name	Black & White			*

Outside Function

Item	Color	Correct	Bad	Neutral
Variables	purple	*		

All

Item	Color	Correct	Bad	Neutral
Operators	silver-green	*		
parenthese/brackets	silver-green	*		
Braces, Semicolon				

If your syntax is correct, you can see a preview of what the renamed file will be by looking at the Test Code block:

In the 'From', it will have a test string but you can change this to any test string you want (do not have to have a physical file, just change the name):

From: c:\testfolder\(e85)RandomName1.txt

The 'To' will show a preview of the renamed file after clicking 'Test'

To: c:\testfolder\RandomName1.txt

So even if you don't own a commercial license, BRU allows you to see an execution of the JavaScript virtually.

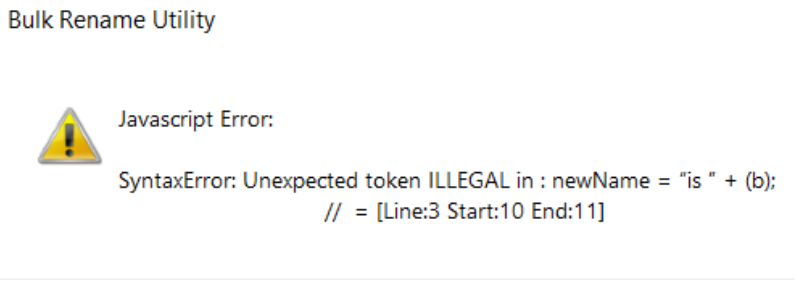
Common Errors

If something doesn't work as it should, here is a checklist. cont.

All

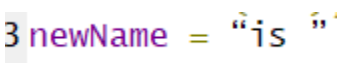
1. In addition to hyphens being turned into 'dashes', I wrote this book using Microsoft Office products. Microsoft uses High ASCII 145 and ASCII 146 for the single quotes instead of the plain text editor usage of ASCII 39. Additionally, Microsoft uses high ASCII 147 and ASCII 148 for the double quotes. Why? Because they look better than the ASCII 34 used in plain text editors.

   What has this got to do with Bulk Rename Utility? Because if you copy and paste an example RegEx or JavaScript that contains one of these characters, it may cause problems. In JavaScript, for example it will generate this error:

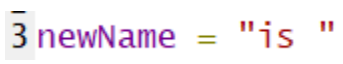


   A clue to this can be found in the JavaScript code entry form…

   You have:



   instead of:



   Notice the color difference? Black and white characters that should be reddish brown are a clear indication of a syntax problem.

   Also see the difference in the double quotes? The more curvy and refined characters belong to the Office products. That is the cause of the error.

About the Format

The layout is very similar to the first volume. If something seems rather small on the page, all photos are high resolution and the PDF can be zoomed (magnified) without distortion to view any difficult to read sections, or for that matter, colored text.

The coloring may be a bit difficult for some people. I am hopeful to come out with a color-friendly version that will reduce some of the lighter colors to darker brown textures, but to get the most out of the book, I recommend this full color version.

There isn't a heck of a lot of choices in colors that work well, so I used what I had. Some of the lighter colors are used for text that I deliberately did not want to distract from other text on the page – for example, the Forum Contributors. Others are used to provide highlight distinction, like the bright green used for the word, 'null'. Whatever, the choice, the book is designed to be zoomed in your reader so that any areas that otherwise would be difficult to read can easily be seen.

If you still have problems with this in your PDF reader then it is time to get another PDF reader. In addition many PDF Readers have accessibility options that can override any color scheme.

It should also be mentioned that any coloring is mainly specific to each section, and in many cases each page. Any meaning therein does not extend to other sections or pages. There are exceptions which will become obvious. The coloring is for clarification and to allow the user to more easily follow the topic of discussion.

The book requires a reader that can reliably render the graphics. If you are using a reader of lesser quality, for example, in the JavaScript sections, you will notice white lines that appear in the colored blocks of text. Readers I recommend are:

Foxit Reader (https://www.techspot.com/downloads/2713-foxit-reader.html)
Downloads CLEAN version of Foxit without added adware

Javelin Reader (https://www.drumlinsecurity.com/javelindownloads.html)

Nitro Pdf Reader (https://download.cnet.com/Nitro-PDF-Reader-64-bit/3000-10743_4-75206289.html)

Adobe Acrobat Reader 2020 Classic (ftp://ftp.adobe.com/pub/adobe/reader/win/)
(on my system Adobe is slow in paging)

Readers not recommended:

MUPDF
SumatraPDF
SLIM PDF Reader
Haihaisoft_PDF_Reader

Expert's Corner —

Bulk Rename Utility

Tim Note:

The following are taken from the public domain BRU Forums. Contributors of the final solutions used will be noted.

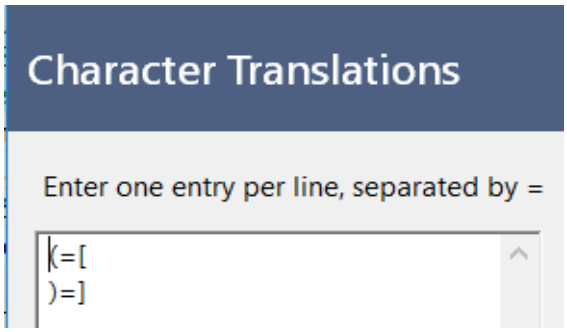
The best method of learning is to study from others. Here are many examples that you can try for yourself. I have tested every single example and have included photos with each to demonstrate that they work Forum posts have been selected based on their usefulness and tutorial benefit.

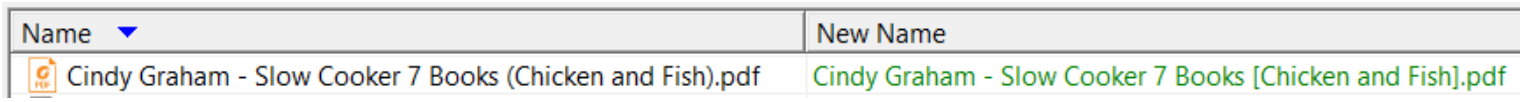
Multiple runs require that criteria settings be reset between runs using the 'Reset' button found next to the 'Rename' button located at the bottom right of the interface.

Bulk Rename Utility

Changing parentheses to brackets - from ( ) to [ ]

Use 'Section #14: Character Translation':





Tim Notes:

1. There are many methods to accomplish this, but this is a good example of using Character Translations.

Forum Contributor: Admin

Bulk Rename Utility

add Alpha order name (alphabet A-Z serial numbering)

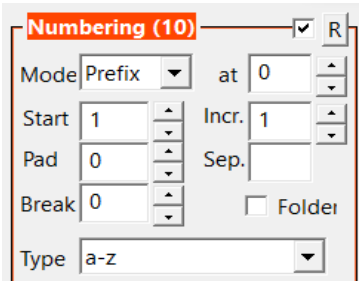
1.text to a.text
2.text to b.text
3.text to c.text

Etc.

'Section #5: Remove' -

First n:1

'Section #10: Numbering' –



Mode: Prefix
Type: a-z

Name	New Name
1.text	a.text
2.text	b.text
3.text	c.text

Forum Contributor: Stefan

Tim Notes:

1. The Heading title states 'A-Z' that would otherwise indicate Mode A-Z for uppercase, but Mode a-z lowercase is used in the example.

Bulk Rename Utility

Remove .com, all brackets, hyphens and dashes and maintain <space> before year

[Word] - 2005.ext	to	Word 2005.ext
[Word] 2006.ext	to	Word 2006.ext
[Word1-Word2] - 2009.ext	to	Word1Word2 2009.ext
[Word1-Word2] 2010.ext	to	Word1Word2 2010.ext
Word1-Word2 - 2007 r.ext	to	Word1Word2 2007 r.ext
Word1-Word2 - 2011.ext	to	Word1Word2 2011.ext
Word1-Word2.com - 2009 r.ext	to	Word1Word2 2009 r.ext
Word1-Word2.com - 2012.ext	to	Word1Word2 2012.ext

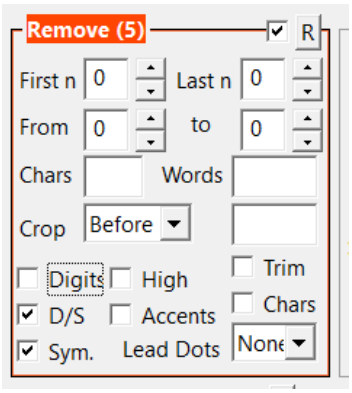
Requires two separate runs:

First run:

'Section #3: Replace' -

Replace: .com
With: null

'Section #5: Remove' –



D/S: Enable
Sym.: Enable

After First Run –

Name	New Name
Word1-Word2.com - 2012.ext	Word1Word2 2012.ext
Word1-Word2.com - 2009 r.ext	Word1Word2 2009 r.ext
Word1-Word2 - 2011.ext	Word1Word2 2011.ext
Word1-Word2 - 2007 r.ext	Word1Word2 2007 r.ext
[Word] 2006.ext	Word 2006.ext
[Word] - 2005.ext	Word 2005.ext
[Word1-Word2] 2010.ext	Word1Word2 2010.ext
[Word1-Word2] - 2009.ext	Word1Word2 2009.ext

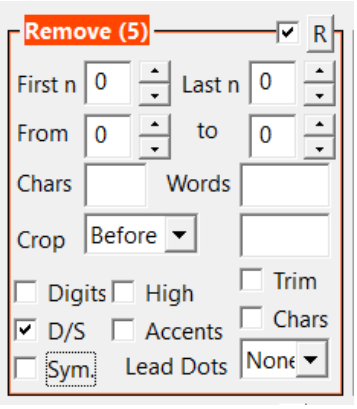
Notice <double space> in Word 2005.ext and Word1Word2 2009.ext? The second run will fix that.

Bulk Rename Utility

Remove .com, all brackets, hyphens and dashes and maintain <space> before year cont.

Second run –

'Section #5: Remove' –



D/S: Enable

After second run:

Name	New Name
Word1Word2 2012.ext	Word1Word2 2012.ext
Word1Word2 2009 r.ext	Word1Word2 2009 r.ext
Word1Word2 2011.ext	Word1Word2 2011.ext
Word1Word2 2007 r.ext	Word1Word2 2007 r.ext
Word 2006.ext	Word 2006.ext
Word 2005.ext	Word 2005.ext
Word1Word2 2010.ext	Word1Word2 2010.ext
Word1Word2 2009.ext	Word1Word2 2009.ext

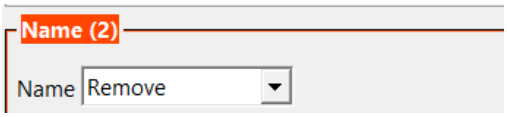
Forum Contributor: Stefan

Bulk Rename Utility

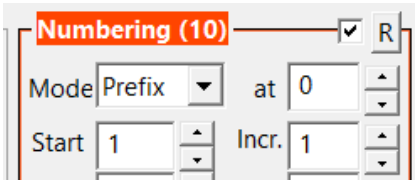
Completely renaming files sequentially (1,2,3...etc)

CMR_04004302.csv to 1.csv
CMR_04004303.csv to 2.csv
CMR_04004304.csv to 3.csv

'Section #2: Name' –



'Section #10: Numbering' –



Mode=Prefix
Start=1
Inc=1

1. Make sure the original names in the right-pane are sorted properly.
   a. Click the 'Name' title column to sort by name and select 'Logical Sorting' from Display Options Menu (if needed).

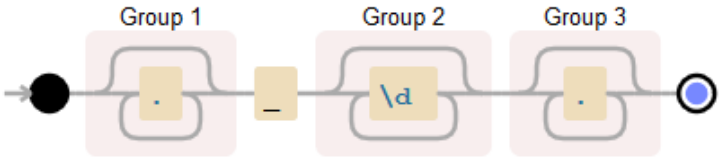
Name	New Name
CMR_04004302.csv	1.csv
CMR_04004303.csv	2.csv
CMR_04004304.csv	3.csv

Tim Contribution –

The following RegEx can be substituted for 'Section #2: Name' above.

Match: (.*)_(\d*)(.*)
Replace: \3

Name	New Name
CMR_04004302.csv	1.csv
CMR_04004303.csv	2.csv
CMR_04004304.csv	3.csv



Forum Contributor: truth

Tim Tidbit:

1. The highest number you can enter in the 'Start' field of 'Section #10: Numbering' is 2147483647 which is the highest number in signed-binary using only 32bits. This doesn't limit the auto increment value applied. Filenames would still be incremented with 2147483648, 2147483649, etc.

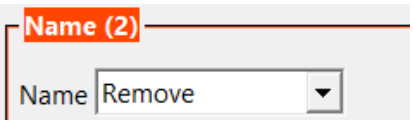
Bulk Rename Utility

Replace Filename with Folder name and append Sequential Number (Autonumber)

File 1.pdf to A Test Folder_Page1.pdf
File 3.pdf to A Test Folder_Page2.pdf
File 4.pdf to A Test Folder_Page3.pdf
File 10.pdf to A Test Folder_Page4.pdf

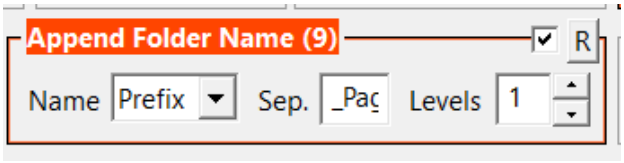
1. Remove Name

'Section #2: Remove' –



2. Append Folder name and Add Separator Text, '_Page'

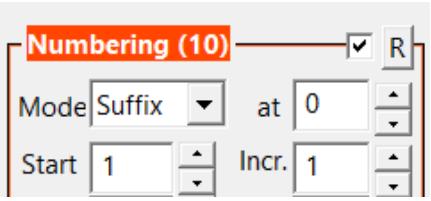
'Section #9: Append Folder Name' -



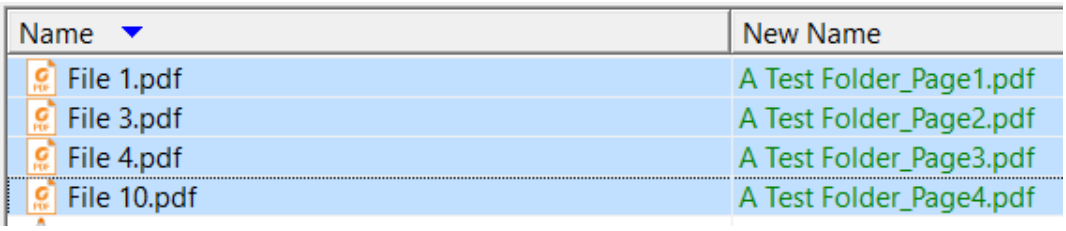
Name: Prefix
Sep: _Page
Levels: 1

3. Set Autonumber to increment +1

'Section #10: Numbering' –



Mode=Suffix
Start=1
Inc=1



Tim Notes:

1. This is done in one run

Forum Contributor: Admin, viral316

Bulk Rename Utility

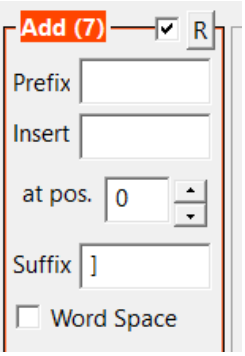
Remove underscore and add brackets around last word in string

foo bar_Disk1.xyz	to	foo bar [Disk1].xyz
foo bar_Disk2.xyz	to	foo bar [Disk2].xyz
foo bar_Disk3.xyz	to	foo bar [Disk3].xyz
goo par_Disk1.xyz	to	goo par [Disk1].xyz
goo par_Disk2.xyz	to	goo par [Disk2].xyz
More of the same_Disk0.xyz	to	More of the same [Disk0].xyz
Test more_DiskA.xyz	to	Test more [DiskA].xyz

1. 'Section: 3: Replace' –

Replace: <underscore>
With: <space> left bracket ' ['

2. 'Section #7: Add' –



Suffix: right bracket '] '

Name	New Name
foo bar_Disk2.xyz	foo bar [Disk2].xyz
foo bar_Disk3.xyz	foo bar [Disk3].xyz
goo par_Disk1.xyz	goo par [Disk1].xyz
goo par_Disk2.xyz	goo par [Disk2].xyz
More of the same_Disk0.xyz	More of the same [Disk0].xyz
Test more_DiskA.xyz	Test more [DiskA].xyz

Forum Contributor: therube

Bulk Rename Utility

Renaming Folders during Move

YYYY-MM-DD 2019-XX-01 thru 2019-XX-31

Observations:

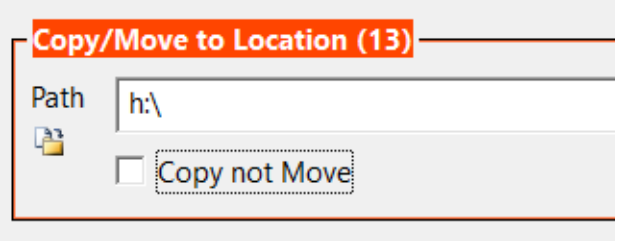
1. Move multiple folders to new location and rename month as required.
2. folder name format inconsequential.
3. Where XX refers to MM 01-12

One run for each month change required.

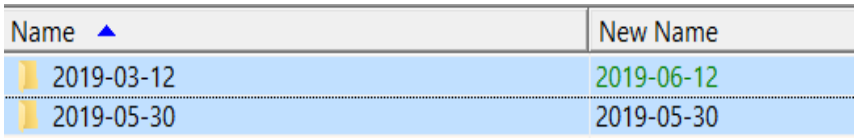
1. 'Section #3: Remove' –

In this example, the month is being changed from March to June.

2. 'Section #13: Copy/Move to Location' –



In this example, folders will be moved to off the root of the H: drive



Contributor Notes:

1. It does not have to be a different location. You can specify the same location if you want which will rename existing folders.
2. Make sure you select all the folders you want to include.
3. Press F5 to refresh and you will see the new folders.

Tim Notes:

1. The renaming has no effect on the files contained herein.
2. Only files can be copied or moved, not the directories themselves with their file contents.
3. If directories are copied/Moved, the behavior of BRU is to create 'empty' directories at the new location. The original directories are left unchanged (regardless if files are contained).
4. If directory does not exist, it will be created.

Forum Contributor: growltiger

Bulk Rename Utility

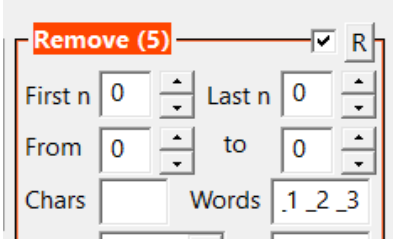
Rename Duplicate filenames W/original filename and Delete them

736.ipt	to	Ignore
736_1.ipt	to	736.ipt
737.ipt	to	Ignore
737_1.ipt	to	737.ipt
737_2.ipt	to	737.ipt
738.ipt	to	Ignore
738_1.ipt	to	738.ipt
738_2.ipt	to	738.ipt
738_3.ipt	to	738.ipt

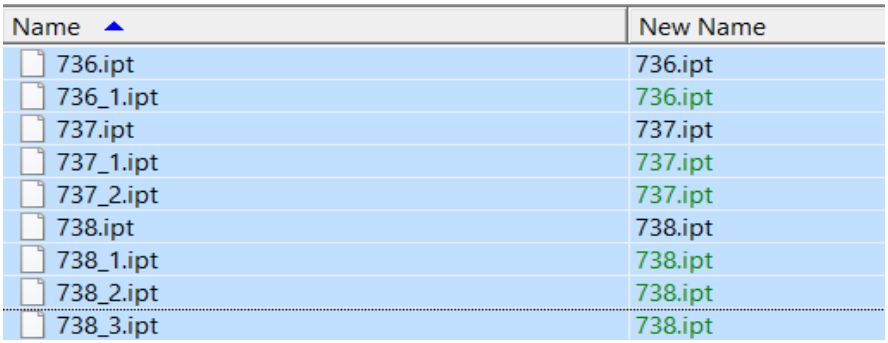
Observations:

1. Folder contains duplicate files. Duplicates are labeled as _1, _2, _3, etc.

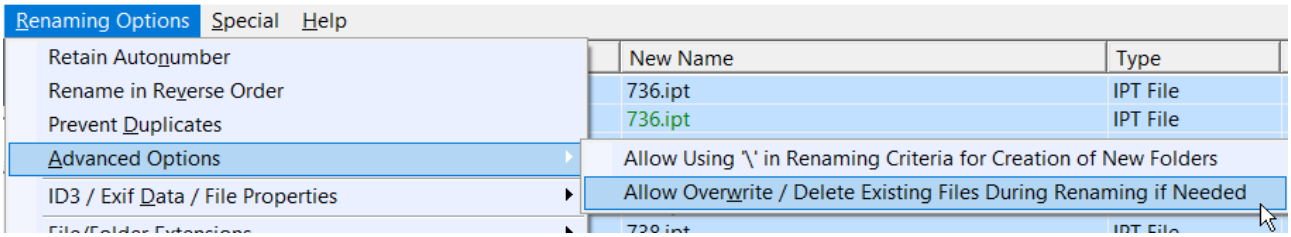
'Section #7: Remove' –



Enter words as _1 _2 _3 separated by a <space> as required for multiple entries in this field.



2. Set 'Allow Overwrite / Delete Existing Files During Renaming if Needed' from 'Advanced Options' of 'Renaming Options Menu'. This will delete the duplicate files as they are renamed.

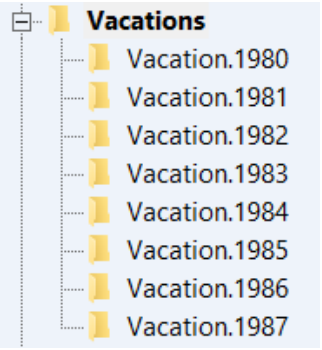


Forum Contributor: therube

Bulk Rename Utility

Rename files based on folder name

Directory structure:



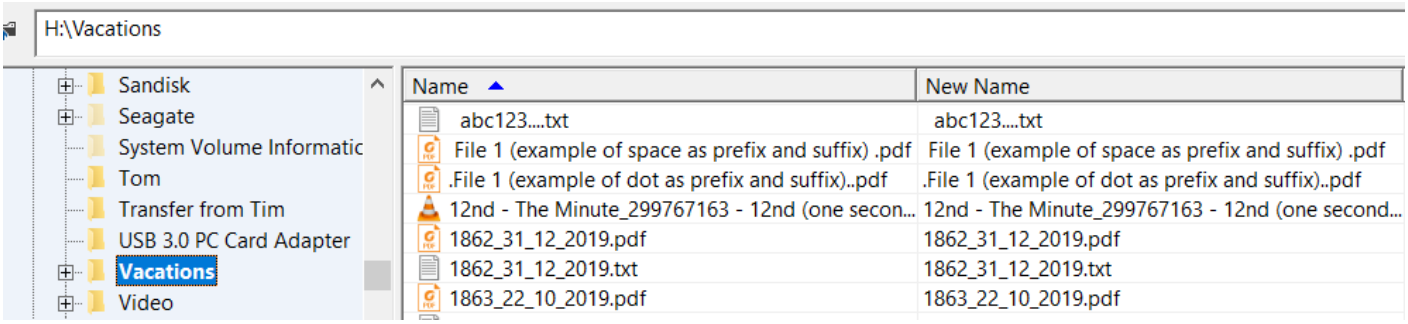
Observations:

1. Root folder named 'Vacations'
2. Subfolders named above.
3. All files contained herein to be renamed with current folder name appended with sequential autonumber.

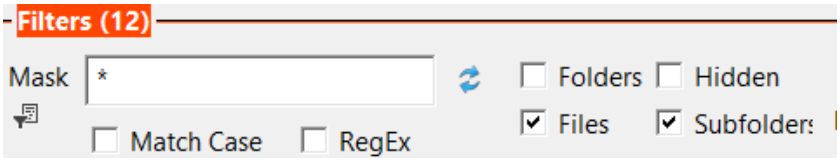
i.e., In the Vacation.1983 folder all files renamed as:

Vacation.1983.001.ext
Vacation.1983.002.ext
Vacation.1983.003.ext
Vacation.1983.004.ext

1. In the Navigation Pane (containing the Tree), select the root folder, "H:\Vacations".



2. In 'Section #12: Filters' – enable only "Files" and "Subfolders" to turn on recursion.

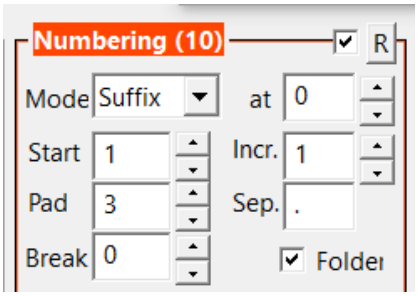


Turn on recursion by selecting Subfolders

Bulk Rename Utility

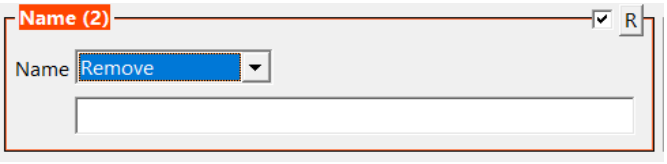
Rename files based on folder name cont.

3. 'Section #10: Numbering' –



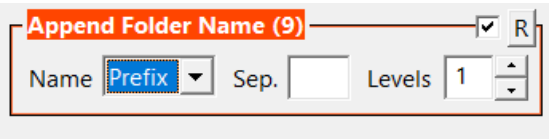
Set Mode = Suffix
Pad = 3
Sep. = . (dot character)
Folder: Enable

4. 'Section #2: Name' –



Name: Remove

5. 'Section #9: Append Folder Name' –



Name: Prefix

6. Make sure you have your files selected (press Ctrl +A or use the 'Select All' option from the Actions Menu).

Name	New Name	Sub Dir.
.File 1 (example of dot as prefix and suffix)..pdf	Vacations.001.pdf	
abc123.....txt	Vacations.002.txt	
test .pdf	Vacations.003.pdf	
Test....pdf	Vacations.004.pdf	
ReadMe.txt	Vacation.1980.001.txt	\Vacation.1980
20191224_182245 (bottom of bag showing melt...	Vacation.1981.001.jpg	\Vacation.1981
20191207_172229.jpg	Vacation.1982.001.jpg	\Vacation.1982
CBS.log	Vacation.1983.001.log	\Vacation.1983
Normal.dot	Vacation.1984.001.dot	\Vacation.1984
Anymus_192638322 - Inkyz.mp3	Vacation.1985.001.mp3	\Vacation.1985
Autoerotique - WTF (K-MAC Trap VIP)_96732707...	Vacation.1985.002.mp3	\Vacation.1985
AWOLNATION - Run (B52 Remix)_217279053 - B...	Vacation.1985.003.mp3	\Vacation.1985
b&B_296170807 - psupac.mp3	Vacation.1985.004.mp3	\Vacation.1985
Word word 01x 88 - variable_word (other words)...	Vacation.1986.001.txt	\Vacation.1986
Word1-Word2.com - 2009 r.ext	Vacation.1987.001.ext	\Vacation.1987
[Word1-Word2] 2010.ext	Vacation.1987.002.ext	\Vacation.1987
[Word] 2006.ext	Vacation.1987.003.ext	\Vacation.1987

You will see the proposed changes in the Content Pane as a preview before you commit to the Rename.

Tim Notes:

1. I sorted the listing on left by Sub Dir. column for clarification (click on 'Sub Dir.' column title in the title bar).

Forum Contributor: Gabriel.

Bulk Rename Utility

Cropping file extension

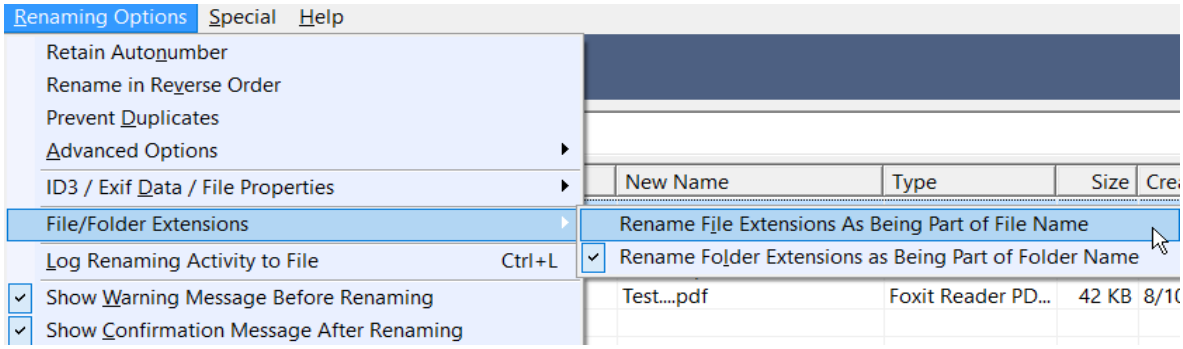
Tim Notes:

Determine When to Select, 'Rename Extensions as Being part of Filename'

When it comes to renaming, BRU treats Files and Folders the same unless something identifies one from the other. With files it is normally the file extension. The period (dot) following the filename identifies a file rather than a folder. In this modern age, however, when filenames and folder names can be longer than the 'eight character name/three character extension' of ye olden days, it is not uncommon for dot characters to be used multiple times within a file or folder name.

When this happens, BRU can get confused. If you have a folder that has a dot character in the name, then set: 'Rename Folder Extensions as Being Part of Folder Name' from the File/Folder Extensions submenu of the Renaming Options Menu. Folders with this naming convention are so common that BRU has already set this as the default.

Not so common are filenames with a dot character in the name itself. When you have filenames that consist of these, then set 'Rename File Extensions as Being Part of Folder Name' from the File/Folder Extensions submenu of the Renaming Options Menu (aka 'Ignore File Extensions'), because, depending on what and how you are renaming, this may or may not have an effect.

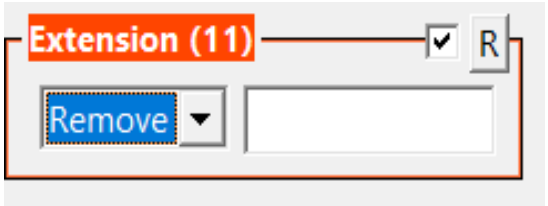


May2018Report.pdf.backup_2018_12_30_17_46_22	to	May2018Report.pdf.
FreightTaxes2018.pdf.backup_2018_08_15_23_47_12	to	FreightTaxes2018.pdf
mp4BigBuckBunny_115k.mov.mp4	to	mp4BigBuckBunny_115k.mov

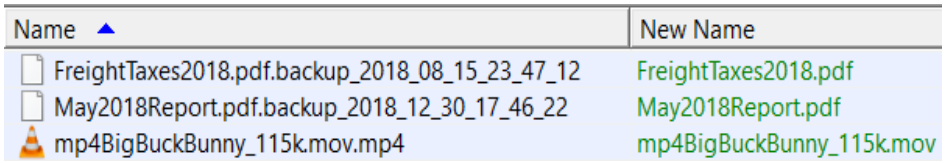
Observations:

1. Since there are dot characters in the filename, need to have Ignore File Extensions set as explained above.

'Section #11: Extension' -



set to Remove



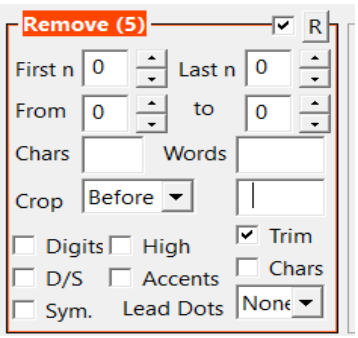
Forum Contributor: therube

Bulk Rename Utility

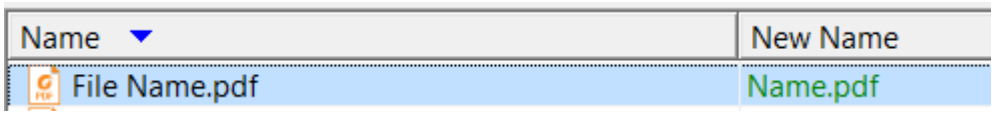
Remove First Word

File Name.pdf to Name.pdf

1. 'Section #5: Remove' –



Set:
Crop: Before
enter <space> in data field.
Trim: Enabled to remove extra <spaces>.



Forum Contributor: Admin

Bulk Rename Utility

Adding space between hyphens anywhere in string

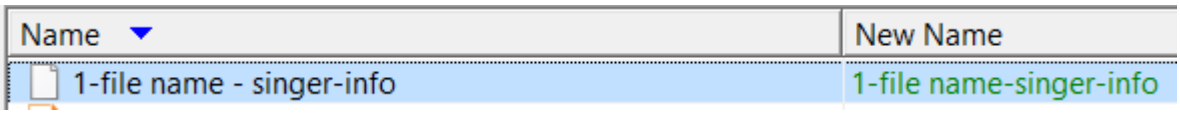
1-file name - singer-info to 1 - file name - singer - info

Two runs:

First run –

1. 'Section #3: Replace' –

Replace: <space> - <space>
With: -



Second run –

1. 'Section #3: Replace' –

Replace: -
With: <space> - <space>

Forum Contributor: therube

Bulk Rename Utility

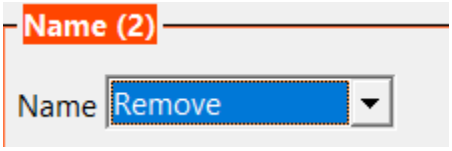
Add Creation Date as suffix

Sample:

3052503155104.Bourjois_FDTAirMat_01_1.png to 2018-08-06 at 12.21.09 AM.png

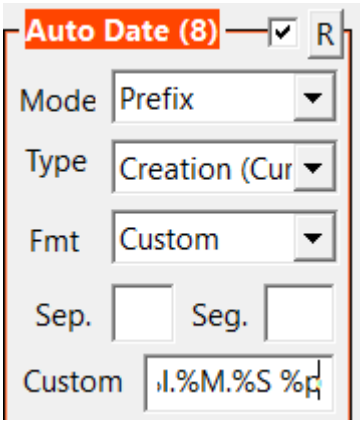
Name	New Name	Type	Size	Created
3052503155104.Bourjois_FDTAirMat_01_1.png	3052503155104.Bourjois_FDTAirMat_01_1.png	PNG File	6 KB	8/6/2018 12:21:09 AM

1. 'Section #2: Name' –



Name: Remove

2. 'Section #8: Auto Date' –



Set:
Mode: Prefix
Type: Creation (Current)
Fmt: Custom
Custom: %Y-%m-%d at %I.%M.%S %p

Tim Notes:

1. For information regarding Custom formats, refer to 'Volume 1 Bulk Rename Utility Operations Manual' under 'Section #8: Auto Date'

Name	New Name
3052503155104.Bourjois_FDTAirMat_01_1.png	2018-08-06 at 12.21.09 AM.png

Forum Contributor: therube

Bulk Rename Utility

Renaming with odd or even numerals using Autonumber

Sample:

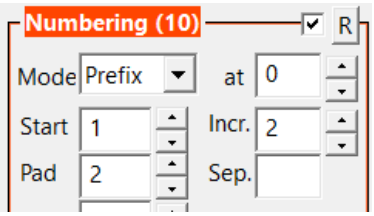
	Odd		Even	
01.jpg	to	01.jpg	or	02.jpg
02.jpg	to	03.jpg	or	04.jpg
09.jpg	to	05.jpg	or	06.jpg
11.jpg	to	07.jpg	or	08.jpg
12.jpg	to	09.jpg	or	10.jpg
15.jpg	to	11.jpg	or	12.jpg

1. 'Section #2: Name' –

Name: Remove

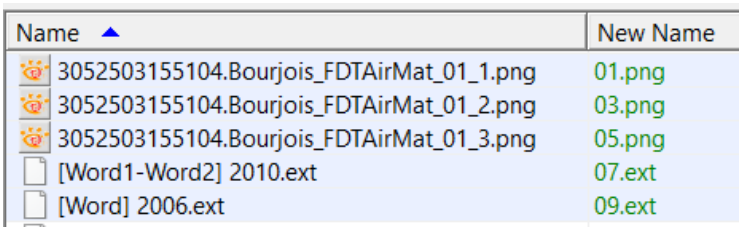
For Odd –

2. 'Section #10: Numbering' -



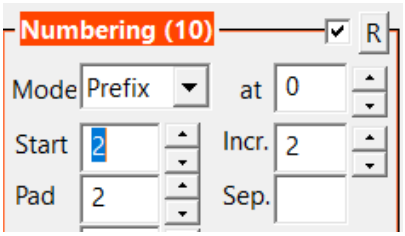
Set:

Start: 1
Pad: 2
Incr(ement): 2



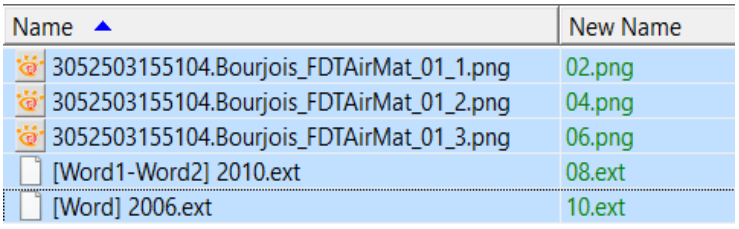
For Even -

2. 'Section #10: Numbering' -



Set:

Start: 2
Pad: 2
Incr(ement): 2



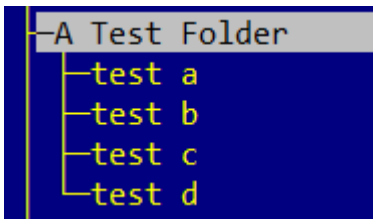
Forum Contributor: jimwillsher (Original Publisher BRU)

Bulk Rename Utility

Use 'Folder' to Reset Autonumber for each Subdirectory's Sequential numbering of Files

Tim Notes:

Directory structure:



Observations:

1. There are a number of files in each subdirectory as well as in the root directory, 'A Test Folder'.
2. Files in each of the directories need to be renamed with a sequential number.
3. Each directory's numbering must be isolated from the other.
   a. The increment value for the sequential count is reset for each directory back to 1

   i.e.,

sample:

subdirectory test a has 3 files
file 1 file 2 file 3 file 4

subdirectory test b has 4 files
file 1 file 2 file 3 file 4

after renaming:

subdirectory test a has 3 files	subdirectory test b has 4 files
001	001
002	002
003	003
	004

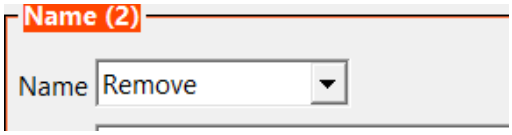
not…

001	004
002	005
003	006
	007

Bulk Rename Utility

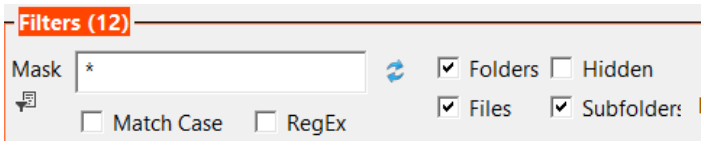
Use 'Folder' to Reset Autonumber for each Subdirectory's Sequential numbering of Files cont.

1. 'Section #2: Name' -



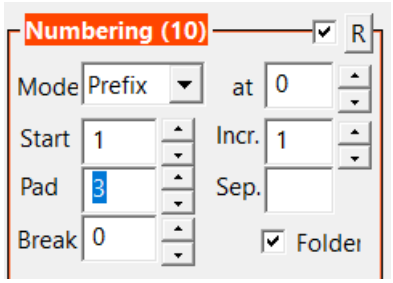
Name: Remove

2. 'Section #12: Filters' –



Subfolders: Enable
Subfolders indicates Recursion so renaming can process all files at once.

3. 'Section #10: Numbering' –



Set:
Mode: Prefix
Start: 1
Pad: 3
Folder: Enabled

This will rename the files in each directory, individually, starting at 001.ext ... 999.ext.

Name	New Name	Sub Dir.
3052503155104.Bourjois_FDTAirMat.png	_001	
[Word1-Word2] 2010.ext	_002	
[Word] 2006.ext	_003	
foo bar_Disk2.xyz	_004	
Vacation.1980	_005	
Vacation.1982	_006	
Word1-Word2.com - 2009 r.ext	_007	
1863_22_10_2019.txt	_001	\test a
738.ipt	_002	\test a
AckleyDustin.txt	_003	\test a
Anymus_192638322 - Inkyz.mp3	_001	\test b
Autoerotique - WTF (K-MAC Trap VIP)_96732707 ...	_002	\test b
AWOLNATION - Run (B52 Remix)_217279053 - B...	_003	\test b
b&B_296170807 - psupac.mp3	_004	\test b
[Word1-Word2] 2010.ext	_001	\test c
[Word] 2006.ext	_002	\test c
Word1-Word2.com - 2009 r.ext	_003	\test c
3052503155104.Bourjois_FDTAirMat_01_3.png	_001	\test d
AWOLNATION - Run (B52 Remix)_217279053 - B...	_002	\test d
b&B_296170807 - psupac.mp3	_003	\test d
test a	test a	
test b	test b	
test c	test c	
test d	test d	

Bulk Rename Utility

Use 'Folder' to Reset Autonumber for each Subdirectory's Sequential numbering of Files cont.

Tim Notes:

1. By indicating 'Folder' in the 'Section #10: Numbering', this resets the sequential counter each time a new directory is processed.
2. Explained extensively under the Folder option of 'Section #10: Numbering' in Volume I.

Forum Contributor: therube

Bulk Rename Utility

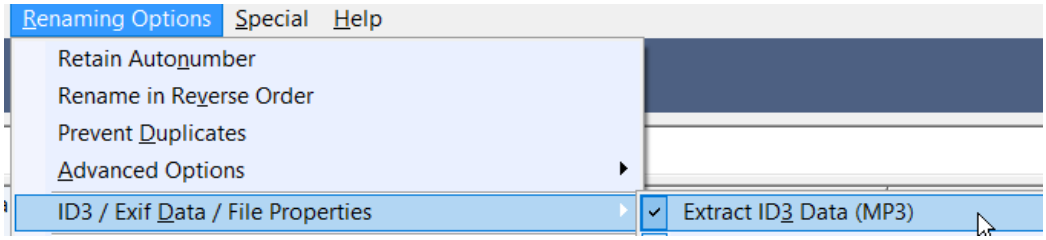
How to add Title from mp3 files to file name Using ID3 Data Extraction

001.mp3	to	001 Vegan Intolerance.mp3
002.mp3	to	002 See What I've Become.mp3
003.mp3	to	003 Tupelo Honey.mp3

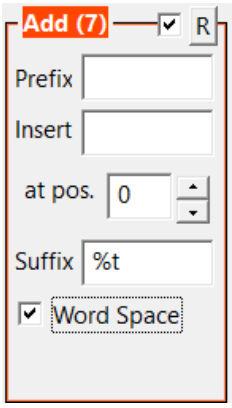
Observations:

1. Each of the MP3 files has a song title.
2. Extract the song title from the ID3 tag.

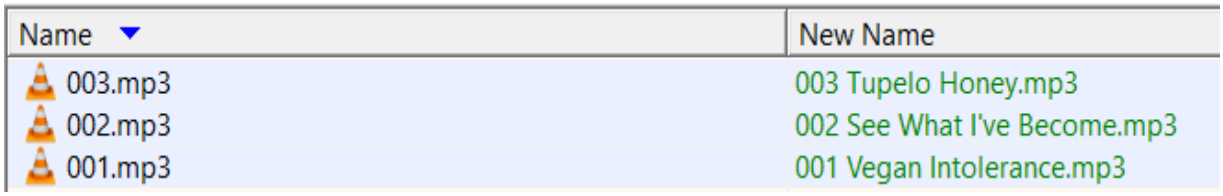
1. Enable 'Extract ID3 Data (MP3)' from the ID3 / Exif Data / File Properties submenu of the Renaming Options Menu



2. 'Section #7: Add' -



Suffix: %t — the 'substitution tag' for Song Title %t
Word Space: Enable — to separate words by a <space> character



Forum Contributor: KenP

Bulk Rename Utility

Autonumber Files based on When Text in string Changes

BRUIEW-intro1	to	BRUIEW-01-intro1
BRUIEW-intro2	to	BRUIEW-02-intro1
BRUIEW-example1	to	BRUIEW-03-example1
BRUIEW-example2	to	BRUIEW-04-example1
BRUIWXXY-intro1	to	BRUIWXXY-01-intro1
BRUIWXXY-example1	to	BRUIWXXY-02-example1
BRUIWXXY-example2	to	BRUIWXXY-03-example2
BRUIWXXY-example3	to	BRUIWXXY-04-example3

Observations:

1. Increment of Numeric padding based on 5th character of string.
   a. When the string's 5th character changes from, i.e., 'E' as in 'BRUIE' to 'W' as in BRUIW
2. Pattern established at 5th character. If this changes to another character, will have to revise.
3. <hyphen> added after numeric padding.

1. 'Section #10: Numbering' –

	New Name
BRUIEW-intro1	BRUIEW-intro1-01
BRUIEW-intro2	BRUIEW-intro2-02
BRUIEW-example1	BRUIEW-example1-03
BRUIEW-example2	BRUIEW-example2-04
BRUIWXXY-intro1	BRUIWXXY-intro1-01
BRUIWXXY-example1	BRUIWXXY-example1-02
BRUIWXXY-example2	BRUIWXXY-example2-03
BRUIWXXY-example3	BRUIWXXY-example3-04

Set:

Mode: Suffix
Start: 1
Inc.: 1
Pad: 2
Sep: -
Break: 5

To move the added numbers to the correct position, run this separately.

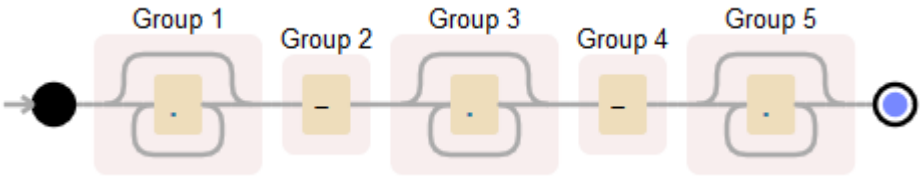
2. Match: (.*)(-)(.*)(-)(.*)
   Replace: \1-\5-\3

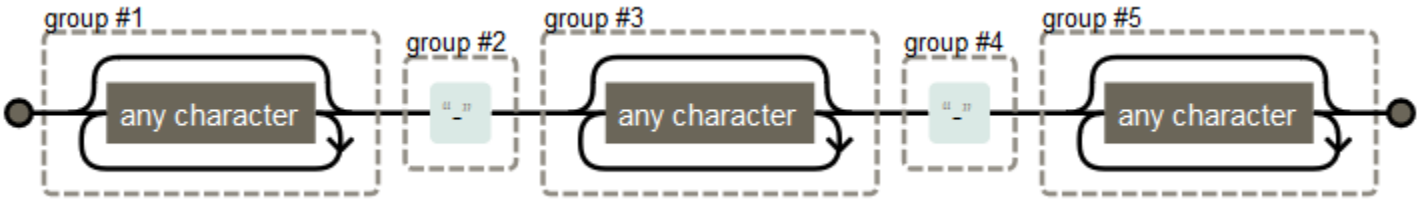
	New Name
BRUIEW-intro1-01	BRUIEW-01-intro1-01
BRUIEW-intro2-02	BRUIEW-02-intro2-02
BRUIEW-example1-03	BRUIEW-03-example1-03
BRUIEW-example2-04	BRUIEW-04-example2-04
BRUIWXXY-intro1-01	BRUIWXXY-01-intro1-01
BRUIWXXY-example1-02	BRUIWXXY-02-example1-02
BRUIWXXY-example2-03	BRUIWXXY-03-example2-03
BRUIWXXY-example3-04	BRUIWXXY-04-example3-04

Bulk Rename Utility

Autonumber Files based on When Text in string Changes cont.

2. Match: (.*)(-)(.*)(-)(.*) cont.





Sample:

```
.............................
Match 1: BRUIEW-01-intro1
Group 1: BRUIEW
Group 2: -
Group 3: 01
Group 4: -
Group 5: intro1
```

Contributor Notes:

1. The "Break" Option allows you to reset the numbering when a character changes. In this example the 5th character is the first to change so setting "Break" at 5 resets the numbering at that point.

Forum Contributor: KenP

Tim Notes:

1. RegEx analysis not displayed. For now just know that:

Capture Group 1 (.*)	Capture Group 2 (-)	Capture Group 3 (.*)	Capture Group 4 (-)	Capture Group 5 (.*)
BRUIEW	-	01	-	intro1

Bulk Rename Utility

Remove extraneous length of text at end of filename when length is known

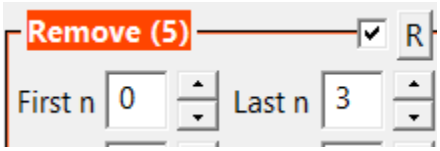
100-dollar-bill-1024x437@2x.jpg to 100-dollar-bill-1024x437.jpg

Observations:

1. Presumes that the '@2x' was added to all files.
2. The makeup of the added text is 3 characters in length

Method #1:

'Section #5: Remove' -



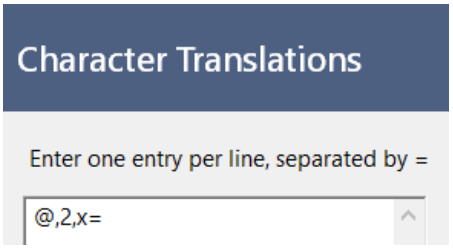
Set:
Last n: 3

Remove last 3 characters

	New Name
100-dollar-bill-1024x437@2x.jpg	100-dollar-bill-1024x437.jpg

Method #2:

'Section #14: Character Translation' –



Enter:
@,2,x=<null>

	New Name
100-dollar-bill-1024x437@2x.jpg	100-dollar-bill-1024x437.jpg

Forum Contributor: Admin

Method #3: Capture all characters up to the '@' into Capture Group 1 and ignore rest.

Tim Contribution –

Match: ^(.*)@.*
Replace: \1

	New Name
100-dollar-bill-1024x437@2x.jpg	100-dollar-bill-1024x437.jpg

Bulk Rename Utility

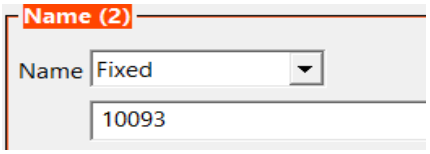
Rename same number Sequence alpha

img_3455.jpg	to	10093.jpg
img_3456.jpg	to	10093a.jpg
img_3457.jpg	to	10093b.jpg
img_3458.jpg	to	10093c.jpg
img_3459.jpg	to	10093d.jpg

Observations:

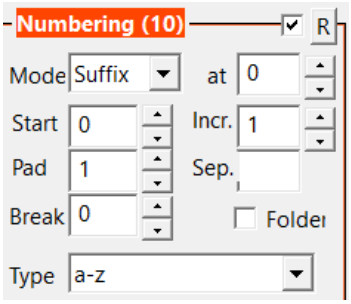
1. Remove original name entirely.
2. Replace with same name padded with Alpha
3. First number has no added padding.

1. 'Section #2: Name' –



Set:
Fixed = 10093

2. 'Section #10: Numbering' –



Set:
Mode: Suffix
Start: 0
Inc.: 1
Pad: 1
Type: a-z

Doesn't work: Start = 0 Pad = 1

	New Name
img_3455.jpg	10093a.jpg
img_3456.jpg	10093a.jpg
img_3457.jpg	10093b.jpg
img_3458.jpg	10093c.jpg
img_3459.jpg	10093d.jpg

Work-Around: Start = 0 Pad = 0

Name	New Name
img_3455.jpg	10093a.jpg
img_3456.jpg	10093b.jpg
img_3457.jpg	10093c.jpg
img_3458.jpg	10093d.jpg
img_3459.jpg	10093e.jpg

Tim Notes:

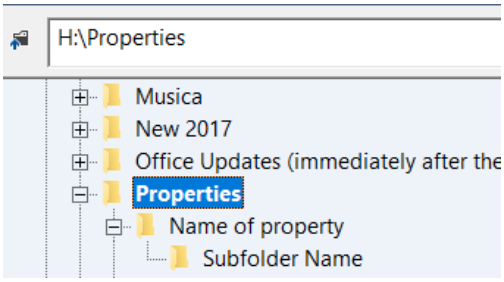
1. At present time, 'Start: 0' does not work as intended. The expected result is 10093 the final result is a duplicated 10093a. 'Start: 0' does not work with 'Type: a-z' or 'Type: A-Z'.
2. See Appendix for a full explanation on how to use Pad (Alpha) in BRU. But for now, the work-around (above) is to have 'Start' of '1' or more using a 'Pad' value of zero or more. Now it will rename appropriately. I have tested this with several hundred files – it will rename from a-z then aa-zz, then aaa-zzz, etc.

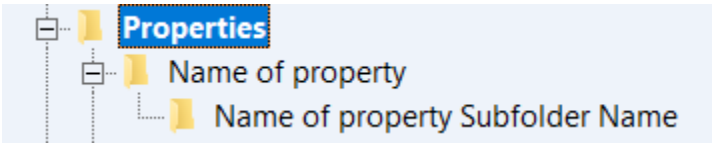
Forum Contributor: Admin

Bulk Rename Utility

Appending Parent Folders Names to All SubFolders Names

a. Folder Structure : To:





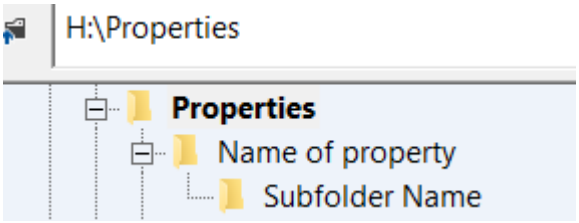
b. Files contained in the 'Subfolder Name' directory should be renamed as:

filename 1.any	to	Name of property Subfolder Name filename 1.any
filename 2.any	to	Name of property Subfolder Name filename 2.any
filename 3.any	to	Name of property Subfolder Name filename 3.any
filename 4.any	to	Name of property Subfolder Name filename 4.any

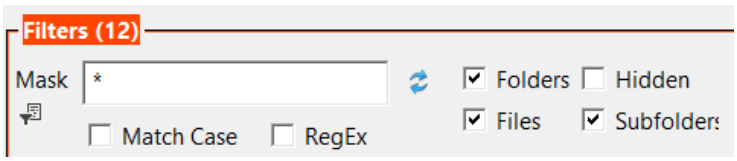
Two Runs:

First run: Rename Folders -

1. In Navigation Pane (Tree Hierarchy) on the left, select the Properties root directory.



2. 'Section #12: Filters'-



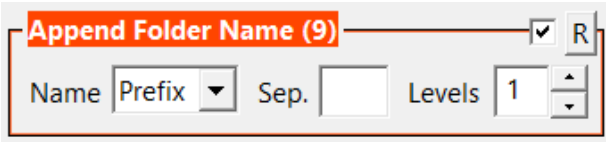
Subfolders: Enable Turn on Recursion

3. Press F5 to Refresh Files List or click on the Filters Refresh button – the blue button to the right of the Filters data field (Mask).

Bulk Rename Utility

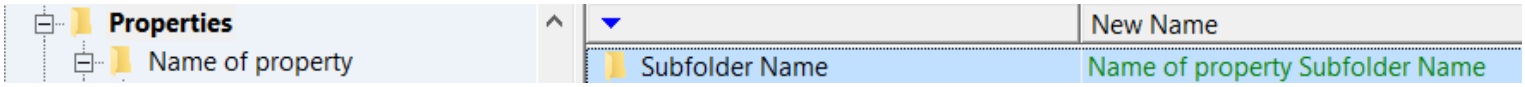
Appending Parent Folders Names to All SubFolders Names cont.

4. 'Section #9: Append Folder Name'



Set:
Name: Prefix
Sep: <space>
Levels: 1

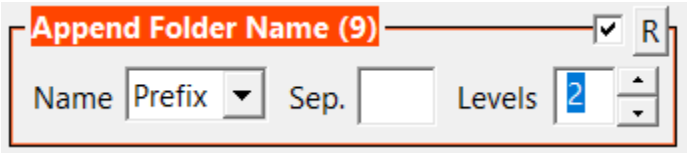
5. In the Content Pane (File List Window) on the right, select those folders you want to rename.



Second Run: Rename the Files within the Subdirectory –

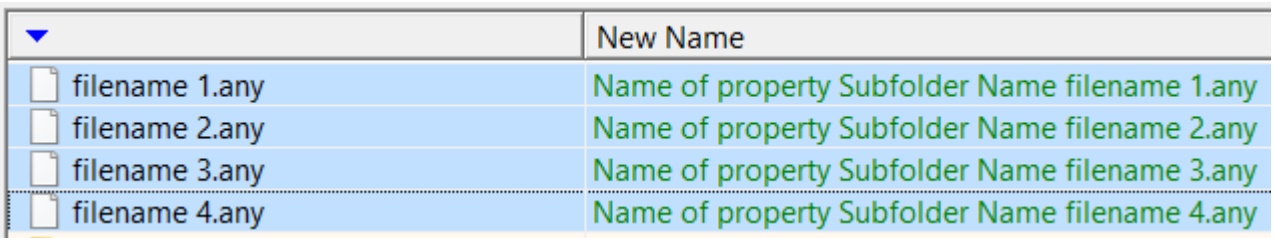
1. 'Section #12: Filters' – no changes

2. 'Section #9: Append Folder Name'



Set:
Name: Prefix
Sep: <space>
Levels: 2

3. Select only those files that you want to rename.



Forum Contributor: GMA.

Bulk Rename Utility

Import Rename-Pairs using Unicode (UTF-16) Files

Trackx.mp3

to

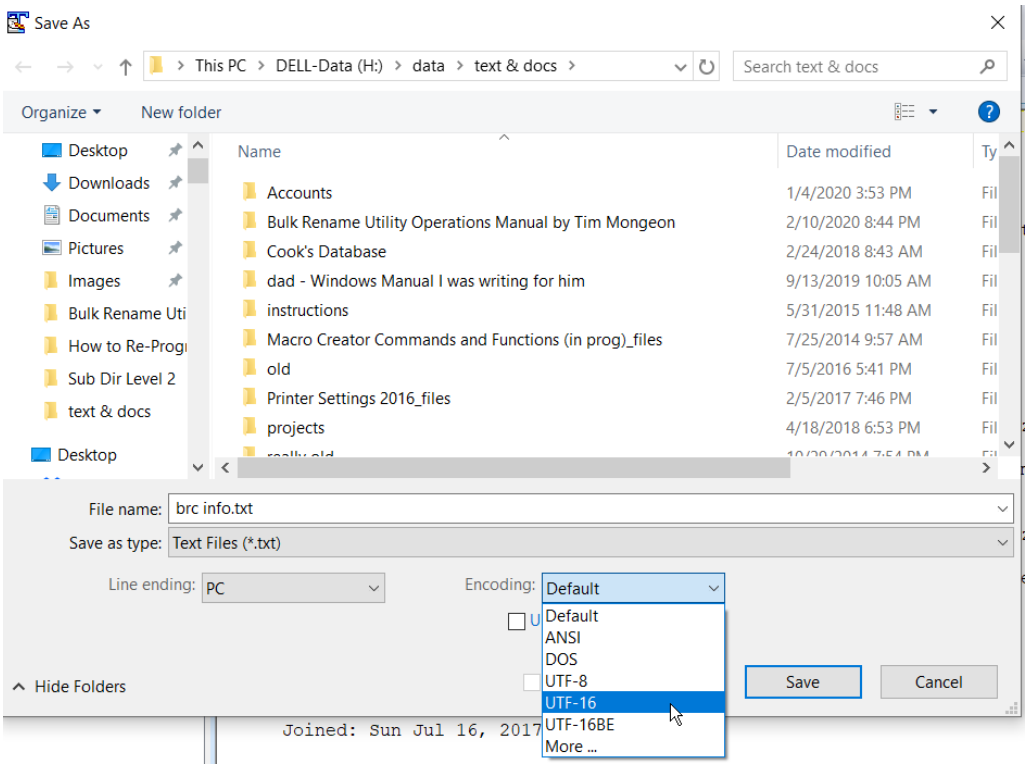
6 Songs For Voice & Piano 'An Die Ferne Geliebte', Op.98 - 1. Auf Dem Hügel Sitz Ich Spähend In E Flat.mp3

Using a Rename-Pairs text file:

1. Create your text file BUT it must be saved as UTF-16 format.

i.e.,

Trackx.mp3|6 Songs For Voice & Piano 'An Die Ferne Geliebte', Op.98 - 1. Auf Dem Hügel Sitz Ich Spähend In E Flat.mp3



Use 'Save As'. Most text editors and word processors have the option to save under different formats. Some have this option under Encoding, others as part of the file formats to Save.

Once the file has been saved in the proper format, it can be imported into BRU.

Contributor Notes:

You can preview the imported list using "View Imported Rename-Pairs" from Action Menu to make sure all characters have been imported correctly.

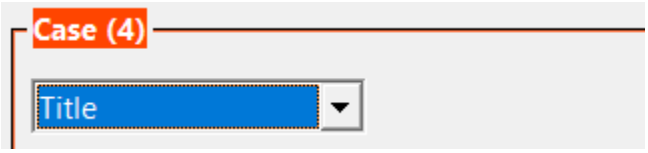
Forum Contributor: Admin

Bulk Rename Utility

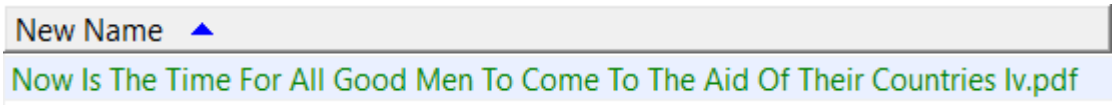
Convert Case to Uppercase for significant words only

now is the time for all good men to come to the aid of their countries iv.pdf

'Section #4: Case' –



Case: Title

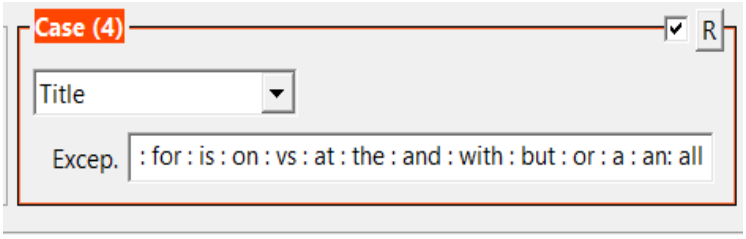


Doesn't work.

There are too many uppercase minor words. Also the Roman Numerals at the end are messed up.

This will:

1. 'Section #4: Case' –



Case: Title

Exception List:

1. Multiple words are delimited by a ':'
2. Note that there is a <space> before and after each word

Exception List -

to : of : by : in : for : is : on : vs : at : the : and : with : but : or : a : an: all : nor : yet : so : as : from

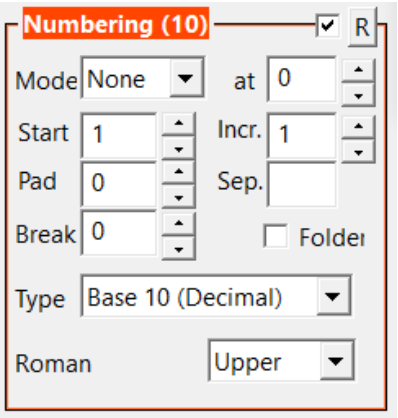
If you find you need more, just add them.

Bulk Rename Utility

Convert Case to Uppercase for significant words only cont.

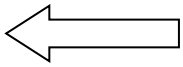
now is the time for all good men to come to the aid of their countries iv.pdf

2. 'Section #10: Numbering' -



Mode: None
Start: 1
Incr.: 1
Roman: Upper

Handles the Roman Numerals if any



Final result looks good:

Forum Contributor: thewordguy

Bulk Rename Utility

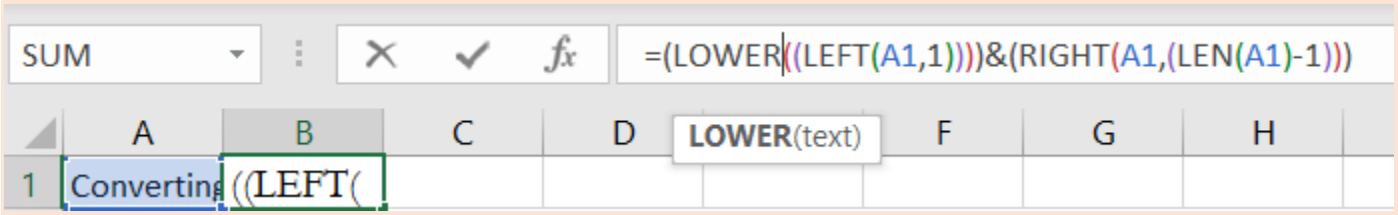
Convert Case to Uppercase for significant words only cont.

A method using Excel or another spreadsheet program to change text to Upper or Lowercase:

1) First, make a text file containing all your filenames.
   a. You can use a program like Ztree (using Batch Command), or Karen's Directory Printer, or PrintFolders
   b. You can use command line, dir /b >filelist.txt (change to current directory first using CD command)
2. Paste the list into Column A of Excel (or LibreOffice Calc, etc.)

	A	B	C	D	E	F	G	H	I	J
1	Converting to epub distorts my jpeg cover dimensions - MobileRead Forums									
2	SOLVED- Why is the washing machine getting out of balance - Kenmore 110 Series Washing Machine - iFixit									
3	Strange problem with Internet Explorer - works only when run as admin									
4	Solved - Internet Explorer 11 Only Opens As Administrator Windows 8 Help Forums									
5	Internet Explorer 11 will only open if I right click and then select Run As Administrator from windows server 2012									
6	IE will only open as administrator									
7	IE 11 ONLY WORKS IF I PRESS RUN ADMINISTRATOR WHY PLS HELP - Microsoft Community									
8	IE11 only opens when run as Administrator - Spiceworks									
9	Internet Explorer only opens as Administrator Solved - Windows 10 Forums									

3. Paste this into cell B1:
   =(LOWER((LEFT(A1,1))))&(RIGHT(A1,(LEN(A1)-1)))
   a. If you are trying to force the first letter into Uppercase, then change "LOWER" to "UPPER" in the formula.



4. Fill/Paste B1 all the way down the page in Column B
   a. Column B is a list of the filenames with the first letter downcased.

Bulk Rename Utility

Convert Case to Uppercase for significant words only cont.

A method using Excel or another spreadsheet program to change text to Upper or Lowercase: cont.

	A	B
1	Converting	converting to epub distorts my jpeg cover dimensions - MobileRead Forums
2	SOLVED- W	sOLVED- Why is the washing machine getting out of balance - Kenmore 110 Series Washing Machine - iFixit
3	Strange pr	strange problem with Internet Explorer - works only when run as admin
4	Solved - In	solved - Internet Explorer 11 Only Opens As Administrator Windows 8 Help Forums
5	Internet Ex	internet Explorer 11 will only open if I right click and then select Run As Administrator from windows server 2012
6	IE will only	iE will only open as administrator
7	IE 11 ONLY	iE 11 ONLY WORKS IF I PRESS RUN ADMINISTRATOR WHY PLS HELP - Microsoft Community
8	IE11 only o	iE11 only opens when run as Administrator - Spiceworks
9	Internet Ex	internet Explorer only opens as Administrator Solved - Windows 10 Forums

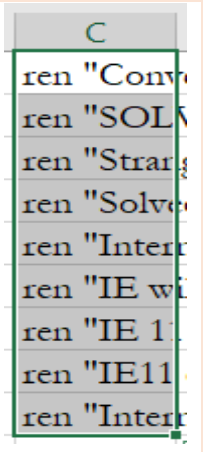
5a. Use Command Line.

a. If all your files are in the same folder, paste this into cell C1:

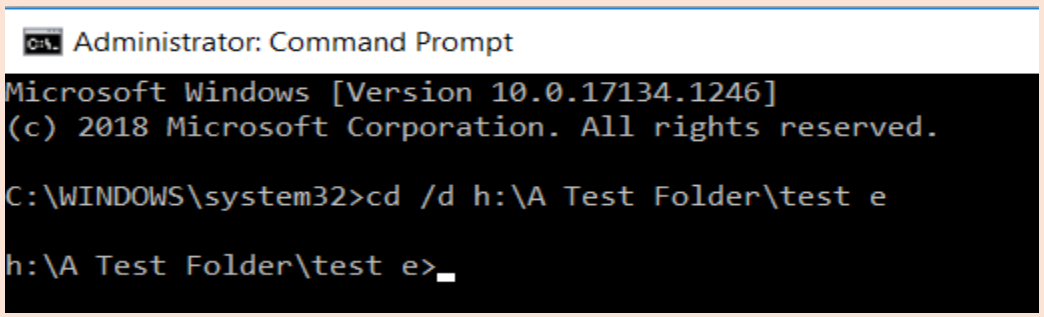
="ren "&CHAR(34)&A1&CHAR(34)&" "&CHAR(34)&B1&CHAR(34)

1. Fill/paste down the rest of column C

C	D	E	F	G	H	I	J	K	L	M	N
ren "Converting to epub distorts my jpeg cover dimensions - MobileRead Forums" "converting to epub distorts my jpeg											
ren "SOLVED- Why is the washing machine getting out of balance - Kenmore 110 Series Washing Machine - iFixit" "sO											
ren "Strange problem with Internet Explorer - works only when run as admin" "strange problem with Internet Explorer -											
ren "Solved - Internet Explorer 11 Only Opens As Administrator Windows 8 Help Forums" "solved - Internet Explorer											
ren "Internet Explorer 11 will only open if I right click and then select Run As Administrator from windows server 2012"											
ren "IE will only open as administrator" "iE will only open as administrator"											
ren "IE 11 ONLY WORKS IF I PRESS RUN ADMINISTRATOR WHY PLS HELP - Microsoft Community" "iE 11											
ren "IE11 only opens when run as Administrator - Spiceworks" "iE11 only opens when run as Administrator - Spicework											
ren "Internet Explorer only opens as Administrator Solved - Windows 10 Forums" "internet Explorer only opens as Adr											



2. Copy column c
3. In a Windows command line window (change directory to current directory if needed)



a. Paste column C into the command line window (Ctrl + v)

1. The moment you do this the commands are run and the files renamed.

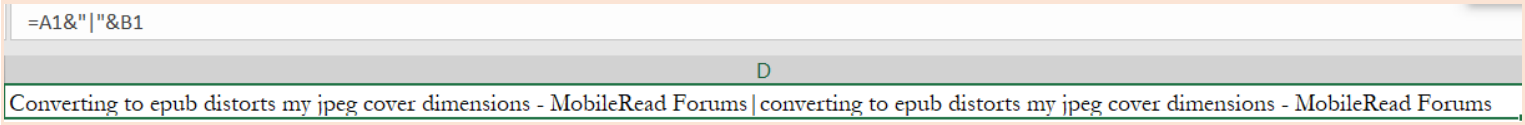
Bulk Rename Utility

Convert Case to Uppercase for significant words only cont.

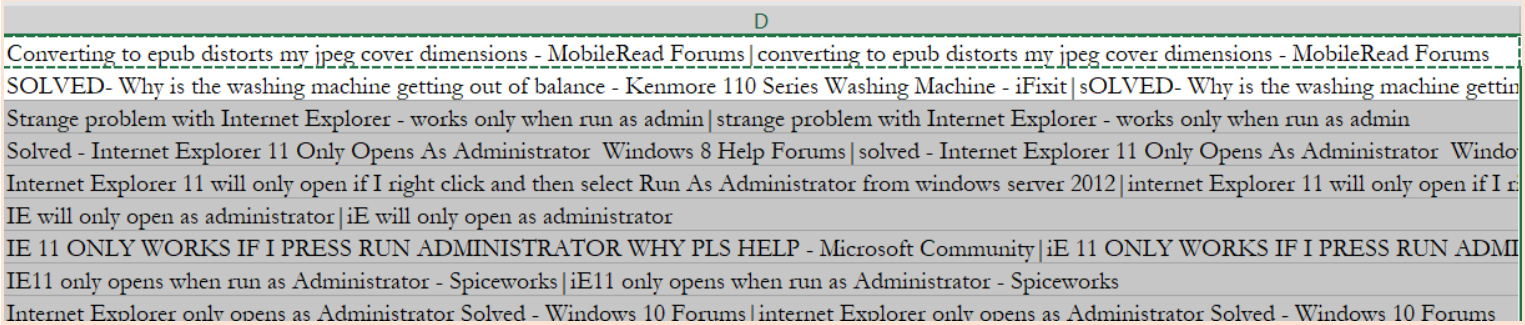
A method using Excel or another spreadsheet program to change text to Upper or Lowercase: cont.

5b. Use BRU Import Rename-Pairs function.

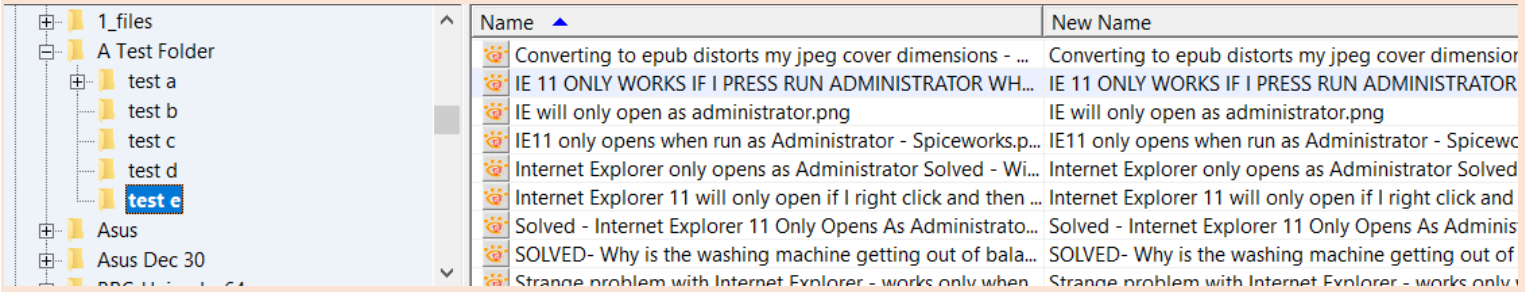
b. If your files are all in different subdirectories or you don't want to use the command line, paste this into cell D1:
=A1&"|"&B1



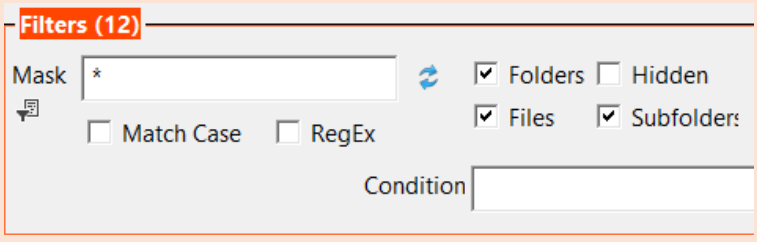
1. Fill/paste D1 down the page in column D



2. Copy/paste Column D into a new textfile and save it as, say, "rename-pairs.txt"
3. Return to BRU
4. In the Naviagtion Pane, navigate to your folder containing your files (in the usual way)



5. Enable the "subfolders" box in the 'Section #12: Filters' (turn on Recursion)

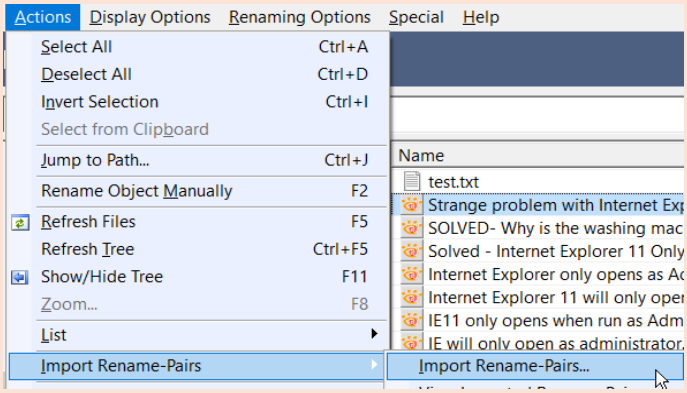


Bulk Rename Utility

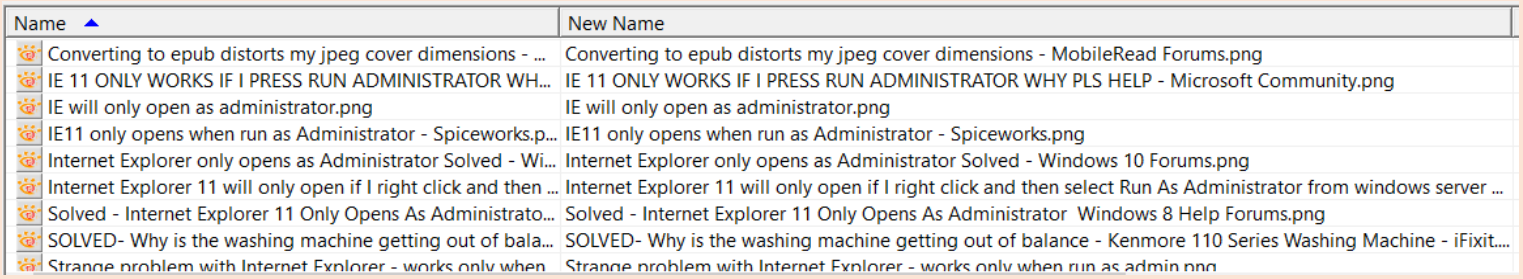
Convert Case to Uppercase for significant words only cont.

A method using Excel or another spreadsheet program to change text to Upper or Lowercase: cont.

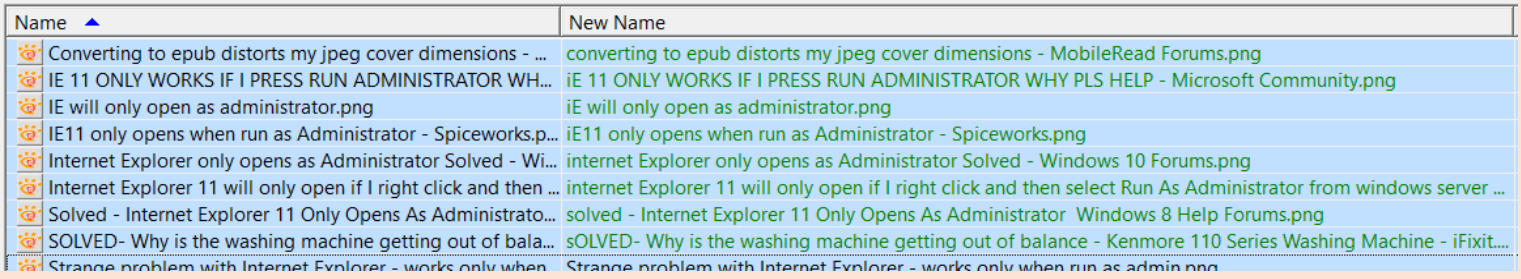
6. Select 'Import Rename-Pairs' through the Actions Menu.



7. Load in the 'rename-pairs.txt' you just created.



8. Highlight all the files in the Content Pane



9. Click "Rename".

Contributor Notes:

1. The formula in B1 just strips-off the first character in the line, changes it to a lowercase then reattaches it.
2. Everything else (e.g. putting capitals on the other words, handling exceptions like 'to, the, of', etc), you have to (first) do yourself using BRU's standard tools.

Forum Contributor: Jack2

Bulk Rename Utility

Pad with Alpha not Numeric

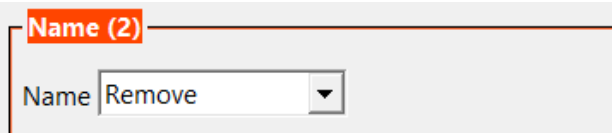
file1 to AAAA
file2 to AAAB
file3 to AAAC

Observations:

1. Remove filename.
2. Replace with padded alpha

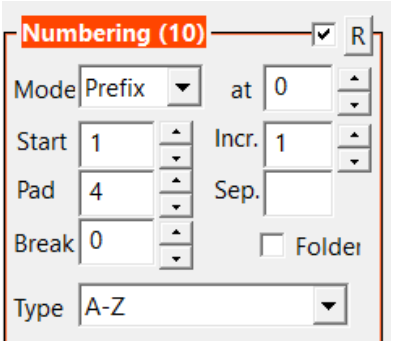
1. In BRU select all the files you want to rename and make sure they're in the order you want them renamed. The files will always be processed in the order of displayed sequence.

2. 'Section #2: Name'

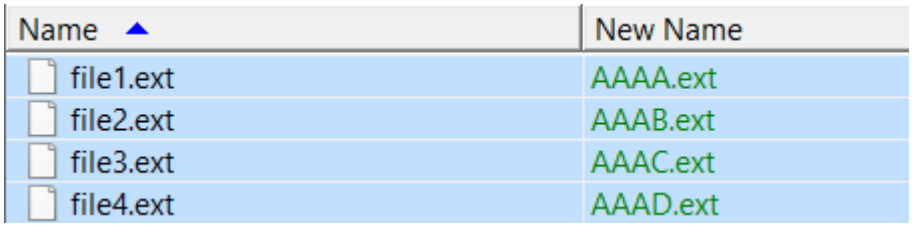


Set Name: Remove

3. 'Section #10: Numbering'



Set:
Mode: Prefix
Start: 1
Inc. 1
Pad: 4
Type: A-Z



Tim Notes:

1. For more information see 'Volume I Bulk Rename Utility Operations Manual' - 'Sorting' under the 'Display Options Menu', 'List - Reposition', under the Actions Menu, 'Apply Random Sort to Current List', also under the Actions Menu, and 'Rename in Reverse Order', under the Renaming Options Menu.

Forum Contributor: KenP

Bulk Rename Utility

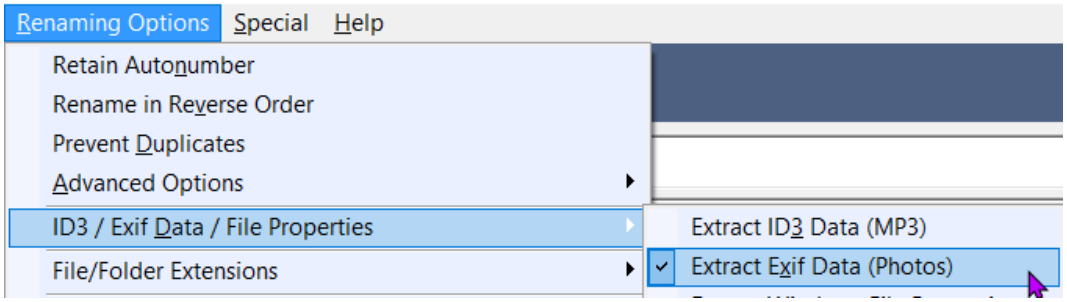
Rename Photo Files with Prefix and date of Photo

DSCN00238.JPG to ARH20081220-00238.JPG

Observations:

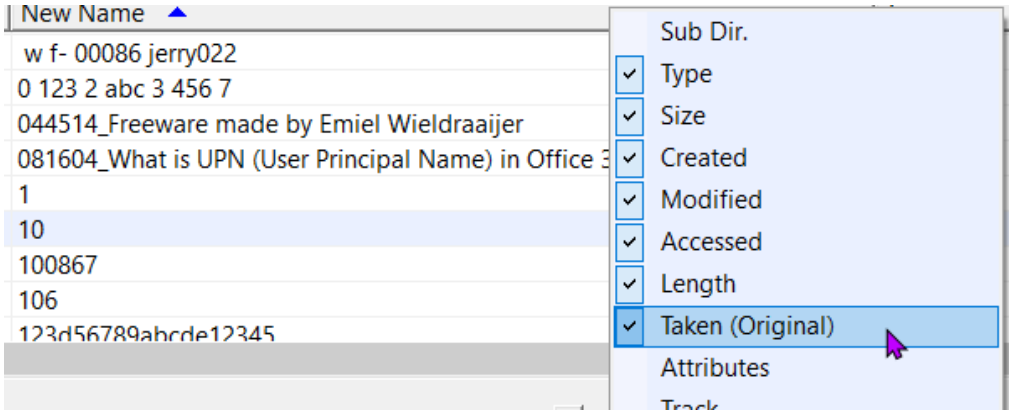
1. DSCN is the original prefix added by the Camera Software at the time the picture is taken.
2. The Date Taken for the photo will need to be extracted from the metadata, i.e. 20081220
3. ARH will be the new prefix
4. -00238 is the original photo number and will be retained.
5. Pattern Established: <Old Prefix> <Date Taken> <Photo Number>

1. Enable 'Extract Exif Data (Photos)' from 'ID3 /Exif data /File Properties' sub menu from the 'Renaming Options Menu'.



Press F5 to refresh the Content Pane.

2. Right click in the Title Bar of the Content Pane and make sure that column for 'Taken (Original)' title heading is Enabled.

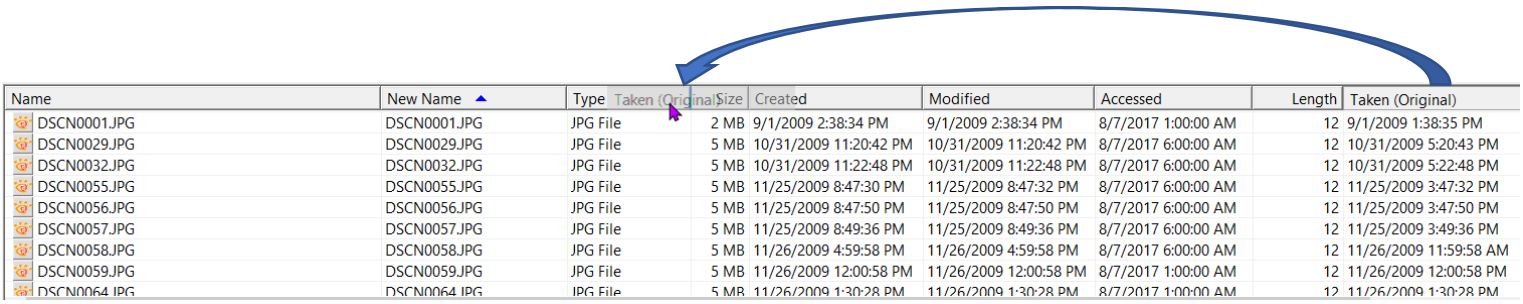


Bulk Rename Utility

Rename Photo Files with Prefix and date of Photo cont.

DSCN00238.JPG to ARH20081220-00238.JPG

3. Move the column if you wish by dragging the title to the desired area:

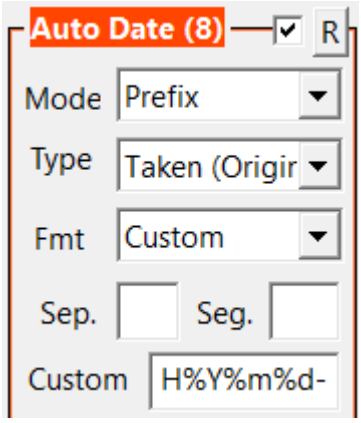


Name	New Name	Type	Taken (Original)	Size	Created	Modified	Accessed	Length	Taken (Original)
DSCN0001.JPG	DSCN0001.JPG	JPG File		2 MB	9/1/2009 2:38:34 PM	9/1/2009 2:38:34 PM	8/7/2017 1:00:00 AM	12	9/1/2009 1:38:35 PM
DSCN0029.JPG	DSCN0029.JPG	JPG File		5 MB	10/31/2009 11:20:42 PM	10/31/2009 11:20:42 PM	8/7/2017 6:00:00 AM	12	10/31/2009 5:20:43 PM
DSCN0032.JPG	DSCN0032.JPG	JPG File		5 MB	10/31/2009 11:22:48 PM	10/31/2009 11:22:48 PM	8/7/2017 6:00:00 AM	12	10/31/2009 5:22:48 PM
DSCN0055.JPG	DSCN0055.JPG	JPG File		5 MB	11/25/2009 8:47:30 PM	11/25/2009 8:47:32 PM	8/7/2017 6:00:00 AM	12	11/25/2009 3:47:32 PM
DSCN0056.JPG	DSCN0056.JPG	JPG File		5 MB	11/25/2009 8:47:50 PM	11/25/2009 8:47:50 PM	8/7/2017 6:00:00 AM	12	11/25/2009 3:47:50 PM
DSCN0057.JPG	DSCN0057.JPG	JPG File		5 MB	11/25/2009 8:49:36 PM	11/25/2009 8:49:36 PM	8/7/2017 6:00:00 AM	12	11/25/2009 3:49:36 PM
DSCN0058.JPG	DSCN0058.JPG	JPG File		5 MB	11/26/2009 4:59:58 PM	11/26/2009 4:59:58 PM	8/7/2017 6:00:00 AM	12	11/26/2009 11:59:58 AM
DSCN0059.JPG	DSCN0059.JPG	JPG File		5 MB	11/26/2009 12:00:58 PM	11/26/2009 12:00:58 PM	8/7/2017 1:00:00 AM	12	11/26/2009 12:00:58 PM
DSCN0064.JPG	DSCN0064.JPG	JPG File		5 MB	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	8/7/2017 1:00:00 AM	12	11/26/2009 1:30:28 PM

Name	New Name	Taken (Original)
DSCN0001.JPG	DSCN0001.JPG	9/1/2009 1:38:35 PM
DSCN0029.JPG	DSCN0029.JPG	10/31/2009 5:20:43 PM
DSCN0032.JPG	DSCN0032.JPG	10/31/2009 5:22:48 PM
DSCN0055.JPG	DSCN0055.JPG	11/25/2009 3:47:32 PM
DSCN0056.JPG	DSCN0056.JPG	11/25/2009 3:47:50 PM
DSCN0057.JPG	DSCN0057.JPG	11/25/2009 3:49:36 PM
DSCN0058.JPG	DSCN0058.JPG	11/26/2009 11:59:58 AM
DSCN0059.JPG	DSCN0059.JPG	11/26/2009 12:00:58 PM
DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM

Post begins:

4. 'Section #8: Auto Date' –



Mode: Prefix
Type: Taken (Original)
Fmt: Custom
Custom: ARH%Y%m%d-

Where:

%Y% = Year as YYYY %m% = Month as MM %d% = Day as DD

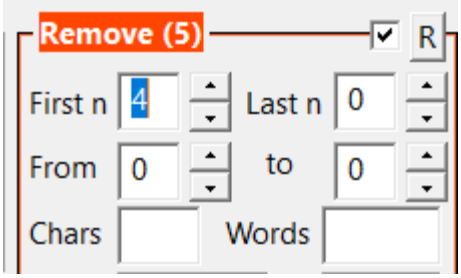
- = literal <hyphen>

Bulk Rename Utility

Rename Photo Files with Prefix and date of Photo cont.

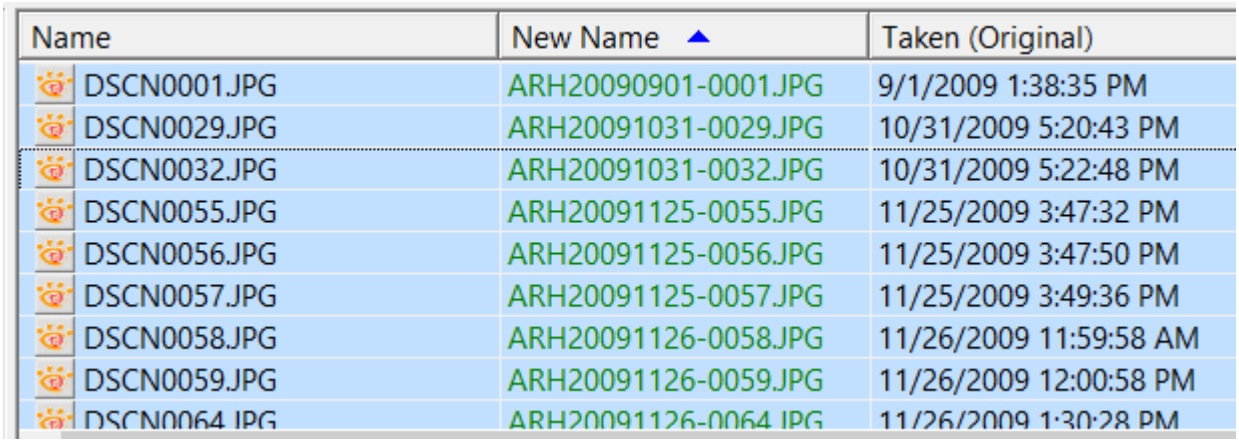
DSCN00238.JPG to ARH20081220-00238.JPG

5. 'Section # 5: Remove' –



First n: 4

Removes the 'DSCN' prefix from filenames



Name	New Name	Taken (Original)
DSCN0001.JPG	ARH20090901-0001.JPG	9/1/2009 1:38:35 PM
DSCN0029.JPG	ARH20091031-0029.JPG	10/31/2009 5:20:43 PM
DSCN0032.JPG	ARH20091031-0032.JPG	10/31/2009 5:22:48 PM
DSCN0055.JPG	ARH20091125-0055.JPG	11/25/2009 3:47:32 PM
DSCN0056.JPG	ARH20091125-0056.JPG	11/25/2009 3:47:50 PM
DSCN0057.JPG	ARH20091125-0057.JPG	11/25/2009 3:49:36 PM
DSCN0058.JPG	ARH20091126-0058.JPG	11/26/2009 11:59:58 AM
DSCN0059.JPG	ARH20091126-0059.JPG	11/26/2009 12:00:58 PM
DSCN0064.JPG	ARH20091126-0064.JPG	11/26/2009 1:30:28 PM

Forum Contributor: AndrewRH

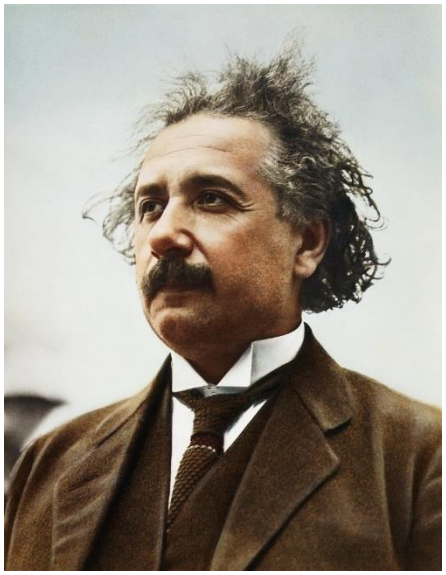
End Section ~

Bulk Rename Utility

[This Page Intentionally Blank]

Expert's Corner-

Regular Expressions



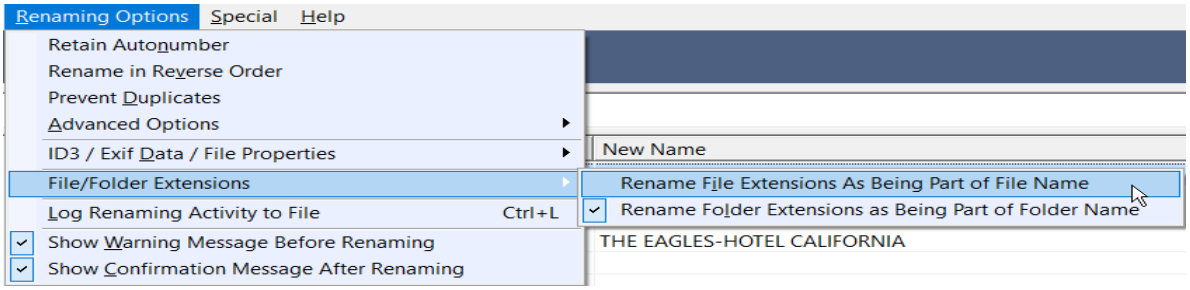
Tim Note:

The following are taken from the public domain BRU Forums. Contributors of the final RegEx used will be noted. I have also contributed quite a few of my own RegEx, notes, observations and insights in this section that were not a part of the forum.

The best method of learning Regular Expressions is to study from others. Here are many examples that you can try for yourself and do your own analysis. I have tested every single example and have included photos with each to demonstrate that they work (or don't work as noted in a few cases but the corrections have been provided). Forum posts have been selected based on their usefulness and tutorial benefit.

Tips:

1. If the examples here do not work when you try them, check your folder and extension settings. Sometimes turning one off or on can mean the difference:



2. Check that you do not have other criteria set that would skew your results.

3. When it comes to RegEx, there is always more than one method of solving a problem. You may find a better solution on your own than the ones provided. Share them with forum members if you do.

4. RegEx uses patterns. That is why the samples are given. The RegEx will work on the patterns presented by these samples. You will have to tweak the RegEx to work with your own samples if they don't adhere to this pattern.

5. If a RegEx is not working with certain strings, make sure that the <hyphen> (-) is not being confused with the dash (–).

6. Multiple runs require that criteria settings be reset between runs using the 'Reset' button.

Note:

1. I do not supply the original posting dates or in many cases, even the same headings.
2. I have sectionalized the RegEx by categories to make it easier to comprehend, but understand, all BRU RegEx has to do with parsing, and some examples will definitely overlap into other categories. It is meant to assist you in your learning endeavor.

My overall conclusions:

The RegEx presented here do not necessarily represent the best method of accomplishing something. Even my own contributions and notes are more or less based on the contributor's code. I try to stay as close as I can to the original intent. Any variations presented, however, can be extremely different. My goal here is not to rewrite the examples but to shine a light on what they do and in many cases how they do it.

In my analysis I have attempted to explain how a certain solution or failure was reached. I do not say this is 100% correct – it is just my interpretation of what I perceive to be the facts based on my overall testing and conclusions. In my defense, however, I also conclude that during my intensive research into these subjects, I have found that most analysis of RegEx by others, even by experts, differs to the degree that the analysis is subjective and open to interpretation. Therefore I encourage you to make your own analysis and draw your own conclusions.

Timothy R. Mongeon

It should also be mentioned that any coloring is mainly specific to each section, and in many cases each page. Any meaning therein does not extend to other sections or pages. There are exceptions which will become obvious. The coloring is for clarification and to allow the user to more easily follow the topic of discussion.

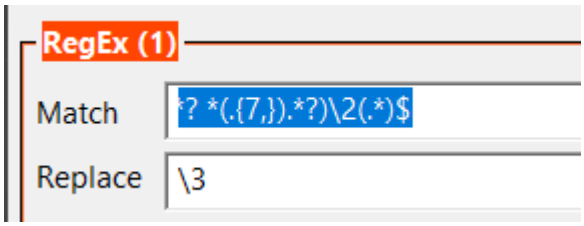
Before Starting:

A Note on Replacement Strings –

In the analysis, very little is mentioned about the replacement strings. This is because, unlike the RegEx, the manner in which they are used – the syntax, does not change. The exception is when a back reference is included in the RegEx expression. There are at least two examples of this in the book.

So what are Replacement strings?

They appear in the Replace data field of 'Section #1: RegEx'



Each element of the Replacement string is either a 'Backreference' or a literal character. Unlike within the RegEx expression, literal characters do not require an escape character to be recognized by the RegEx Engine, PCRE, in BRU.

What are Literal Characters?

Literal characters are those characters that you can reproduce on the keyboard – i.e., A,b, <space> <hyphen>^&#9, etc.

What are Backreferences?

Backreferences are 'stand-in's for the Capture Groups defined in the RegEx through the use of enclosed parentheses around a section within the expression.

What are Capture Groups (aka marked Groups)?

Each enclosed parentheses form a Capture Group (with the exception of certain special groups i.e. Non-Marking groups, case insensitivity, Lookarounds, etc.). Each defined Capture Group are referenced by a number in the order in which they are evaluated.

i.e. (….) (…..) (…..) Defines three Capture Groups. Once the expression contained within each is evaluated, the
1 2 3 Capture Group holds that value (maximum number of Capture Groups in an expression = 9).

They are referenced using the Backreference syntax as \1 \2 \3

When you use a Backreference. whether it is in the RegEx itself or more commonly in the Replacement String, it returns the value of that Capture Group.

Using Literals in the Replacement String:

You are not limited to just using Backreferences in the Replacement String. As explained above, any literal can be entered in the data field as well.

If for example, you had 3 capture groups with the following values:

\1 = Cat \2 = Mouse \3 = Ball

and you had a Replacement string of -

\1\2\3

The result is: CatMouseBall

If you had a Replacement string of –	or	or
\1<space> \2 <space> \3 <space>	\1-\2-\3	(\1)<space>(\2)<space>(\3)
The result is:		
Cat Mouse Ball	Cat-Mouse-Ball	(Cat) (Mouse) (Ball)

You could not have:

(\)1 or \(1) Because the syntax for a backreference is \1

The Real power of the Replacement string is once you have the Capture Groups values, the New name can be put back together in any order you desire:

If you have a Replacement string of –

\2<space>\1<space>\3

The result is: Mouse Cat Ball

Finally,

You re not limited to how many times a Capture Group can be used as a Backreference in a Replacement String. You could have:

\1<space>\3<space>\1<space>\1

The result is: Cat Ball Cat Cat

Be aware and study the Replacements Strings (and RegEx) closely so you don't miss a literal such as a dot or <space>

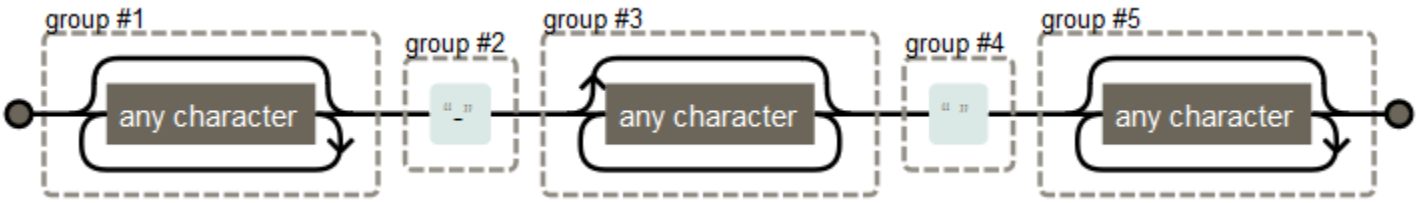
Well I guess that does it. So this is why Replacement Strings will not be, for the most part, included as part of the analysis. They just don't change in behavior.

Railroad Diagrams - Visualizations

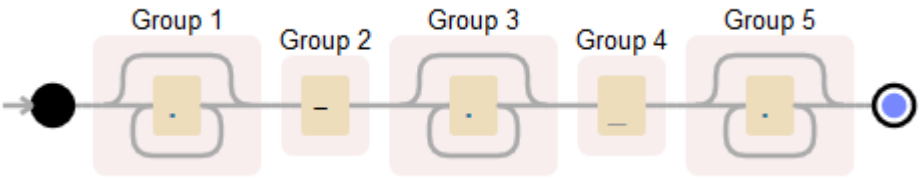
The purpose of these is to visually depict the RegEx. There are two sets I use from two different testing programs in this book (see sources below). Here is an example of both for the same RegEx:

Match: (.*)(-)(.*?)( )(.*)

1.



2.



The first one is more detailed and descriptive. This one will be displayed whenever possible or I feel the reader would benefit by its inclusion. Unfortunately, it doesn't support Lookbehinds or the Case Insensitive work-around that was illustrated in Volume I. Also newer additions to PCRE, such as Referenced Groups, Relative Conditionals, possessive quantifiers, Subroutine Calls and probably other features are also not supported at this time. RegEx' that are not supported will only be represented by the second Visualization program diagram only.

The second is easier to follow. This will be included for almost all examples that use RegEx.

Together, these are especially useful for those few examples that do not include a full analysis to assist the reader with their own self analysis, and for those that do, it helps to understand how the RegEx works in conjunction with any provided analysis. Both have advantages and disadvantages. The Second program's Visualizer, for example, doesn't discern between Greedy and Lazy quantifiers while the first program does.

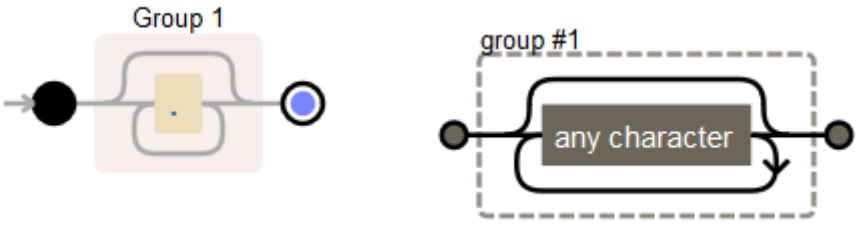
Tim Notes:

1. In some cases the RegEx Visualizations are too long to fit properly without shrinking. If I feel the shrinkage is 'excessive' making the diagram difficult to comprehend, I will provide an 'Expanded' version as well. I will provide both in certain circumstances if I feel the reader can benefit by viewing the complete visual along side the expanded versions.

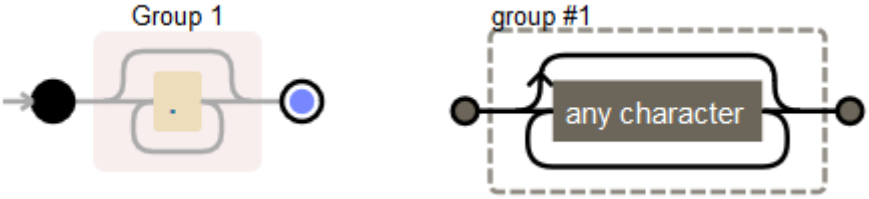
Source Material this section: //regexper.com, //www.debuggex.com

Railroad Diagrams – Visualizations cont.

(.) <dot>



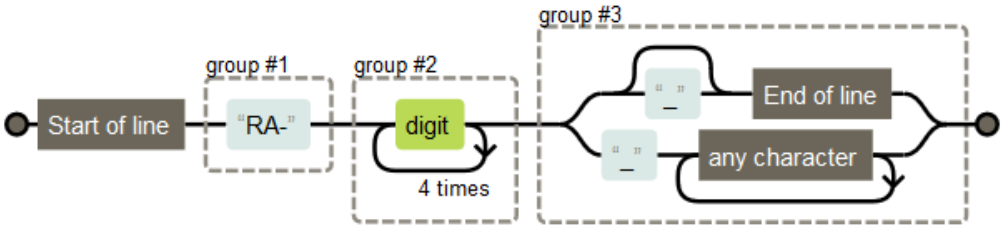
(.*) Greedy

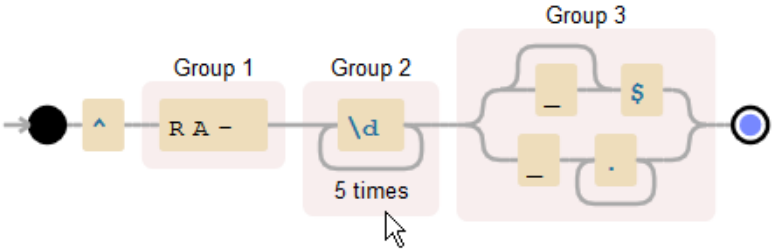


(.*?) Lazy

For example, the Second visualization type depicts Greedy and Lazy with no differences, shown as a grey line that represents the path of the RegEx Engine, optional or otherwise. The second appropriately discerns the differences by displaying the different paths – optional or otherwise.

When it comes to repetitions of a match (loop) by a Quantifier typically, the more descriptive first Visualizer will display one iteration less i.e., '4 times', than what is specified, i.e. {5}, because it counts the expression, i.e., 'numeric digit', '\d', as the first match, so the number '4' represents the additional matches. The other type does display the specified value.

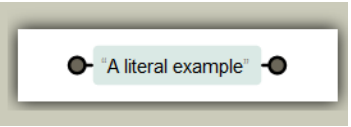




Railroad Diagrams – Visualizations cont.

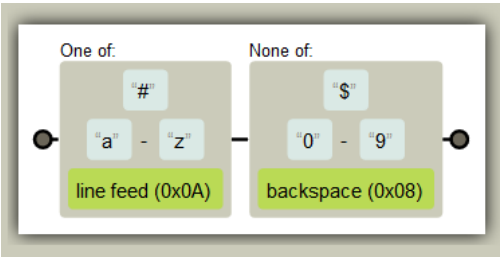
Understanding the Paths

With a RegEx, the entire expression is evaluated from Left to Right. The Expression may also consist of additional sub-expressions that may be enclosed in Groups. The preferred path is always a straight line from beginning to end but many factors can cause this path to stray from the 'straight and 'narrow', the least of which is backtracking. But certain conditions can also interfere with this intentionally including quantifiers, alternates, ranges, etc. The path represented by the visualizations shows both the straight path and any other paths that may be taken by the RegEx engine dependent on the conditions set forth.

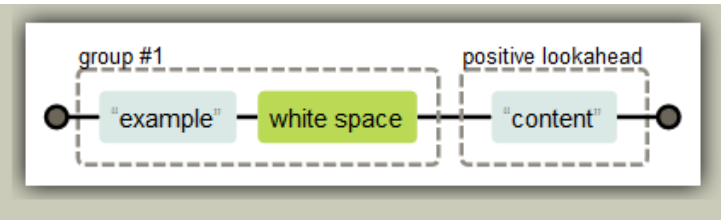


Literals match an exact string of text. They're displayed in a light blue box, and the contents are quoted (to make it easier to see any leading or trailing whitespace)

Escape sequences are displayed in a green box and contain a description of the type of character(s) they will match.



Character sets will match or not match a collection of individual characters. They are shown as a box containing literals and escape sequences. The label at the top indicates that the character set will match "One of" the contained items or "None of" the contained items.



Subexpressions are indicated by a dotted outline around the items that are in the expression. Captured subexpressions are labeled with the group number they will be captured under. Positive and negative lookahead are labeled as such.

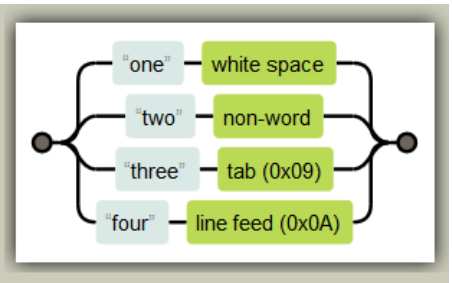
Source Material this section: //regexper.com

Railroad Diagrams – Visualizations cont.

Alternation (Branch Testing)

Understanding the Paths

With an Alternation the Path branches off into prospective options.



Alternation provides choices for the regular expression. It is indicated by the path for the expression fanning out into a number of choices.

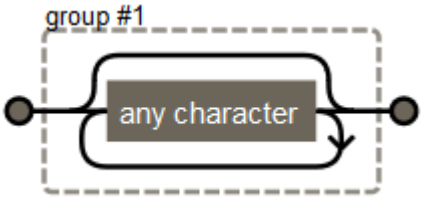
Quantifiers

Quantifiers indicate if part of the expression should be repeated or optional. They are displayed similarly to Alternation, by the path through the diagram branching (and possibly looping back on itself). Unless indicated by an arrow on the path, the preferred path is to continue going straight.

Understanding the Paths

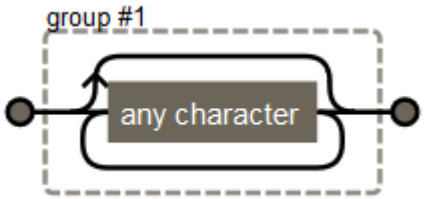
The addition of a quantifier always creates additional paths depending on whether it is Greedy, made Lazy or made Optional. In some cases the RegEx can loop back upon itself as indicated by the lower path *under* the expression. In other cases, an optional path may be available as indicated by the upper path *over* the expression. Again, what paths are available are dependent on the conditions set forth.

(.*) Example of Zero or More Greedy Quantifier



The zero-or-more quantifier matches any number of repetitions of the pattern.

(.*?) Example of Zero or More Non-Greedy Quantifier



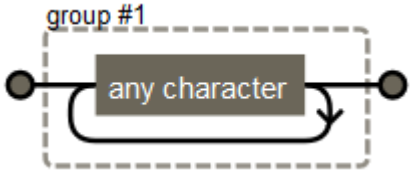
The zero-or-more quantifier matches any number of repetitions of the pattern. The purpose doesn't change, only the available path.

Railroad Diagrams – Visualizations cont.

Quantifiers cont.

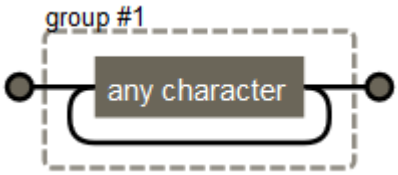
Understanding the Paths cont.

(.+) Example of Required Greedy Quantifier (One or More Repetitions)



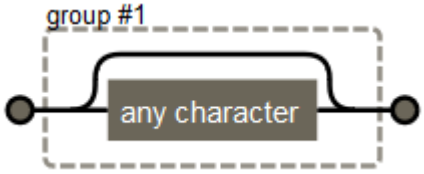
The required quantifier matches one or more repetitions of the pattern. Note that it does not have the path that allows the pattern to be skipped (this would be the upper path that appears over, i.e., 'any character') like the zero-or-more quantifier.

(.+?) Example of Required Non-Greedy Quantifier (One or More Repetitions)



The required quantifier matches one or more repetitions of the pattern. Note that it does not have the path that allows the pattern to be skipped like the zero-or-more quantifier. The purpose doesn't change from above, only the available path.

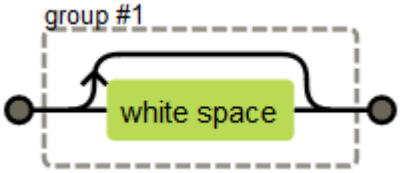
(.?) Example of Optional Quantifier (or another example, \s?)



This also depicts two potential paths. The direct path represented by the straight line which would directly match against the <dot> and the optional path represented by the curved top line that indicates the expression is optional and if required would 'skip' it entirely – because of the Lazy quantifier – ' ? ' which indicates to match the pattern at most once.

The optional quantifier matches the pattern at most once. Note that it does not have the path that allows the pattern to loop back on itself (this would be the bottom path that appears under, i.e. 'any character', like the zero-or-more or required quantifiers.

(\s??) Example of Optional Non-Greedy Quantifier

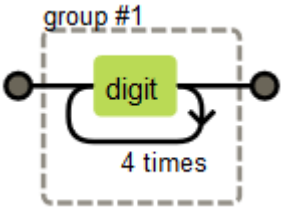


The optional quantifier matches the pattern at most once. Note that it does not have the path that allows the pattern to loop back on itself like the zero-or-more or required quantifiers.

Railroad Diagrams – Visualizations cont.

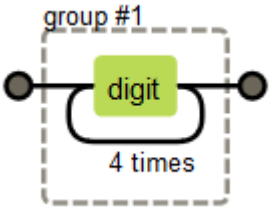
Ranges

(\d{5}) Example of Range Greedy Quantifier



The ranged quantifier specifies a number of iterations the pattern may be repeated. The two examples provided here both have a range of "{5,10}", the label for the looping branch indicates the number of iterations that branch may be followed. The values are one less than specified in the expression since the pattern would have to be matched once before repeating it is an option. So, for these examples, the pattern would be matched once and then the loop would be followed 4 to 9 iterations, for a total of 5 to 10 matches of the pattern.

(\d{5}?) Example of Range Non-Greedy Quantifier



The ranged quantifier specifies a number of iterations the pattern may be repeated. The two examples provided here both have a range of "{5,10}", the label for the looping branch indicates the number of iterations that branch may be followed. The values are one less than specified in the expression since the pattern would have to be matched once before repeating it is an option. So, for these examples, the pattern would be matched once and then the loop would be followed 4 to 9 iterations, for a total of 5 to 10 matches of the pattern. The purpose doesn't change from the above, only the path is made optional instead of required.

Tim Notes:

1. With the other Visualization program, the range of, i.e., {5,10}, displays as '5' iterations and not as '4'. This is because it doesn't discern between the first match and the remainder of the iterations.

Source Material this section: //regexper.com, //www.debuggex.com

Regex Buddy

Regex Buddy by Just Great Software, is another program I heavily relied on for both testing with BRU and for providing visual depictions of the matching and debugging process.

The following should be noted:

1. Regex Buddy always includes the extension as part of the filename - because it deals with text not filenames generally, so the extension, '.zip' is just another part of the string of text. Under Regex Buddy I had to remove any anchor $ (End of Line or EOL) along with some file extensions, otherwise the Captures would not match up with BRU.

   Be aware of this, because I may have inadvertently left some in. So if you find that the Regex Buddy depictions do not match the BRU output or the sample data, it would probably be due to this.

2. Regex Buddy uses string positioning rather than character positioning. Both BRU and RegEx are set for using the same RegEx Engine, it is just that Regex Buddy begins the count starting at position 0, BOL, instead of character position 1. This throws off the count obviously by one character when referencing a character position.

   Why is this significant?

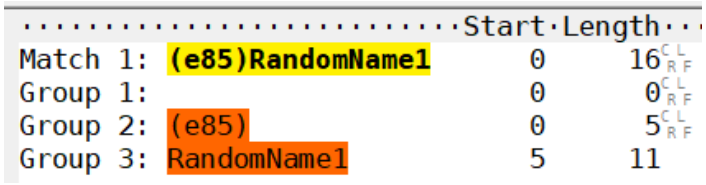
   Because in the examples I illustrated using Regex Buddy, you may notice this discrepancy. I account for this in my notation as much as I can.

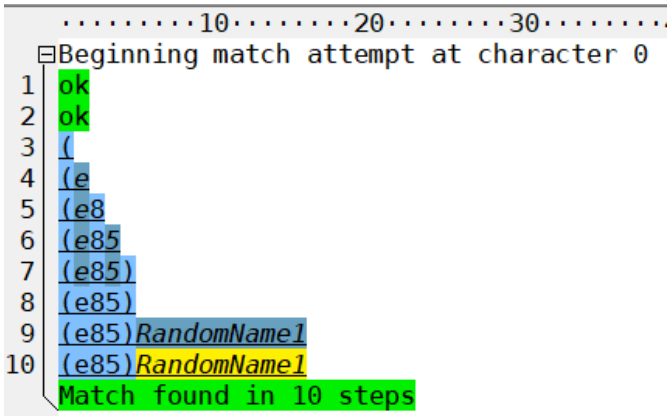
   For example, when referring to the, i.e., 9th character position in Regex Buddy, I would reference this using BRU at position 10.

3. I used Regex Buddy set for the PCRE 7.5-7.6 Language for compatibility with BRU. BRU uses PCRE v7.5.

Here are documentation examples I did from Regex Buddy so you will be able to distinguish them in the book:

Match: `(.*?)(\(e\d\d\))(.*)`





BRU and String Position

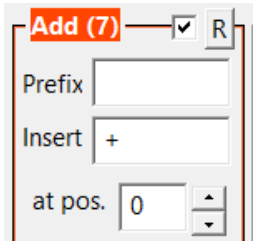
For most of the BRU Renaming Criteria, position 0 is referred to as the 'Prefix Position' and the last position in the string is referred to as the 'Suffix Position'

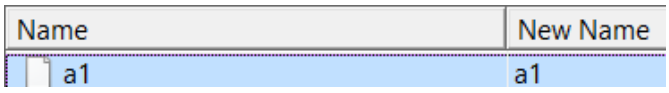
Here is how the criteria uses the position function -

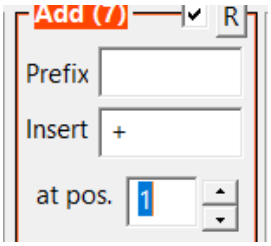
'Section #7: Add' –

Function: Insert:
at pos:

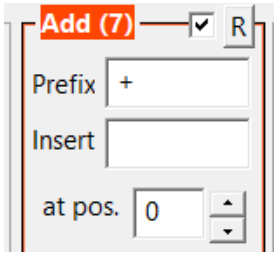
Position 0 – BOL aka Prefix position, – no inserting of characters. Insertion begins at Position 1 in BRU.

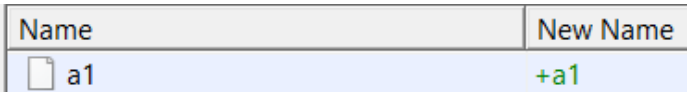






You can also use:





BRU and String Position

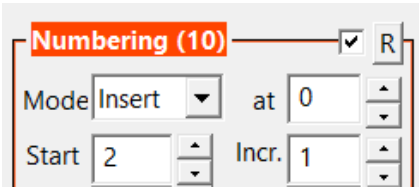
Here is how the criteria uses the position function – cont.

'Section #10: Numbering' –

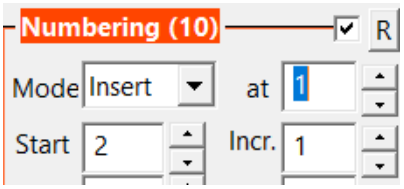
Function: Mode: Insert
at:

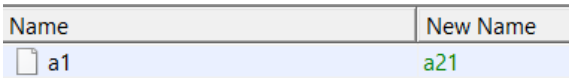
Position 0 aka Prefix position is the first position occupied by a character. Numbering, even if active, is not allowed to be inserted at position 0 and is instead reserved for the Prefix.

Position 0 yields nothing (no change in New Name)

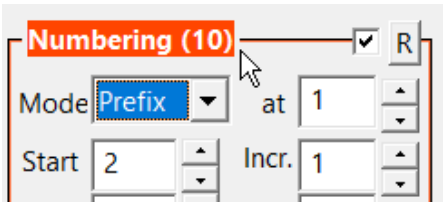


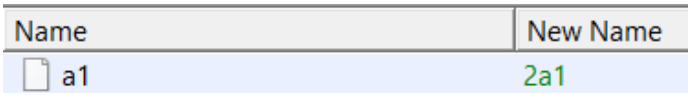
Position begins at Position 1.





Therefore, you can't have '2a1' unless you use Mode: Prefix





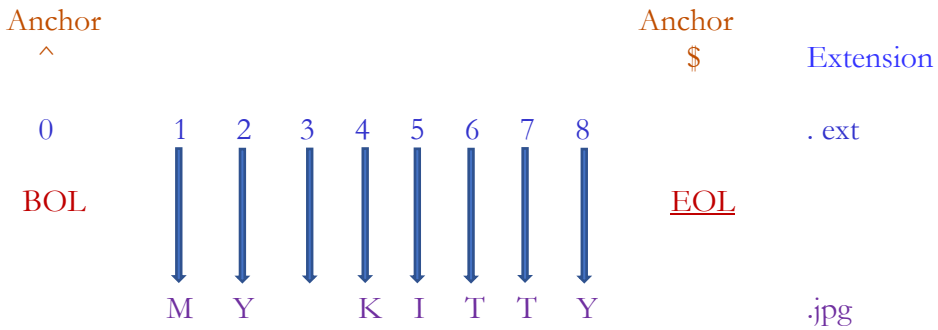
Tim Notes:

1. Activation refers to entering a value into the criteria that enables that criteria to be included with the renaming process. An active criteria can easily be identified by a red border around the criteria and the name of that criteria highlighted also in red. Just because the criteria is active, however, does not mean that the data entered will have any effect.

Character Position vs String Position

Regular Expressions (RegEx)

This is how an 8 character string is expressed in RegEx:



The first position, Position 0, is referred to as the Beginning of Line or BOL. It is indicated by using the ' ^ ' carat Metacharacter. The last position is referred to as the End of Line or EOL. It has no numbered character position and is indicated by the ' $ ' dollar sign Metacharacter. It comes right before the extension of a filename. Both of these are called 'anchors'.

All of the positions in between are character positions.

As you will see in many of the analysis, when the current position is at the EOL it may be because the entire string matched. BRU will not display any name change because there have been no changes. A position of either BOL or EOL are null values. In RegEx, these null values cannot be seen.

For example,

Match: (.*)
Replace: \1

will match up and capture the entire string, i.e., 'a1'.

It will display as:

Name	New Name
a1	a1

If you wanted to see that this did actually match, use a Replacement String of \1 <space>

Name	New Name
a1	a1

Character Position vs String Position

Regular Expressions (RegEx) cont.

Now change the Match to:

Match: (.*^)
Replace: \1

Name	New Name
a1	a1

Same as before, but instead of matching to end of string and staying there, by specifying the BOL Metacharacter, it backtracked to the beginning of the string and matched at BOL. This did not affect the name change so BRU shows no change.

If you want to verify the current position is at BOL, then add a dot character at the end. This will match against any character – and will display the position as a result:

Match: (.*^.)
Replace: \1

Name	New Name
a1	a

The <dot> matched against the 'a' of 'a1' thereby verifying that the current position before the <dot> was evaluated, was held by BOL.

I use this trick a lot during my analysis of the examples to verify my current character position.

If I change the Match to:

Match: (.*a)
Replace: \1

Name	New Name
a1	a

It will backtrack back to the beginning of the string and match against the 'a'.

Tim Notes:

1. Backtracking occurred because when the current position was already at EOL, the RegEx Engine couldn't go forward any further to match against the ' ^ ' or the 'a' so it moved back through the string to try and satisfy the Match.

Character Position vs String Position

Regular Expressions (RegEx) cont.

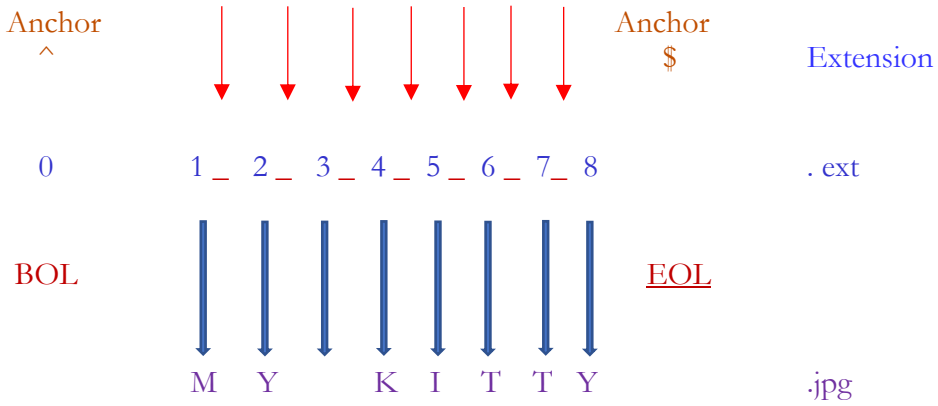
Tim Notes:

I don't want to confuse anyone, so I left this following information in the notations because, for the most part, this book will not delve into this matter too deeply except where necessary.

When talking about positions, there are position *between* each character position that the RegEx Engine can move to. It does this when moving forward or backtracking, testing a previous or next character, etc. Each of these positions are a null value just as BOL and EOL are. Most of these movements are intermediary steps that have no real effect on the outcome so I leave these out in the analysis unless absolutely necessary. I do mention them on occasion referring to a movement '**Just before**' or '**Just after**'. I also might just mention them as a resulting null value.

Just so you know:

This is how an 8 character string is expressed in movement positions in RegEx:



The RegEx Sections

Everything comes down to parsing. I have a definition of parsing somewhere in the book already, but what it comes down to is this – breaking apart the string (filename) into its individual textual data pieces and isolating these pieces. The Replacement String is then used to:

- Re-order the sections of the string
- Remove sections of the string
- Add text or other characters to sections of the strings
- Substitute characters for other characters within the string

The RegEx sections in the book are divided as follows:

Date Manipulation
Examples specifically involved with dates.

String Manipulation (Parsing)
Examples that I use to illustrate parsing for string manipulation and reordering. The RegEx performs the parsing and the Replacement String performs the re-ordering.

Find Occurrences
Specifically use the RegEx to locate a specific area within the string for purposes of adding, removing, substituting.

Filter RegEx
Specific only to 'Section #12: Filters'

Remove
Parse string to isolate. Capture values that may be retained. Don't capture values that will be removed. Use the Replacement String to create a New Name using those values retained and not include those values to be removed.

Add
Parse string to isolate. Add characters through Replacement String.

Element Substitution
Find more than one occurrence in string. Isolate. Use Replacement String to replace.

Uncategorized
Just as the heading indicates. It will be an inclusion of any and all the above that didn't fit into any one category and may also include BRU as well.

Tim Additions

Tim Notes
My personal opinions, observations, insights and addendum information.

Tim Contribution
Much more detailed than what would be in Tim Notes. Usually my own inclusions of examples and original RegEx or variations on RegEx already presented by the Contributor and may also include corrected RegEx to failed attempts by the Contributor. Much of these may be accompanied by my personal Analysis.

Tim Analysis
Take it for what it is – an opinion. You may have your own. My hope is to help readers understand how and sometimes possibly why a RegEx comes to an evaluated conclusion.

There are several forms:

Narrated or Summary Analysis
Instead of going through the steps, I may present the analysis as a summary opinion.

Expression Analysis

This is typically how a RegEx analysis is done. Take each sectional part of the expression and evaluate it as a whole.
i.e.,

(\d\d\d\d-\d\d-\d\d) Get year, month and day

Component Analysis
Yes I made up this term. It is somewhat controversial because I am trying to read the mind and track the direction of a steaming locomotive called the PCRE RegEx Engine. I can't account for every twist and turn – but I attempt to narrow down *what* it does, but where I fail is in my understanding of why it does what it does. I always say that it has a mind of its own - moving back and forth (backtracking) through the string making every attempt to satisfy the RegEx to the point (almost) of illogic. But hey, that's just me.

I break down the RegEx into its individual components. Then I take each component – some refer to these as tokens – but I find the term too broad – and track the evaluated value. I then add the next component to that – slowly building back the original RegEx, piece by piece into each sub-expression. I Look for changes in the component's values as I move through both the string and the RegEx just as I think the RegEx Engine is doing – of course I am not even close because it is doing so much more than I can document. For example, one RegEx I analyzed in 10 steps whereas the engine analyzed in 52 steps. So I base my analysis on the changing values rather than as a step by step process.

My conclusions agree with the final evaluated outcome. The values I come up with are sound even if my technique may be a bit flawed. So why Include it? It is a great learning tool – it is how I learned RegEx. When you study my analysis or better yet do your own, you will learn too.

I don't indicate in the book which analysis approach I use I just do it.

Contributor Analysis
Sometimes I will include the original Contributor's analysis with slight editing perhaps or if heavily edited I may use the term 'Contributor and Tim Analysis'.

Additional Notes:

1. It's nice to have the RegEx and look at the samples and try them for yourself, but it is quite another to understand how they work. If you understand this, then you are learning Regular Expressions. Soon you will be able to create your own. Of course that might put the forums out of business ☺

2. I have extensively added personal material either through notations (Tim Notes) as well as many of my own analysis (Tim Analysis) where indicated. Some analysis' are a combination of the Contributor's along with my own while some examples will only show the Contributor's analysis.

3. Some examples show no analysis whatsoever. This is because either the subject matter involved in the expressions have been previously discussed or they are left for you to make your own analysis. Most have accompanying analysis, though.

4. Then there are the <wink> <wink> those RegEx that work but dang if you can figure how it does. Believe me there are many of the Contributor's who can secretly attest to that. You try this and you try that – you add something here and Eureka! Just don't ask them how they did it. They don't know. – Or is it just me … :{

5. The format is the same throughout the book. They have all been heavily edited.

6. This book is a tutorial, and although it utilizes material from the forums, it is not the forums. There would be no purpose in just copying and pasting information from there.

   Instead of relying on a poster's request for solving a problem, the examples are presented in a universal format using samples, and presenting the techniques involved in renaming those samples to the desired outcome.

   The forums on the other hand are for fulfilling requests and solving problems:

   I have a problem – I want this and that – here's what I did and didn't do, and please help.

   That is what the forums are for. But this volume is a 'BRU By Example', so the concentration is on the techniques leaving out the requests and any other extraneous information.

7. Remember, that this is a companion volume to Volume I. Volume I has a pretty thorough RegEx manual that can be found in the Appendix. It is expected that the reader uses both volumes to get a full understanding of the Regular Expressions language.

8. Contributor's of the final solutions are noted. Examples where there are multiple Contributors are also noted, generally, in a single notation, and not – this one did this and this one did that, although exceptions have been made.

9. 'Observation' section takes in consideration not only personal observations, but puts forward the desired outcome (expressed by the original Poster or Requester) and in some examples illustrates the pattern that will be used in the RegEx for those samples.

10. There may be Contributors who disagree with my analysis. I may have errors or perhaps a misunderstanding of the intentions – that may be true – it is hard to look over someone else's work and be 100% precise. It is also difficult in some cases to understand what the RegEx engine is doing; values change as the evaluation of the expression progresses. In any respect, there is one thing I can say for certain is that all of the examples have been personally tested and all of them working except where specifically noted, and with much of those, I have presented solutions and alternatives.

11. The RegEx presented span from the most simple to the advanced. One thing the reader learns is that there is normally more than one method to obtain a desired outcome. The RegEx are also not supposed to be the best – in fact in some examples I show long meandering steps leading to a useful conclusion.

My Philosophy

In my recent discussion with a colleague of whom I have had many a debate, we discussed what the purpose of the book is or should be.

From these discussions I think it is important to share with you what the RegEx section, especially, is meant to portray to you the reader.

1. To teach. To show you as I have learned and still learning.

Debate # 1 – Should all of the examples, including my own contributions, have a an applicable purpose

I start with a an example from one of our forum experts.
From there depending on the example, I may digress into other areas that are not specifically linked to the originating example, but still important nonetheless.

These digressions may not have any practical value.

But my intent is to show you possibilities.

Some of these digressions lead from one topic to another that advance to an eventual conclusion that ends that particular topic - or it may be picked up, expounded upon or even started over using another example from another forum contributor. Other examples may just be that. Examples. But even those, I will most likely provide a full analysis.

Debate #2 – Should all the RegEx be 'Safe'

Is it Safe? Asks the Nazi Dentist portrayed by Laurence Olivier to Dustin Hoffman, being tortured.

In another discussion we debated whether an example should always be 'safe' or is 'dangerous' allowed? What is meant by this is, should the pattern presented by the RegEx always conform to the specific file(s) to be renamed, thereby removing the risk of inadvertent files being matched and renamed as well? My answer is no. Whether something is 'safe' or 'dangerous' should be left up for the reader to decide. The reader has the tools available to limit the scope either through direct selection, filtering, moving files out of the way physically or using just plain common sense.

If I just used 100% verifiably proven safe RegEx from the forums, this over 1800 page book would be a pamphlet. Besides I try to stay away from examples that are too rigid - too specific - too restrictive in nature that the pattern matched would be too limited to really learn from, unless they serve a purpose in the topic presented.

2. To provide examples that you can try yourself.

This is where the 'BRU by Example' principle comes into play. I encourage all to try them in BRU - just to see for yourself what they do or don't do.

3. To have the knowledge to do it yourself.

This is a given.

My Philosophy cont.

Debate #3 - Analysis: Full Disclosure or Need to Know

Another debate we had was in my analysis. Should the analysis be more detail focused than it already is?

For example, there is an example in the book:

> Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title)

At one point when discussing a variation that used conditional Alternates, I ask the question:

> 'So, question. If the first Alternate matched, why would it end up selecting the Third Alternate, if the RegEx Engine goes with First come, first served?'
>
> The simple explanation is that it didn't match - a least not enough to satisfy the match. When the expression is provided with alternates, upon any failure to match, the RegEx Engine will seek out the next alternate without trying to satisfy the match with further permutations. If the expression did not have any alternates, then the RegEx Engine would continue to try and satisfy the match.

I then go on to explain and prove this using a further breakdown analysis. In this analysis I talk about .+? and its effect on various things happening but I keep it brief. My colleague argued that .+? is doing so much more and having a greater impact on when an Alternate is failing and why, just to give you one example. And this person is right. I am oversimplifying it.

But I argued back that I am more focused on the values changing depending on the outcome of pieces being added to the building of the RegEx, and in turn those values being changed as these pieces come together, than I am about every single aspect of what the RegEx Engine does to get those values. A lot of intermediary steps.

I have or will mention, depending upon where this little disclaimer appears in the book, the fact that I am not interested in documenting those intermediary steps - important, relevant or otherwise. For example, I analyzed a RegEx in 10 steps while the RegEx Engine actually performed over 50 steps. I find no value in documenting 50 steps and I am sure, that you would not want that either, no matter how much accuracy it would add.

Conclusion:

I explain to my colleague that the analysis' are an opinion - mine. You, the reader, can have your own by doing your own self-analysis - and again, I encourage you to do just that. Right or wrong, it is just an opinion. Take it for what it is.

My Philosophy cont.

Debate #4 – Some of the Title Headings may be Misleading as to the Content

I did three things in relation to the headings.

1. In some instances I kept the original Poster's Heading.
   This may account for some confusion because most Posters at the time don't really understand what their problem is and have trouble defining it for others. But if at all possible, I do try to use the original heading.

2. I edited them.
   If I saw that the title is overlong (many of them are extremely long even edited down) or incoherent, I may change it around a little bit.

3. I replaced it.
   To be more in line with what I discuss under the topic, I may have changed the content so much that the original title no longer applies.

In any regard, this book stands on its own. It is not meant to be an index to the Forums.

My Personal Observations

Part of my career was spent in music. I was a pianist, keyboardist and percussionist and taught same throughout the years at various times. Nothing unusual in that. However, I didn't have the use of my little fingers on both hands. For those of you that are not musicians, let me inform you that this is a great hindrance in both learning and performing. For the keyboards and piano, I had to adjust stretching my fingers farther out more than necessary and adapting most chording to four fingers instead of five. Of course, certain chords could never be played but I adapted so that they sounded similar.

As for teaching? Just because you can't do, doesn't mean you can't teach, as the old adage goes.

My point is, that I am not intimidated at learning something in the unconventional way. This book is proof of that. My methodology may not be 100% accurate for the purists out there, but it does work. My entire method is an adaptation.

Final Conclusion:

By looking at the examples presented, by familiarizing yourself with the Metacharacters, by studying the analysis and performing your own self-analysis, you *will* learn Regular Expressions. I would suggest having the RegEx Manual of Volume I and/or RegEx Manual Volume II open as you read the various sections related to RegEx.

Along with all of the other numerous and various topics I present, this book is nothing I have ever seen before in a technical volume. I believe that it is a unique, all-inclusive volume – although I could be wrong – but personally, in all of the books I have owned, seen, read etc., I have never come across anything like it. It is my hope that you will favorably agree.

Testing the Waters

Before starting, here is a quick rundown of what to expect in the coming pages. It uses a summary analysis.

Metallica - Enter Sandman to Enter Sandman – Metallica

Match: (.*)\s-\s(.*)
Replace: \2 - \1

Simply put the first part - the RegEx string, parses the string (breaks it apart) so portions can be isolated and the values of those portions 'captured'.

The second part - the Replacement string, is used to re-arrange those parts, add new characters, or remove characters by not including some parts that were isolated but not referenced back in the Replacement String.

It is the Replacement string that determines what you see under the New name column. (If successful, New name will be Green - if not, New Name will be in normal black text, meaning that the RegEx may have failed to match, or if it did match, there was no change in the name, and if in error, New Name may be in Red as Invalid). When you click on Rename - that is what you get.

The evaluation is done by what is called the RegEx Engine - think of it as a train on a track. It moves forward and backwards trying to satisfy a match. As it does, however, it can change values that were previously captured.

(.*) the <dot> Metacharacter will match against any character. The ' * ' is a Greedy Quantifier that will match zero or more times meaning - capture as much of the filename (string) as possible. It appears at the beginning to 'eat up the string and capture it in the first Capture Group - the first set of parentheses indicated by (.*). This would be Capture Group 1 referenced in the Replacement String as \1.

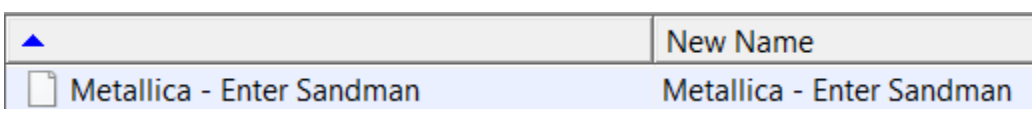
First it matches against the 'M' of 'Metallica'. Next it captures the entire string in Capture Group 1 by moving forward until it has reached the end of the string (called EOL - End of Line).

Metallica <space> <hyphen> <space> Enter Sandman EOL

Capture Group 1 = <entire string value>

Metallica <space> <hyphen> <space> Enter Sandman EOL

Because Capture Group 1 is essentially the same name as the original, in BRU, New name will not reflect any change.



Testing the Waters

Metallica - Enter Sandman to Enter Sandman – Metallica

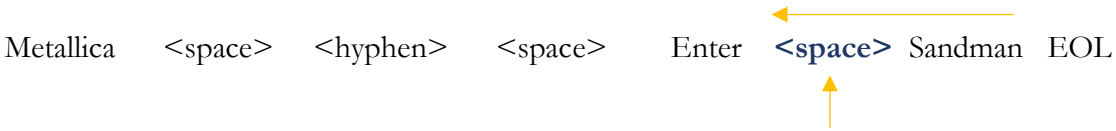
Match: (.*)\s-\s(.*)
Replace: \2 - \1

The <spaces> between the components that make up the RegEx are equally important. The \s is a <space> Metacharacter. It will search the string for a <space> character, but since it has reached the end, it moves backward (this is called backtracking) and will match against the first <space> it finds.

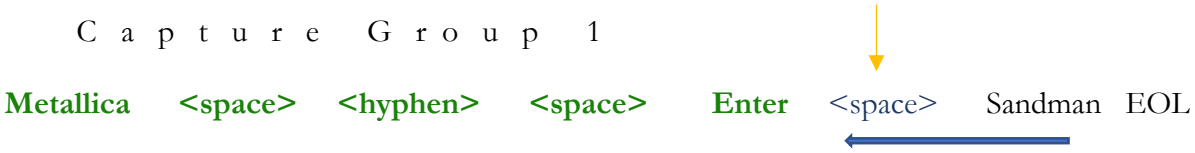
In this example -

Metallica - Enter Sandman

The first <space> it encounters moving backwards is the **<space>** after 'Enter'.



This changes the value of Capture Group 1 to the preceding characters leading up to that <space>

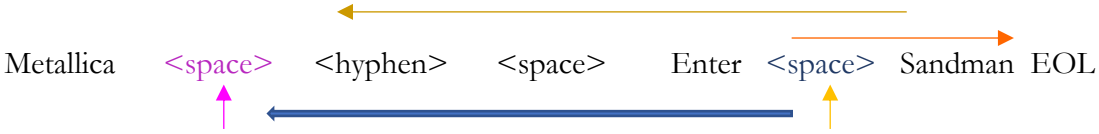


so now ….

Capture Group 1 = Metallica - Enter

The \s value (the <space>) has no parentheses so the value is not captured (retained).

The Next component, the <hyphen> changes things again. The RegEx Engine moves forward through the string looking to match a <hyphen> - it can't find any and reaches the end once more, so it moves backwards again this time it finds a <hyphen> located after 'Metallica <space>' . This again is not 'Captured'.



This changes the **PREVIOUS** \s from the <space> **after 'Enter'** to the **<space> after Metallica**.

You see how the values change?

Testing the Waters

Metallica - Enter Sandman to Enter Sandman – Metallica

Match: (.*)\s-\s(.*)
Replace: \2 - \1

So when the <space> changed, this in turn changed Capture Group 1 from **'Metallica - Enter'** to **'Metallica'**

C a p t u r e G r o u p 1

Metallica <space> <hyphen> <space> Enter Sandman

to:

Metallica <space> **<hyphen>** **<space>** **Enter** <space> Sandman EOL

Now the word 'Metallica' has been isolated (parsed).

Capture Group 1

Metallica <space> <hyphen> <space> Enter <space> Sandman EOL

The Next component is also a <space>, the \s.

It is important to note when performing an analysis, where, at what position in the string, is the RegEx Engine. The last match was the <hyphen>. Thus, the current position of where the RegEx Engine is on the 'track' is at that point:

Metallica - Enter Sandman

Moving forward, the \s will match against the <space> that follows the <hyphen> preceding Enter - again not captured.

Why not Capture the ' <space> <hyphen> <space> ' ?

It may not be needed. Simple as that. But, if we want a <hyphen> or <space> in New name, these can be added back in using the Replacement String or because it is 'removed' from being captured, it gives the flexibility of adding anything else in substitution.

Testing the Waters

Metallica - Enter Sandman to Enter Sandman – Metallica

Match: (.*)\s-\s(.*)
Replace: \2 - \1

So What's left to evaluate?

The (.*)

This indicates to the RegEx Engine to move forward and capture from that point on to the end of the string, the EOL (remember it is Greedy and will match zero or more times – meaning, 'eat everything in sight'). Because the ' .* ' is in parentheses, this value gets 'captured' in a another Capture Group, designated as Capture Group 2.

So each parentheses group in this RegEx represents a Capture Group, but just be aware that there are actually parentheses groups in RegEx that are NOT capture Groups. These are called Non marking Groups.

The rest of the string is captured in Capture Group 2 from the current position forward (currently at the <space> preceding 'Enter')

Capture Group 1 Capture Group 2

Metallica <space> <hyphen> <space> **Enter <space> Sandman** EOL

Not Captured

resulting in:

Capture Group 2 = 'Enter Sandman'

We have now isolated 'Metallica' and 'Enter Sandman'. The Parsing is finished. The RegEx String is finished and the filename string is finished - why?

#1 - no more components of the RegEx String remain - all have been evaluated

#2 - reached EOL and 'typically' when .* is used at the end of the RegEx the Engine cannot backtrack

#3 - the filename string is what is referred to as *exhausted* - meaning no more matches can be made even if there were more components of the RegEx String remaining.

This is where the fun comes in. The RegEx has done its job. There are no errors.

Testing the Waters

Metallica - Enter Sandman to Enter Sandman – Metallica

Match: (.*)\s-\s(.*)
Replace: \2 - \1

What is left to do is to use the Replacement String to piece together (rearrange) the filename in the order you want.

Remember the Replacement string can reference the values 'captured' by referencing them using the format:

\ <reference number>

Where:

reference number refers to the Capture Group designation, i.e. \1 represents Capture Group 1

In addition, literal characters (a literal is a character that can be reproduced on a keyboard) can be added. In this example, both the <space> and <hyphen> literal characters have been added back into the Replacement String

Here is what we have:

Capture Group 1 = Metallica

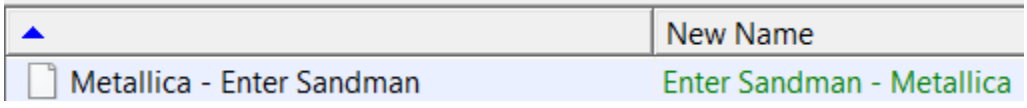
Capture Group 2 = Enter Sandman

Replacement String = \2 <space> <hyphen> <space> \1

The Final result is that the two words of the string have been rearranged and separated by the …

<space> <hyphen> <space>

= Enter Sandman - Metallica

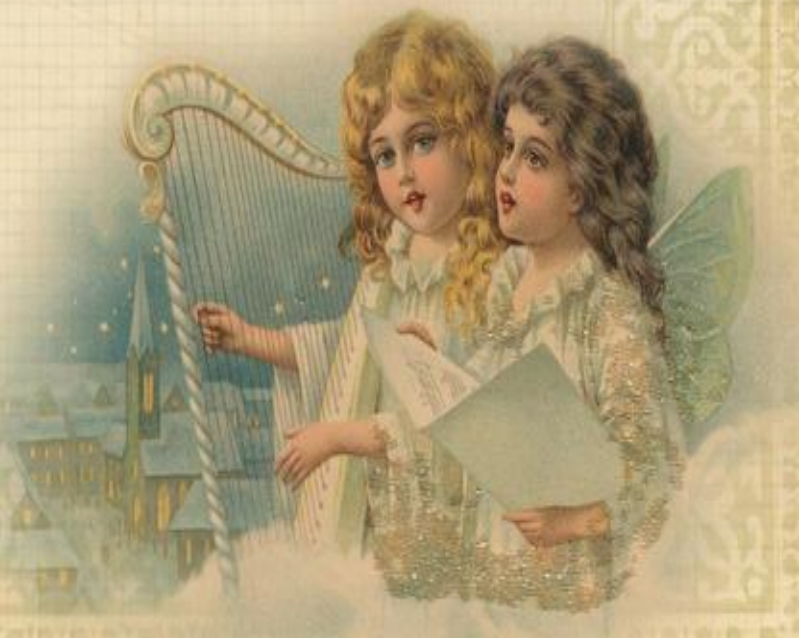


Tim Notes:

1. Most of the examples in this book won't be as simplified or as elaborately illustrated as this. You are expected to read the RegEx Manuals located in the Appendix of each volume and any corresponding material that may be indicated.

Enjoy.

Forum Contributor for the RegEx: therube

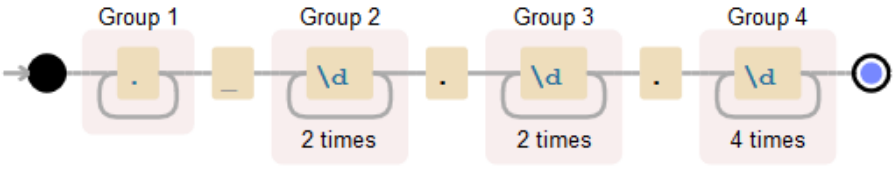
Date Manipulation

Date Manipulation

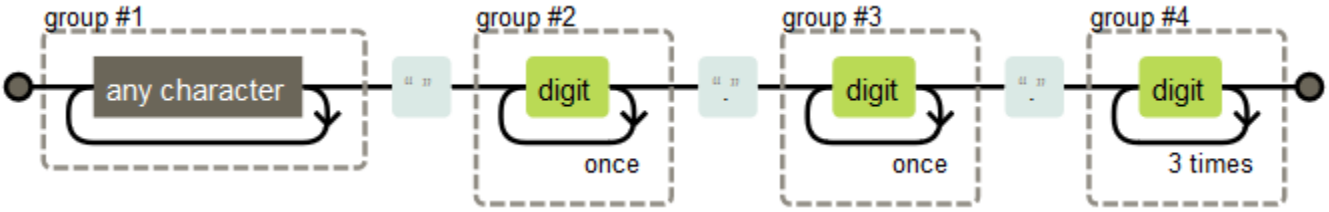
Move Date M.D.Y to front

Mobile Production 01.03.2015.pdf to 2015.01.03 Mobile Production.pdf

Match: (.+) (\d{2})\.(\d{2})\.(\d{4})
Replace: \4.\2.\3 \1 <space> after \3

Name	New Name
Mobile Production 01.03.2015.pdf	2015.01.03 Mobile Production.pdf





Sample:

```
.....................................
Match 1: Mobile Production 01.03.2015
Group 1: Mobile Production
Group 2: 01
Group 3: 03
Group 4: 2015
```

Forum Contributor: Admin

Tim Analysis:

1. .+	Match against characters until…	Capture Group 1 = Mobile Production
2. \d{2}	.. 2 numeric digits (Month) followed by ….	Capture Group 2 = 01
3. \.	.. a dot character (Not Captured), followed by …	
4. \d{2}	.. 2 numeric digits (Day) followed by …	Capture Group 3 = 03
5. \.	.. a dot character (Not Captured), followed by …	
6. \d{4}	.. 4 numeric digits (Year)	Capture Group 4 = 2015

Tim Notes:

1. A literal dot is preceded by an Escape Character \ in the RegEx. The Escape character is not required in the Replacement String for the same literal dot character.

Date Manipulation

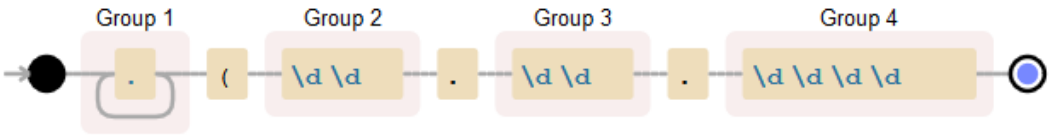
Convert Date (M.D.Y) to (Y.M.D)

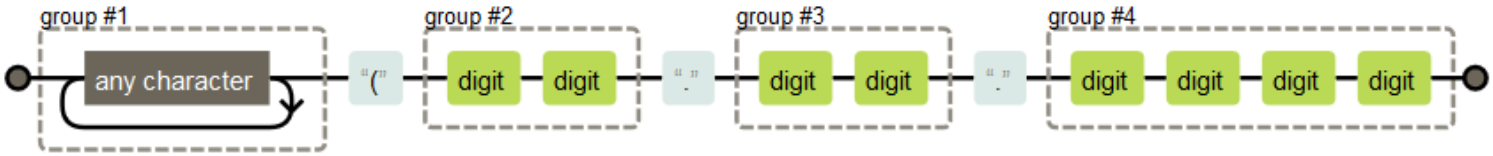
Production company1 (30.11.2015).pdf to Production company1 (2015.11.30).pdf

Example #1

Match: (.+)\((\d\d)\.(\d\d)\.(\d\d\d\d)
Replace: \1 (\4.\3.\2)

Name	New Name
Production company1 (30.11.2015).pdf	Production company1 (2015.11.30).pdf





Sample:

```
........................................
Match 1: Production company1 (30.11.2015
Group 1: Production company1
Group 2: 30
Group 3: 11
Group 4: 2015
```

Remove parentheses – use \1 \4.\3.\2

Name	New Name
Production company1 (30.11.2015).pdf	Production company1 2015.11.30.pdf

Forum Contributor: Hasse

Date Manipulation

Convert Date (M.D.Y) to (Y.M.D) cont.

Using: Production company1 (30.11.2015).pdf

Match: (.+)\((\d\d)\.(\d\d)\.(\d\d\d\d)

Where:

\1 = Production company1 <space> \2 = 30 \3 = 11 \4 =2015

Tim Analysis:

1. .	Match against any character	Capture Group 1 = P
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \(	Match against right parenthesis (Not Captured) followed by…	Changes Capture Group 1 = Production Company1<space>
4. \d\d	.. Match against Day followed by…	Capture Group 2 = 30
5. \.	.. Match against literal <dot> (Not Captured) followed by…	
6. \d\d	.. Match against Month followed by…	Capture Group 3= 11
7. \.	.. Match against literal <dot> (Not Captured) followed by…	
8. \d\d\d\d	.. Match against Year	Capture Group 4 = 2015

Tim Notes:

1. You can see the similarity between this example and 'Move Date M.D.Y to front'. One of the main differences is that here each numeric digit is represented singularly by \d and in the former, the numeric digits are represented by a set of,

   i.e., \d{2} for \d\d.

2. A literal left parenthesis is represented by using the Escape character as in \(

   a. Because this is common knowledge, literals will not be discussed in the notations as much from this point forward in the book except where a literal can be mistaken as something else. For example, here at first glance it appeared that there was a nested Group (parentheses within parentheses), but this wasn't the case.

3. Didn't need to bother with the right parenthesis since it was not used in the Rename process and the Year was already captured. A lot of the RegEx examples would have used ' .* ' to capture the remaining string or account for it specifically using ' \) ' or something similar.

Date Manipulation

Convert Date (M.D.Y) to (Y.M.D) cont.

Addition:

The following will also work: (Tim Contribution for Replace string and Admin for RegEx)

Example #2

Match: (.+)(\d{2})\.(\d{2})\.(\d{4}) Same RegEx used in 'Move Date M.D.Y to Front' ..
Replace: \1\4.\3.\2 .. just different replace string

Name	New Name
Production company1 (30.11.2015).pdf	Production company1 (2015.11.30).pdf

Finally ….

To illustrate how easy the RegEx can be used on a sample that does not have parentheses:

Tim Contribution –

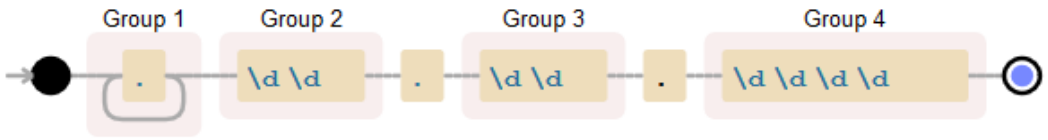
Filename without parentheses

Production company1 30.11.2015.pdf to Production company1 2015.11.30.pdf

Example #3

Match: (.+)(\d\d).(\d\d)\.(\d\d\d\d)
Replace: \1 \4.\3.\2 - <space> after \1

Name	New Name
Production company1 30.11.2015.pdf	Production company1 2015.11.30.pdf



Sample: (same for both Example #2 and Example #3)

```
.....................................
Match 1: Production company1 30.11.2015
Group 1: Production company1
Group 2: 30
Group 3: 11
Group 4: 2015
```

Date Manipulation

Convert Date (M.D.Y) to (Y.M.D) cont.

Tim Contribution – cont.

Filename without parentheses cont.

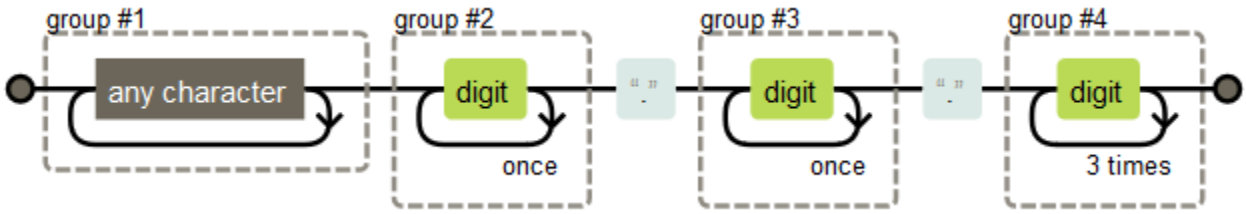
Using: Production company1 (30.11.2015).pdf

Example #1:

Match: (.+)\((\d\d)\.(\d\d)\.(\d\d\d\d)

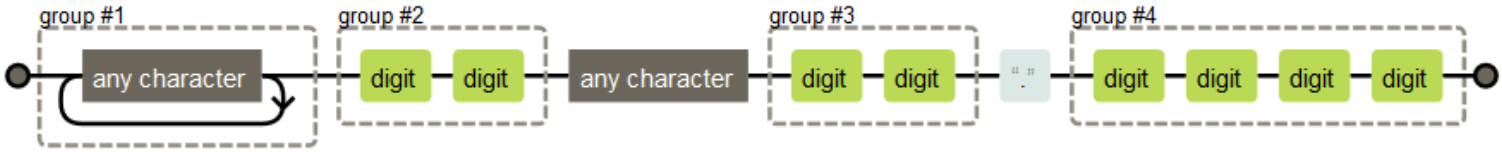
Example: #2:

Match: (.+)(\d{2})\.(\d{2})\.(\d{4})



Example #3:

Match: (.+)(\d\d).(\d\d)\.(\d\d\d\d)



Tim Notes:

1. In the first example, the parentheses and dot characters were matched in the expression and not captured; the parentheses and dots were then added back in using the Replacement string.

2. In the second example, the left parenthesis was captured in the expression but not the right parenthesis. The dot characters were matched but not captured. Both the dots and the right parenthesis were added back in using the Replacement string.

3. In this last example, no parentheses were in the sample. The RegEx parsed the string, ignored the dot characters and replaced them in the Replacement string.

Date Manipulation

Convert Date (M.D.Y) to (Y.M.D) cont.

Tim Contribution – cont.

Filename without parentheses cont.

Using: Production company1 (30.11.2015).pdf

Example #2

Match: (.+)(\d{2})\.(\d{2})\.(\d{4})

Where:

\1 = Production Company1<space> \2 = 30 \3 = 11 \4 = 2015

Tim Analysis:

1. .	Match against any character	Capture Group 1 = P
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \d	Match against Numeric Digit	Capture Group 2 = 5 Changes Capture Group 1 = Production Company1 (30.11.201
4. {2}	Match last expression two iterations	Capture Group 2 = 15 Changes Capture Group 1 = Production Company1 (30.11.20
5. \.	Match against literal <dot> (Not Captured) followed by…	
6. \d	Match against Numeric Digit	Capture Group 3 = 2 Changes Capture Group 2 = 11 Changes Capture Group 1 = Production Company1 (30.
7. {2}	Match last expression two iterations	Capture Group 3 = 20 Capture Group 2 = Unchanged Capture Group 1 = Unchanged
8. \.	Match against literal <dot> (Not Captured) followed by…	
9. \d	Match against Numeric Digit	Capture Group 4 = 2 Changes Capture Group 3 = 11 Changes Capture Group 2 = 30 Changes Capture Group 1 = Production Company1 <space>
7. {4}	Match last expression four iterations	Capture Group 4 = 2015 All Others = Unchanged

Date Manipulation

Convert Date (M.D.Y) to (Y.M.D) cont.

Tim Contribution – cont.

Filename without parentheses cont.

Using: Production company1 30.11.2015.pdf

Example #3

Match: (.+)(\d\d).(\d\d)\.(\d\d\d\d)

Where:

\1 = Production Company1<space> \2 = 30 \3 = 11 \4 = 2015

Tim Analysis:

1. .	Match against any character	Capture Group 1 = P
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \d\d	.. Match against two Numeric Digits followed by…	Capture Group 2 = 15 Changes Capture Group 1 = Production Company1 30.11.20
4. .	Match against any character (Not Captured) = 5	Changes Capture Group 2 = 01 Changes Capture Group 1 = Production Company1 30.11.2
5. \d\d	.. Match against two numeric digits followed by… Changes Character Not Captured = <dot>	Capture Group 3 = 20 Changes Capture Group 2= 11 Changes Capture Group 1 = Production Company1 30.
6. \.	.. Match against literal <dot> (Not Captured) followed by… This is the <dot> after '11'	Changes Capture Group 3 = 11 Changes Capture Group 2 = 30 Changes Capture Group 1 = Production Company1 <space>
7. \d\d\d\d	.. Match against 4 numeric digits	Capture Group 4 = 2015 All Others = Unchanged

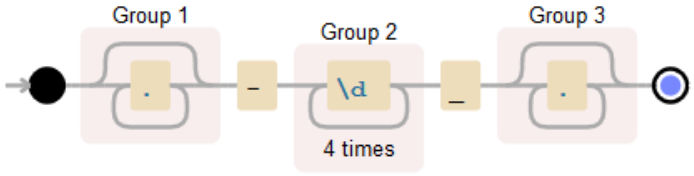
Date Manipulation

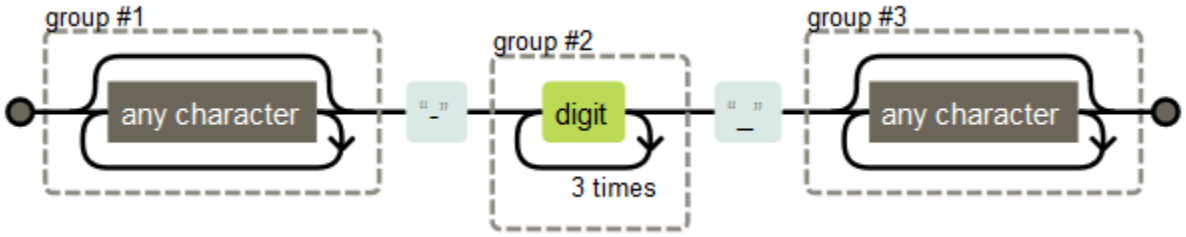
Convert M-D-Y to Y-M-D

05-27-2016_095636.pdf to 2016-05-27_095636.pdf

Match: (.*)-(\d{4})_(.*)
Replace: \2-\1_\3

Name	New Name
05-27-2016_095636.pdf	2016-05-27_095636.pdf





Sample:

```
Match 1: 05-27-2016_095636
Group 1: 05-27
Group 2: 2016
Group 3: 095636
```

Forum Contributor: KenP

Tim Analysis:

1. .*	Match against all characters until …	Capture Group 1 = 05-27
2. -	.. <hyphen> (Not Captured), followed by…	
3. \d{4}	.. 4 numeric digits (Year), followed by…	Capture Group 2 = 2016
4. _	.. underscore (Not Captured)	
5. .*	Capture remainder of string	Capture Group 3 = 095636

Date Manipulation

Tim Note –

1. The following pages contain good examples of using \d to parse date information

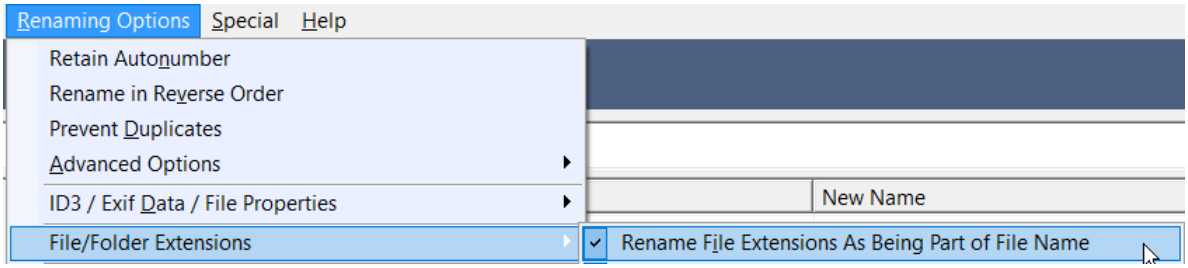
Move Date to Front

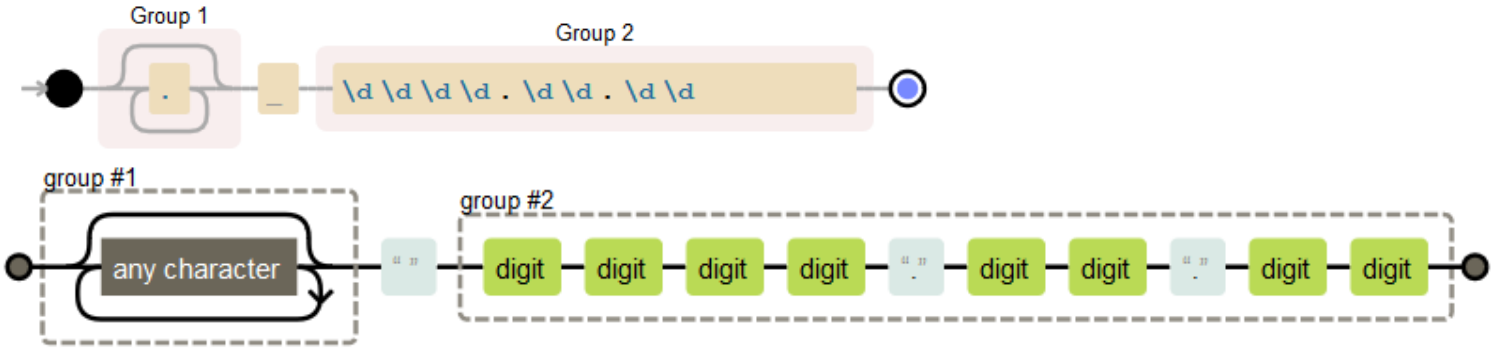
Metallica vs. Manchester Apollo 1988.10.08 to 1988.10.08 - Metallica vs. Manchester Apollo

Match: (.*) (\d\d\d\d\.\d\d\.\d\d)
Replace: \2 - \1

Name	New Name
Metallica vs. Manchester Apollo 1988.10.08.pdf	1988.10.08 - Metallica vs. Manchester Apollo.pdf

Because the samples use dots within the filenames, you have to enable 'Rename File extensions As Part of File Name' under the 'ID3 /Edif Data / File Properties' submenu of the 'Renaming Options Menu'.





Sample:

```
.....................................................
Match 1: Metallica vs. Manchester Apollo 1988.10.08
Group 1: Metallica vs. Manchester Apollo
Group 2: 1988.10.08
```

Forum Contributor: therube

Date Manipulation

Move Date to Front cont.

Using: Metallica vs. Manchester Apollo 1988.10.08

Match: (.*) (\d\d\d\d\.\d\d\.\d\d)

Where:

\1 = Metallica vs. Manchester Apollo \2 = 1988.10.08

Tim Analysis:

1. .	Match against any character	Capture Group 1 = M
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against <space> (Not Captured) This is the <space> following 'Apollo'	Changes Capture Group 1 = Metallica vs. Manchester Apollo
3. \d\d\d\d	Match against 4 numeric digits	Capture Group 2 = 1988
4. \.	Match against a literal <dot>	Capture Group 2 = 1988.
5. \d\d	Match against two numeric digits	Capture Group 2 = 1988.10
6. \.	Match against a literal <dot>	Capture Group 2 = 1988.10.
7. \d\d	Match against two numeric digits	Capture Group 2 = 1988.10.08 Capture Group 1 = Unchanged

Tim Notes:

1. There will be typos. Count on it. This was a massive project and if I had three more months maybe I would have caught more of them - maybe not.

   I had been over this document many times - too numerous to tell you. Each time through I would find a few more. But after a while I felt like that woman in Andromeda Strain where the solution was before her, but her Epilepsy prevented her from seeing it.

   An example... page 125, in the analysis of Example #1, step #2 should obviously be Capture Group 1 = 26, not Capture Group 2. This is not only common sense if you read through the analysis, but for every example, I provide multiple sources of information that allow you to discern the truth from fiction. If you look at the sample run photo for the example #1 on page 124, it clearly shows '26' as the value for Capture Group 1. Same for 'Where... Capture Group 1 = 26' appearing on page 125.

   So if something doesn't look right, look at the other information presented and in all likelihood it may be a minor typo.

2. In this revision, this typo has been fixed, but I wanted the reader to understand what to do if confronted by something that doesn't make sense or causes confusion.

Date Manipulation

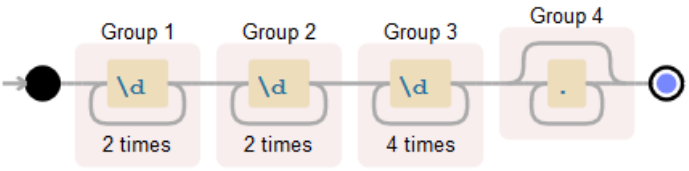
Convert DMY to Y-M-D

26062015_1.pdf to 2015-06-26_1.pdf

Example #1

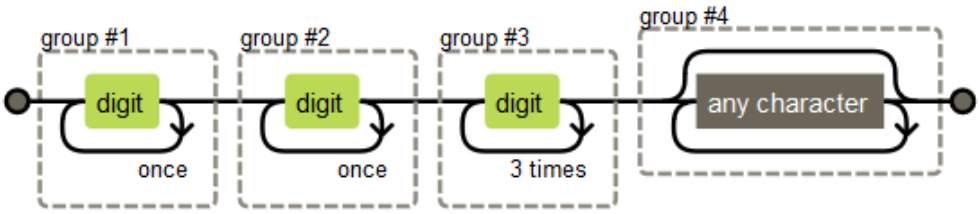
Match: (\d{2})(\d{2})(\d{4})(.*)
Replace: \3-\2-\1\4

Name	New Name
26062015_1.pdf	2015-06-26_1.pdf



Sample:

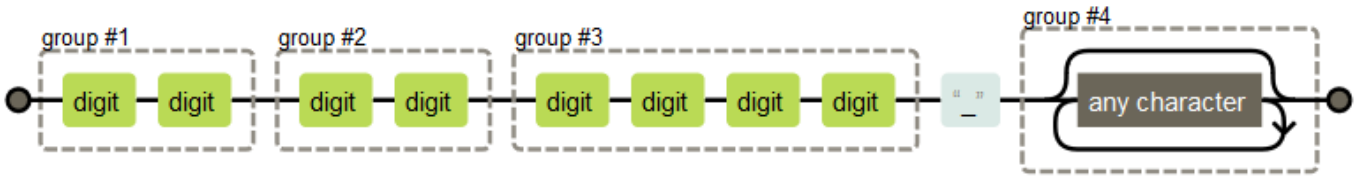
```
.....................
Match 1: 26062015_1
Group 1: 26
Group 2: 06
Group 3: 2015
Group 4: _1
```



Example #2

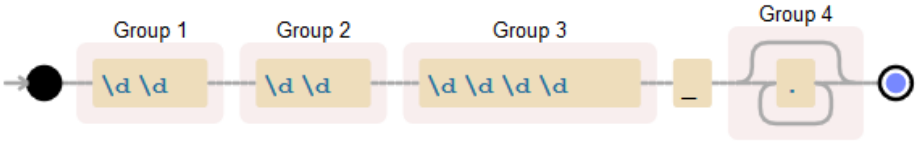
Match: (\d\d)(\d\d)(\d\d\d\d)_(.*)
Replace: \3-\2-\1_\4

Name	New Name
26062015_1.pdf	2015-06-26_1.pdf



Sample:

```
.....................
Match 1: 26062015_1
Group 1: 26
Group 2: 06
Group 3: 2015
Group 4: _1
```



Forum Contributor: @pepemendes

Date Manipulation

Convert DMY to Y-M-D cont.

Using: 26062015_1.pdf

Example #1

Match: (\d{2})(\d{2})(\d{4})(.*)

Where:

\1 = 26 \2 = 06 \3 = 2015 \4 = _1

Tim Analysis:

1. \d	Match against Numeric Digit	Capture Group 1 = 2
2. {2}	Match last expression two iterations	Capture Group 1 = 26
3. \d	Match against Numeric Digit	Capture Group 2 = 0
4. {2}	Match last expression two iterations	Capture Group 2 = 06
5. \d	Match against Numeric Digit	Capture Group 3 = 2
6. {4}	Match last expression four iterations	Capture Group 3 = 2015
7. .*	Match against remaining string	Capture Group 4 = _1

Using: 26062015_1.pdf

Example #2

Match: (\d\d)(\d\d)(\d\d\d\d)_(.*)

Where:

\1 = 26 \2 = 06 \3 = 2015 \4 = 1

Tim Analysis:

1. \d\d	Match against two Numeric Digits	Capture Group 1 = 26
2. \d\d	Match against two Numeric Digit	Capture Group 2 = 06
3. \d\d\d\d	Match against four Numeric Digits	Capture Group 3 = 2015
4. _	Match against <underscore> (Not Captured)	
5. .*	Match against remaining string	Capture Group 4 = 1

Date Manipulation

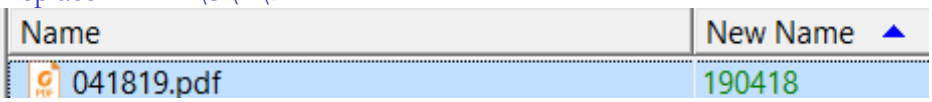
Convert MDY into YMD so the files will list Chronologically

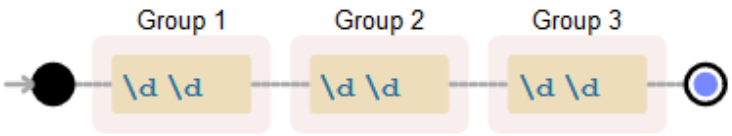
041819 to 190418

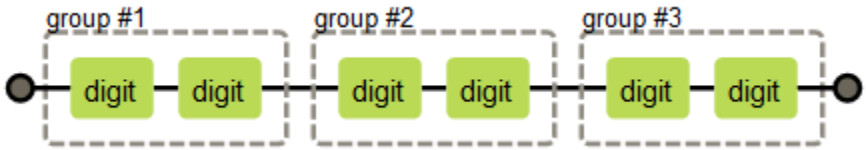
Observations:

1. Capture each part of the date and rearrange them.

Match: (\d\d)(\d\d)(\d\d)
Replace: \3\1\2







Sample:

```
.................
Match 1: 041819
Group 1: 04
Group 2: 18
Group 3: 19
```

Forum Contributor: therube

Date Manipulation

Convert MDY into YMD so the files will list Chronologically cont.

041819 to 190418

Match: (\d\d)(\d\d)(\d\d)

Where:

\1 = 04 \2 = 18 \3 = 19

Tim Analysis:

1. \d\d	Match against two Numeric Digits	Capture Group 1 = 04
2. \d\d	Match against two Numeric Digit	Capture Group 2 = 18
3. \d\d	Match against two Numeric Digits	Capture Group 3 = 19

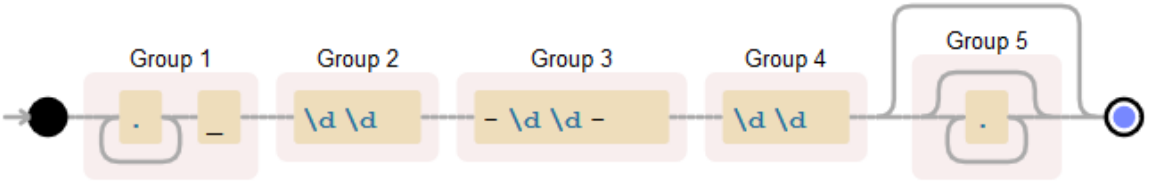
Date Manipulation

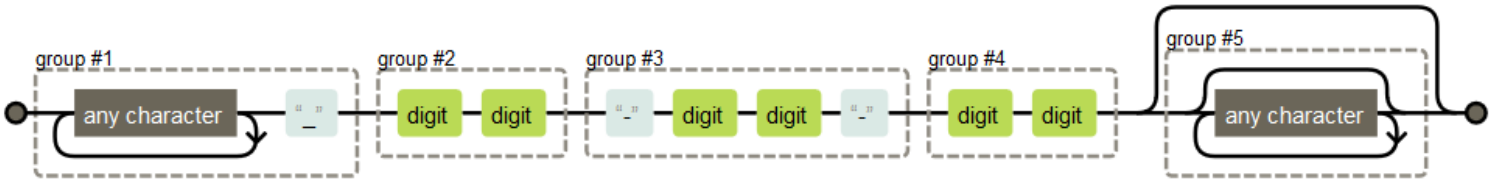
Convert date from mm-dd-yy to yyyy-mm-dd

file_04-14-12.pdf	to	file_12-14-04.pdf
file_06-23-12-signed.pdf	to	file_12-23-06-signed.pdf
file_01-09-05.pdf	to	file_05-09-01.pdf

Match: (.+_)(\d\d)(-\d\d-)(\d\d)(.*)?
Replace: \1\4\3\2\5

Name	New Name
file_06-23-12-signed.pdf	file_12-23-06-signed.pdf
file_04-14-12.pdf	file_12-14-04.pdf
file_01-09-05.pdf	file_05-09-01.pdf





Sample:

```
................................
Match 1: file_06-23-12-signed
Group 1: file_
Group 2: 06
Group 3: -23-
Group 4: 12
Group 5: -signed
```

Forum Contributor: Stefan

Date Manipulation

Convert date from mm-dd-yy to yyyy-mm-dd cont.

Using: file_04-14-12.pdf

Match: (.+_)(\d\d)(-\d\d-)(\d\d)(.*)?

Where:

\1 = file_ \2 = 04 \3 = -14- \4 = 12 \5 = null

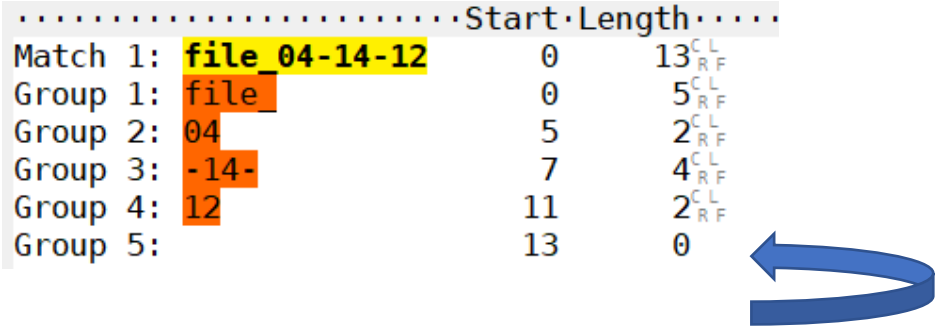
Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. _	Match against <underscore> Backtracks to match the underscore	Changes Capture Group 1 = file_
4. \d\d	Match against two Numeric Digits	Capture Group 2= 04
5. -	Match against <hyphen>	Capture Group 3 = -
6. \d\d	Match against two Numeric Digits	Capture Group 3 = -14
7. -	Match against <hyphen>	Capture Group 3 = -14-
8. \d\d	Match against two Numeric Digits	Capture Group 4 = 12
9. .*	Match against remaining string (Greedy)	Capture Group 5 = null
10. ?	But not too Greedy	Capture Group 5 = No Change

Tim Notes:

1. Capture Group 5 is used to capture any remaining text for those files that have text following the numeric digits. Otherwise a Zero Occurrence Match takes place resulting in a null value.

Sample:



2. A Zero Occurrence Match (some refer to it as Zero Length Matches) takes place, for example, when a Lazy Quantifier is used, i.e., ' ? ' but no text is captured as a result. The Lazy Quantifier makes the match optional resulting in a null value. Other Quantifiers, including the ' * ' which will match zero or more times, may also result in a null value depending on the evaluation.

Date Manipulation

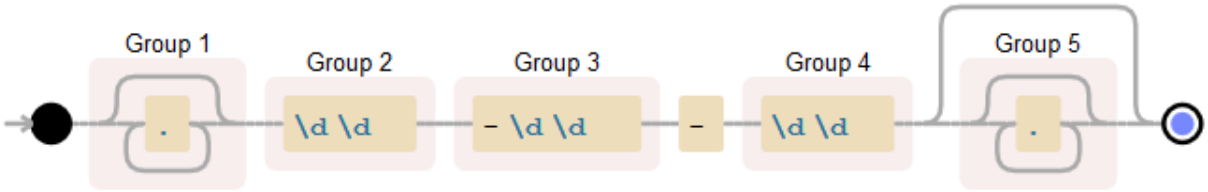
Convert date from mm-dd-yy to yyyy-mm-dd cont.

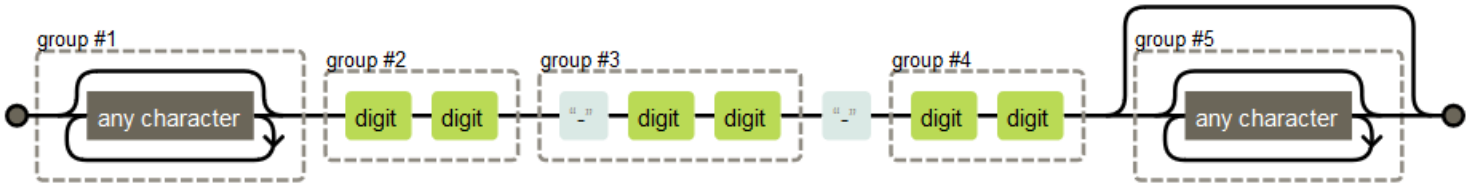
Similar:

Axiom Black Cartridge Receipt 01-09-12.pdf	to	Axiom Black Cartridge Receipt 2012-01-09-.pdf
Calendars packing slip 09-29-11.pdf	to	Calendars packing slip 2011-09-29-.pdf

Match: (.*)(\d\d)(-\d\d)-(\d\d)(.*)?
Replace: \120\4-\2\3\5

Name	New Name
Calendars packing slip 09-29-11.pdf	Calendars packing slip 2011-09-29.pdf
Axiom Black Cartridge Receipt 01-09-12.pdf	Axiom Black Cartridge Receipt 2012-01-09.pdf





Sample:

```
                                                  Start Length
Match 1: Axiom Black Cartridge Receipt 01-09-12       0     38
Group 1: Axiom Black Cartridge Receipt                0     30
Group 2: 01                                          30      2
Group 3: -09                                         32      3
Group 4: 12                                          36      2
Group 5:                                             38      0
```

Forum Contributor: Stefan

Date Manipulation

Convert date from mm-dd-yy to yyyy-mm-dd cont.

Using: Calendars packing slip 09-29-11.pdf

Match: (.*)(\d\d)(-\d**d)-(\d**\d)(.*)?

Where:

\1 = Calendars packing slip \2 = 09 \3 = -29 \4 = 11 \5 = null

Tim Analysis:

1. .	Match against any character	Capture Group 1 = C
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. \d\d	Match against two Numeric Digits Backtracks.	Capture Group 2= 11 Changes Capture Group 1 = Calendars packing slip 09-29-
4. -	Match against <hyphen>	Capture Group 3 = - Changes Capture Group 2 = 29 Changes Capture Group 1 = Calendars packing slip 09-
5. \d\d	Match against two Numeric Digits	Capture Group 3= -11
6. -	Match against <hyphen> (Not Captured)	Changes Capture Group 3 = -29 Changes Capture Group 2 = 09 Changes Capture Group 1 = Calendars packing slip<space>
6. \d\d	Match against two Numeric Digits	Capture Group 4 = 11
7. .*	Match against remaining string (Greedy)	Capture Group 5 = null
8. ?	But not too Greedy	Capture Group 5 = No Change

Just so you understand, I won't do this with all analysis, but it is good for you to know if you don't already…

Why does Capture Group 1's Value change in step 3 (and subsequently in step 4 and 6)?

a. When the RegEx Engine has reached the EOL, but there are more sub-expressions to evaluate, it backtracks to satisfy the match if possible.

b. If a Capture Group already contains a value, and the value captured by the newly matched sub-expression contains all or part of the value in an existing capture Group, then that value or partial value is removed from the previous.

c. In step 3, the RegEx Engine backtracks through the string to match against the first two numeric digits encountered. This is the '11' of '09-29-11'. Remember that prior to this, Capture Group 1's value was the entire string, 'Calendars packing slip 09-29-11'. When Capture Group 2's value became the '11', this removed the '11' from Capture Group 1 giving Capture Group 1 a changed value of 'Calendars packing slip 09-29-'

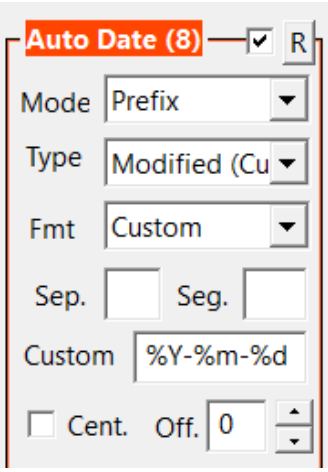
Date Manipulation

Insert Current 'Modified Date' and move Date within Filename

TEUROMED_24hours 0123 45.png to TEUROMED_24hours 2020-01-04 0123 45.png
TWORLD_1week 0723 151.png to TWORLD_1week 2020-01-13 0723 151.png

Requires two separate runs.

First run:



'Section #5: Auto Date'

Mode: Prefix
Type: Modified (Curr)
Fmt: Custom

Custom: %Y-%m-%d (There's a <space> after the %d)

After Run #1

Name	New Name
TWORLD_1week 0723 151.png	2020-01-13 TWORLD_1week 0723 151.png
TEUROMED_24hours 0123 45.png	2020-01-04 TEUROMED_24hours 0123 45.png

Second Run:

RegEx

Match: ^(\d{4}-\d{2}-\d{2})(.*)((\d{4}) (\d{3}|\d{2}))$
Replace: \2\1\3

Name	New Name
2020-01-13 TWORLD_1week 0723 151.png	TWORLD_1week 2020-01-13 0723 151.png
2020-01-04 TEUROMED_24hours 0123 45.png	TEUROMED_24hours 2020-01-04 0123 45.png

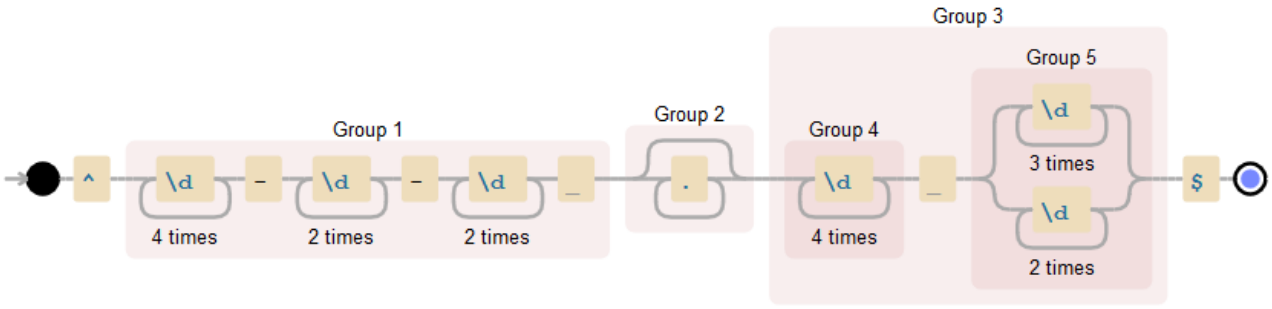
Date Manipulation

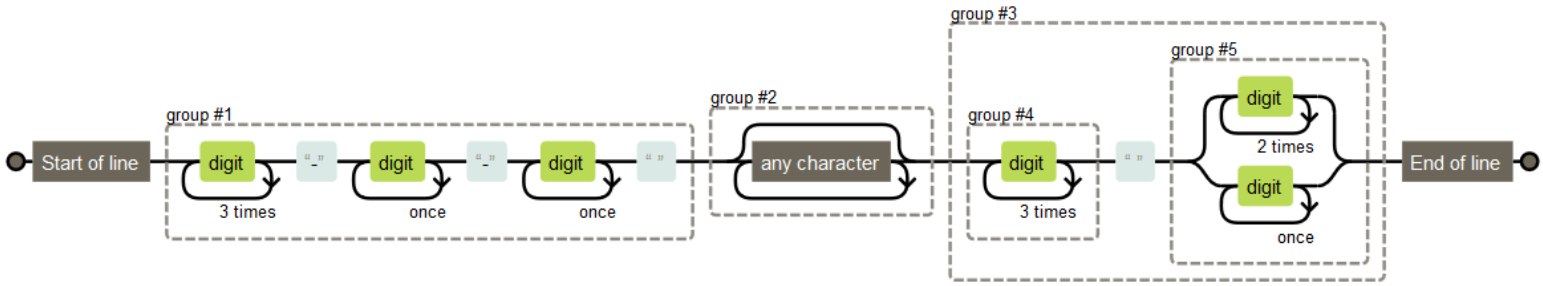
Insert Current 'Modified Date' and move Date within Filename cont.

Second Run: cont.

RegEx

Match: ^(\d{4}-\d{2}-\d{2})(.*)((\d{4}) (\d{3}|\d{2}))$





Sample:

```
Match 1: 2020-01-04 TEUROMED_24hours 0123 45
Group 1: 2020-01-04
Group 2: TEUROMED_24hours
Group 3: 0123 45
Group 4: 0123
Group 5: 45
```

Contributor Notes:

1. The | means OR, so the last part of the RegEx (\d{3}|\d{2}) will match 3 numeric digits or 2 numeric digits.
2. Don't forget to clear your Auto Date settings before making the second run.
3. This should match file names where the last characters are either 2 or 3 numeric digits.
4. Modified Date is not necessarily the Current date and as you can see it differs on the two different files in this example.

Forum Contributor: KenP

Date Manipulation

Insert Current 'Modified Date' and move Date within Filename cont.

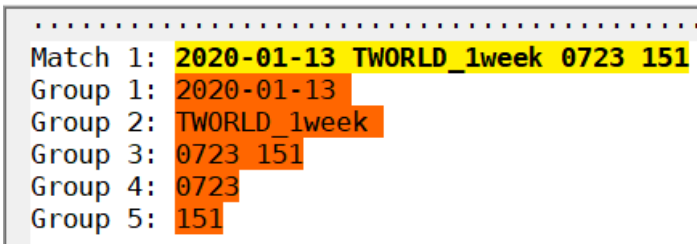
Using : 2020-01-13 TWORLD_1week 0723 151.png

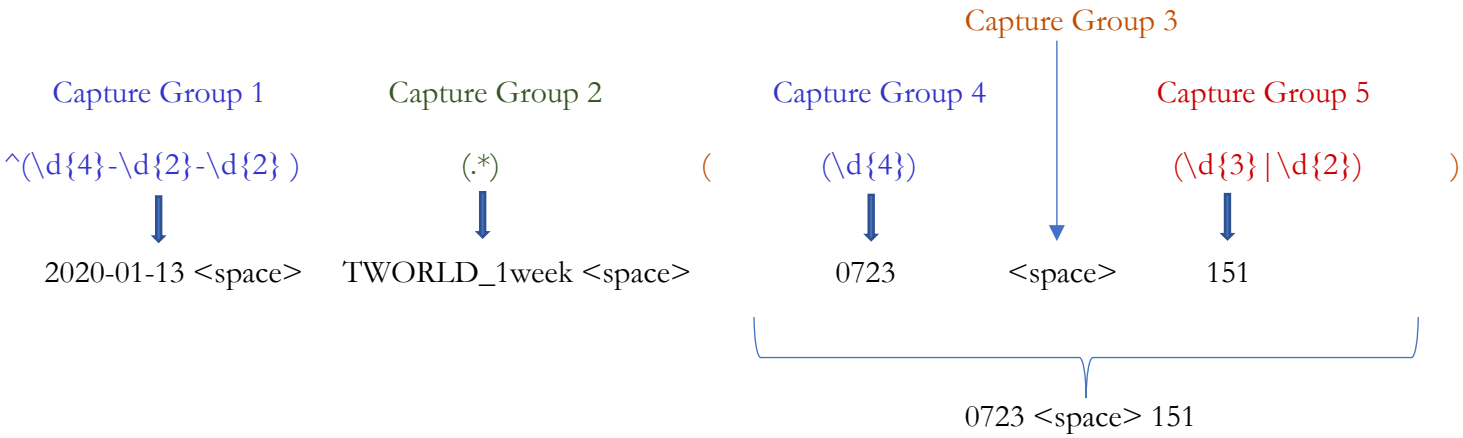
Match: ^(\d{4}-\d{2}-\d{2})(.*)((\d{4}) (\d{3}|\d{2}))$

Where:

\1 = 2020-01-13 <space> \2 = TWORLD_1week <space> \3 = 0723 151 \4 = 0723 \5 = 151

Breaks down:





This uses an Alternate for Capture Group 5

This	OR	This
\d{3}	|	\d {2}

Tim Notes:

1. For further information on Anchors or Alternates, refer to RegEx Manual found in Volume I and Volume II.

Date Manipulation

Insert Current 'Modified Date' and move Date within Filename cont.

Using : 2020-01-13 TWORLD_1week 0723 151.png

Match: ^(\d{4}-\d{2}-\d{2})(.*)((\d{4}) (\d{3}|\d{2}))$

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. \d{4}	Match against 4 numeric digits followed by …	Capture Group 1 = 2020
3. -	.. Match against <hyphen> followed by …	Capture Group 1 = 2020-
4. \d{2}	.. Match against 2 numeric digits followed by …	Capture Group 1 = 2020-01
5. -	.. Match against <hyphen> followed by …	Capture Group 1 = 2020-01-
6. \d{2}	.. Match against 2 numeric digits	Capture Group 1 = 2020-01-13
7. <space>	Match against a <space> character This is the <space> after '13'	Capture Group 1 = 2020-01-13 <space>
8. (.*)	Match against remainder of string Current position = EOL	Capture Group 2 = <space> TWORLD_1week 0723 <space> 151.png
9. \d{4}	Match against 4 numeric digits followed by …	Capture Group 3 = 0723 Changes Capture Group 2 = TWORLD_1week <space>
	Because Capture Group 3 is enclosed in an Outer Capture Group, this creates Capture Group 4	Capture Group 4 = 0723
10. <space>	.. <space> This is the <space> after '0723'	Capture Group 3 = 0723 <space> Capture Group 4 = No Change
11. (\d{3}|\d{2}	... Match against *either* three numeric digits or 2 numeric digits… Backtracks because already at EOL.	Capture Group 5 = 151 Capture Group 4 = No Change Changes Capture Group 3 = 0723 151 All Others = No Change
12. $	…from the end of the string (Must be at EOL after the match).	

Tim Notes:

1. When I state 'Beginning of line' or "Must be at EOL' in my analysis, what I am really stating is that the first match must be made at the start of the string (BOL) and the last match must be made at the end of the string. If either statement is False, the RegEx Fails. So if, for example, it didn't find a 3 numeric digit number or 2 numeric digit number at the end of the string, the statement, \d{3}|\d{2}$ would be False and the RegEx would fail.

2. Capture Group 3 is evaluated as a single expression inclusive of all values held by any captured Sub-Expressions. This means that the value of '0723' evaluated from Capture Group 4, (\d{4}) along with the captured <space> character that was captured as part of Capture Group 3 only and was not captured by either sub-expressions, and '151' evaluated from Capture Group5, (\d{3}|\d{2}) are 'grouped' together and result in a single value of '0723 151'. In other words a nested Capture Group takes its value inclusive of all values of the Capture Groups within.

Date Manipulation

Rename DDMMYYYY to YYYYMMDD

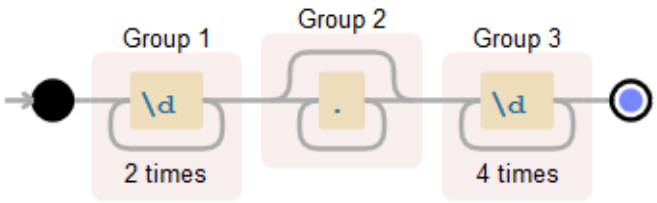
Tim Notes:

1. This also explains that a Restrictive RegEx is sometimes preferable to a more universal RegEx when you want to limit filename inclusion to prevent problems.

01JAN2009	to	2009JAN01
02JAN2009	to	2009JAN02
13NOV2013	to	2013NOV13

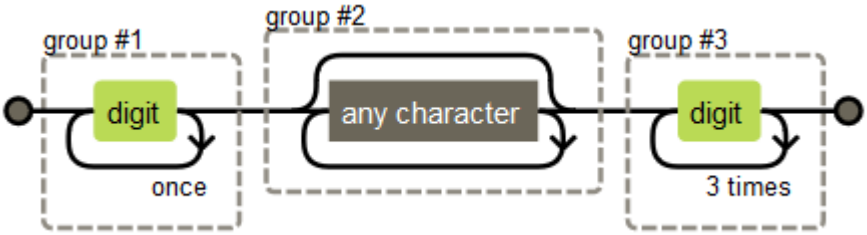
Match: (\d{2})(.*)(\d{4})
Replace: \3\2\1

Name	New Name
01JAN2009	2009JAN01
02JAN2009	2009JAN02
13NOV2013	2013NOV13



Sample:

```
....................
Match 1: 01JAN2009
Group 1: 01
Group 2: JAN
Group 3: 2009
```



Contributor Notes:

1. Useful when name-formats don't change but if there are multiple name formats, use with caution, especially when using with filenames:

Group1	matches 1st 2 Numeric digits anywhere
Group2	matches either Any Text/No Text until…
Group3	.. matches Last 4 Numeric digits anywhere

Date Manipulation

Rename DDMMYYYY to YYYYMMDD cont.

Using: 01JAN2009

Match: (\d{2})(.*)(\d{4})

Where:

\1 = 01 \2 = JAN \3 = 2009

Tim Analysis:

1. \d {2}	Match against 2 numeric digits	Capture Group 1 = 01
1..	Match against any character	Capture Group 2 = J
2. *	Make it Greedy	Capture Group 2 = JAN2009
	Current position = EOL	
3. \d {4}	Match against 4 numeric digits	Capture Group 3 = 2009
	Backtracks.	Changes Capture Group 2 = JAN

Date Manipulation

Rename DDMMYYYY to YYYYMMDD cont.

Problem:

So while text from \1 to \3 is matched & re-ordered, text outside that range is removed from New Name. These files got inadvertently included and the text cut off.

Name	New Name
Text12abc3456Text	3456abc12
aa123456zz	345612
000003	000300

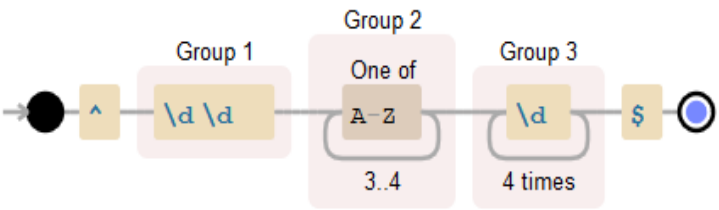
Text12abc3456Text got changed to 3456abc12
aa123456zz got changed to 345612
000003 got changed to 000300

Solution – revise the RegEx to be more restrictive to the pattern -

As an example, here's a more specific match:

Match: ^(\d\d)([A-Z]{3,4})(\d{4})$
Replace: \3\2\1

Name	New Name
01JAN2009	2009JAN01
02JAN2009	2009JAN02
13NOV2013	2013NOV13
Text12abc3456Text	Text12abc3456Text
aa123456zz	aa123456zz
000003	000003



Sample:

```
.....................
Match 1: 01JAN2009
Group 1: 01
Group 2: JAN
Group 3: 2009
```

Contributor Analysis:

1. ^ ensures filename starts with 2 Numeric digits (Beginning of Line or BOL)
2. $ ensures filename ends with 4 Numeric digits (End of Line or EOL)
3. [A-Z{3,4}] Matches All Text-in between month and year only if the text is made up of 3 or 4 uppercase letters.

Forum Contributors: truth, gocyclones

Date Manipulation

Rename DDMMYYYY to YYYYMMDD cont.

Solution – revise the RegEx to be more restrictive to the pattern – cont.

Match: ^(\d\d)([A-Z]{3,4})(\d{4})$

Where:

Where:

\1 = 01 \2 = JAN \3 = 2009

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. \d \d	Match against 2 numeric digits	Capture Group 1 = 01
3. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = J
4. {3,4}	Specifies a minimum of 3 and maximum of 4 match repetitions	Capture Group 2 = JAN
5. \d {4}	Match against 4 numeric digits	Capture Group 3 = 2009
6. $	Must be at EOL (End of Line)	

Tim Notes:

1. No backtracking.

Date Manipulation

dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric

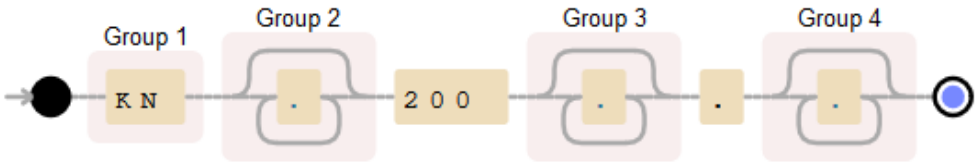
KN21Nov2008.1.pdf to 1KN211108.pdf

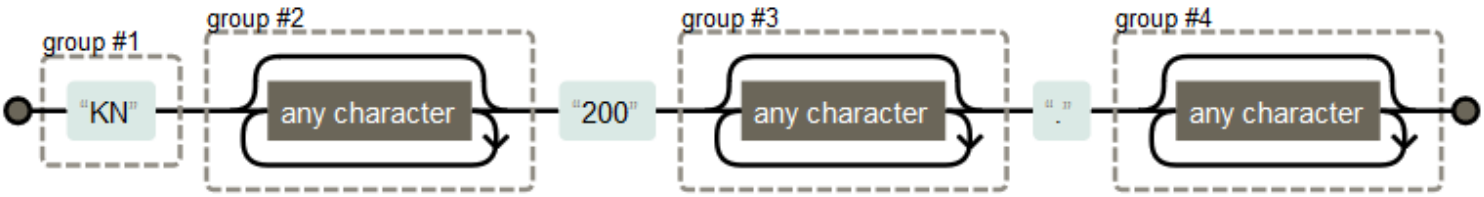
In 2 runs:

First run:

Match: (KN)(.*)200(.*)\.(.*)
Replace \4\1\20\3

Name	New Name
KN21Nov2008.1.pdf	1KN21Nov08.pdf





Sample:

```
.........................
Match 1: KN21Nov2008.1
Group 1: KN
Group 2: 21Nov
Group 3: 8
Group 4: 1
```

Where:

\1 = KN \2 = 21Nov \3 = 8 \4 = 1

Contributor and *Tim Analysis:*

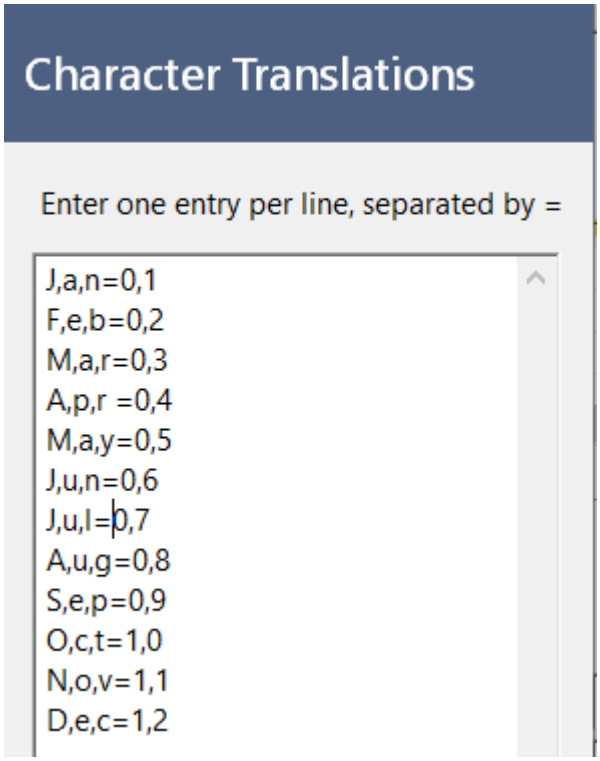
1. KN	Matches the 'KN' in the filename	Capture Group 1 = KN
2. .*	Captures and matches the '21Nov'	Capture Group 2 = 21Nov2008.1
3. 200	Match the 200 but don't capture	Changes Capture Group 2 =21Nov
4. .*	Match till…	Capture Group 3 = 8.1
5. \.	.. literal dot but don't capture	Changes Capture Group 3 = 8
6. .*	Capture remainder of string	Capture Group 4 = 1

Date Manipulation

dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric cont.

Second run:

'Section #14: Special – Character Translation' –



Enter as:

J,a,n=0,1
F,e,b=0,2
M,a,r=0,3
A,p,r =0,4
M,a,y=0,5
J,u,n=0,6
J,u,l=0,7
A,u,g=0,8
S,e,p=0,9
O,c,t=1,0
N,o,v=1,1
D,e,c=1,2

This will the translate the 'Nov' in your 1KN21Nov08.pdf string to '11' yielding your desired result: 1KN211108.pdf

Tim Notes:

1. You could also spell out the complete month, i.e. J,a,n,u,a,r,y=0,1
2. This will pick up 'Separate but Equal' inadvertently as '09arate but Equal'
   a. Use 'Section #12: Filters' for better selection.
   b. If the sample files to be renamed, use a delimiter between dates, then add that to the character translation.
      i.e., If sample was '1KN21-Nov08.pdf', then entry would be.. -,N,o,v=1,1
3. When using Character Translation, there must be no trailing <spaces> after each line.

Forum Contributor: jimwillsher (Original Publisher of BRU), spotburst

Date Manipulation

dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric cont.

Tim Contribution –

Expanded Samples:

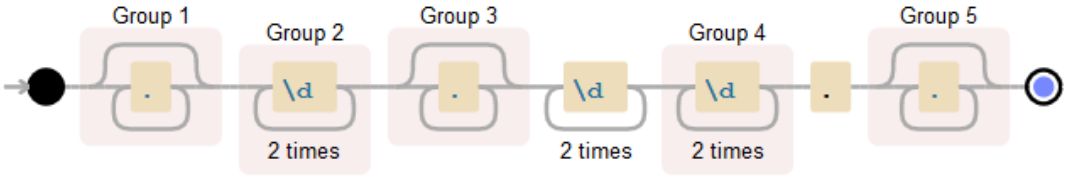
KN21Nov2008.1.pdf
XV21Dec2011.1.pdf
QP05Jul1911.3.pdf

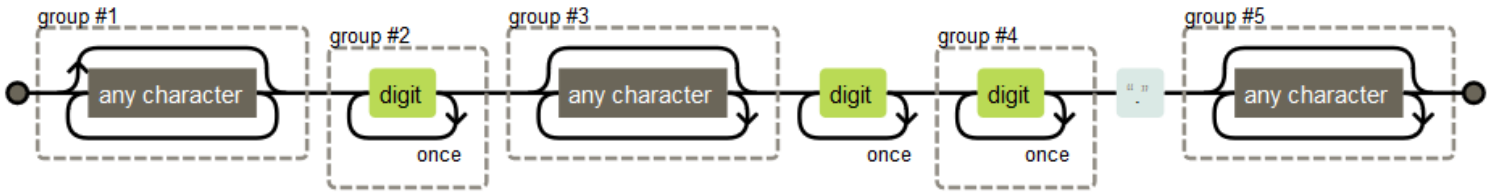
1. The above RegEx is very restricted to the sample.

Less Restrictive:

Match: (.*?)(\d{2})(.*)\d{2}(\d{2})\.(.*)
Replace: \5\1\2\3\4

Name	New Name
XV21Dec2011.1.pdf	1XV21Dec11.pdf
KN21Nov2008.1.pdf	1KN21Nov08.pdf
QP05Jul1911.3.pdf	3QP05Jul11.pdf





Sample:

```
.......................
Match 1: KN21Nov2008.1
Group 1: KN
Group 2: 21
Group 3: Nov
Group 4: 08
Group 5: 1
```

Date Manipulation

dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric cont.

Tim Contribution – cont.

Less Restrictive: cont.

Using: KN21Nov2008.1.pdf

Match: (.*?)(\d{2})(.*)\d{2}(\d{2})\.(.*)

Where:

\1 = KN \2 = 21 \3 = Nov \4 = 08 \5 = 1

Tim Analysis:

1. .	Match against any character	Capture Group 1 = K
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \d{2}	Match against 2 numeric digits	Capture Group 2 = 21
		Changes Capture Group 1 = KN
5. .*	Match against remaining string	Capture Group 3 = Nov2008.1
6. \d{2}	Match against 2 numeric digits (Not Captured) = '08'	
		Changes Capture Group 3 = Nov20
7. \d{2}	Match against 2 numeric digits	Capture Group 4 = 08
	Changes Not Captured numeric digits = '20'	Changes Capture Group 3 = Nov
8. \.	Match against literal <dot> character (Not Captured)	
9. .*	Match against remaining string	Capture Group 5 = 1

Date Manipulation

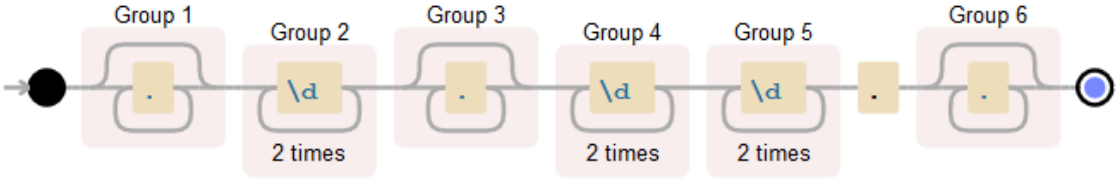
dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric cont.

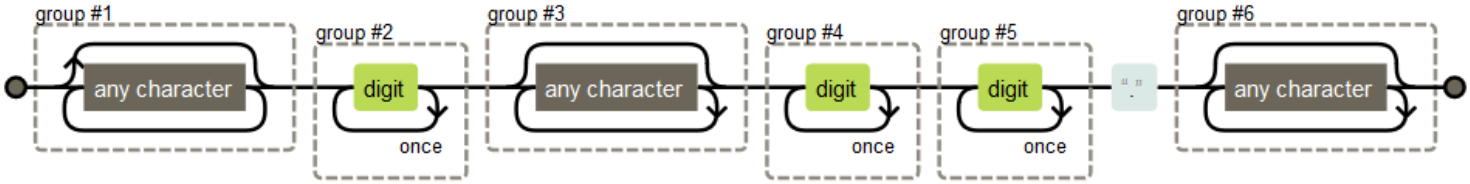
Tim Contribution – cont.

If you wanted to keep the century just change to:

Match: (.*?)(\d{2})(.*)(\d{2})(\d{2})\.(.*)
Replace: \6\1\2\3\4\5

Name	New Name
XV21Dec2011.1.pdf	1XV21Dec2011.pdf
KN21Nov2008.1.pdf	1KN21Nov2008.pdf
QP05Jul1911.3.pdf	3QP05Jul1911.pdf





Sample:

```
.......................
Match 1: KN21Nov2008.1
Group 1: KN
Group 2: 21
Group 3: Nov
Group 4: 20
Group 5: 08
Group 6: 1
```

Tim Notes:

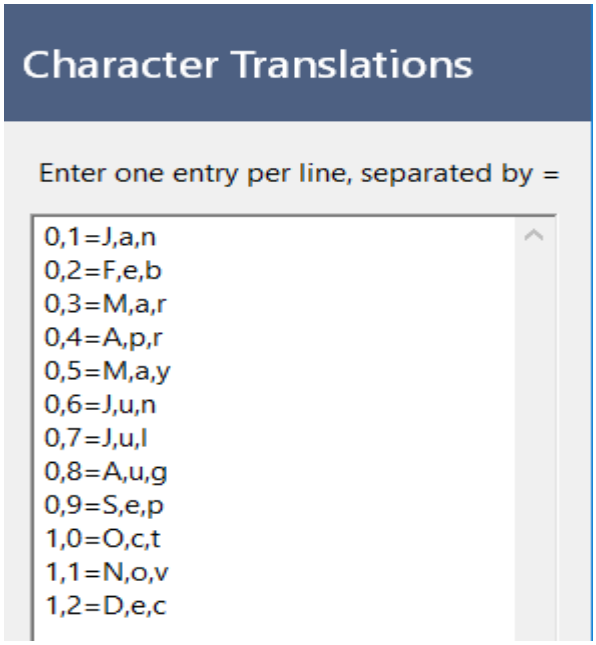
1. This is the same RegEx listed under 'less restrictive', except that the \d{2} that was previously not captured = '20' in step 7 of the analysis, is captured (\d{2}).

Date Manipulation

dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric cont.

Tim Contribution – cont.

In case you were wondering, Character Translation will not work if trying to reverse back to the written date:

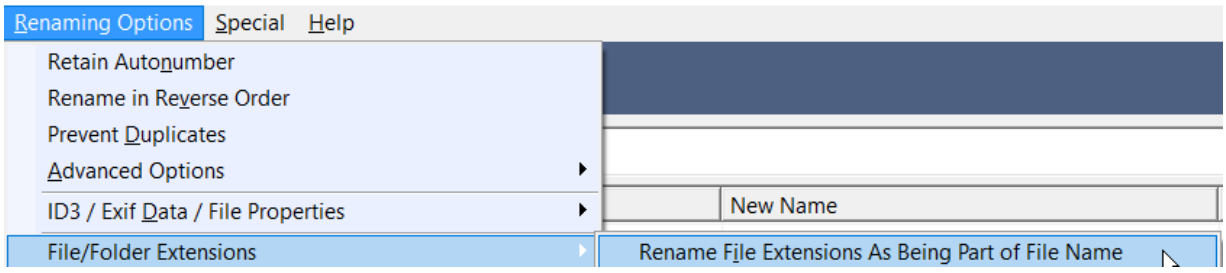


It can't distinguish between the Day as 21, the Month as the 11th and the Year as 08. It ignores the 2 because the combination of 21 doesn't fit the translation (there isn't a translation for the 2,1), so the next pair, 11 converts to Nov, and the next pair, 10, converts to Oct. It ignores the 8 because it doesn't fit the translation (there is no 8 = in the Character Translation).

Tim Notes:

1. There is a JavaScript example, 'Convert YMD to Written Date', that can, but it needs to be tweaked because it is programmed for a full date of YYYYMMDD. For more information, refer to the JavaScript section.

2. Although the samples present with <dot> characters in the filenames, you are instructed NOT to enable 'Rename File Extensions As Being part of File Name' in order to achieve the desired outcome.



Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front

name dd.mm.yy to yyyy-mm-dd name
name ddmmyy to yyyy-mm-dd name

Observations:

1. Add century to YY
2. Remove any dots in date and replace with <hyphens>
3. For dates with no delimiters, add <hyphens>
4. 1997 to 1999 shouldn't end as 2097 to 2099 but remain as century 19
   a. You have to run RegEx more than once using different replacement strings to reflect desired century.
5. Rearrange string so date is in front of filename

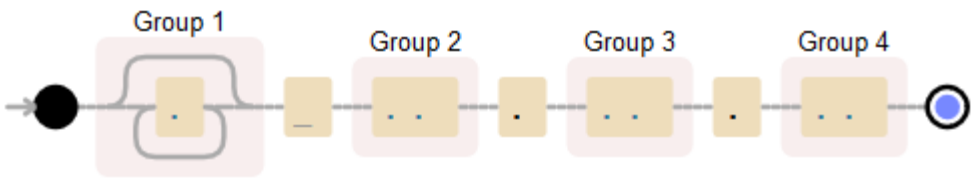
For "name dd.mm.yy" format, try this:

Match: (.*) (..)\.(..)\.(..)
Replace: 19\4-\3-\2 \1

Name	New Name
name ddmmyy.pdf	name ddmmyy.pdf
name dd.mm.yy.pdf	19yy-mm-dd name.pdf
name dd.mm.yy evenmore.pdf	19yy-mm-dd name.pdf
name 21.08.04extracrap.pdf	1904-08-21 name.pdf
name 21.08.04.pdf	1904-08-21 name.pdf

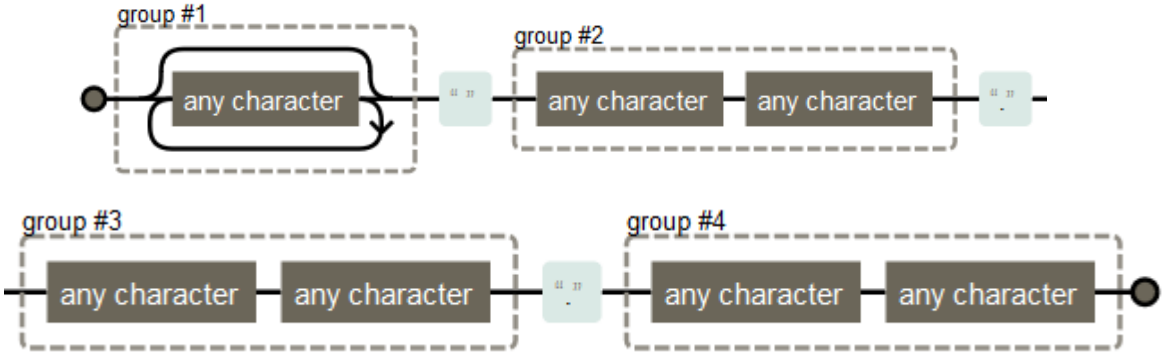
Sample

```
.....................
Match 1: name dd.mm.yy
Group 1: name
Group 2: dd
Group 3: mm
Group 4: yy
```



If the date is after year 2000, be sure to change the 19\4 with 20\4. If you have a mixture of dates, you will have to run the program twice, cherry picking the 1900's from the 2000's.

Expanded:



Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

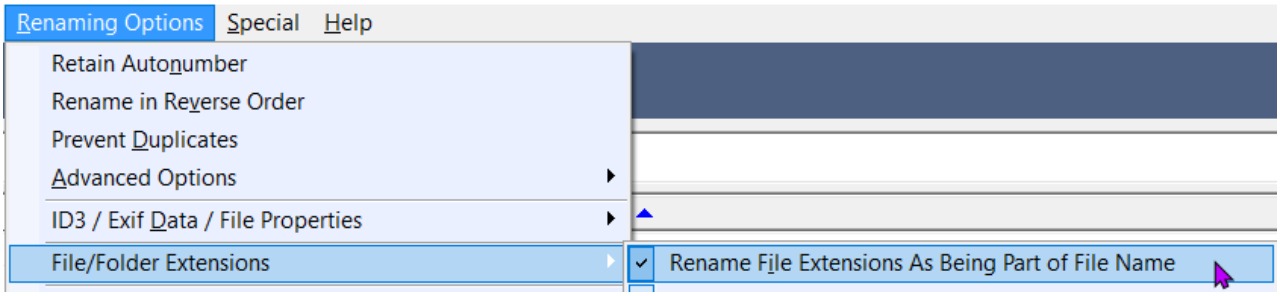
Using: name dd.mm.yy

Match: (.*) (..)\.(..)\.(..)

Where:

\1 = name \2 = dd \3 = mm \4 = yy

For this sample, you have to enable, 'Rename File Extension As Being Part of File Name' from the 'File / Folder Extensions sub-menu' of the 'Renaming Options Menu'.



Tim Analysis:

1..	Match against any character	Capture Group 1 = n
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'name'	Changes Capture Group 1 = name
4. ..	Match against any 2 characters	Capture Group 2 = dd
5. \.	Match against a literal <dot> character (Not Captured) This is the <dot> after 'dd'	
6. ..	Match against any 2 characters	Capture Group 3 = mm
7. \.	Match against a literal <dot> character (Not Captured) This is the dot after 'mm'	
8. ..	Match against any 2 characters	Capture Group 3 = yy

Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

Tim Notes:

1. Note that the samples using 'dd' or 'mm' or 'yy' in the string, i.e., 'name dd.mm.yy', will not be included from here on because they are not numeric digits but were used only to illustrate the required patterns.

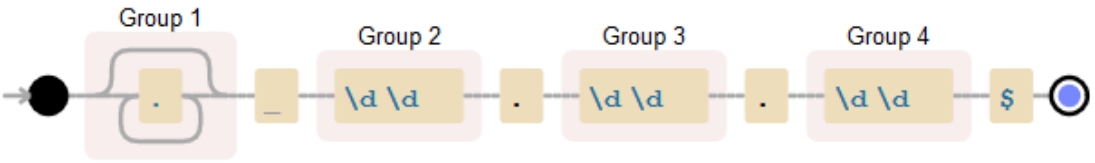
Samples:

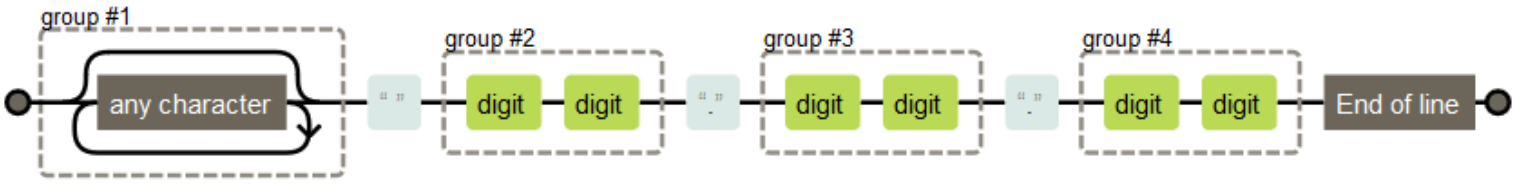
name 31.02.13.pdf	name 230700.pdf	name 210804
name 21.08.04.pdf	name 15.11.15evenmore.pdf	name 21.08.04extracrap.pdf

To get rid of any non numbers in the date positions use \d instead of . in the parentheses.
To prevent matching filenames with more than 6 numbers after the <space>, end the RegEx with $ (EOL)

Match: (.*) (\d\d)\.(\d\d)\.(\d\d)$
Replace: 19\4-\3-\2 \1

Name	New Name
name 31.02.13.pdf	1913-02-31 name.pdf
name 230700.pdf	name 230700.pdf
name 210804	name 210804
name 21.08.04extracrap.pdf	name 21.08.04extracrap.pdf
name 21.08.04.pdf	1904-08-21 name.pdf
name 15.11.15 evenmore.pdf	name 15.11.15 evenmore.pdf





Sample:

```
.........................
Match 1: name 31.02.13
Group 1: name
Group 2: 31
Group 3: 02
Group 4: 13
```

Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

To get rid of any non numbers in the date positions use \d instead of . in the parentheses. cont.

Using: name 31.02.13.pdf

Match: (.*) (\d\d)\.(\d\d)\.(\d\d)$

Where:

\1 = name \2 = 31 \3 = 02 \4 = 13

Tim Analysis:

1..	Match against any character	Capture Group 1 = n
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'name'	Changes Capture Group 1 = name
4. \d\d	Match against 2 numeric digits	Capture Group 2 = 31
5. \.	Match against a literal <dot> character (Not Captured) This is the <dot> after '31'	
6. \d\d	Match against 2 numeric digits	Capture Group 3 = 02
7. \.	Match against a literal <dot> character (Not Captured) This is the <dot> after '02'	
8. \d\d	Match against 2 numeric digits	Capture Group 4 = 13
9. \.	Match against a literal <dot> character (Not Captured) This is the <dot> after '13'	
10. $	Must be at EOL (End of Line)	

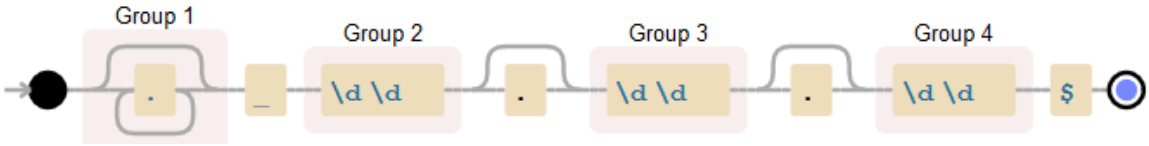
Date Manipulation

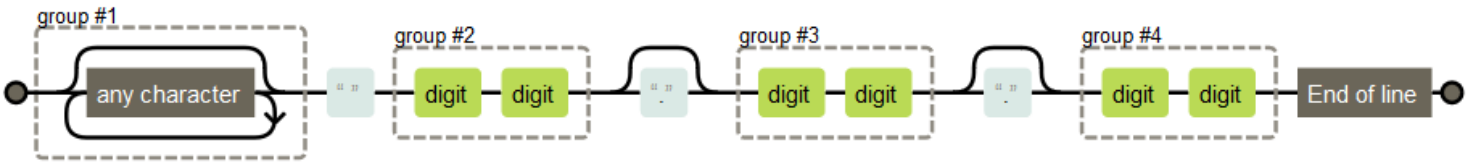
Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

A simple way to allow it to match both period or no period separators is to make them optional with ?

Match: (.*) (\d\d)\.?(\d\d)\.?(\d\d)$
Replace: 19\4-\3-\2 \1

Name	New Name
name 31.02.13.pdf	1913-02-31 name.pdf
name 230700.pdf	1900-07-23 name.pdf
name 210804	1904-08-21 name
name 21.08.04extracrap.pdf	name 21.08.04extracrap.pdf
name 21.08.04.pdf	1904-08-21 name.pdf
name 15.11.15 evenmore.pdf	name 15.11.15 evenmore.pdf





Sample:

```
.....................
Match 1: name 230700
Group 1: name
Group 2: 23
Group 3: 07
Group 4: 00
```

Under Regex Buddy for this test, had to remove anchor $ (EOL) because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.pdf' is just another part of the string of text.

Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

A simple way to allow it to match both period or no period separators is to make them optional with ? cont.

Using: name 230700.pdf

Match: (.*) (\d\d)\.?(\d\d)\.?(\d\d)$

Where:

\1 = name \2 = 23 \3 = 07 \4 = 00

Tim Analysis:

1..	Match against any character	Capture Group 1 = n
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'name'	Changes Capture Group 1 = name
4. \d\d	Match against 2 numeric digits	Capture Group 2 = 23
5. \.	Match against a literal <dot> character (Not Captured) – No match	All capture Groups = No Value
6. ?	Make it optional.	All Capture Groups = Restored

Tim Notes:

1. No <dot> characters. By making it optional, the RegEx Engine repositions as if the sub-expression was never evaluated – thus optional. This can clearly be seen in the debugging output of the Regex Buddy program on the next page.

7. \d\d	Match against 2 numeric digits	Capture Group 3 = 07
8. \.	Match against a literal <dot> character (Not Captured) – No match	
9. ?	Make it optional.	All Capture Groups = Restored
10. \d\d	Match against 2 numeric digits	Capture Group 4 = 00
11. $	Must be at EOL (End of Line)	

Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

A simple way to allow it to match both period or no period separators is to make them optional with ? cont.

Using: name 230700.pdf

```
.....................
Match 1: name 230700
Group 1: name
Group 2: 23
Group 3: 07
Group 4: 00
```

Match: (.*) (\d\d)\.?(\d\d)\.?(\d\d)$

Where:

\1 = name \2 = 23 \3 = 07 \4 = 00

Tim Analysis: cont.

1.. Match against any character
2. * Make it Greedy

3. <space> Match against a <space> character (Not Captured)
Backtracks. This is the <space> after 'name'

4. \d\d Match against 2 numeric digits

5. \. Match against a literal <dot> character (Not Captured) – No match
6. ? Make it optional.

7. \d\d Match against 2 numeric digits

8. \. Match against a literal <dot> character (Not Captured) – No match
9. ? Make it optional.

10. \d\d Match against 2 numeric digits

```
Beginning match attempt at character 0
 1 name 230700
 2 name 230700
 3 name 230700backtrack
 4 name 23070
 5 name 23070
 6 name 23070backtrack
 7 name 2307
 8 name 2307
 9 name 2307backtrack
10 name 230
11 name 230
12 name 230backtrack
13 name 23
14 name 23
15 name 23backtrack
16 name 2
17 name 2
18 name 2backtrack
19 name
20 name
21 name backtrack
22 name
23 name
24 name
25 name 2
26 name 23
27 name 23
28 name 23ok
29 name 230
30 name 2307
31 name 2307
32 name 2307ok
33 name 23070
34 name 230700
35 name 230700
36 name 230700ok
Match found in 36 steps
```

Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

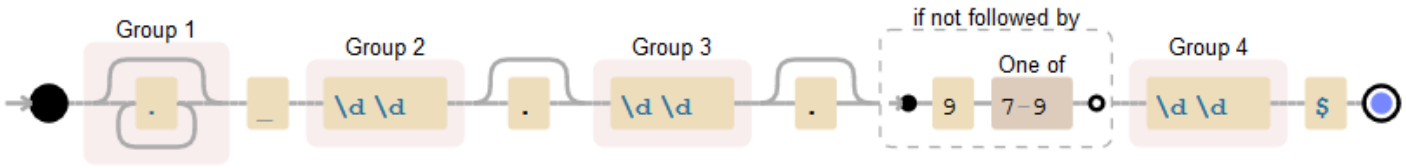
If the sample changes to ….

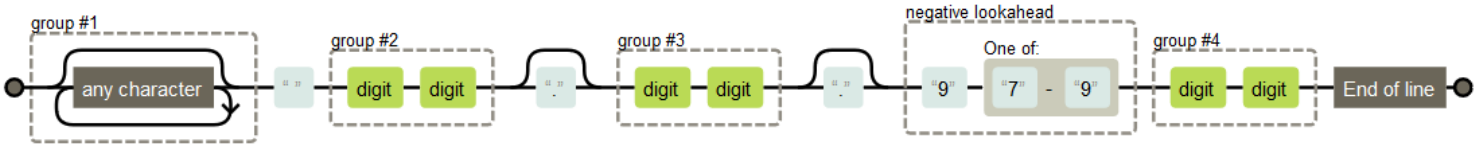
1900's	2000's
name 21.08.98.pdf	name 21.08.04.pdf
name 210899	name 210804
name 31.02.97.pdf	name 31.02.13.pdf
name 21.08.98extracrap.pdf	name 230700.pdf
name 15.11.97 evenmore.pdf	

… do the first pass with:

Match: (.*) (\d\d)\.?(\d\d)\.?(?!9[7-9])(\d\d)$
Replace: 20\4-\3-\2 \1

Name	New Name
name 31.02.97.pdf	name 31.02.97.pdf
name 210899	name 210899
name 21.08.98extracrap.pdf	name 21.08.98extracrap.pdf
name 21.08.98.pdf	name 21.08.98.pdf
name 15.11.97 evenmore.pdf	name 15.11.97 evenmore.pdf
name 31.02.13.pdf	2013-02-31 name.pdf
name 230700.pdf	2000-07-23 name.pdf
name 210804	2004-08-21 name
name 21.08.04.pdf	2004-08-21 name.pdf





Sample:

```
.........................
Match 1: name 31.02.13
Group 1: name
Group 2: 31
Group 3: 02
Group 4: 13
```

Under Regex Buddy for this test, had to remove anchor $ (EOL) because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.pdf' is just another part of the string of text.

Contributor Notes:

1. (?!9[7-9]) This tells it to NOT match any dates 97-99) - negative lookahead
2. This will change all filenames NOT 97-99 to yyyy-mm-dd name format and they won't match the second pass.

Date Manipulation

Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

Using: name 31.02.13.pdf

Match: (.*) (\d\d)\.?(\d\d)\.?(?!9[7-9])(\d\d)$

Where:

\1 = name \2 = 31 \3 = 02 \4 = 13

Tim Analysis:

1..	Match against any character	Capture Group 1 = n
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'name'	Changes Capture Group 1 = name
4. \d\d	Match against 2 numeric digits	Capture Group 2 = 31
5. \.	Match against a literal <dot> character (Not Captured) This is the <dot> after '31'	All capture Groups = Unchanged
6. ?	Make it optional. Since the <dot> character was matched, the option is ignored.	
7. \d\d	Match against 2 numeric digits	Capture Group 3 = 02
8. \.	Match against a literal <dot> character (Not Captured) This is the <dot> after '02'	
9. ?	Make it optional. Since the <dot> character was matched, the option is ignored.	All Capture Groups = Unchanged
10. (?!9[7-9])	Negative Lookahead. (Non-marking Group) Look to the right that the next Character is 'not' a '9' followed by a '7', '8' or '9' = 1 = True	

Tim Notes:

1. This indicates that if the year is '97 '98 or '99, then fail the RegEx (step 10 = False). On this first run, only want to match dates that are in the years ranging from 2000 - 2099. If the RegEx Fails, the string is ignored (will not be included in the Renaming function) , and processing will continue with the next string.

11. \d\d	Match against 2 numeric digits	Capture Group 4 = 13
12. $	Must be at EOL (End of Line)	

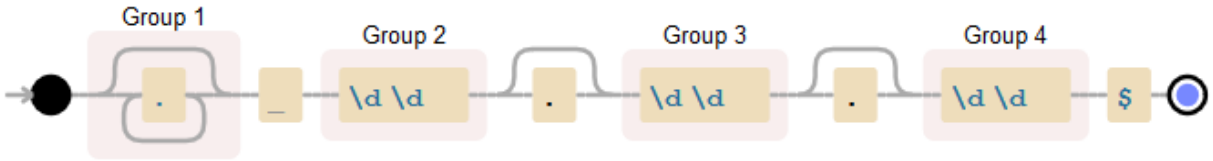
Date Manipulation

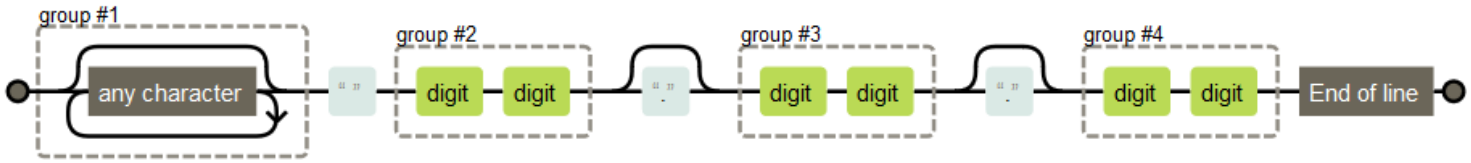
Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD and Rearrange to Front cont.

For the second pass all that are left are 97-99 so use:

Match: (.*) (\d\d)\.?(\d\d)\.?(\d\d)$
Replace: 19\4-\3-\2 \1

Name	New Name
name 31.02.97.pdf	1997-02-31 name.pdf
name 210899	1999-08-21 name
name 21.08.98extracrap.pdf	name 21.08.98extracrap.pdf
name 21.08.98.pdf	1998-08-21 name.pdf
name 15.11.97 evenmore.pdf	name 15.11.97 evenmore.pdf
2013-02-31 name.pdf	2013-02-31 name.pdf
2000-07-23 name.pdf	2000-07-23 name.pdf
2004-08-21 name	2004-08-21 name
2004-08-21 name.pdf	2004-08-21 name.pdf





Sample:

```
.........................
Match 1: name 31.02.97
Group 1: name
Group 2: 31
Group 3: 02
Group 4: 97
```

Tim Notes:

1. For more information on Lookarounds, refer to 'Regular Expressions Manual' found in Appendix of 'Volume 1 Bulk Rename Utility Operations Manual'.

2. This RegEx is the same as the one used with, 'A simple way to allow it to match both period or no period' earlier in this section. Please refer to that for the full analysis and other notations.

Forum Contributor: Glenn

Date Manipulation

Tim Contribution –

Using Custom date with BRC

I am including this because the BRC manual is lacking in its instruction of using Custom Dates.

I don't delve a lot into BRC as much as I do with other topics. BRC is a separate program that is a command line version of BRU. There are advantages and disadvantages to both programs. The Biggest advantage that BRC has over BRU is that BRU is limited to a single renaming function. BRC is not. It can perform multiple renaming functions at one time – one after the other in the order specified. BRC is not limited to the Order of Evaluation as BRU is. This means that you can have, for example, Remove performed before Replace. In BRU this is not possible. Neither as of this writing, support the Global switch, which would allow multiple matches of a string based on one RegEx, but as stated BRC can run one RegEx after the other. More importantly, it can run these in memory, essentially setting up the final format of the string to be renamed using the /Execute of the BRC statement.

BRU's advantages are that it is GUI, meaning that you can see what you are doing. It is also easier for beginners. And if you want the full benefit of BRU's power, the JavaScript function is only available with BRU (low-cost addition).

/Pattern in BRC has it limitations as well:

BRC will only process one /Pattern switch. If Multiple /Pattern switches are specified, only the last will be recognized.

This means there is not much flexibility when renaming single files using BRC.

A better implementation would have been:

> /Pattern:file1 /Pattern:file2

rather than the current:

> /Pattern: "file1 file2"

For example, to rename filenames that contain <spaces> is not possible using /Pattern in BRC because BRC uses the <space> as a delimiter in the /Pattern switch.

> So "*12 34*"

Is interpreted in the /Pattern as two separate filespecs instead of one:

> *12 – names ending with '12'
> 34* – names beginning with "34'

BRU, it should be mentioned, also shares this same limitation with its 'Section #12: Filters' function.

Tim Notes:

1. Just like BRU, BRC /Pattern is case sensitive. Back*.log is not the same as back*.log

Date Manipulation

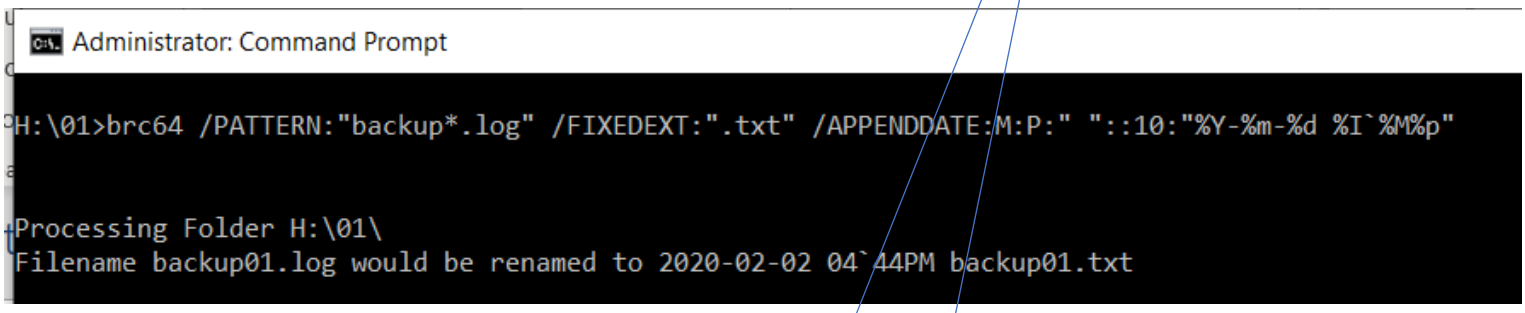
Using Custom date with BRC cont.

Using: backup01.log

Command line:

Filter Replace Extension Append the Date as Prefix using 'Custom' Format string

```
brc64 /PATTERN:"backup*.log" /FIXEDEXT:".txt" /APPENDDATE:M:P:" "::10:"%Y-%m-%d %I`%M%p"
```



1. This indicates to change the extension from .log to .txt
2. Append the creation date as a prefix using a 'Custom' date Format string in the form, YYYY-MM-DD `HH MM SS
3. Using a Separator of a <space> as specified as " : <space> "
4. Segment parameter is left blank when using a 'Custom' format string.
   a. This is indicated above as ' :: ' preceding the '10'. Normally a segment would be specified here, i.e. :-: <hyphen>
   b. When a 'Custom' format string is to be specified, using '10' as the format number, any 'Segment'(s) are inclusive as part of that 'Custom' format string itself.
   c. When a 'Custom' Format string is used, multiple characters can be entered anywhere in the string.

      1. You will notice the inclusion of the <hyphen> character in the 'Custom' format string:

         "%Y-%m-%d %I`%M%p"

      2. A <space> character is added between the date and time.
      3. An accent character is added before the Minutes = `

Tim Notes:

1. Capitalized keywords are not required:

```
brc64 /Pattern:"backup*.log" /FixedExt:".txt" /AppendDate:M:P:" "::10:"%Y-%m-%d %I`%M%p"
```

.. will work as well.

2. order of switches is not important either:

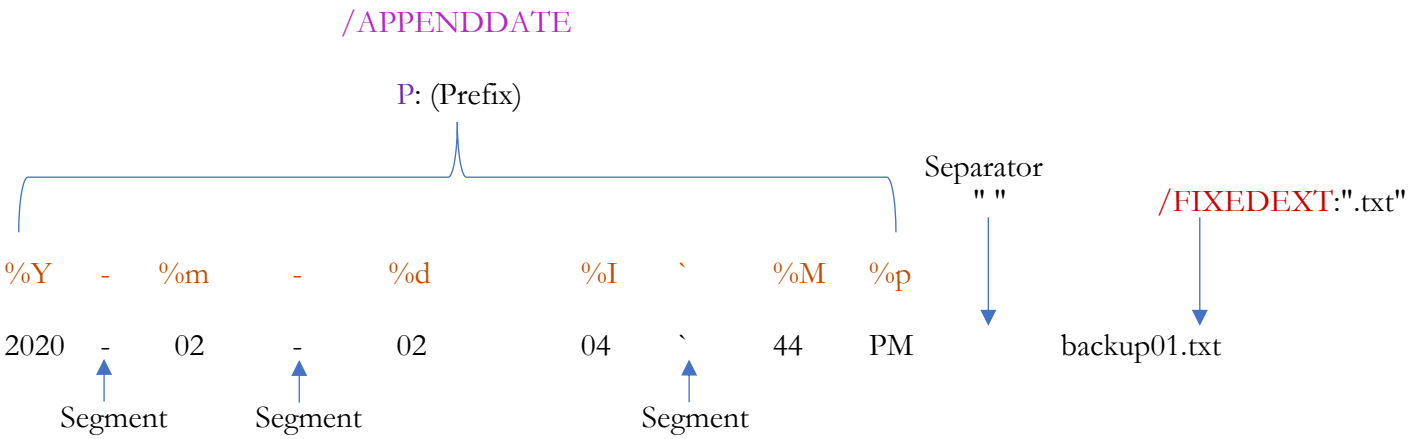
```
BRC64.exe /APPENDDATE:M:P:" ":-:10:"%Y-%m-%d %I`%M%p" /PATTERN:"backup*.log" /FIXEDEXT:".txt"
```

Date Manipulation

Using Custom date with BRC cont.

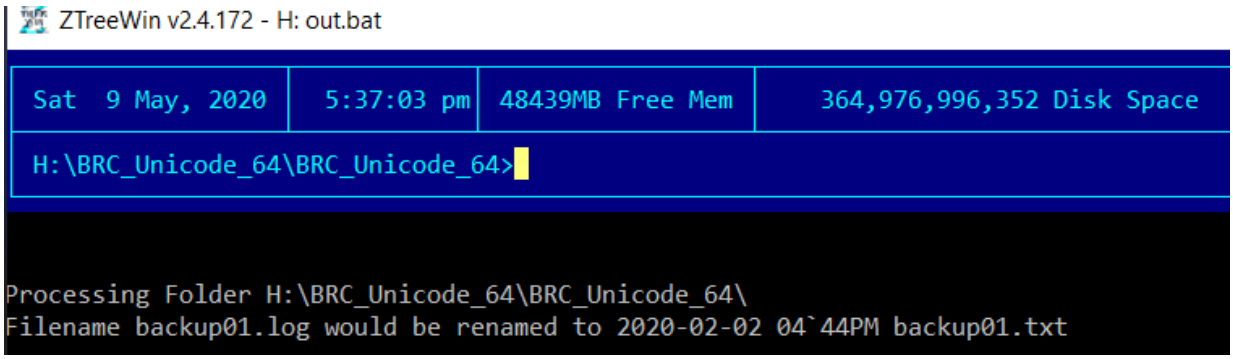
Using: backup01.log

Command line: cont.



In a Batch File:

```
@ECHO OFF
REM // Specify path to BRC64.exe
SET BRC64=H:\BRC_Unicode_64\BRC_Unicode_64\brc64.exe

brc64 /PATTERN:"backup*.log" /FIXEDEXT:".txt" /APPENDDATE:M:P:" "::10:"%%Y-%%m-%%d %%I`%%M%%p"
```



Tim Notes:

1. When used in a Batch file, variables must be specified using 'two' percentage signs rather than the one required for the Command Line.

Date Manipulation

Using Custom date with BRC cont.

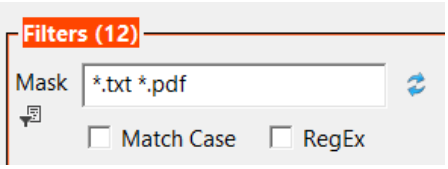
Using: backup01.log

Command line: cont.

Where:

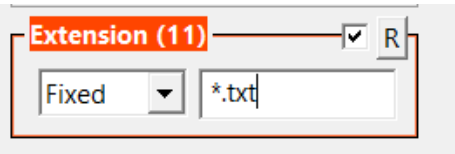
/PATTERN:
The patterns of files to match (with multiple values). e.g. /PATTERN:"*.jpg *.tiff *.bmp". If omitted then "*" is used which matches any file.

This is the equivalent of the 'Section #12: Filters' of BRU.



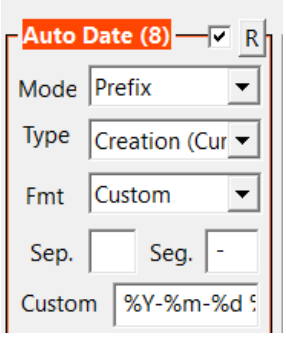
/FixedExt
This will completely remove the filename extension, and replace it with the specified extension.

This is the equivalent of the 'Fixed' specification of 'Section #11: Extension' of BRU



/APPENDDATE
Append a date automatically to the name. You can append a variety of dates to the filenames, either at the beginning (Prefix) or the end (Suffix). For example, you could prefix the filenames with their Modified date, or you could suffix them with the Creation date.

This is the equivalent of the 'Section #8: Auto Date' of BRU.



Mode: Prefix
Type: Creation (Current)
Fmt: Custom
Sep.: <space>
Seg.: <hyphen>
Custom: %Y-%m-%d %I`%M%p

Date Manipulation

Using Custom date with BRC cont.

Using: backup01.log

Command line: cont.

Syntax of using /APPENDDATE

Type:Mode:Separator:Segment:Format:Custom

Type: –
Optional. The type of date to append, these being Created Date, Modified Date, Accessed Date or the date Now. Default is Created Date if omitted.

Mode: –
Required. Prefix, Suffix, or Both. Must be specified.

Separator: –
Optional. The text to place between the name and the Autodate text. Can be blank.

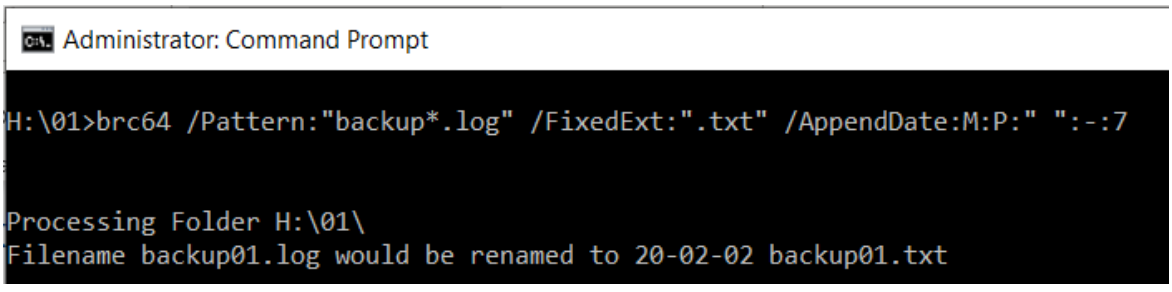
Segment: –
Optional. The text to place between each of the sub-elements of the text. Can be blank. Segment is not used with 'Custom' format. If Custom format is used the segment is ignored and must be specified as part of the Custom format.

Format - 1-10
Required. 'Formats' 1-9 are presets. Format 10 is a user-specified 'Custom' Format string

Formats 1-9 are:

1 –	DayMonthYear	6 –	MonthDayYearHourMinuteSecond
2 –	DayMonthYearHourMinute	7 –	YearMonthDay
3 –	DayMonthYearHourMinuteSecond	8 –	YearMonthDayHourMinute
4 –	MonthDayYear	9 –	YearMonthDayHourMinuteSecond
5 –	MonthDayYearHourMinute		

i.e., brc64 /Pattern:"backup*.log" /FixedExt:".txt" /AppendDate:M:P:" ":-:7



Date Manipulation

Using Custom date with BRC cont.

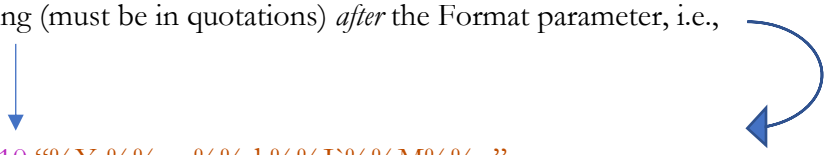
Using: backup01.log

Command line: cont.

Syntax of using /APPENDDATE cont.

Type:Mode:Separator:Segment:Format:Custom

Format – 10
This indicates that a 'Custom' date Format is to be used in place of the presets 1-9. The syntax is to place the 'Custom' date string (must be in quotations) *after* the Format parameter, i.e.,



10:"%Y-%%m-%%d %%I`%%M%%p"

This is omitted if you are not specifying Format=10.

So for example, if you wanted to suffix the filename with the day name and the month you could enter a 'Custom' date of

"%a%M"

You can enter other text in the filename, and the symbols above will be substituted with the values. So, you could enter a custom value of "Created on " %a, %d %B, %Y" to get "Created on Tuesday, 25th March, 2004.

Tim Notes:

Proper syntax of %

In a Command Line:

1. Two percent signs without anything in between are treated like a single): %%2 ~ %2
2. A single percent sign is used to indicate variables, i.e. %A

In a Batch File:

1. A single percent sign is used in batch files to represent command line parameters: %1, %2, ...
2. Two percent signs with any characters in between them are interpreted as a variable: %A%
3. The %* modifier is a unique modifier that represents all arguments passed in a batch file.

With thanks to RegexNinja

Date Manipulation

Using Custom date with BRC cont.

Using: backup01.log

Command line: cont.

Syntax of using /APPENDDATE cont.

Format – 10 cont.

List of Codes

Week:

%a Abbreviated weekday name	%A Full Weekday name	%U Week number of year (00-53) with Sunday as the first day of the week
%w Weekday (0-6), with Sunday=0	%W Week number of year (00-53) with Monday as the first day of the week	

Date:

%b Abbreviated month name	%B Full month name	%d Day of Month (01-31)
%j Day of Year (01-366)	%m Month number (01-12)	%y Year, with no century indicator (00-99)
%Y Year, with century indicator (e.g. 2004)		

Time:

%H Hour in 24-hour format (00-23)	%I Hour in 12-hour format (01-12)	%M Minute (00-59)
%p AM/PM Indicator	%S Seconds (00-59)	%z Time zone name

Percentage:

%% Percentage sign

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format

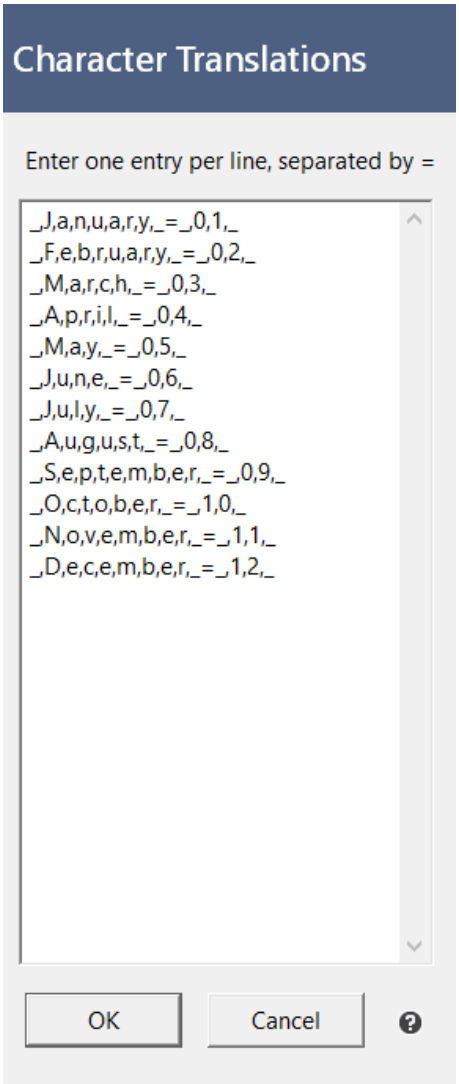
Tim Notes:

1. I have already presented a technique for doing this. See:

'dd_mmm_yyyy to yymmdd convert abbreviated Written date to Numeric'

However, this uses a modified technique that I thought readers could benefit from.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf



Observations:

1. Ensure that a "month name" is not being used otherwise.
   i.e., Mayoral.race.txt renamed to 05oral.race.txt

   solution:

   Include the underscore character before and after the month using 'Section #14: Character Translation'

2. This solution will only work for samples using the pattern:

   underscore_<written month> underscore

3. Additional pattern for the RegEx:

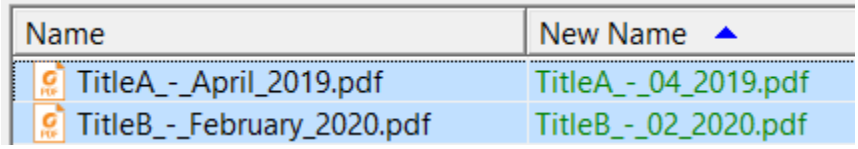
   underscore <hyphen> underscore <written month> underscore

4. The RegEx will match against years from 1970-2020

1. 'Section #14: Character Translation' –

```
_,J,a,n,u,a,r,y,_=_,0,1,_
_,F,e,b,r,u,a,r,y,_=_,0,2,_
_,M,a,r,c,h,_=_,0,3,_
_,A,p,r,i,l,_=_,0,4,_
_,M,a,y,_=_,0,5,_
_,J,u,n,e,_=_,0,6,_
_,J,u,l,y,_=_,0,7,_
_,A,u,g,u,s,t,_=_,0,8,_
_,S,e,p,t,e,m,b,e,r,_=_,0,9,_
_,O,c,t,o,b,e,r,_=_,1,0,_
_,N,o,v,e,m,b,e,r,_=_,1,1,_
_,D,e,c,e,m,b,e,r,_=_,1,2,_
```

This output is just using Character Translation without the RegEx

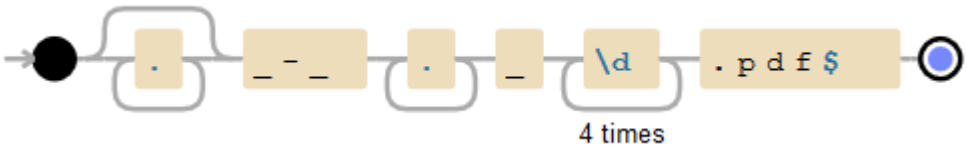


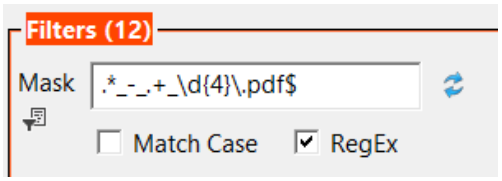
Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

2. 'Section #12: Filters' –



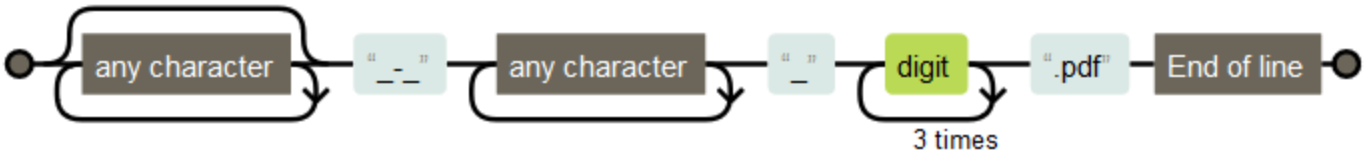


Mask: .*_-_.+_\d{4}\.pdf$
RegEx: Enabled

Tim Notes:

1. This only filters out the display to make it easier for file selection. It is not required.

Use the Filter Refresh blue button (located to the right of 'Mask' field) or just press F5 to see file list.



Using: TitleA_-_April_2019.pdf

Tim Analysis:

1. 1..	Match against any character	Filter = T
2. *	Make it Greedy Current position = EOL	Filter = <entire string>
3. _-_	Match against the pattern, underscore <hyphen> underscore Backtracks.	Filter = TitleA_-_
4. .	Match against any character	Filter = TitleA_-_A
5. +	Make it Greedy Current Position = EOL	Filter = TitleA_-_April_2019.pdf
6. _	Match against the underscore character Backtracks.	Filter = TitleA_-_April_
7. \d{4}	Match against 4 numeric digits	Filter = TitleA_-_April_2019
8. \.	Match against literal <dot> character	Filter = TitleA_-_April_2019.
9. pdf	Match against the string 'pdf'	Filter = TitleA_-_April_2019.pdf
10. $	Must be at EOL (End of Line)	

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

3. RegEx

Because of the order of evaluation, it is *this* step that gets performed first, not the Character Translation. The RegEx will perform the requested manipulation of the portions of the string. This manipulation will consist of switching around the year for the month. After this, the Character Translation will convert the written month of the date to a numeric date.

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Replace: \1 - \3\2

Name	New Name
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

After RegEx

Name	New Name
TitleA_-_April_2019.pdf	TitleA - 2019_04_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_02_.pdf

Immediately following RegEx, After Character Translation

Here is the Regex Buddy display which allows you to see the RegEx a little bit clearer using a color coding system.

Match:

```
^(.*?)_-(_[JFMASOND[a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
```

Where:

Plain Text or literal text =	White
Regular Expression sub-expression or Metacharacters =	Light Blue
Character Classes =	Yellow
Group level nesting =	
Level 1 =	Light Green
Level 2 =	Yellow

Samples:

```
.............................
Match 1: TitleA_-_April_2019
Group 1: TitleA
Group 2: _April_
Group 3: 2019
Group 4: 20
```

```
.............................
Match 1: TitleB_-_February_2020
Group 1: TitleB
Group 2: _February_
Group 3: 2020
Group 4: 20
```

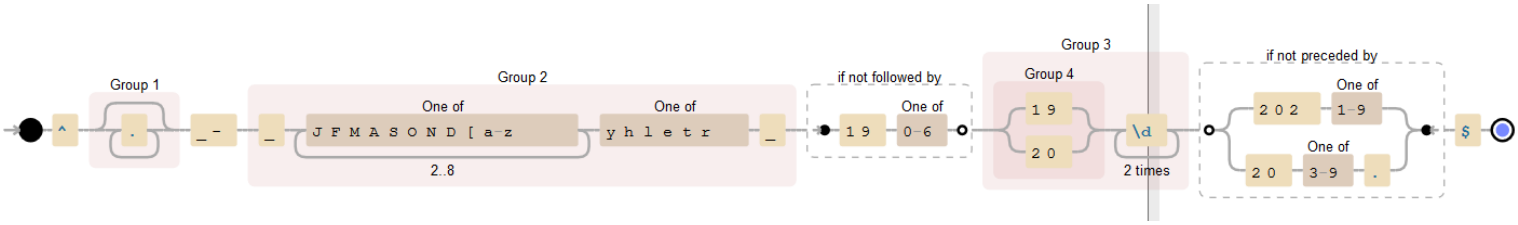
Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

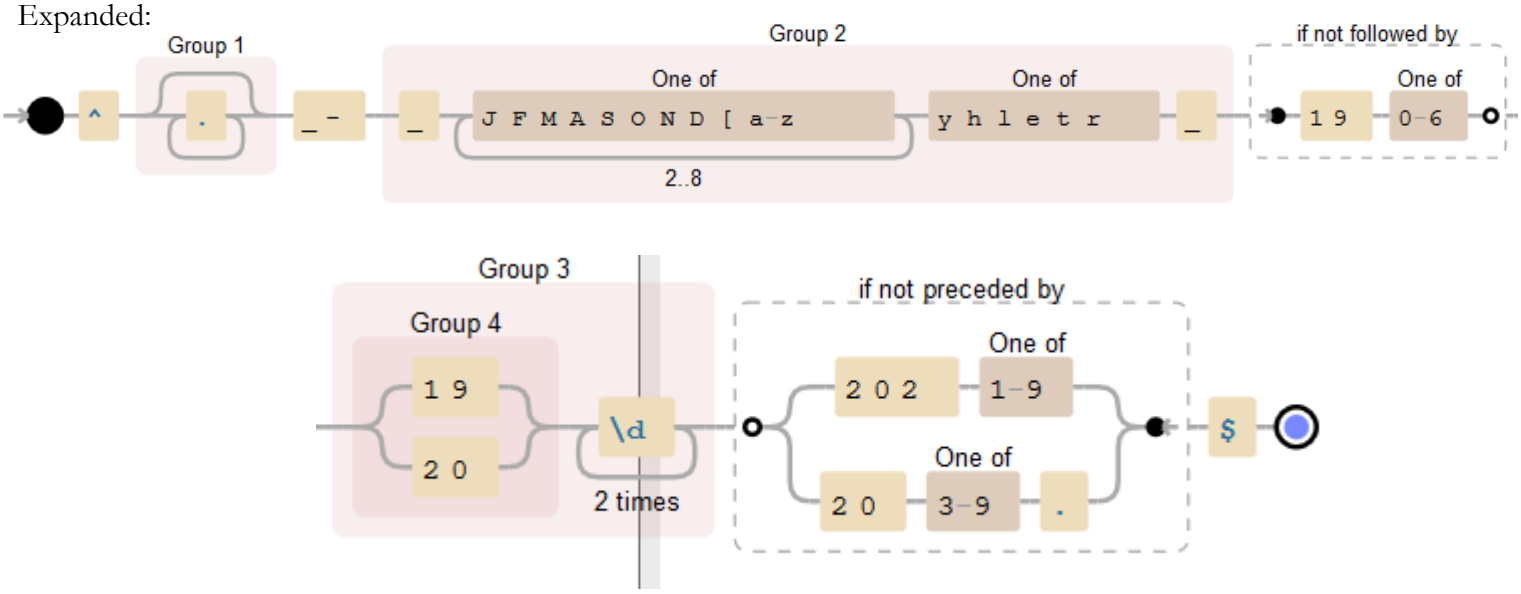
TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

3. RegEx cont.

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$



Expanded:



Samples

```
Match 1: TitleB_-_February_2020
Group 1: TitleB
Group 2: _February_
Group 3: 2020
Group 4: 20
```

```
Match 1: TitleA_-_April_2019
Group 1: TitleA
Group 2: _April_
Group 3: 2019
Group 4: 20
```

Where:

Capture Group 1 is 'reserved' for any preceding text
Capture Group 2 is 'reserved' for the Month
Capture Group 3 and 4 are nested and 'reserved' for the Year and Century

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down:

This is one of those few RegEx when broken down doesn't make it any simpler.

Anchor
^

Capture Group 1 — (.*?)

_-

Capture Group 2 — (_ [JFMASOND] [a-z] {2,8}? [yhletr] _)

Capture Group 3 — (

Negative Lookahead
Non-marking Group — (?! 19 [0-6])

Capture Group 4 — ((19 | 20) \d{2})

)

Negative Lookbehind
Non-Marking Group — (?<! 202 [1-9] | 20 [3-9] .)

Anchor
$

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down: cont.

Sub-Expression	Value	Capture Group	Notations
^			BOL Anchor
(.*?)	= TitleA	1	Capture up to 1st underscore
_-			Matches underscore followed by <hyphen>
_	= _	2	
[JFMASOND]	= _A	2	Match against first letter of Month
[a-z]{2,8}?[yhletr]	= _April	2	Match against subsequent letters of Month
_	= _April_	2	
(?!19[0-6])	= TRUE		Validate - Not Years 1900 – 1969 Negative Lookahead. Non-Marking Group
((19|20)\d{2})	= 2019	3,4	Capture Year Nested. Capture Group 4 = 20 Capture Group 3 = 2019
(?<!202[1-9]|20[3-9].)	= TRUE		Validate - Not years 2021 – 2099 Negative Lookbehind. Non-Marking Group
$			EOL Anchor

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down: cont.

Get Month –

Class [JFMASOND]

J = January, June, July
F = February
M = March, May
A = April, August
S = September
O = October
N = November
D = December

Class [a-z] {2,8}

Class [yhletr]

y = January, February, May, July
h = March
l = April
e = June
t = August
r = September, October, November, December

The First character of the month has already been captured.
The next [a-z] captures the second character.
This match is repeated from a minimum of 2 to 8 iterations.

Where:

The minimum represents the months with the shortest number of letters
= M ay (2 additional characters)

.. to the month with the longest number of letters
= S eptember (8 additional characters)

Class [yhletr]

Months ending in y = January, February, May, July,
Months ending in h = March
Months ending in l = April
Months ending in e = June
Months ending in t = August
Months ending in r = September, October, November, December

Tim Notes:

1. I don't see a purpose for this because the [a-z] {2,8} has already captured the full month. Unless this is a type of validation to make sure a valid month is matched?

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down: cont.

Get Year –

Negative Lookahead
Non-marking Group

Class
(?! 19 [0-6])

After capturing the written month, a validation is made for the year. This is because the RegEx is only supposed to cover the range of 1970 to 2020

Class Range 0-6

Look ahead that the year is **NOT** 19 0 = 190 - covers 1900 - 1909
1 = 191 - covers 1910 - 1919
2 = 192 - covers 1920 - 1929
3 = 193 - covers 1930 - 1939
4 = 194 - covers 1940 - 1949
5 = 195 - covers 1950 - 1959
6 = 196 - covers 1960 - 1969

Once the validation is made, the year is captured using an Alternate.

This uses an Alternate for Capture Group 3 & 4:

This	OR	This	
19	|	20	\d {2}

The value of 19 or 20 covering the first part of the century from 1970 (1900-1969 have already been validated) through the 2000's. The second part of the century is captured through \d {2}.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

TitleA_-_April_2019.pdf to TitleA - 2019-04.pdf
TitleB_-_February_2020.pdf to TitleB - 2020-02.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down: cont.

Get Year – cont.

The Next part validates the 2000's year further.

Negative Lookbehind
Non-Marking Group

Alternate
Class Class <dot>
(?<! 202 [1-9] | 20 [3-9] .)

The limit placed on the year 2000 is up to 2020. Anything after the year 2020 will not be accepted. This is done through a validation using a Negative Lookbehind. Validates against years 2021 through 2099.

After the year is captured, it looks back to validate that the year, if in the century 20, does **NOT** fall within the years:

This uses an Alternate for a Non-Marking Group:

This			OR	This		
						<dot>
?<! 202	[1-9]		|	?<! 20	[3-9]	.

Year			Years		
202	1	= 2021	20	3	= 2030 - 2039
202	2	= 2022	20	4	= 2040 - 2049
202	3	= 2023	20	5	= 2050 - 2059
202	4	= 2024	20	6	= 2060 - 2069
202	5	= 2025	20	7	= 2070 - 2079
202	6	= 2026	20	8	= 2080 - 2089
202	7	= 2027	20	9	= 2090 - 2099
202	8	= 2028			
202	9	= 2029			

The <dot> would be representative of each individual year 2031, 2032, …2039 etc. pertaining to each decade 2030-2090.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Using: TitleA_-_April_2019.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Where:

\1 = TitleA \2 = _April_ \3 = 2019 \4 = 20

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = T
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. _	Match against underscore (Not Captured)	Changes Capture Group 1 = TitleA
6. <space>	Match against a <space> character (Not Captured)	
7. _	Match against underscore	Capture Group 2 = _
8. [JFMASOND]	Match against a class consisting of single letters, 'J', 'F', 'M', 'A', 'S', 'O', 'N' or 'D'	Capture Group 2 = _A

Tim Notes:

1. For a full explanation of this, see Get Month, on previous pages. Tests the next character for a match. The 'A' in 'April' matches.

9. [a-z]	Match against a class consisting of a lowercase letter.	Capture Group 2 = _Ap
10. {2,8}	Specifies a minimum of 2 and maximum of 8 match repetitions	Capture Group 2 = _April

Tim Notes:

1. For a full explanation of this, see Get Month, on previous pages. Matches against the lowercase letters for 2 to 8 iterations. This captures, 'p', 'r', 'i' and 'l' of 'April'.

11. ?	Make the maximum optional. Will only perform the minimum permutations as required.	Capture Group 2 = _Apr
12. [yhletr]	Match against a class consisting of 'y', 'h', 'l', 'e', 't' or 'r'	Capture Group 2 = _April

Tim Notes:

1. Full explanation of steps 11 and 12 will follow. See Next page. Also see, Get Month, on previous pages.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Using: TitleA_-_April_2019.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Where:

\1 = TitleA \2 = _April_ \3 = 2019 \4 = 20

Tim Analysis: cont.

Tim Notes:

1. Using, ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]), The minimum required can clearly be seen here:

Name	New Name
TitleA_-_April_2019.pdf	_April.pdf
TitleA_-_August_2002.pdf	_August.pdf
TitleA_-_December_1995.pdf	_Dece.pdf
TitleA_-_July_1974.pdf	_July.pdf
TitleA_-_June_1974.pdf	_June.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	_Nove.pdf
TitleA_-_October_2010.pdf	_Octobe.pdf
TitleA_-_September_1987.pdf	_Sept.pdf
TitleB_-_February_2020.pdf	_Febr.pdf

April –

step 8	captures the 'A'
[a-z] {2-8}	captures 'pril'
?	changes to 'Apr'
[yhletr])	Matches the 'l' of 'April' so changes the match to capture the letters *leading up* to the 'l' = 'Apr' + 'i' + 'l' The additional letter captured is the 'i' leading up to the 'l' of 'April'

Dece –

step 8	captures the 'D'
[a-z] {2-8}	captures 'ecember'
?	changes to 'Dec'
[yhletr])	Matches the 'e' of 'December' so changes match to capture the letters leading up to the 'e' = 'Dec' + 'e' There were no additional letters captured leading up to the second 'e' of 'December'

2. For a full explanation of this, see, Get Month, on previous pages.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Using: TitleA_-_April_2019.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Replace: \1 - \3\2

Where:

\1 = TitleA \2 = _April_ \3 = 2019 \4 = 20

Tim Analysis: cont.

Tim Notes:

1. The analysis has revealed a problem. This, as it stands now, will not match against the month of May. The month of May has been covered in the match of {2} minimum using step 8 to capture the 'M' and [a-z]{2,8} to capture the 'ay'. The next sub-expression, [yhletr] has nothing left to match up against. the RegEx will fail. The sub-expression is also redundant because [a-z] {2,8} captures the month. There's no need to match against the ending character of the Month.

This can be verified by the output:

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr])
Replace: \2

Name	New Name
TitleA_-_April_2019.pdf	_April.pdf
TitleA_-_August_2002.pdf	_August.pdf
TitleA_-_December_1995.pdf	_Dece.pdf
TitleA_-_July_1974.pdf	_July.pdf
TitleA_-_June_1974.pdf	_June.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	_Nove.pdf
TitleA_-_October_2010.pdf	_Octobe.pdf
TitleA_-_September_1987.pdf	_Sept.pdf
TitleB_-_February_2020.pdf	_Febr.pdf

Name	New Name
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_August_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_December_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_July_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_June_.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_November_.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_October_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_September_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

Even the final output from the full RegEx string, confirms this.

The initial sample base wasn't large enough, so this problem went unnoticed.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Using: TitleA_-_April_2019.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Replace: \1 - \3\2

Where:

\1 = TitleA \2 = _April_ \3 = 2019 \4 = 20

Tim Analysis: cont.

Tim Notes:

There are solutions:

1. Change the Quantifier to {1,8}. The [a-z] will match a minimum of one iteration, so step 8 captures the 'M' and [a-z] will capture the 'a' and [yhletr] will capture the 'y'.

Name	New Name
TitleA_-_April_2019.pdf	_Apr.pdf
TitleA_-_August_2002.pdf	_August.pdf
TitleA_-_December_1995.pdf	_Dece.pdf
TitleA_-_July_1974.pdf	_Jul.pdf
TitleA_-_June_1974.pdf	_June.pdf
TitleA_-_May_2019.pdf	_May.pdf
TitleA_-_November_2003.pdf	_Nove.pdf
TitleA_-_October_2010.pdf	_Oct.pdf
TitleA_-_September_1987.pdf	_Sept.pdf
TitleB_-_February_2020.pdf	_Febr.pdf

2. Remove the class [yhletr]. so the RegEx becomes, ^(.*?)_-(_[JFMASOND][a-z]{2,8}?)

Name	New Name
TitleA_-_April_2019.pdf	_Apr.pdf
TitleA_-_August_2002.pdf	_Aug.pdf
TitleA_-_December_1995.pdf	_Dec.pdf
TitleA_-_July_1974.pdf	_Jul.pdf
TitleA_-_June_1974.pdf	_Jun.pdf
TitleA_-_May_2019.pdf	_May.pdf
TitleA_-_November_2003.pdf	_Nov.pdf
TitleA_-_October_2010.pdf	_Oct.pdf
TitleA_-_September_1987.pdf	_Sep.pdf
TitleB_-_February_2020.pdf	_Feb.pdf

As a side note:

Interesting enough, this also gives us a three character abbreviation for the month. This, again, is because of the ' ? ', which makes minimum permutations required, so step 6 captures the first character, and [a-z] {2,8}? captures the next two characters.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Using: TitleA_-_April_2019.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Replace: \1 - \3\2

Where:

\1 = TitleA \2 = _April_ \3 = 2019 \4 = 20

Tim Analysis: cont.

Tim Notes:

For now let's continue with the analysis.

13. _	Match against underscore (Not Captured)	All Capture Groups = Unchanged
14. (?!19[06])	Negative Lookahead. Look to the right that the next Character is 'not' a string '19' followed by a class consisting of either a '0' or a '6'. = '20' + '1' = True.	All Capture Groups = Unchanged

Tim Notes:

1. Validate against years 1900-1969.
2. For a full explanation of this, see, Get Year, on previous pages.

15. (19|20)	Alternate. (see Previous pages for explanation)		
	a. 19	Match against the string '19' = '20' = Fail.	Capture Group 4 = No Value Capture Group 3 = No Value
	or		
	b. 20	Match against the string '20'	Capture Group 4 = 20 Capture Group 3 = 20
16. \d{2}	Match against 2 numeric digits		Capture Group 4 = Unchanged Capture Group 3 = 2019

Tim Notes:

1. Match against Century. Capture Year.
2. For a full explanation of this, see, Get Year, on previous pages.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Using: TitleA_-_April_2019.pdf

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Replace: \1 - \3\2

Where:

\1 = TitleA \2 = _April_ \3 = 2019 \4 = 20

Tim Analysis: cont.

17. (?<!202[1-9]|20[3-9].)

Alternate (see Previous pages for explanation)
a. ?<!202[1-9] Negative Lookbehind.
Look to the left that the last
character is 'not' a member of a
class consisting of a numeric digit
'1' – '9' preceded by a string,
'202' = '9' and '201' = True

or

a. ?<!20[3-9]. Negative Lookbehind.
Look to the left that the last
character is 'not' any character
preceded by a string '20'
followed by a member of a class
consisting of a numeric digit '3' – '9'
= '9' and '20' + '1' = True

Tim Notes:

1. This validates against the years 2021 – 2029 (First Alternate) and 2030 = 2099 (Second Alternate). Since both branches are True, the first one is selected, although it doesn't matter which because in either case, the evaluation continues.

2. For a full explanation, see, Get Year, on previous pages.

18. $ Must be at EOL (End of Line)

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974	TitleA_-_May_2019	TitleA_-_August_2002	TitleA_-_July_1974
TitleA_-_September_1987	TitleA_-_April_2019	TitleA_-_October_2010	TitleB_-_February_2020
TitleA_-_December_1995	TitleA_-_November_2003	TitleA_-_November_2033	

Match: ^(.*?)_-(_[JFMASOND][a-z]{2,8}?[yhletr]_)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Replace: \1 - \3\2

Name	New Name
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_August_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_December_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_July_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_June_.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_November_.pdf
TitleA_-_November_2033.pdf	TitleA_-_November_2033.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_October_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_September_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

Tim Analysis: cont.

Okay, so what about those issues of not matching against 'May' and possible redundancies?

I found that if I modify it to this, it removes the redundancies and still match all intended samples including 'May':

Modified: ^(.*?)_-(_[JFMASOND][a-z]{2,8} _)(?!19[06])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$
Original: ^(.*?)_-(_[JFMASOND][a-z]{2,8} ?[yhletr] _)(?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Name	New Name
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_August_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_December_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_July_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_June_.pdf
TitleA_-_May_2019.pdf	TitleA - 2019_May_.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_November_.pdf
TitleA_-_November_2033.pdf	TitleA_-_November_2033.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_October_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_September_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

Tim Notes:

1. November 2033 is to be ignored because it exceeds the allowed year range of 1970-2020.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974	TitleA_-_May_2019	TitleA_-_August_2002	TitleA_-_July_1974
TitleA_-_September_1987	TitleA_-_April_2019	TitleA_-_October_2010	TitleB_-_February_2020
TitleA_-_December_1995	TitleA_-_November_2003	TitleA_-_November_2033	

Tim Notes:

1. You should also be reminded that after the RegEx finishes, the Character Translation will be performed and the final result is:

	New Name
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_August_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_December_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_July_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_June_.pdf
TitleA_-_May_2019.pdf	TitleA - 2019_May_.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_November_.pdf
TitleA_-_November_2033.pdf	TitleA_-_November_2033.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_October_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_September_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

Wait. There's a new problem. It seems that the year 2033, although ignored by the RegEx, was not ignored by the Character Translation. The Character Translation has no way to determine a good file from a bad. It only looks to convert the written month into a numeric.

The only way I can see preventing this from happening, is not from the RegEx side, but either through a different filter or manually unselecting those files before performing the final Renaming operation.

I think I will try a new filter.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974	TitleA_-_May_2019	TitleA_-_August_2002	TitleA_-_July_1974
TitleA_-_September_1987	TitleA_-_April_2019	TitleA_-_October_2010	TitleB_-_February_2020
TitleA_-_December_1995	TitleA_-_November_2003	TitleA_-_November_2033	

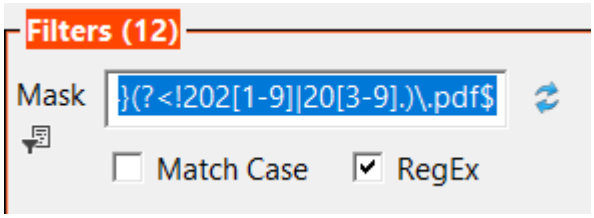
Tim Notes: cont.

Here is a corrected Filter:

Filter: .*_-_.+_\d{4}(?<!202[1-9]|20[3-9].)\.pdf$

Comparison -

Original Filter: .*_-_.+_\d{4} \.pdf$
Modified Filter: .*_-_.+_\d{4} (?<!202[1-9]|20[3-9].) \.pdf$



Name	New Name
TitleA_-_April_2019.pdf	TitleA_-_April_2019.pdf
TitleA_-_August_2002.pdf	TitleA_-_August_2002.pdf
TitleA_-_December_1995.pdf	TitleA_-_December_1995.pdf
TitleA_-_July_1974.pdf	TitleA_-_July_1974.pdf
TitleA_-_June_1974.pdf	TitleA_-_June_1974.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	TitleA_-_November_2003.pdf
TitleA_-_October_2010.pdf	TitleA_-_October_2010.pdf
TitleA_-_September_1987.pdf	TitleA_-_September_1987.pdf
TitleB_-_February_2020.pdf	TitleB_-_February_2020.pdf

I simply added in the validation for the years 2021-2099.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974	TitleA_-_May_2019	TitleA_-_August_2002	TitleA_-_July_1974
TitleA_-_September_1987	TitleA_-_April_2019	TitleA_-_October_2010	TitleB_-_February_2020
TitleA_-_December_1995	TitleA_-_November_2003	TitleA_-_November_2033	

New Expanded Sample Base Inclusions:

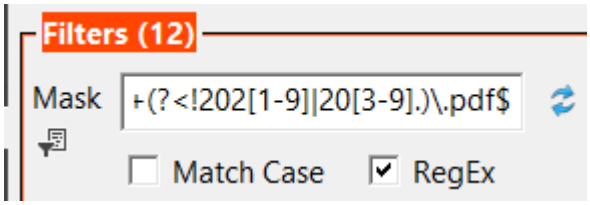
TitleA - April 2019.pdf
TitleB - January 8, 2019.pdf

This requires a whole new RegEx and filter.

1. Use the same Character Translation.

2. Need new filter (again)

Filter: .*[_]-[_].+(?<!202[1-9]|20[3-9].)\.pdf$



Comparison -

Original Filter:	.*	_-_	.+	_\d{4}		\.pdf$
Modified Filter:	.*	_-_	.+	_\d{4}	(?<!202[1-9]|20[3-9].)	\.pdf$
New Filter:	.*	[_]-[_]	.+		(?<!202[1-9]|20[3-9].)	\.pdf$

Name	New Name
TitleA - April 2019.pdf	TitleA - April 2019.pdf
TitleA_-_April_2019.pdf	TitleA_-_April_2019.pdf
TitleA_-_August_2002.pdf	TitleA_-_August_2002.pdf
TitleA_-_December_1995.pdf	TitleA_-_December_1995.pdf
TitleA_-_July_1974.pdf	TitleA_-_July_1974.pdf
TitleA_-_June_1974.pdf	TitleA_-_June_1974.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	TitleA_-_November_2003.pdf
TitleA_-_October_2010.pdf	TitleA_-_October_2010.pdf
TitleA_-_September_1987.pdf	TitleA_-_September_1987.pdf
TitleB - January 8, 2019.pdf	TitleB - January 8, 2019.pdf
TitleB_-_February_2020.pdf	TitleB_-_February_2020.pdf

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974	TitleA_-_May_2019	TitleA_-_August_2002	TitleA_-_July_1974
TitleA_-_September_1987	TitleA_-_April_2019	TitleA_-_October_2010	TitleB_-_February_2020
TitleA_-_December_1995	TitleA_-_November_2003	TitleA_-_November_2033	

New Expanded Sample Base Inclusions: cont.

TitleA - April 2019.pdf
TitleB - January 8, 2019.pdf

3. RegEx

Match:
^(.*?)[_]-[_]([JFMASOND][a-z]{2,8}?[yhletr])(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Replace: \1 - \4_\2_

Name	New Name
TitleA - April 2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_August_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_December_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_July_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_June_.pdf
TitleA_-_May_2019.pdf	TitleA_-_May_2019.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_November_.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_October_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_September_.pdf
TitleB - January 8, 2019.pdf	TitleB - 2019_January_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

After RegEx

Tim Notes:

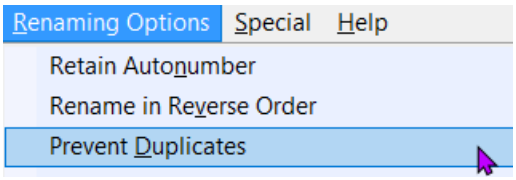
1. May didn't match – same problem as before. Because of this, Character Translation below for this entry will also be in error.

Name	New Name
TitleA - April 2019.pdf	TitleA - 2019_04_.pdf
TitleA_-_April_2019.pdf	TitleA - 2019_04_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_08_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_12_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_07_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_06_.pdf
TitleA_-_May_2019.pdf	TitleA_-_05_2019.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_11_.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_10_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_09_.pdf
TitleB - January 8, 2019.pdf	TitleB - 2019_01_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_02_.pdf

After Character Translation

Tim Notes:

1. There will be a Duplicate File conflict with 2019_04 unless you have 'Prevent Duplicates' enabled:



Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

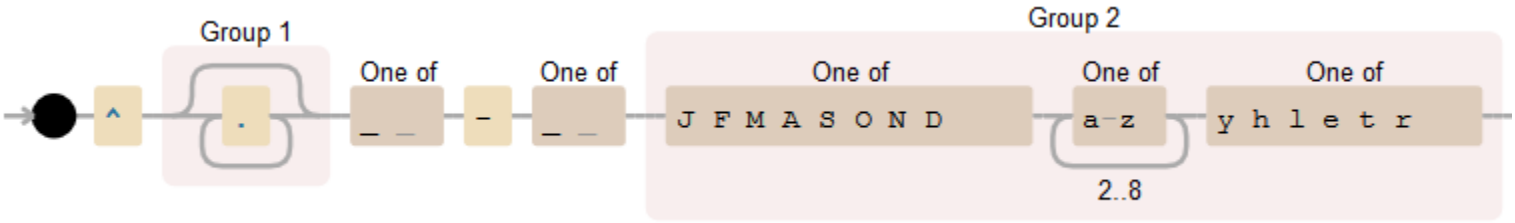
TitleA_-_June_1974
TitleA_-_May_2019
TitleA_-_August_2002
TitleA_-_July_1974
TitleA_-_September_1987
TitleA_-_April_2019
TitleA_-_October_2010
TitleB_-_February_2020
TitleA_-_December_1995
TitleA_-_November_2003
TitleA_-_November_2033

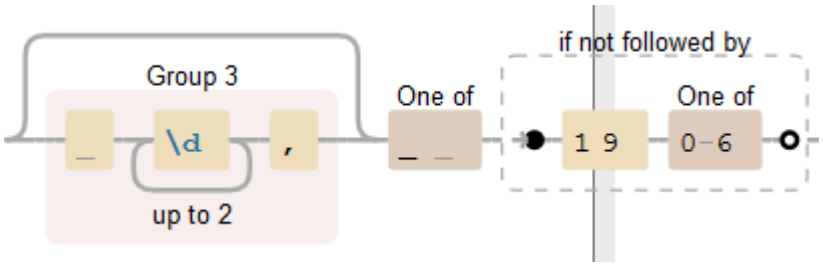
New Expanded Sample Base Inclusions: cont.

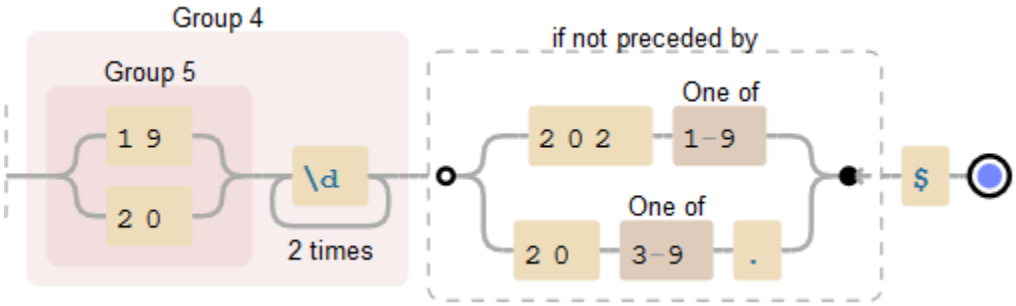
TitleA - April 2019.pdf
TitleB - January 8, 2019.pdf

Match:
^(.*?)[_]-[_]([JFMASOND][a-z]{2,8}?[yhletr])(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Expanded:







Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974 TitleA_-_May_2019 TitleA_-_August_2002 TitleA_-_July_1974
TitleA_-_September_1987 TitleA_-_April_2019 TitleA_-_October_2010 TitleB_-_February_2020
TitleA_-_December_1995 TitleA_-_November_2003 TitleA_-_November_2033

New Expanded Sample Base Inclusions: cont.

TitleA - April 2019.pdf
TitleB - January 8, 2019.pdf

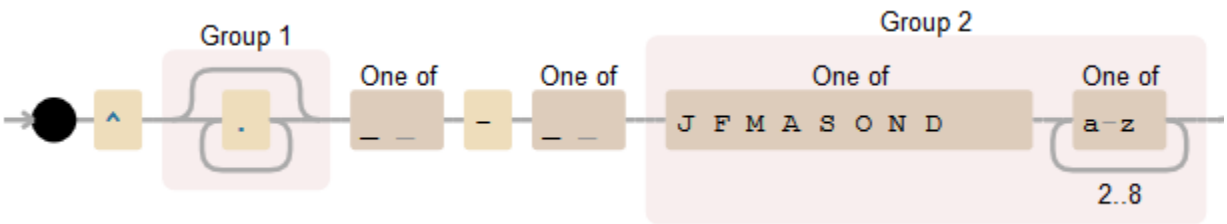
Match:
^(.*?)[_]-[_]([JFMASOND][a-z]{2,8}?[yhletr])(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Same problems with redundancy and the month of 'May'. Fixed by removing, `?[yhletr]` :

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Name	New Name
TitleA - April 2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_April_2019.pdf	TitleA - 2019_April_.pdf
TitleA_-_August_2002.pdf	TitleA - 2002_August_.pdf
TitleA_-_December_1995.pdf	TitleA - 1995_December_.pdf
TitleA_-_July_1974.pdf	TitleA - 1974_July_.pdf
TitleA_-_June_1974.pdf	TitleA - 1974_June_.pdf
TitleA_-_May_2019.pdf	TitleA - 2019_May_.pdf
TitleA_-_November_2003.pdf	TitleA - 2003_November_.pdf
TitleA_-_October_2010.pdf	TitleA - 2010_October_.pdf
TitleA_-_September_1987.pdf	TitleA - 1987_September_.pdf
TitleB - January 8, 2019.pdf	TitleB - 2019_January_.pdf
TitleB_-_February_2020.pdf	TitleB - 2020_February_.pdf

Changes Visualization for that section to:



Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

Expanded Sample Base:

TitleA_-_June_1974	TitleA_-_May_2019	TitleA_-_August_2002	TitleA_-_July_1974
TitleA_-_September_1987	TitleA_-_April_2019	TitleA_-_October_2010	TitleB_-_February_2020
TitleA_-_December_1995	TitleA_-_November_2003	TitleA_-_November_2033	

New Expanded Sample Base Inclusions: cont.

TitleA - April 2019.pdf
TitleB - January 8, 2019.pdf

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Samples

```
.................................Start·Length···
Match 1: TitleA_-_December_1995      0     22
Group 1: TitleA                      0      6
Group 2: December                    9      8
Group 3 did not participate in the match
Group 4: 1995                       18      4
Group 5: 19                         18      2
```

```
.................................Start·Length··
Match 1: TitleA - April 2019         0     19
Group 1: TitleA                      0      6
Group 2: April                       9      5
Group 3 did not participate in the match
Group 4: 2019                       15      4
Group 5: 20                         15      2
```

```
.................................Start·Length··
Match 1: TitleA_-_September_1987     0     23
Group 1: TitleA                      0      6
Group 2: September                   9      9
Group 3 did not participate in the match
Group 4: 1987                       19      4
Group 5: 19                         19      2
```

```
.................................
Match 1: TitleB - January 8, 2019
Group 1: TitleB
Group 2: January
Group 3:  8,
Group 4: 2019
Group 5: 20
```

Where:

Capture Group 1 is 'reserved' for any preceding text
Capture Group 2 is 'reserved' for the Month
Capture Group 3 is 'reserved' for those strings that have a Day of the Month, otherwise No Value.
Capture Group 4 and 5 are nested and 'reserved' for the Year and Century.

Tim Notes:

1. Capture Group 3's 'No Value' displays in the Regex Buddy program as 'Group 3 did not participate in the match'.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Comparison

New Sub-expression	Original Sub-expression	Change
^	^	None
(.*?)	(.*?)	None
[_]-[_]	_-	[underscore <space] - [underscore <space>
([JFMASOND][a-z]{2,8}?[yhletr])	(_[JFMASOND][a-z]{2,8}?[yhletr]_)	underscores removed
(\d{1,2},){0,1}		<space> \d{1,2}<comma> {0,1}
[_]		[underscore <space]
(?!19[0-6])	(?!19[0-6])	None
((19|20)\d{2})	((19|20)\d{2})	None
(?<!202[1-9]|20[3-9].)	(?<!202[1-9]|20[3-9].)	None
$	$	None

Won't match against the month of 'May' and same redundancy problem. Changed to:

([JFMASOND][a-z]{2,8})

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down:

Anchor
^

Capture Group 1 (.*?)

Class [_<space>]

\-

Class [_ <space>]

Capture Group 2 ([JFMASOND] [a-z] {2,8})

Capture Group 3 (<space> \d {1,2} ,) {0,1}

Class [_ <space>]

Capture Group 4

Negative Lookahead
Non-marking Group

Class (?! 19 [0-6])

Capture Group 5 ((19 | 20) \d{2})

Negative Lookbehind
Non-Marking Group

Alternate
Class (?<! 202 [1-9] |
Class 20 [3-9] .)

Anchor
$

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

TitleA - April 2019.pdf
TitleB - January 8, 2019.pdf

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Breaks down: cont.

Sub-Expression	Value	Capture Group	Notations
^			BOL Anchor
(.*?)	= TitleB	1	Capture up to 1st underscore
[_]			Match underscore or <space>
-			Match <hyphen>
[_]			Match underscore or <space>
[JFMASOND]	= J	2	Match against first letter of Month
[a-z]{2,8}	= January	2	Match against subsequent letters of Month
\d{1,2}, {0,1}	= 8,	3	Match against day of Month. The {0,1} makes it optional for strings that don't have it – Repeat Match zero or one iteration.
[_]			Match underscore or <space>
(?!19[0-6])	= TRUE		Validate - Not Years 1900 – 1969 Negative Lookahead. Non-Marking Group
((19|20)\d{2}))	= 2019	4,5	Capture Year Nested. Capture Group 5 = 20 Capture Group 4 = 2019
(?<!202[1-9]|20[3-9].)	= TRUE		Validate - Not years 2021 – 2099 Negative Lookbehind. Non-Marking Group
$			EOL Anchor

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

Using: TitleB - January 8, 2019.pdf

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Where:

\1 = TitleB \2 = January \3 = 8, \4 = 2019 \5 = 20

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = T
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [_]	Match against a class consisting of an (Not Captured) a. underscore b. <space> = <space>. This is the <space> after 'TitleB'	Changes Capture Group 1 = TitleB
6. -	Match against <hyphen> (Not Captured)	
7. [_]	Match against a class consisting of an (Not Captured) a. underscore b. <space> = <space>. This is the <space> preceding 'January'	

Tim Notes:

1. Steps 5 through 7 match against strings that have:

<space> - <space> January
_-_September

8. [JFMASOND]	Match against a class consisting of single letters, 'J', 'F', 'M', 'A', 'S', 'O', 'N' or 'D'	Capture Group 2 = J

Tim Notes:

1. For a full explanation of this, see Get Month, on previous pages. Tests the next character for a match. The 'J' in 'January' matches.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

Using: TitleB - January 8, 2019.pdf

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Where:

\1 = TitleB \2 = January \3 = 8, \4 = 2019 \5 = 20

Tim Analysis: cont.

9. [a-z]	Match against a class consisting of a lowercase letter.	Capture Group 2 = Ja
10. {2,8}	Specifies a minimum of 2 and maximum of 8 match repetitions	Capture Group 2 = January

Tim Notes:

1. For a full explanation of this, see Get Month, on previous pages. Matches against the lowercase letters for 2 to 8 iterations. This captures, 'a', 'n', 'u', 'a', 'r' and 'y' of 'January'.

11. <space>	Match against a <space> character	Capture Group 3 = <space>
12. \d	Match against a numeric digit	Capture Group 3 = <space> 8
13. {1,2}	Repeat Match for a minimum of 1 iteration and a maximum of 2 iterations	Capture Group 3 = Unchanged

Tim Notes:

1. This covers strings that have a Day of the Month. The {1,2) accounts for the format D or DD for the day, i.e. 10.

14. ,	Match against <comma>	Capture Group 3 = <space>8,
15. {0,1}	Repeat the entire sub-expression zero or one iteration.	

Tim Notes:

1. This is really the equivalent of using the ' ? ' Quantifier, because it makes the entire expression optional. Why? Because it accounts for those strings that do not have a day of the month specified, i.e. TitleA_-_April_2019.pdf. If it didn't have this, the RegEx would fail to match against these strings.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

Using: TitleB - January 8, 2019.pdf

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Where:

\1 = TitleB \2 = January \3 = 8, \4 = 2019 \5 = 20

Tim Analysis: cont.

16. [_] Match against a class consisting of an (Not Captured)
a. underscore
b. <space> = <space>. This is the <space> preceding '2019'

Tim Notes:

1. This matches strings that have either a <space> or an underscore:

April_2019.pdf
April 2019.pdf

17. (?!19[06]) Negative Lookahead. All Capture Groups = Unchanged
Look to the right that the next character is 'not' a string '19' followed by a class consisting of either a '0' or a '6'. = '20' + '1' = True.

Tim Notes:

1. Validate against years 1900-1969.
2. For a full explanation of this, see, Get Year, on previous pages.

18. (19|20) Alternate. (see Previous pages for explanation)
a. 19 Match against the string '19' = '20' = Fail. Capture Group 5 = No Value
Capture Group 4 = No Value

or

b. 20 Match against the string '20' Capture Group 5 = 20
Capture Group 4 = 20

19. \d{2} Match against 2 numeric digits Capture Group 5 = Unchanged
Capture Group 4 = 2019

Tim Notes:

1. Match against Century. Capture Year.
2. For a full explanation of this, see, Get Year, on previous pages.

Date Manipulation

Use Character Translation to Convert Written Month to Numeric Format cont.

New Expanded Sample Base Inclusions: cont.

Using: TitleB - January 8, 2019.pdf

Match: ^(.*?)[_]-[_]([JFMASOND][a-z]{2,8})(\d{1,2},){0,1}[_](?!19[0-6])((19|20)\d{2})(?<!202[1-9]|20[3-9].)$

Where:

\1 = TitleB \2 = January \3 = 8, \4 = 2019 \5 = 20

Tim Analysis: cont.

17. (?<!202[1-9]|20[3-9].)

Alternate (see Previous pages for explanation)

a. ?<!202[1-9] Negative Lookbehind.
Look to the left that the last
character is 'not' a member of a
class consisting of a numeric digit
'1' – '9' preceded by a string,
'202' = '9' and '201' = True

or

a. ?<!20[3-9]. Negative Lookbehind.
Look to the left that the last
character is 'not' any character
preceded by a string '20'
followed by a member of a class
consisting of a numeric digit '3' – '9'
= '9' and '20' + '1' = True

Tim Notes:

1. This validates against the years 2021 – 2029 (First Alternate) and 2030 = 2099 (Second Alternate). Since both branches are True, the first one is selected, although it doesn't matter which because in either case, the evaluation continues.

2. For a full explanation, see, Get Year, on previous pages.

18. $ Must be at EOL (End of Line)

This has been a Tim Contribution – along with ..

Forum Contributors: Admin, therube, RegexNinja

String Manipulation (Parsing)

String Manipulation (Parsing)

Tim Notes: Parsing

Here is a great way to diagnose a string: Parsing. Parsing is isolating each element of the string.

Note: Must have 'Rename File Extensions as Being Part of File Name (Ignore Extension) ON if string contains <dots>

Otherwise:

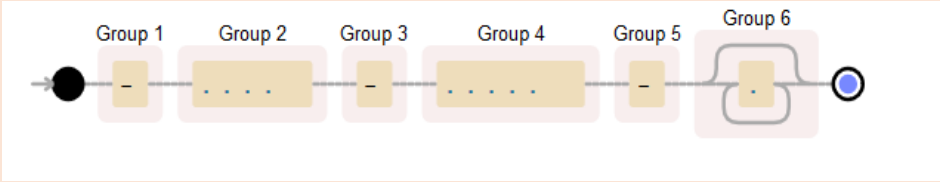
2016.11.3jpg

(.) = 2.3 instead of just 2

i.e. EXAMPLE SHOW-JANE-WAYNE-CATEGORYB.jpg

Use (….) to isolate each part of the string then replace that with the appropriate (.*) (\d{#}), etc.

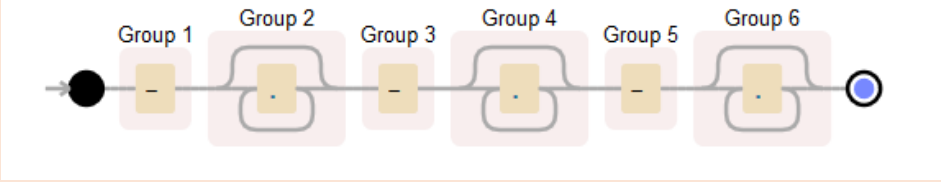
Match: (-)(....)(-)(.....)(-)(.*)



1. (-) – locate the first <hyphen> in the string
2. Verified by (….) = JANE
3. (-) - Find next <hyphen>
4. Verified by (…..) = WAYNE
5. (-) – Find final <hyphen>
6. (.*) - Gather up rest of string = CATEGORYB.jpg

Now substitute (.*) for the (….) because these are strings

Match: (-)(.*)(-)(.*)(-)(.*)



Capture Group 1 = - (first <hyphen>)
Capture Group 2 = JANE
Capture Group 3 = - (second <hyphen>)
Capture Group 4 = WAYNE
Capture Group 5 = - (third <hyphen>)
Capture Group 6 = CATEGORYB.jpg

Now you have parsed the string and can put it together the way you want.

\2-\4 = JANE-WAYNE

Name	New Name
EXAMPLE SHOW-JANE-WAYNE-CATEGORYB.jpg	JANE-WAYNE.jpg

\2-\4 = JOHN DOE

Name	New Name
SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg	JOHN-DOE.jpg

(Apply same RegEx to different sample)

Note: you could also try combinations of (\w) or (\W) for words and non-words and \d for numeric digits

String Manipulation (Parsing)

Re-arrange String parts

Numbers 12 - Colors Red - Letters ABCDE to Colors Red - Letters ABCDE - Numbers 12

Match: (.*?)(-)(.*?)(-)(.*)

Various Replace Strings:

Replace: \1 - \5

Name	New Name
Numbers 12 - Colors Red - Letters ABCDE.pdf	Numbers 12 - Letters ABCDE.pdf

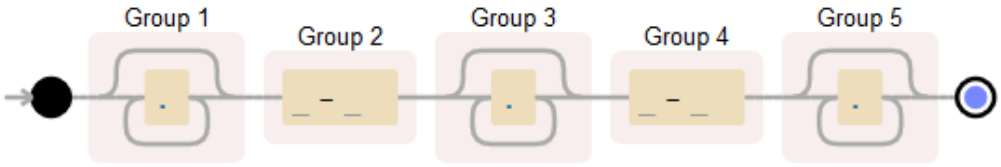
Replace: \3 - \1

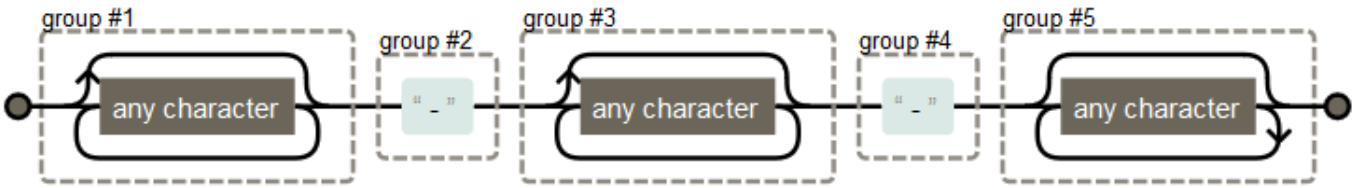
Name	New Name
Numbers 12 - Colors Red - Letters ABCDE.pdf	Colors Red - Numbers 12.pdf

Replace: \5 - \1 - \3

Name	New Name
Numbers 12 - Colors Red - Letters ABCDE.pdf	Letters ABCDE - Numbers 12 - Colors Red.pdf

etc....





Sample:

```
.....................................................
Match 1: Numbers 12 - Colors Red - Letters ABCDE
Group 1: Numbers 12
Group 2:  -
Group 3: Colors Red
Group 4:  -
Group 5: Letters ABCDE
```

Forum Contributors: therube, ajdedo

String Manipulation (Parsing)

Re-arrange String parts cont.

Using: Numbers 12 - Colors Red - Letters ABCDE

Match: (.*?)(-)(.*?)(-)(.*)

Where:
\1 = Numbers 12 \2 = <space> - <space> \3 = Colors Red \4 = <space> - <space> \5 = Letters ABCDE

Tim Analysis:

1..	Match against any character	Capture Group 1 = N
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. <space>	Match against a <space> character	Capture Group 2 = <space>
	This is the <space> after "Numbers'	Changes Capture Group 1 = Numbers
5. <hyphen>	Match against <hyphen) character	Capture Group 2 = <space> <hyphen>
	This is the <hyphen> after '12 <space>	Changes Capture Group 1 = Numbers 12 <space>
6. <space>	Match against a <space> character	Capture Group 2 = <space> <hyphen> <space>
	This is the <space> after 'Numbers –'	Changes Capture Group 1 = Numbers 12
7..	Match against any character	Capture Group 3 = C
8. *	Make it Greedy	Capture Group 3 = Colors Red - Letters ABCDE
9. ?	But not too Greedy	Capture Group 3 = null
	Backtracks to position just before 'C'	
10. <space>	Match against a <space> character	Capture Group 4 = <space>
	This is the <space> after 'Colors'	Changes Capture Group 3 = Colors
11. <hyphen>	Match against <hyphen) character	Capture Group 4 = <space> <hyphen
	This is the <hyphen> after 'Red <space>'	Changes Capture Group 3 = Colors Red
12. <space>	Match against a <space> character	Capture Group 4 = <space> <hyphen> <space>
	This is the <space> after 'Red'	
13. .*	Match against remaining string	Capture Group 5 = Letters ABCDE

String Manipulation (Parsing)

Moving part of a Filename to another location

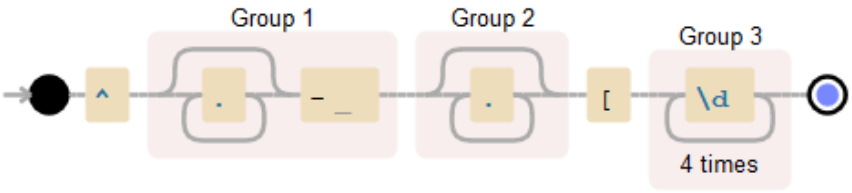
Artist - Title [quality][year] to Artist - year - Title [quality]

i.e.

Aalon - Cream City [320KBPS][1977] to Aalon - 1977 - Cream City [320KBPS]

Match: ^(.*-)(.*)\[(\d{4})
Replace: \1\3 - \2

Name	New Name
Aalon - Cream City [320KBPS][1977]	Aalon - 1977 - Cream City [320KBPS]



Sample:

```
Match 1: Aalon - Cream City [320KBPS][1977
Group 1: Aalon -
Group 2: Cream City [320KBPS]
Group 3: 1977
```

Contributor and *Tim Analysis:*

1. ^ : Denotes the start of the string (Beginning of Line or BOL).
2. (.*-) This is Capture Group 1. The .*- matches any character or <space> any number of times up-to and including the - (<hyphen> followed by a <space>)

3. (.*) This is Capture Group 2. The .* matches any character or <space> any number of times.

4. \[This stops Capture Group 2 from matching anything after and including the first square bracket, this is not captured because it's not needed in the Replace String.

5. (\d{4}) This is Capture Group 3 and matches the date. The \d catches a numeric digit and the {4} tells the RegEx engine to catch the previous character 4 iterations, so \d{4} matches 4 numeric digits.

Where:

\1 = (Artist).
\3 = year followed by a <space> <hyphen> <space>
\2 = Title and Quality.

Basically it's just a matter of parsing the entire file name into reusable groups (Capture Groups are in parentheses and numbered according to their position in the RegEx), then placing those groups in the Replace String in the order you want them. You can place literal characters in-between the groups, for instance the value of Capture 3 is followed by <space> <hyphen> <space> which in turn is followed by the value of Capture Group 2 or .. \3 - \2

Forum Contributor: KenP

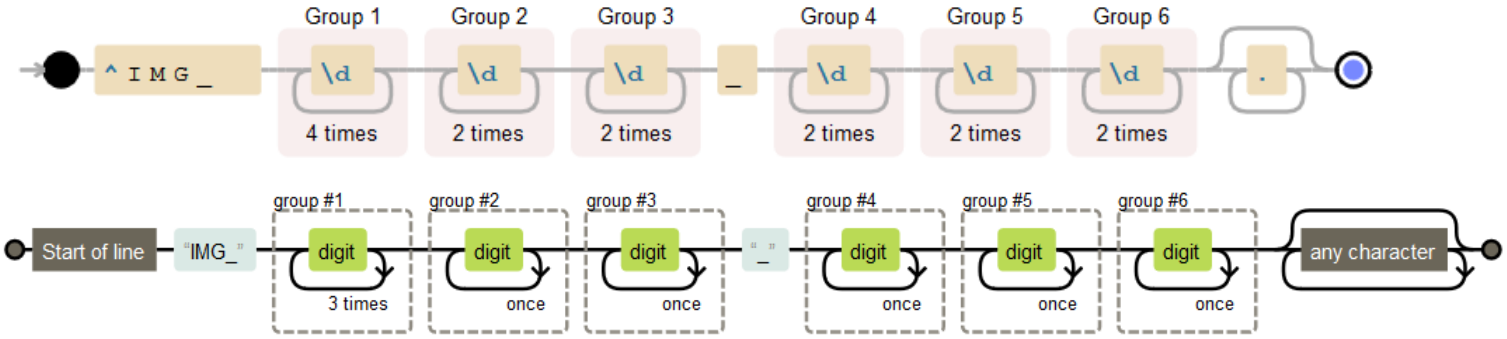
String Manipulation (Parsing)

Rename Camera Image Files

IMG_20181024_134730384_HDR.jpg to 2018-10-24 at 13_47_30.jpg

Match: ^IMG_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2}).*
Replace: \1-\2-\3 at \4_\5_\6

Name	New Name
IMG_20181024_134730384_HDR.jpg	2018-10-24 at 13_47_30.jpg



Sample:

```
.........................................
Match 1: IMG_20181024_134730384_HDR.jpg
Group 1: 2018
Group 2: 10
Group 3: 24
Group 4: 13
Group 5: 47
Group 6: 30
```

Contributor Notes:

1. Careful, this trims everything after the final '_ ' + 6 Numeric Digits so '384_HDR' is removed from '_134730'

Tim Notes:

1. The ^ is an achor and here it is used to indicate the 'Beginning of String' (Beginning of Line or BOL) as opposed to the $ anchor used to indicate the 'End of the String' also called 'EOL'. It forces the evaluation to begin at the first character position.

Forum Contributor: the_Badluck

String Manipulation (Parsing)

Rename Camera Image Files cont.

Using: IMG_20181024_134730384_HDR.jpg

Match: ^IMG_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2}).*

Where:

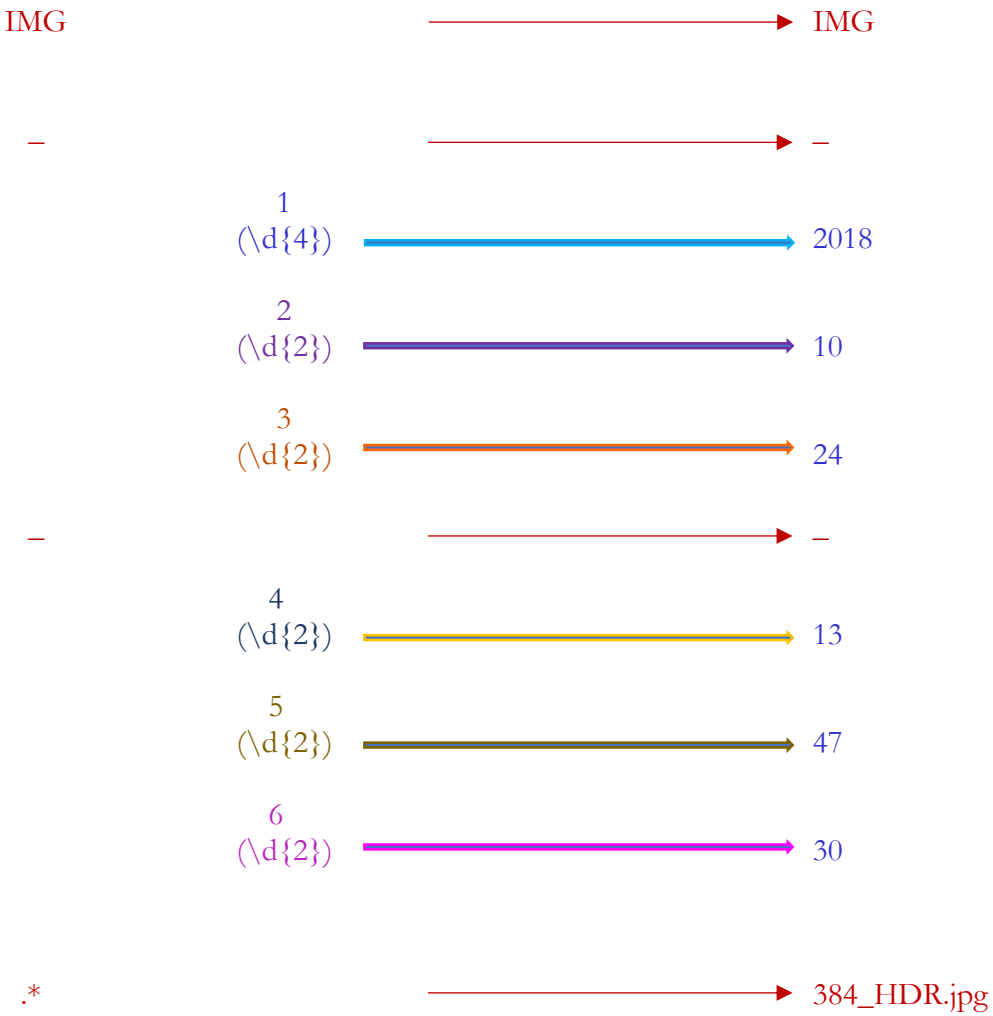
\1 = 2018 \2 = 10 \3 = 24 \4 = 13 \5 = 47 \6 = 30

Breaks down:

Anchor

^

Not Captured Captured



String Manipulation (Parsing)

Rename Camera Image Files cont.

Using: IMG_20181024_134730384_HDR.jpg

Match: ^IMG_(\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2}).*

Where:

\1 = 2018 \2 = 10 \3 = 24 \4 = 13 \5 = 47 \6 = 30

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. IMG	Match against a string 'IMG' (Not Captured) = 'IMG'	
3. _	Match against the underscore character (Not Captured) This is the underscore afte 'IMG'	
4. \d{4}	Match against 4 numeric digits	Capture Group 1 = 2018
5. \d{2}	Match against 2 numeric digits	Capture Group 2 = 10
6. \d{2}	Match against 2 numeric digits	Capture Group 3 = 24
7. _	Match against the underscore character (Not Captured) This is the underscore after '24'	
8. \d{2}	Match against 2 numeric digits	Capture Group 4 = 13
9. \d{2}	Match against 2 numeric digits	Capture Group 5 = 47
10. \d{2}	Match against 2 numeric digits	Capture Group 6 = 30
11. .*	Match against remaining string (Not Captured) = '384_HDR'	

String Manipulation (Parsing)

Switch Artist - Title to Title - Artist

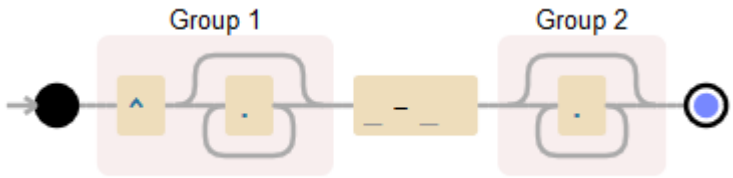
Alanis Morissette - You Oughta Know.mp3 to You Oughta Know - Alanis Morissette.mp3

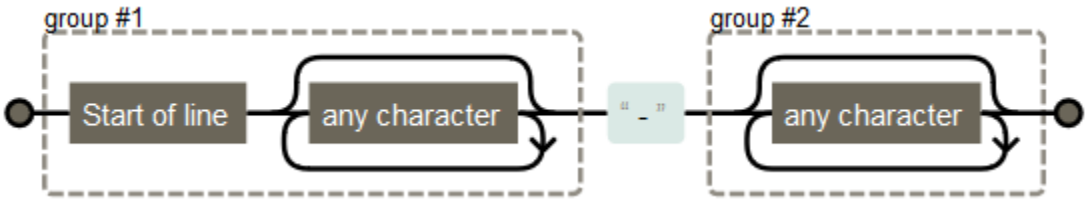
Match: (^.*) - (.*)
Replace: \2 - \1

Name	New Name
Alanis Morissette - You Oughta Know.mp3	You Oughta Know - Alanis Morissette.mp3

Sample:

```
.....................................................
Match 1: Alanis Morissette - You Oughta Know
Group 1: Alanis Morissette
Group 2: You Oughta Know
```





Forum Contributors: Admin, therube

Where:

\1 = Alanis Morissette \2 = You Oughta Know

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = A
3. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'Oughta'	Changes Capture Group 1 = Alanis Morissette - You Oughta
5. <hyphen>	Match against <hyphen) character (Not Captured) This is the <hyphen> after 'Morissette <space>' Changes Not Captured in step 4 to <space> after 'Morissette <space> <hyphen>'	Changes Capture Group 1 = Alanis Morissette <space>
6. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Morissette <space><hyphen>' Changes Not Captured in step 4 to <space> after 'Morissette'	Changes Capture Group 1 = Alanis Morissette
7. .*	Match against remaining string	Capture Group 2 = You Oughta Know

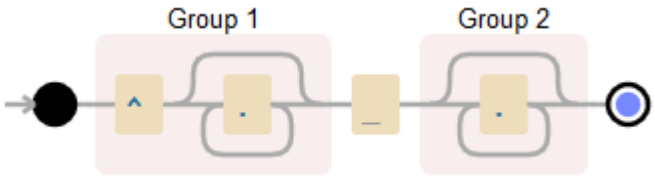
String Manipulation (Parsing)

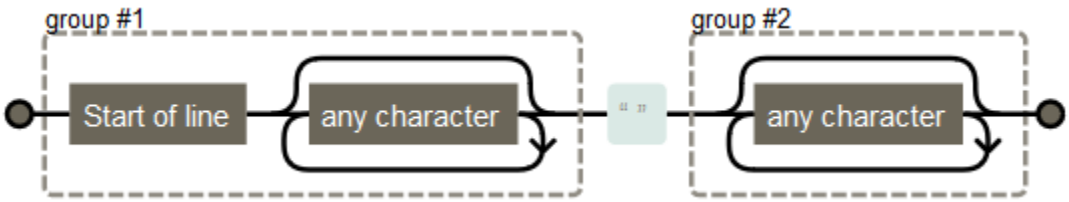
Change around string but add Zeros to single digits

Neon 1-6-2019.mp3 to 2019-01-06_Neon.mp3

Match: (^.*) (.*)
Replace: \2_\1

Name	New Name
Neon 1-6-2019.mp3	1-6-2019 _ Neon.mp3





Sample:

```
.......................
Match 1: Neon 1-6-2019
Group 1: Neon
Group 2: 1-6-2019
```

Contributor Analysis:

1. Find anything up to the (final) <space>, then everything that follows.
2. Reverse the two found items, inserting a _ between

Forum Contributors: Admin, therube

Tim Notes

1. This is not what was wanted. Not even close. The Contributor only parsed and rearranged the string adding an underscore. The poster wanted a zero before all single numeric digits and the year prefixed. The parsing as it stands now, won't allow the zero appended before the '6' and the year can't be prefixed because it is part of Capture Group 2.

   Even if I used a Replace String of - Replace: 0\2_\1 = 01-6-2019
   .. back reference 2 (Capture Group 2) includes the '6' so the zero can't be inserted and the year has not been parsed.

String Manipulation (Parsing)

Change around string but add Zeros to single digits cont.

Using: Neon 1-6-2019.mp3

Match: (^.*) (.*)

Where:

\1 = Neon \2 = 1-6-2019

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = N
3. *	Make it Greedy	Capture Group 1 = <entire string>
	Current position = EOL	
4. <space>	Match against a <space> character (Not Captured)	Changes Capture Group 1 = Neon
	Backtracks. This is the <space> after 'Neon'	
5. .*	Match against remaining string	Capture Group 2 = 1-6-2019

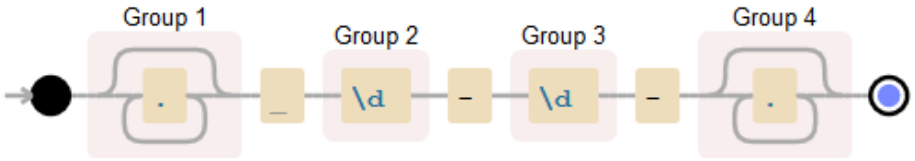
String Manipulation (Parsing)

Change around string but add Zeros to single digits cont.

Tim Contribution – This will:

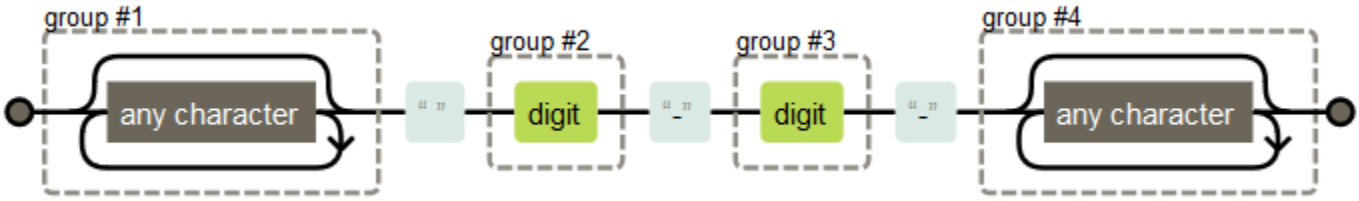
Match: (.*) (\d{1})-(\d{1})-(.*)
Replace: \4-0\2-0\3_\1

Name	New Name
Neon 1-26-2019.mp3	Neon 1-26-2019.mp3
Neon 1-6-2019.mp3	2019-01-06_Neon.mp3



Sample:

```
.....................
Match 1: Neon 1-6-2019
Group 1: Neon
Group 2: 1
Group 3: 6
Group 4: 2019
```



Tim Analysis:

1. Gather up string to first <space>, but do not capture <space>
2. Find a single numeric digit and capture it
3. Find <hyphen> but do not capture it
4. Find a single numeric digit and capture it
5. Find <hyphen> but do not capture it
6. Gather up rest of string and capture it
7. Put together the string in the order wanted including <hyphens>, Zeros and underscore

Capture Group 1 = Neon
Capture Group 2 = 1
Capture Group 3 = 6
Capture Group 4 = 2019

Ignores any that do not fit the RegEx - n<space>#-#-n

Where:

n = string data
# = single numeric digit

Tim Notes:

1. Could have used (\d{4}) instead of (.*) at the end to capture the 2019 if I knew the last 4 numeric digits of the string are always going to be a 4 numeric digit year.

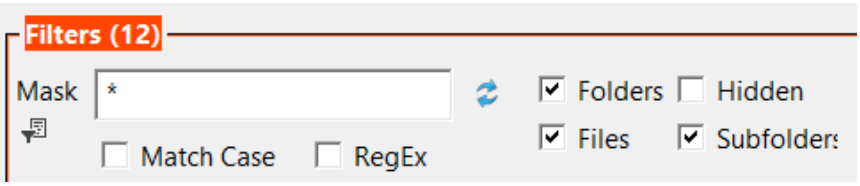
2. The string with '26' is ignored because '26' is two numeric digits not a single numeric digit number, which is what was wanted.

String Manipulation (Parsing)

Change format to both Folders and the files within

id_day_month_year to year_month_day_id

Turn on Recursion: Subfolders: Enable

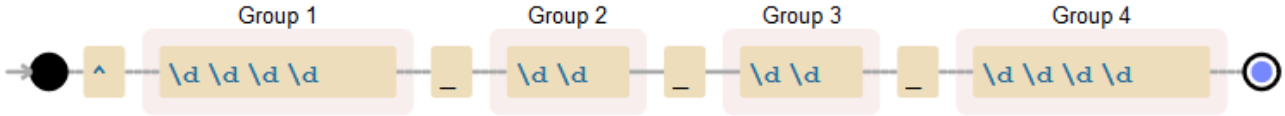


Match: ^(\d\d\d\d)_(\d\d)_(\d\d)_(\d\d\d\d)
Replace: \4_\3_\2_\1

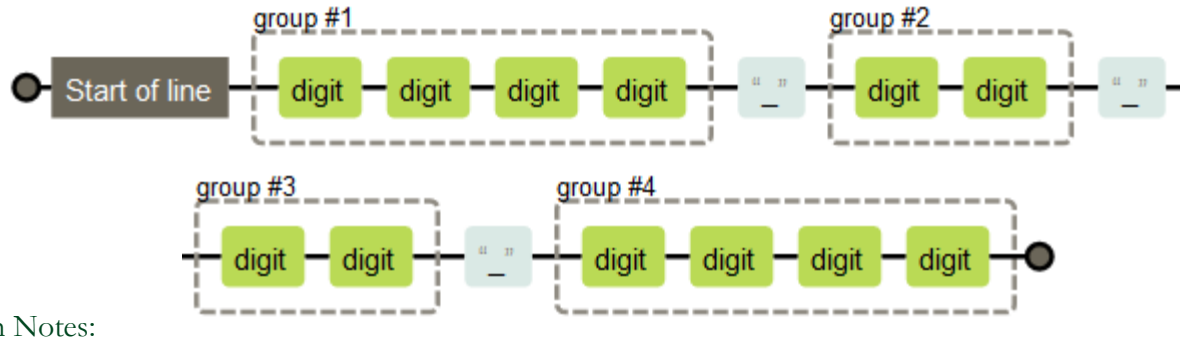
Name	New Name
1862_31_12_2019	2019_12_31_1862
1863_22_10_2019	2019_10_22_1863
1862_31_12_2019.pdf	2019_12_31_1862
1862_31_12_2019.txt	2019_12_31_1862
1863_22_10_2019.pdf	2019_10_22_1863
1863_22_10_2019.txt	2019_10_22_1863

Sample:

```
Match 1: 1862_31_12_2019
Group 1: 1862
Group 2: 31
Group 3: 12
Group 4: 2019
```



Expanded:



Tim Notes:

1. If you only want to affect the folders then change Match to : ^(\d\d\d\d)_(\d\d)_(\d\d)_(\d\d\d\d)$

Forum Contributor: therube

String Manipulation (Parsing)

Change format to both Folders and the files within cont.

Sample base:

1862_31_12_2019 1862_31_12_2019.pdf 1862_31_12_2019.txt 1863_22_10_2019
1863_22_10_2019.pdf 1863_22_10_2019.txt

Using: 1862_31_12_2019

Match: ^(\d\d\d\d)_(\d\d)_(\d\d)_(\d\d\d\d)

First let's simplify this:

Match: ^ (\d {4})_(\d {2})_(\d{2})_(\d{4})

Where:

\1 = 1862 \2 = 31 \3 = 12 \4 = 2019

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. \d {4}	Match against 4 numeric digits	Capture Group 1 = 1862
3. _	Match against underscore (Not Captured)	
4. \d {2}	Match against 2 numeric digits	Capture Group 2 = 31
5. _	Match against underscore (Not Captured)	
6. \d {2}	Match against 2 numeric digits	Capture Group 3 = 12
7. _	Match against underscore (Not Captured)	
8. \d {2}	Match against 2 numeric digits	Capture Group 4 = 2019

String Manipulation (Parsing)

Place parentheses around year in string

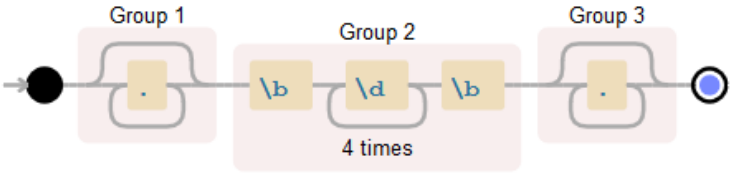
Alien Directors Cut 1979 1080p.mkv	to	Alien Directors Cut (1979) 1080p.mkv
Alien Directors Cut 1979.mkv	to	Alien Directors Cut (1979).mkv
Alien Directors Cut 1979 720p.mkv	to	Alien Directors Cut (1979) 720p.mkv

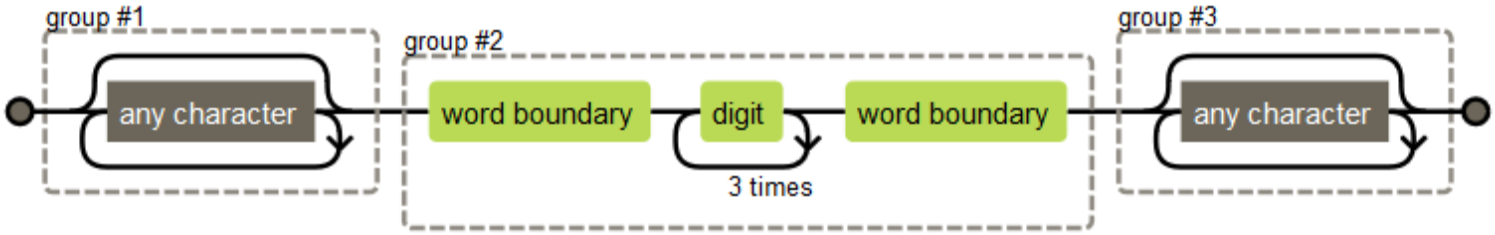
Match: (.*)(\b\d{4}\b)(.*)
Replace: \1(\2)\3

Name	New Name
Alien Directors Cut 1979 1080p.mkv	Directors Cut (1979) 1080p.mkv
Alien Directors Cut 1979 720p.mkv	Directors Cut (1979) 720p.mkv
Alien Directors Cut 1979.mkv	Directors Cut (1979).mkv

Sample:

```
Match 1: Alien Directors Cut 1979 1080p
Group 1: Alien Directors Cut
Group 2: 1979
Group 3:  1080p
```





Forum Contributors: therube, @trekerie

Where:
\1 = Alien Director's Cut<space> \2 = 1979 \3 = <space> 1080p

Tim Analysis using first sample:

1. .*	Matches all characters until….	Capture Group 1 = Alien Director's Cut<space>
2. \b\d{4}\b	.. a word boundary inclusive of the 4 numeric digit Year	Capture Group 2 = 1979
3. .*	Capture remainder of string	Capture Group 3 = <space> 1080p

Tim Notes:

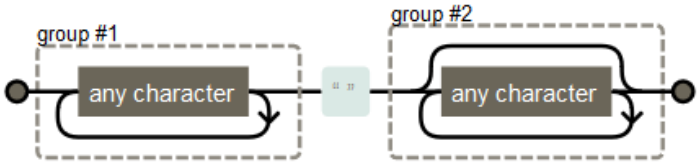
1. The Word Boundary, as defined here, includes the characters following the <space> after Director's Cut, inclusive of the Year and ending the boundary. Here it only serves to capture the Year, but because it is a word boundary it does not include <spaces>. If a quantifier was used instead, it would pick up the <space> and possibly more characters than what was required. By using Word boundaries you limit the match.

String Manipulation (Parsing)

Place parentheses around year in string cont.

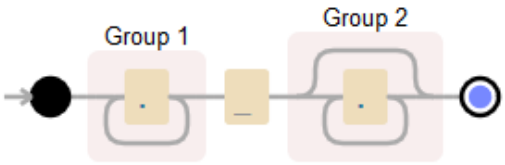
Tim Contribution –

Variations: Enclose last string portion in parentheses



Match: (.+) (.*)
Replace: \1 (\2)

Name	New Name
Alien Directors Cut 1979 1080p.mkv	Alien Directors Cut 1979 (1080p).mkv
Alien Directors Cut 1979 720p.mkv	Alien Directors Cut 1979 (720p).mkv
Alien Directors Cut 1979.mkv	Alien Directors Cut (1979).mkv



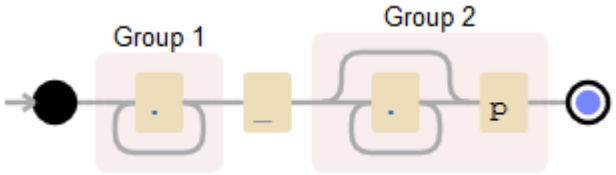
Sample:

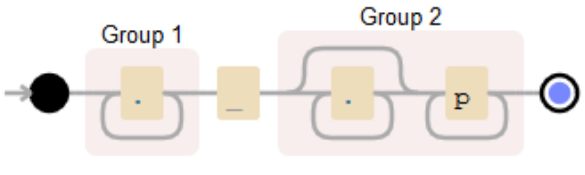
```
Match 1: Alien Directors Cut 1979 1080p
Group 1: Alien Directors Cut 1979
Group 2: 1080p
```

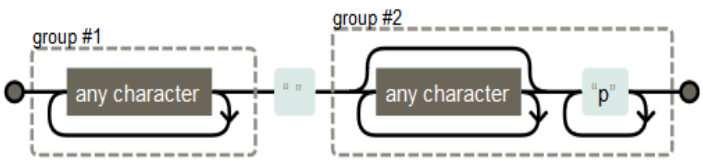
Variations: Enclose only those strings that contain a 'p' in the final string (movie format – ignore others)

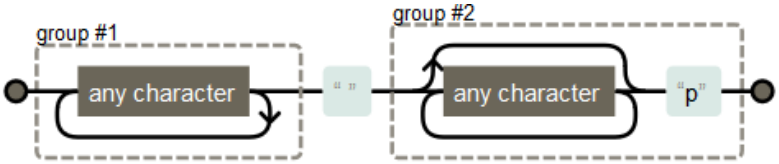
Match: (.+) (.*p+) or more restrictive Match: (.+) (.*?p)
Replace: \1 (\2) Replace: \1 (\2)

Name	New Name
Alien Directors Cut 1979 1080p.mkv	Alien Directors Cut 1979 (1080p).mkv
Alien Directors Cut 1979 720p.mkv	Alien Directors Cut 1979 (720p).mkv
Alien Directors Cut 1979.mkv	Alien Directors Cut 1979.mkv









Sample:

```
Match 1: Alien Directors Cut 1979 1080p
Group 1: Alien Directors Cut 1979
Group 2: 1080p
```

Sample:

```
Match 1: Alien Directors Cut 1979 1080p
Group 1: Alien Directors Cut 1979
Group 2: 1080p
```

String Manipulation (Parsing)

Place parentheses around year in string cont.

Tim Contribution – cont.

Using: Alien Directors Cut 1979 1080p.mkv

Variations: Enclose last string portion in parentheses cont.

Match: (.+) (.*)

Where:

\1 = Alien Directors Cut 1979 \2 = 1080p

Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after '1979'	Changes Capture Group 1 = Alien Directors Cut 1979
4. .*	Match against remaining string	Capture Group 2 = 1080p

Variations: Enclose only those strings that contain a 'p' in the final string (movie format – ignore others) cont.

Match: (.+) (.*p+)

Where:

\1 = Alien Directors Cut 1979 \2 = 1080p

Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after '1979'	Changes Capture Group 1 = Alien Directors Cut 1979
4. .	Match against any character	Capture Group 2 = 1
5. *	Make it Greedy	Capture Group 2 = 1080p
6. p	Match against string 'p'	Capture Group 2 = Unchanged
7. +	Make it Greedy	Capture Group 2 = Unchanged

String Manipulation (Parsing)

Place parentheses around year in string cont.

Tim Contribution – cont.

Using: Alien Directors Cut 1979 1080p.mkv

Variations: Enclose only those strings that contain a 'p' in the final string (movie format – ignore others) cont.

Match: (.+) (.*?p)

Where:

\1 = Alien Directors Cut 1979 \2 = 1080p

Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after '1979'	Changes Capture Group 1 = Alien Directors Cut 1979
4. .	Match against any character	Capture Group 2 = 1
5. *	Make it Greedy	Capture Group 2 = 1080p
6. ?	But not too Greedy Positions just before the '1080p'	Capture Group 2 = null
7. p	Match against string 'p'	Capture Group 2 = 1080p

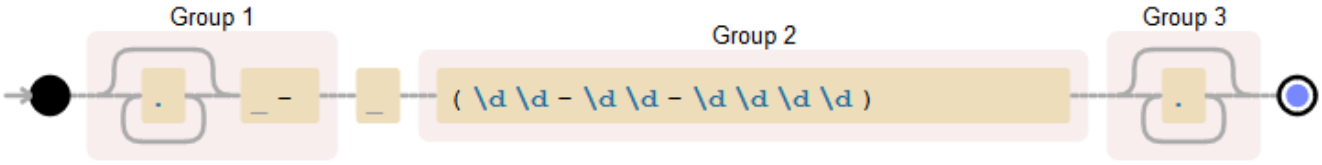
String Manipulation (Parsing)

Within string - Title first , date second

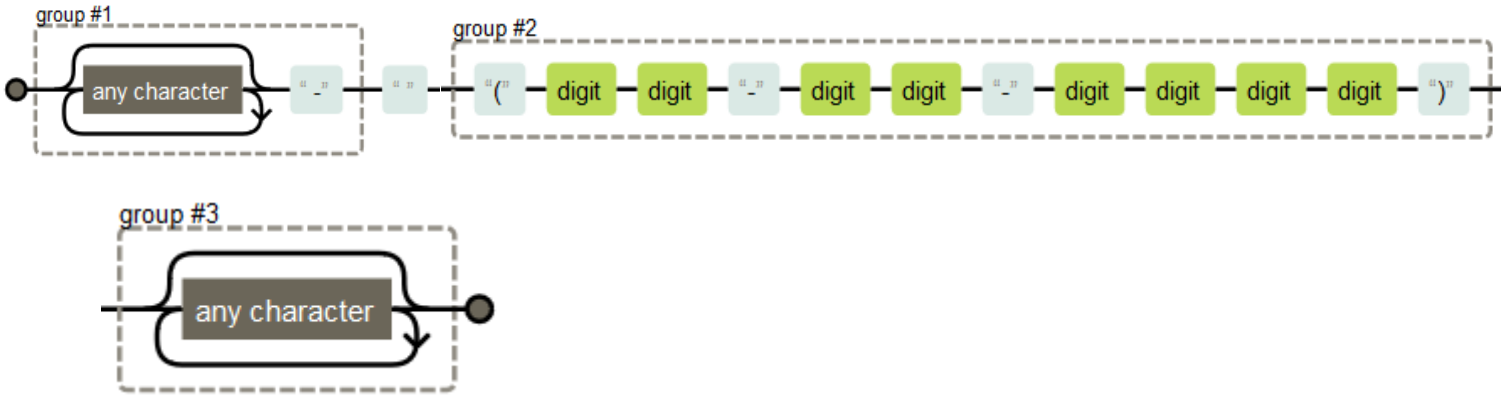
s01e01 - (10-02-1959) Where Is Everybody.avi to s01e01 - Where Is Everybody (10-02-1959).avi
s01e02 - (10-09-1959) One For The Angels.avi to s01e02 - One For The Angels (10-09-1959).avi
s01e04 - (10-23-1959) Sixteen Millimeter Shrine.avi to s01e04 - Sixteen Millimeter Shrine (10-23-1959).avi

Match: (.* -) (\(\d\d-\d\d-\d\d\d\d\))(.*)
Replace: \1\3 \2

Name	New Name
s01e01 - (10-02-1959) Where Is Everybody.avi	s01e01 - Where Is Everybody (10-02-1959).avi
s01e02 - (10-09-1959) One For The Angels.avi	s01e02 - One For The Angels (10-09-1959).avi
s01e04 - (10-23-1959) Sixteen Millimeter Shrine.avi	s01e04 - Sixteen Millimeter Shrine (10-23-1959).avi



Expanded:



Sample:

```
.................................................
Match 1: s01e01 - (10-02-1959) Where Is Everybody
Group 1: s01e01 -
Group 2: (10-02-1959)
Group 3:  Where Is Everybody
```

Forum Contributor: therube

String Manipulation (Parsing)

Within string - Title first , date second cont.

Using: s01e01 - (10-02-1959) Where Is Everybody.avi

Match: (.* -) (\(\d\d-\d\d-\d\d\d\d\))(.*)

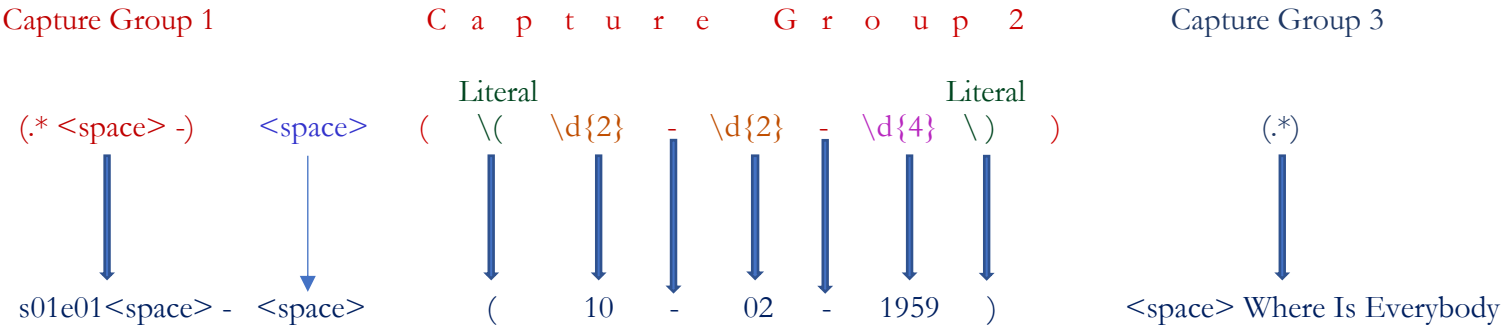
Where:

\1 = s01e01 - \2 = (10-02-1959) \3 = <space> Where is Everybody

First, let's simplify it:

Match: (.* -) (\(\d{2}-\d{2}-\d{4}\))(.*)

Breaks down:



Tim Analysis:

1..	Match against any character	Capture Group 1 = s
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character Backtracks. This is the <space> after 'Is'	Capture Group 1 = s01e01 - (10-02-1959) Where Is
4. -	Match against the <hyphen>	Capture Group 1 = s01e01 - (10-02-
5. <space>	Match against a <space> character (Not Captured) This is the <space> after '1959)'	Changes Capture Group 1 = s01e01 - (10-02-1959)
6. \(	Match against left parenthesis	Capture Group 2 = (Changes Capture Group 1 = s01e01 -<space>
7. \d{2}	Match against 2 numeric digits	Capture Group 2 = (10
8. -	Match against the <hyphen>	Capture Group 2 = (10-
9. \d{2}	Match against 2 numeric digits	Capture Group 2 = (10-02
10. -	Match against the <hyphen>	Capture Group 2 = (10-02-
11. \d {4}	Match against 4 numeric digits	Capture Group 2 = (10-02-1959
12. \)	Match against right parenthesis	Capture Group 2 = (10-02-1959)
13. .*	Match against remaining string	Capture Group 3 = <space> Where Is Everybody

String Manipulation (Parsing)

Tim Contribution –

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name

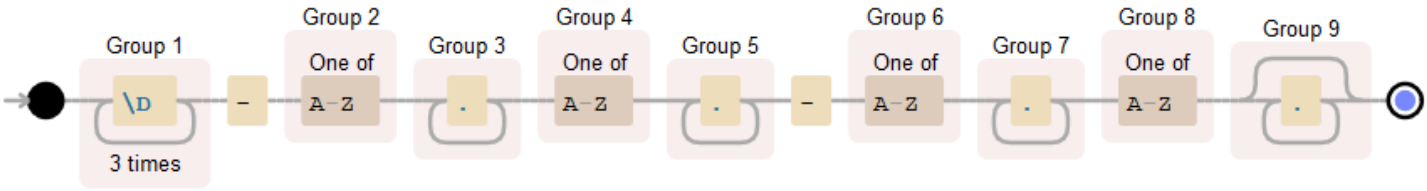
A challenge. Originally solved using Powerscript by therube. However since this was in the BRU forum, I decided to take it on.

For two names:

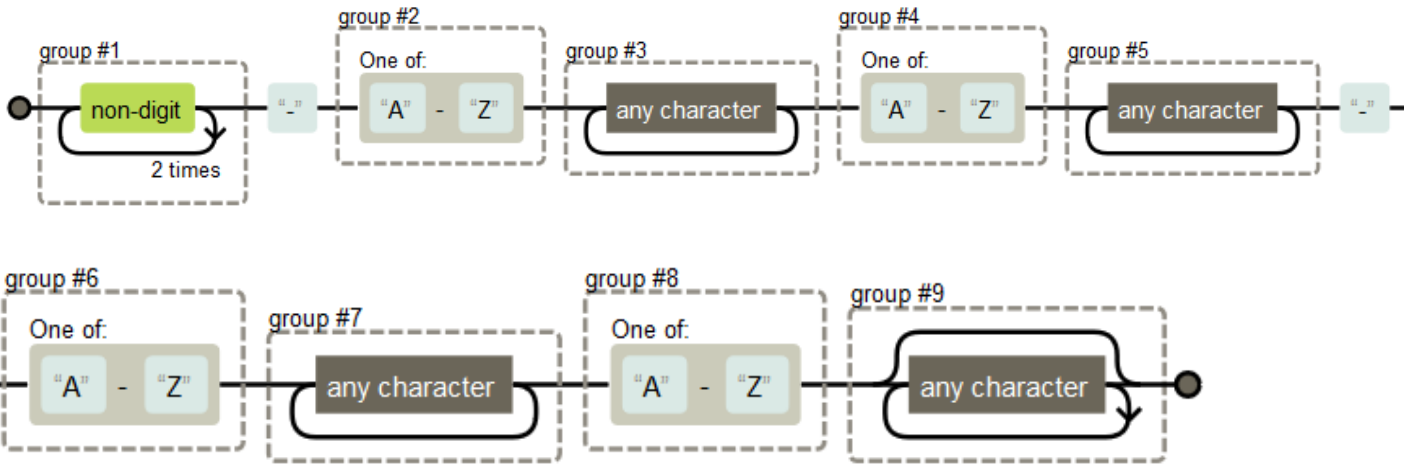
GER-MartinSchmidt-NinaWagner to GER - Martin Schmidt and Nina Wagner

Match: (\D{3})-([A-Z])(.+?)([A-Z])(.+?)-([A-Z])(.+?)([A-Z])(.*)
Replace: \1 - \2\3 \4\5 and \6\7 \8\9

Name	New Name
GER-MartinSchmidt-NinaWagner	GER - Martin Schmidt and Nina Wagner



Expanded -



Tim Notes:

1. The reason I also do an expanded version is because I feel it is important to see the entire sequence then show the enlarged version to study properly.

String Manipulation (Parsing)

Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For two names: cont.

Sample:

```
.....................................
Match 1: GER-MartinSchmidt-NinaWagner
Group 1: GER
Group 2: M
Group 3: a
Group 4: r
Group 5: tinSchmidt
Group 6: N
Group 7: i
Group 8: n
Group 9: aWagner
```

Tim Notes:

1. Another one I use for tutoring where all of the Capture Groups have been used up to illustrate parsing.

Where:

\1 = GER
\2 = M
\3 = artin
\4 = S
\5 = chmidt
\6 = N
\7 = ina
\8 = W
\9 = agner

String Manipulation (Parsing)

Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For two names: cont.

Using: GER-MartinSchmidt-NinaWagner

Match: (\D{3})-([A-Z])(.+?)([A-Z])(.+?)-([A-Z])(.+?)([A-Z])(.*)

Where:

\1 = GER	\2 = M	\3 = artin	\4 = S	\5 = chmidt	\6 = N
\7 = ina	\8 = W	\9 = agner			

Tim Analysis:

1. \D	Match against Non-Numeric Digit	Capture Group 1 = G
2. {3}	Repeat last match 3 iterations (\D)	Capture Group 1 = GER
3. -	Match against the <hyphen> (Not Captured) This is the <hyphen> following 'GER'	
4. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = M
5. .	Match against any character	Capture Group 3 = a
6. +	Make it Greedy	Capture Group 3 = artinSchmidt-NinaWagner
7. ?	But not too Greedy	Capture Group 3 = a
8. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 4 = S Changes Capture Group 3 = artin
9. .	Match against any character	Capture Group 5 = c
10. +	Make it Greedy	Capture Group 5 = chmidt-NinaWagner
11. ?	But not too Greedy	Capture Group 5 = c
12. -	Match against the <hyphen> (Not Captured) This is the <hyphen> following 'Schmidt'	Changes Capture Group 5 = chmidt
8. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 6 = N
9. .	Match against any character	Capture Group 7 = i
10. +	Make it Greedy	Capture Group 7 = ina-Wagner
11. ?	But not too Greedy	Capture Group 7 = i
12. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 8 = W Changes Capture Group 7 = ina
13. .*	Match against remaining string	Capture Group 9 = agner

String Manipulation (Parsing)

Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For three names:

UK-PeterFord-JeremyJordan-LisaWendt to UK - Peter Ford, Jeremy Jordan and Lisa Wendt

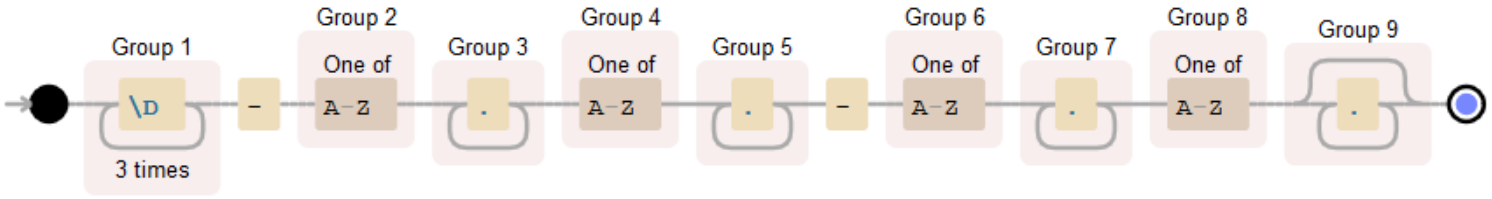
This one I will use to illustrate the methodology of problem solving (trial and error☺). The final result will be more efficient using only 7 Capture Groups.

This one was harder dealing with three names – please not a fourth.

Using what I learned in the 'Two Names':

Match: (\D{3})-([A-Z])(.+?)([A-Z])(.+?)-([A-Z])(.+?)([A-Z])(.*)
Replace: \1 - \2\3 \4\5 and \6\7 \8\9

Name	New Name
UK-PeterFord-JeremyJordan-LisaWendt	ord - Jeremy Jordan and Lisa Wendt



Sample:

```
.....................................
Match 1: ord-JeremyJordan-LisaWendt
Group 1: ord
Group 2: J
Group 3: e
Group 4: r
Group 5: emyJordan
Group 6: L
Group 7: i
Group 8: s
Group 9: aWendt
```

Well, it looks like the RegEx needs a bit of tweaking.

Tim Notes:

1. This is the same RegEx used for 'Two Names'. I just wanted to see how it acted on 'Three names' to give me a starting point.

String Manipulation (Parsing)

Tim Contribution – cont.

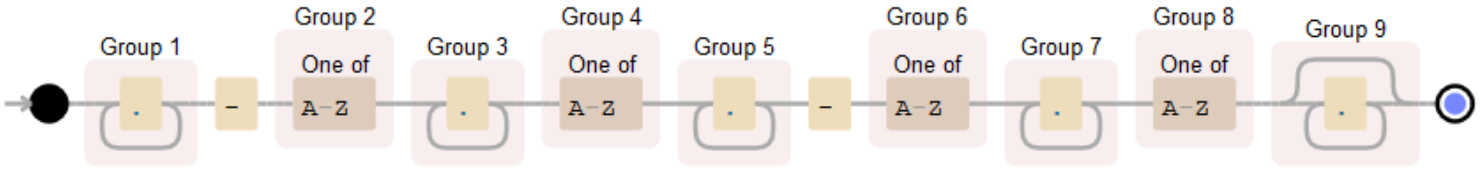
Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For three names: cont.

Using: UK-PeterFord-JeremyJordan-LisaWendt

Match: (.+)-([A-Z])(.+?)([A-Z])(.+?)-([A-Z])(.+?)([A-Z])(.*)
Replace: \1 - \2\3 \4\5 and \6\7 \8\9

```
UK-PeterFord - Jeremy Jordan and Lisa Wendt
```



Sample:

```
........................................
Match 1: UK-PeterFord-JeremyJordan-LisaWendt
Group 1: UK-PeterFord
Group 2: J
Group 3: e
Group 4: r
Group 5: emyJordan
Group 6: L
Group 7: i
Group 8: s
Group 9: aWendt
```

This is not going to work. I ran out of Capture Groups. I need to redo. Instead of having each Capture Group capture an initial and the rest of the name, I need each Capture Group to capture a full part of each name.

String Manipulation (Parsing)

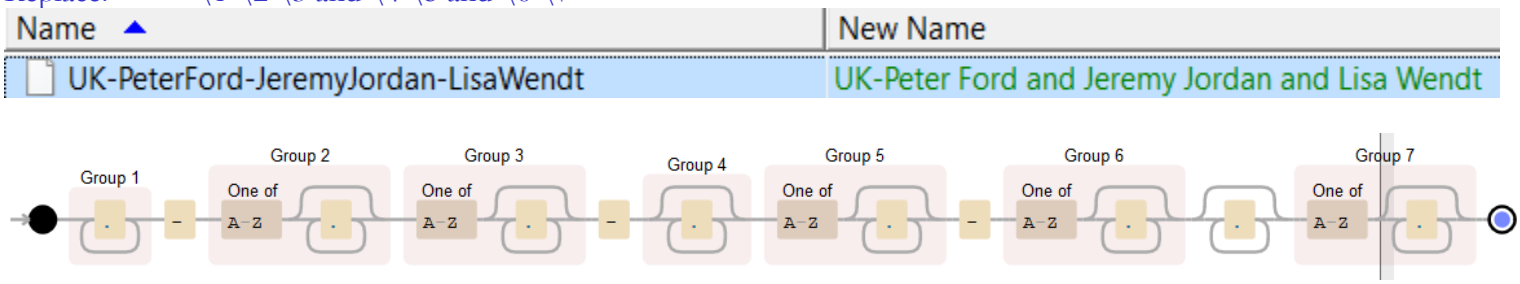
Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

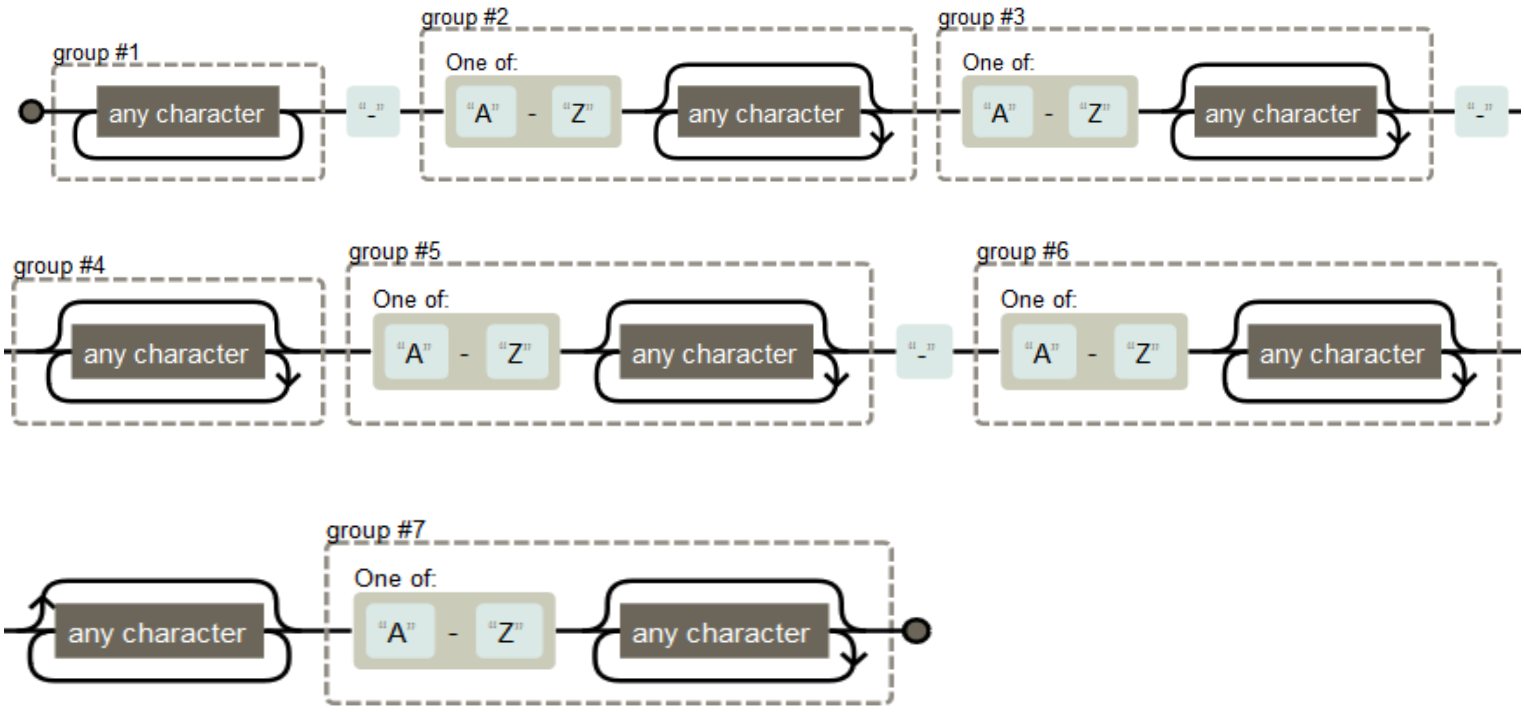
For three names: cont.

UK-PeterFord-JeremyJordan-LisaWendt

Match: (.+?)-([A-Z].*)([A-Z].*)-(.*)([A-Z].*)-([A-Z].*).*?([A-Z].*)
Replace: \1-\2 \3 and \4 \5 and \6 \7



Expanded:



String Manipulation (Parsing)

Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For three names: cont.

UK-PeterFord-JeremyJordan-LisaWendt

Match: (.+?)-([A-Z].*)([A-Z].*)-(.*)([A-Z].*)-([A-Z].*).*?([A-Z].*)

Sample:

```
.........................................
Match 1: UK-PeterFord-JeremyJordan-LisaWendt
Group 1: UK
Group 2: PeterFor
Group 3: d
Group 4: JeremyJorda
Group 5: n
Group 6: LisaWend
Group 7: t
```

Got It! ---- This one is more universal and will work with any Country's abbreviation – i.e., USA

Where:

\1 = UK
\2 = Peter
\3 = Ford
\4 = Jeremy
\5 = Jordan
\6 = Lisa
\7 = Wendt

String Manipulation (Parsing)

Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For three names: cont.

Using: UK-PeterFord-JeremyJordan-LisaWendt

Match: (.+?)-([A-Z].*)([A-Z].*)-(.*)([A-Z].*)-([A-Z].*).*?([A-Z].*)

Where:

\1 = UK \2 = Peter \3 = Ford \4 = Jeremy \5 = Jordan
\6 = Lisa \7 = Wendt

Tim Analysis:

1. .	Match against any character	Capture Group 1 = U
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = Beginning of string
4. -	Match against the <hyphen> (Not Captured) This is the <hyphen> following 'UK'	Changes Capture Group 1= UK
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = P
6. .	Match against any character	Capture Group 2 = Pe
7. *	Make it Greedy	Capture Group 2 = PeterFord-JeremyJordan-LisaWendt
8. [A-Z]	Match against a class consisting of an uppercase letter Backtracking	Capture Group 3 = W Changes Capture Group 2 = PeterFord-JeremyJordan-Lisa
9. .	Match against any character	Capture Group 3 = We
10. *	Make it Greedy	Capture Group 3 = Wendt
11. -	Match against the <hyphen> (Not Captured) This is the <hyphen> following 'Jordan'	Changes Capture Group 3= Jordan Changes Capture Group 2 = PeterFord-Jeremy
12. .	Match against any character	Capture Group 4 = L
13. *	Make it Greedy	Capture Group 4 = LisaWendt
14. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 5 = W Changes Capture Group 4 = Lisa
15. .	Match against any character	Capture Group 5 = We
16. *	Make it Greedy	Capture Group 5 =Wendt

Tim Notes:

1. Wherever mentioned, beginning of string refers to Beginning of Line or BOL.

String Manipulation (Parsing)

Tim Contribution – cont.

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

For three names: cont.

Using: UK-PeterFord-JeremyJordan-LisaWendt

Match: (.+?)-([A-Z].*)([A-Z].*)-(.*)([A-Z].*)-([A-Z].*).*?([A-Z].*)

Where:

\1 = UK \2 = Peter \3 = Ford \4 = Jeremy \5 = Jordan
\6 = Lisa \7 = Wendt

Tim Analysis: cont.

17. -	Match against the <hyphen> (Not Captured) This is the <hyphen> following 'Jordan' Changes the <hyphen> in step 11 to the <hyphen> following 'Ford'	Changes Capture Group 5= Jordan Changes Capture Group 4 = Jeremy Changes Capture Group 3 = Ford Changes Capture Group 2 = Peter Capture Group 1 = Unchanged = UK
18. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 6 = L All other Groups = Unchanged
15. .	Match against any character	Capture Group 6 = Li
16. *	Make it Greedy	Capture Group 6 = LisaWendt
17. .	Match against any character (Not Captured) = 't' Backtracking	Changes Group 6 = LisaWend All other Groups = Unchanged
18. *	Make it Greedy (Not Captured) = EOL	Changes Capture Group 6 = LisaWendt
19. ?	But not too Greedy (Not Captured) = EOL – no change	
20. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 7 = W Changes Capture Group 6 = Lisa All other Groups = Unchanged
21. .	Match against any character	Capture Group 7 = We
22. *	Make it Greedy	Capture Group 7 = Wendt

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

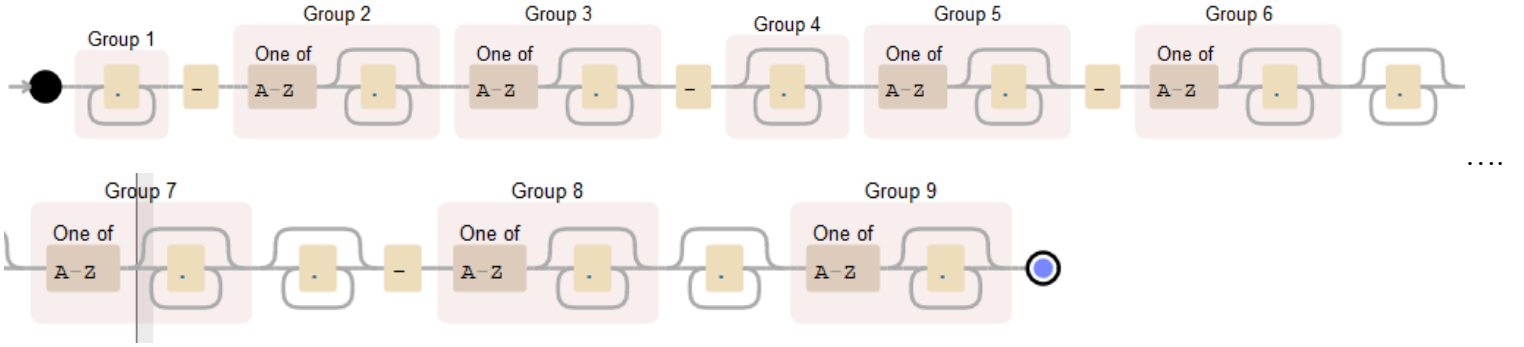
In case You are wondering, here is the RegEx to handle a fourth name:

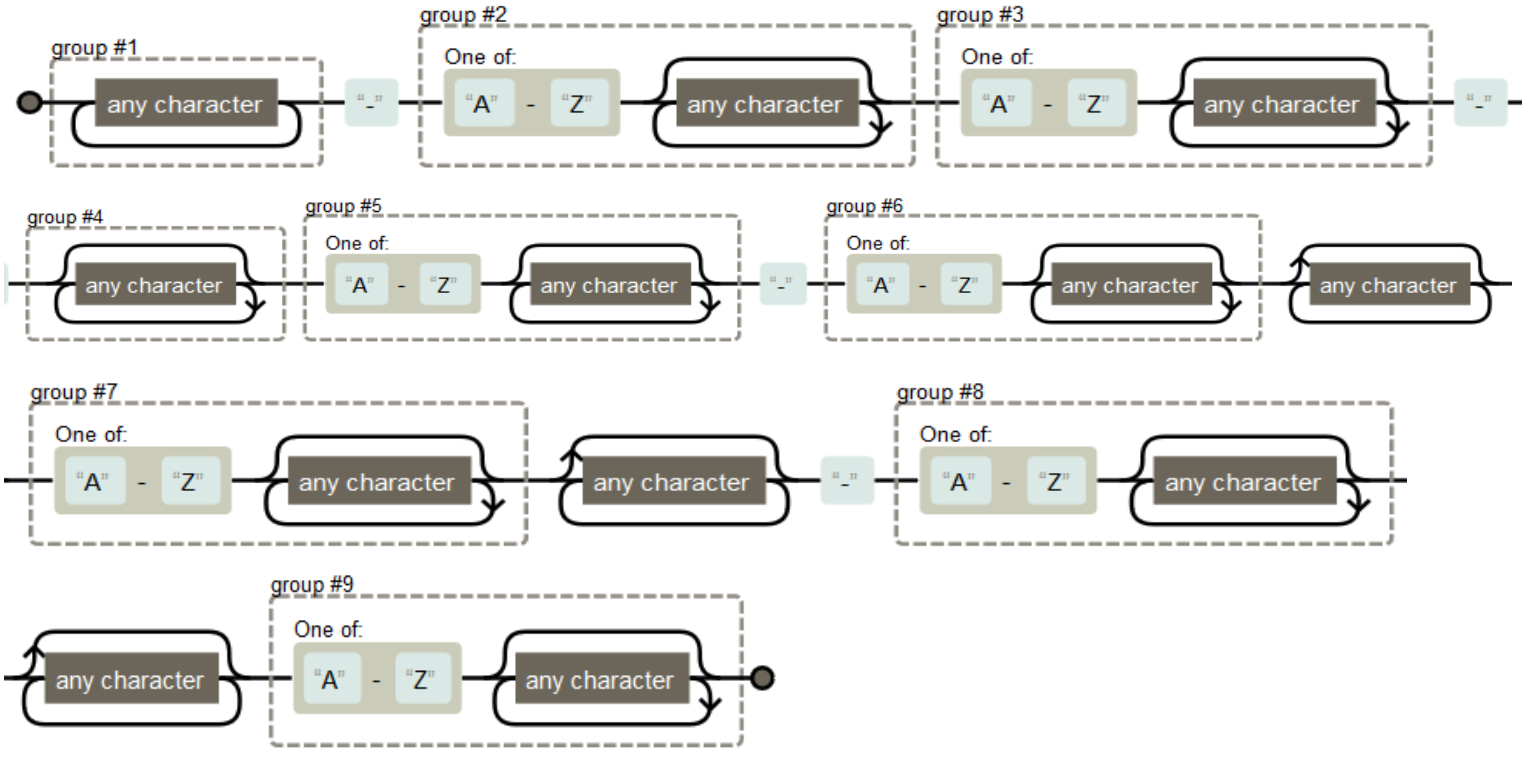
String = UK-PeterFord-JeremyJordan-LisaWendt-RogerSterling

Match: (.+?)-([A-Z].*)([A-Z].*)-(.*)([A-Z].*)-([A-Z].*).*?([A-Z].*).*?-([A-Z].*).*?([A-Z].*)
Replace: \1-\2 \3 and \4 \5 and \6 \7 and \8 \9

Name	New Name
UK-PeterFord-JeremyJordan-LisaWendt-RogerSterling	UK-Peter Ford and Jeremy Jordan and Lisa Wendt and Roger Sterling

Expanded:





String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

In case You are wondering, here is the RegEx to handle a fourth name: cont.

String = UK-PeterFord-JeremyJordan-LisaWendt-RogerSterling

Match: (.+?)-([A-Z].*)([A-Z].*)-(.*)([A-Z].*)-([A-Z].*).*?([A-Z].*).*?-([A-Z].*).*?([A-Z].*)

Sample:

```
.....................................................
Match 1: UK-PeterFord-JeremyJordan-LisaWendt-RogerSterling
Group 1: UK
Group 2: PeterFor
Group 3: d
Group 4: JeremyJorda
Group 5: n
Group 6: LisaWend
Group 7: t
Group 8: RogerSterlin
Group 9: g
```

Where:

\1 = UK
\2 = Peter
\3 =Ford
\4 =Jeremy
\5 = Jordan
\6 = Lisa
\7 = Wendt
\8 = Roger
\9 = Sterling

4 is my maximum. No more Capture Groups.

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Another Contributor threw his hat into the ring with this BRC equivalent:

Here's a batch using Brc32 to handle anywhere from 1-10 names!

CAN-FirstLast.docx	to	CAN - First Last.docx
CAN-First Last.docx	to	Ignore
CAN - FirstLast.docx	to	Ignore
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	to	UK - Jo Lii1, Jo Lii2, Jo Lii3, Jo Lii4, Jo Lii5 and Jo Lii6bru

Observations:

1. Remove /Execute to preview[3] renames, & edit line2 with a folder-path:
2. Ignores already-renamed files (or any file with <spaces>).
3. No need to move them after renaming

Batch file:

```
@echo off
cd/d "H:\A Test Folder"
set reg1=/Regexp:"^([A-Z]{2,})-([A-Z][^A-Z -]*?)([A-Z].*)$:\1}\2 \3"
set reg2=/Regexp:"^(.*[}][^-]*)-([A-Z][^A-Z -]*?)([A-Z].*)$:\1, \2 \3"
set max10=%reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2%
set reg3=/Regexp:"(.*[}].*), (.*):\1 and \2"
set reg4=/Regexp:"(.*)[}](.*):\1 - \2"
brc32 /Pattern:*-*.docx %reg1% %max10% %reg3% %reg4% /Execute
pause>nul
```

Forum Contributor: bru

Tim Analysis: General Analysis of the Batch file:

1. cd	Change to the folder containing the files to rename
2. set	Assign a RegEx to each environmental variable identified as: reg1, reg2, max10, reg3 and reg4
3. brc32	Executes the Brc program
4. /Pattern: *.*.docx	Process all Word docx files found in directory
5. %regx%	Execute all RegEx in order specified
6. /Execute	This parameter executes the Renaming Process. Leaving it off will provide a text display of which files are renamed and which are not - the Preview Mode.

Tim Notes:

1. reg2 is executed for each possible name found in string – up to 10 as defined in max10
2. Each RegEx adds the Replace string at the end
3. You can always send the preview to a text-file before using /execute by appending, > preview.txt to the end.

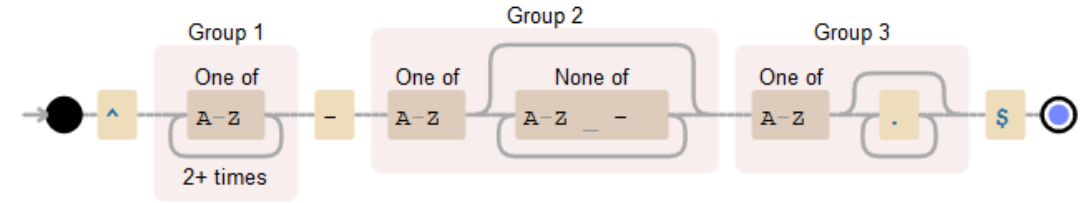
String Manipulation (Parsing)

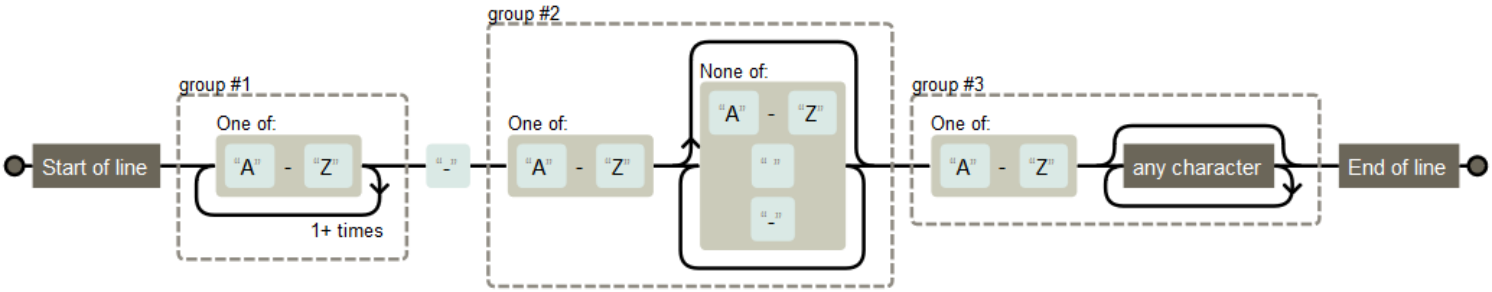
Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Tim Analysis:

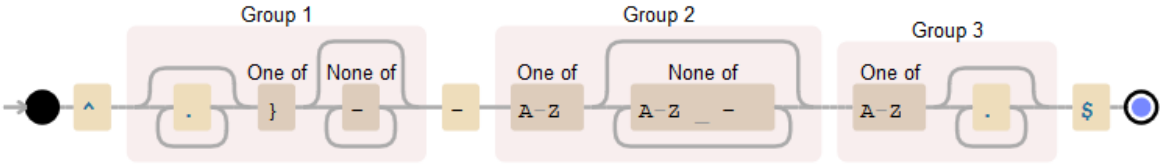
The RegEx breaks down as:

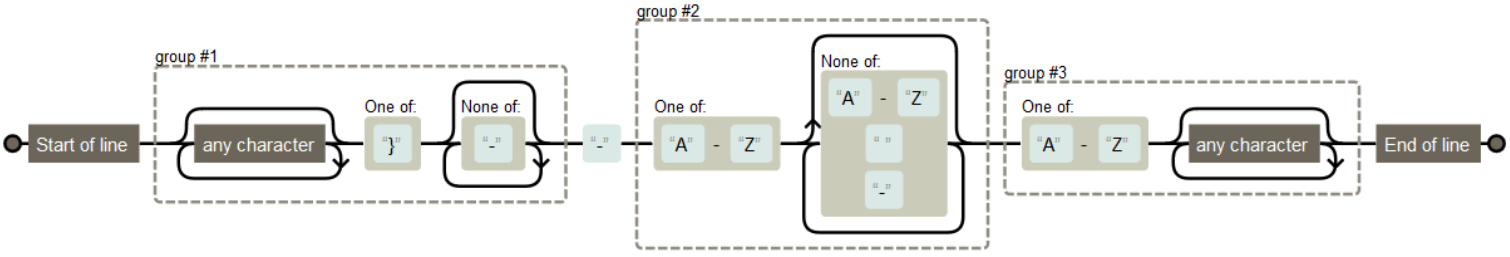
reg1:
Match: ^([A-Z]{2,})-([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1}\2 \3





reg2:
Match: ^(.*[}][^-]*)-([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1, \2 \3





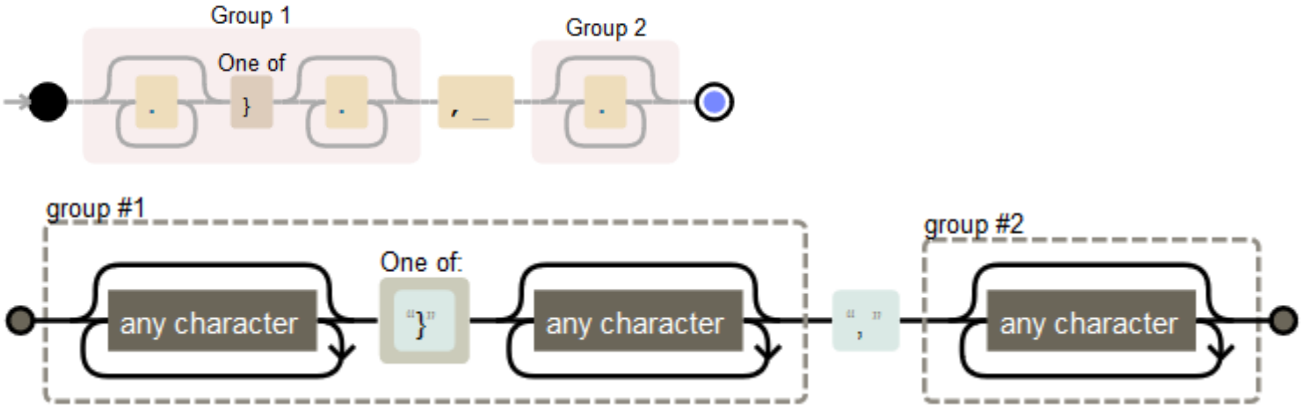
String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

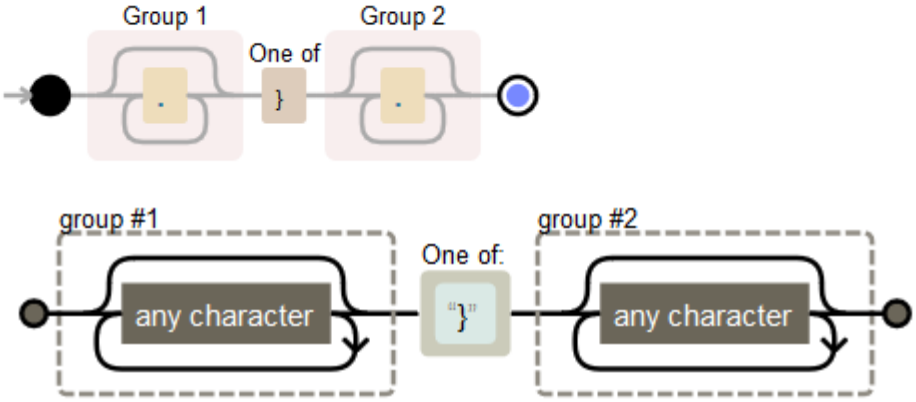
Tim Analysis: cont.

The RegEx breaks down as: cont.

reg3:
Match: (.*[}].*), (.*)
Replace: \1 and \2



reg4:
Match: (.*)[}](.*)
Replace: \1 - \2



String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Tim Analysis: cont.

Observations:

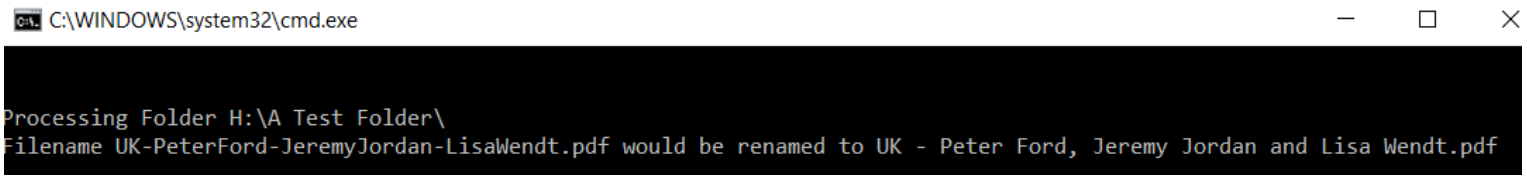
1. reg1 executes first followed by max10, reg3 and reg4.

2. Max10 will act sort of like a GOSUB statement, in programming terms, where control is passed to another RegEx, a SUBROUTINE, in programming terms, and when that RegEx finishes, control is returned back, RETURN statement, in programming terms, and then processng will continue.

3. reg2 is called to run multiple times within Max10 until the string is exhausted – no more matches, or there are no more calls left, at which time reg3 and reg4 will execute.

Using sample: UK-PeterFord-JeremyJordan-LisaWendt.pdf (to show the flexibility I used a different sample)

After running the batch file and with /Execute removed, it displays a preview:



To rename the file, rerun the batch file with the /Execute restored.

Final result:

UK - Peter Ford, Jeremy Jordan and Lisa Wendt.pdf

Tim Notes:

1. Subroutine as used here is a programmer's term not to be confused with Subroutine, an actual RegEx function.

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Using sample: UK-PeterFord-JeremyJordan-LisaWendt.pdf

Tim Analysis: cont.

reg1 –

Match: ^([A-Z]{2,})-([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1}\2 \3

Name	New Name
UK-PeterFord-JeremyJordan-LisaWendt.pdf	UK}Peter Ford-JeremyJordan-LisaWendt.pdf

Where:

\1 =UK \2 = Peter \3 =Ford-JeremyJordan-LisaWendt

Capture Group 1:	Capture Group 2	Capture Group 3
([A-Z]{2,})	([A-Z][^A-Z -]*?)	([A-Z].*)

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 1 = U
3. {2,}	Limits the repetitions to 2 or more. This means that in order to be TRUE, the RegEx must match against at least two uppercase letters in sequence.= TRUE = 'UK'.	Capture Group 1 = UK
4. -	<hyphen> (Not Captured) followed by ..	
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = P
6. [^A-Z-]	Negated Class.	
	[^A-Z] Continue Moving forward but ignore next uppercase letter.	Capture Group 2 = Pe
	[^-] Continue Moving forward but ignore next <hyphen>.	Capture Group 2 = Unchanged
7. *	Make Greedy	Capture Group 2 = Peter
8. ?	But not too Greedy	Capture Group 2 = P
9. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = F Changes Capture Group 2 = Peter
10. .*	Capture remaining string	Capture Group 3 = Ford-JeremyJordan-LisaWendt
11. $	Must be at EOL (End of Line)	

Conclusion: reg1-

a. Substitutes a } for the first <hyphen> after the first two uppercase letters in string.
b. <space> is added between first run-on word group by Replacement String.

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Original sample: UK-PeterFord-JeremyJordan-LisaWendt.pdf

Tim Analysis: cont.

reg2 –

Match: ^(.*[}][^-]*)-([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1, \2 \3

Run #1 Sample after reg1 = UK}Peter Ford-JeremyJordan-LisaWendt.pdf

Name	New Name
UK}Peter Ford-JeremyJordan-LisaWendt.pdf	UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf

Where:

\1 =UK}Peter Ford \2 = Jeremy \3 =Jordan-LisaWendt

Run #2 Sample after reg2 Run #1 = UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf

Name	New Name
UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf	UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf

Where:

1 = UK}Peter Ford, \2 = Jeremy \3 = Jordan-LisaWendt

After Run #2 – String is Exhausted – No more Renames. Will return back and Process reg3 next.

Name	New Name
UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf	UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf

Where:

\1 = UK}Peter Ford, Jeremy Jordan \2 = Lisa \3 =Wendt

Conclusion: reg2 –

a. substitutes a , for each remaining <hyphen> (after reg1). after the first two uppercase letters in string.
b. Adds <spaces> between all subseqent run-on word groups (Uppercase words) (up to 10 runs of reg2).
c. Each run of reg2 performs the 'substitute comma, add <space> moving forward through the string from left to right until the string is exhausted – no more renames can be done – RegEx Fails.
d. <space> is added between current run-on word group by Replacement String in each run.

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Original sample: UK-PeterFord-JeremyJordan-LisaWendt.pdf
Current sample: UK}Peter Ford-Jeremy Jordan-LisaWendt.pdf (After reg1)
After reg2 Run #1: UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf

Tim Analysis: cont.

1st Run

reg2 – Match: ^(.*[}][^-]*)-([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1, \2 \3

Name	New Name
UK}Peter Ford-JeremyJordan-LisaWendt.pdf	UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf

Where:

\1 = UK}Peter Ford, \2 = Jeremy \3 = Jordan-LisaWendt

Capture Group 1:	Capture Group 2	Capture Group 3
(.*[}][^-]*)	([A-Z][^A-Z -]*?)	([A-Z].*)

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .	. Match against single character	Capture Group 1 = U
3. *	Make it Greedy until …	Capture Group 1 = entire string
4. [}]	.. Match against a class consisting of a right curly brace character	Capture Group 1 = UK}
5. [^-]	Ignore next <hyphen>	Capture Group 1 = UK]P
6. *	Make it Greedy until …	Capture Group 1 = UK}Peter Ford
7. -	.. <hyphen> (Not Captured) followed by ..	
8. [A-Z]	Match against a class consisting of an uppercase letter, followed by ..	Capture Group 2 = J
9. [^A-Z-]	Negated Class.	
	[^A-Z] Continue Moving forward but ignore next uppercase letter.	Capture Group 2 = Je
	[^-] Continue Moving forward but ignore next <hyphen>.	Capture Group 2 = Unchanged
10. *	Make Greedy	Capture Group 2 = Jeremy
11. ?	But not too Greedy	Capture Group 2 = J
12. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = J Changes Capture Group 2 = Jeremy
13. .*	Capture remaining string	Capture Group 3 = Jordan-LisaWendt
14. $	Must be at EOL (End of Line)	

Before reg2 1st run: UK}Peter Ford-JeremyJordan-LisaWendt.pdf
After the string is exhausted: UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Original sample: UK-PeterFord-JeremyJordan-LisaWendt.pdf
Current sample: UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf (After reg2: Run#1)
After reg2 Run #2: UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf String is now Exhausted

Tim Analysis: cont.

2nd Run

reg2 – Match: ^(.*[}][^-]*)-([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1, \2 \3

	New Name
UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf	UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf

Where:

\1 = UK}Peter Ford, Jeremy Jordan \2 = Lisa \3 =Wendt

Capture Group 1:	Capture Group 2	Capture Group 3
(.*[}][^-]*)	([A-Z][^A-Z -]*?)	([A-Z].*)

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .	. Match against single character	Capture Group 1 = U
3. *	Make it Greedy until …	Capture Group 1 = entire string
4. [}]	.. Match against a class consisting of a right curly brace character	Capture Group 1 = UK}
5. [^-]	Ignore next <hyphen>	Capture Group 1 = UK]P
6. *	Make it Greedy until …	Capture Group 1 = UK}Peter Ford, Jeremy Jordan
7. -	.. <hyphen> (Not Captured) followed by ..	No Change
8. [A-Z]	Match against a class consisting of an uppercase letter, followed by ..	Capture Group 2 = L
9. [^A-Z-]	Negated Class.	
	[^A-Z] Continue Moving forward but ignore next uppercase letter.	Capture Group 2 = Li
	[^-] Continue Moving forward but ignore next <hyphen>.	Capture Group 2 = Unchanged
10. *	Make Greedy	Capture Group 2 = Lisa
11. ?	But not too Greedy	Capture Group 2 = L
12. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = W Changes Capture Group 2 = Lisa
13. .*	Capture remaining string	Capture Group 3 = Wendt
14. $	Must be at EOL (End of Line)	

Before reg2 2nd run: UK}Peter Ford, Jeremy Jordan-LisaWendt.pdf
After the string is exhausted: UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Original sample: UK-PeterFord-JeremyJordan-LisaWendt
Current sample: UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf (After reg2: Run #2 exhausts)
After reg3: UK}Peter Ford, Jeremy Jordan and Lisa Wendt.pdf

Tim Analysis: cont.

reg3 – Match: (.*[}].*), (.*)
Replace: \1 and \2

Name	New Name
UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf	UK}Peter Ford, Jeremy Jordan and Lisa Wendt.pdf

Where:

\1 = UK}Peter Ford, Jeremy Jordan \2 = Lisa Wendt

Capture Group 1: Capture Group 2

(.*[}.*) (.*)

1. .	Match against single character	Capture Group 1 = U
3. *	Make it Greedy until …	Capture Group 1 = entire string
4. [}]	.. Match against a class consisting of a right curly brace character	Capture Group 1 = UK}
5. .	Match against single character	Capture Group 1 = UK}P
6. *	Make it Greedy until …	Capture Group 1 = entire string
7. ,	Match against comma (Don't Capture)	Changes Group 1 = UK}Peter Ford, Jeremy Jordan
8. <space>	Match against <space> (Don't Capture)	
9. .*	Capture remaining string	Capture Group 2 = Lisa Wendt

Conclusion: reg3 –

a. Parses the string to be able to insert the 'and' between 'Jordan' and 'Lisa'

Tim Notes:

Subsequent runs of reg3 would result in:

Name	New Name
UK}Peter Ford, Jeremy Jordan and Lisa Wendt.pdf	UK}Peter Ford and Jeremy Jordan and Lisa Wendt.pdf

This could be done in the batch by adding:

```
set max-and = %reg3% %reg3% %reg3%                    Set to run 3 times or until string exhausted
brc32 /Pattern:*-*.pdf %reg1% %max10% %max-and% %reg4%
```

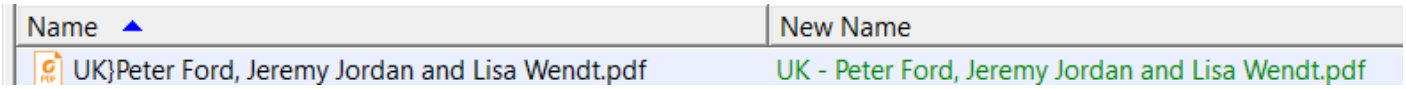
String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Original sample: UK-PeterFord-JeremyJordan-LisaWendt
Current sample: UK}Peter Ford, Jeremy Jordan and Lisa Wendt.pdf (After Reg3)
After reg4: UK - Peter Ford, Jeremy Jordan and Lisa Wendt.pdf

Tim Analysis: cont.

reg4 – Match: (.*)[}](.*)
Replace: \1 - \2



Where:

\1 = UK \2 = Peter Ford, Jeremy Jordan and Lisa Wendt.pdf

1. .*	Match against string until ..	Capture Group 1 = entire string
2. [}]	.. class consisting of a right curly brace (Not Captured)	Changes Capture Group 1 = UK
3. .*	Capture remainder of string	Capture Group 2 = Peter Ford, Jeremy Jordan and Lisa Wendt.pdf

Conclusion: reg4 –

a. Parses the 'UK' from the rest of the string.
b. The Replace String restores <hyphen> with added <spaces> after 'UK'

Tim Notes:

1. I normally don't cover BRC (Bulk Rename Command) but because it is related to this challenge, it's inclusion is required.
2. One of the many adavantages of BRC over the GUI version is that multiple RegEx can be performed within a single run as illustrated here.
3. There is a brc64 version available and that is what I used for testing.
4. Had to change the original [|] for [}] in reg2 because the | (pipe character) is an invalid character in filenames.

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Post Continues

Contributer adapted the script to handle:

COUNTRY - FirstName

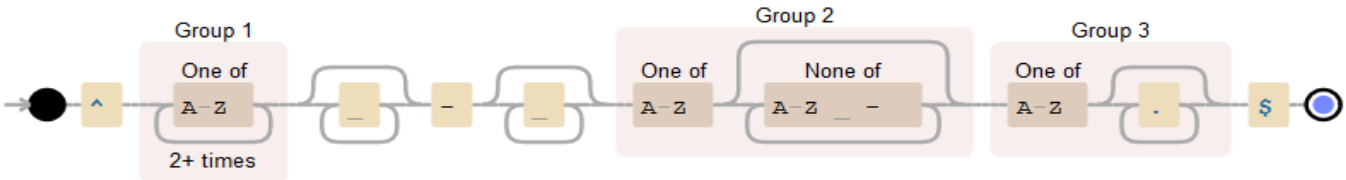
i.e., USA – FlashGordon.docx

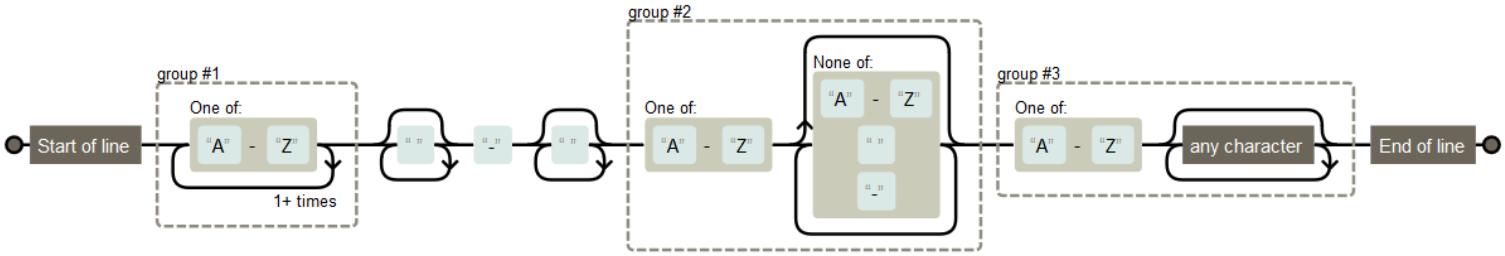
For BRC Change reg1 to:

/Regexp:"^([A-Z]{2,}) *- *([A-Z][^A-Z -]*?)([A-Z].*)$:\1}\2 \3"

BRU Equivalent would be:

Match: ^([A-Z]{2,}) *- *([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1}\2 \3





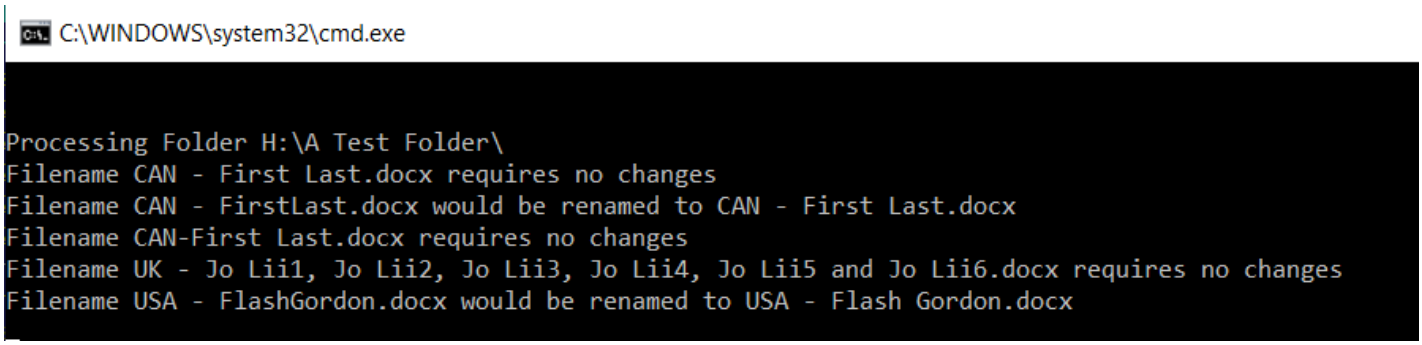
String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Post Continues cont.

Here is the updated batch file:

```
@echo off
cd/d "H:\A Test Folder"
set reg1=/Regexp:"^([A-Z]{2,}) *- *([A-Z][^A-Z -]*?)([A-Z].*)$:\1}\2 \3"
set reg2=/Regexp:"^(.*[}][^-]*)-([A-Z][^A-Z -]*?)([A-Z].*)$:\1, \2 \3"
set max10=%reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2%
set reg3=/Regexp:"(.*[}].*), (.*):\1 and \2"
set reg4=/Regexp:"(.*)[}](.*):\1 - \2"
brc32 /Pattern:*-*.docx %reg1% %max10% %reg3% %reg4% /Execute
pause>nul
```



Forum Contributor: bru

String Manipulation (Parsing)

Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name cont.

Using: USA – FlashGordon.docx

Tim Analysis:

Match: ^([A-Z]{2,}) *- *([A-Z][^A-Z -]*?)([A-Z].*)$
Replace: \1}\2 \3

Name	New Name
USA - FlashGordon.docx	USA}Flash Gordon.docx

Where: \1 = USA \2 = Flash \3 = Gordon

Capture Group 1:	Capture Group 2	Capture Group 3
([A-Z]{2,})	([A-Z][^A-Z -]*?)	([A-Z].*)

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 1 = U
3. {2,}	Limits the repetitions to 2 or more. This means that in order to be TRUE, the RegEx must match against at least two uppercase letters in sequence.= TRUE = 'USA'.	Capture Group 1 = USA
4.<space>	Match against the <space> character (Not Captured) = <space>	
5. *	Make it Greedy until…	No Changes
6. -	.. <hyphen> (Not Captured) followed by.. = <space><hyphen>	No Changes
7.<space>	Match against the <space> character (Not Captured) = <space><hyphen><space>	No Changes
8. *	Make it Greedy until…	No Changes
9. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = F
10. [^A-Z-]	Negated Class.	
	[^A-Z] Continue Moving forward but ignore next uppercase letter.	Capture Group 2 = Fl
	[^-] Continue Moving forward but ignore next <hyphen>. There are not any more <hyphens>. This was a hold-over from original reg1, but it does no harm here because it is ignored.	Capture Group 2 = Unchanged
11. *	Make Greedy	Capture Group 2 = Flash
12. ?	But not too Greedy	Capture Group 2 = F
13. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = G Changes Capture Group 2 = Flash
14. .*	Capture remaining string	Capture Group 3 = Gordon
15. $	Must be at EOL (End of Line)	

Conclusion: reg1-

a. Removes the <space><hyphen><space> and adds the } after the 'Country' designation
b. <space> is added between first run-on word group by Replacement String.

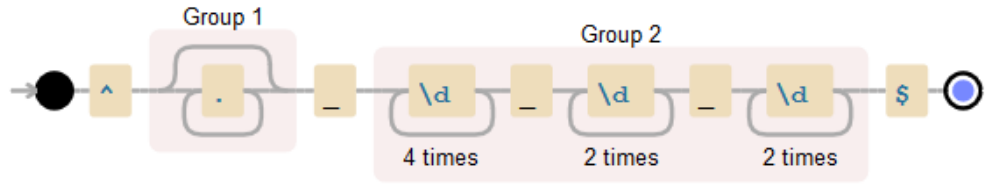
String Manipulation (Parsing)

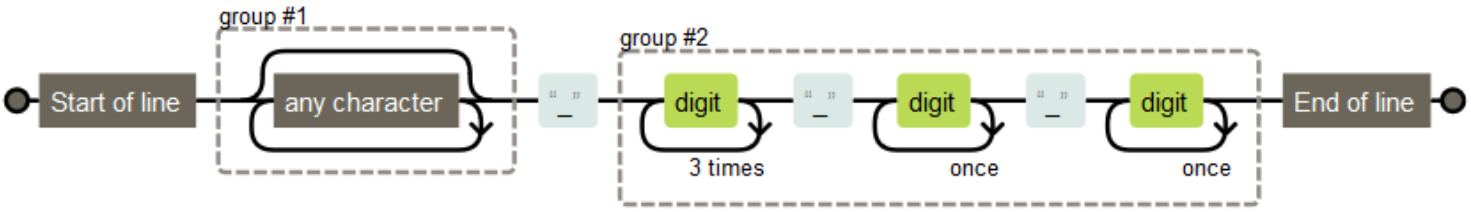
Move Underscored Date to beginning of String

abcdefgh, abcdefgh_abcdefgh_1_2016_09_22.docx to 2016_09_22-abcdefgh, abcdefgh_abcdefgh_1.docx

Match: ^(.*)(?:_)(\d{4}_\d{2}_\d{2})$
Replace: \2-\1

	New Name
abcdefgh, abcdefgh_abcdefgh_1_2016_09_22.docx	2016_09_22-abcdefgh, abcdefgh_abcdefgh_1.docx





Sample:

```
Match 1: abcdefgh, abcdefgh_abcdefgh_1_2016_09_22
Group 1: abcdefgh, abcdefgh_abcdefgh_1
Group 2: 2016_09_22
```

Under Regex Buddy for this test, had to remove anchor $ because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.docx' is just another part of the string of text.

Forum Contributor: KenP

String Manipulation (Parsing)

Move Underscored Date to beginning of String cont.

abcdefgh, abcdefgh_abcdefgh_1_2016_09_22.docx to 2016_09_22-abcdefgh, abcdefgh_abcdefgh_1.docx

Match: ^(.*)(?:_)(\d{4}_\d{2}_\d{2})$

Where:

\1 = abcdefgh, abcdefgh_abcdefgh_1 \2 = 2016_09_22

Tim Analysis:

1. ^	Start at beginning of filename (Beginning of Line or BOL)	
2. .*	Match all characters until…	Capture Group 1 = abcdefgh, abcdefgh_abcdefgh_1_2016_09
3. _	Ignore and move forward past the underscore character until …. a. Non-Marking Group	
4. \d{4}	.. 4 numeric digits (the Year) followed by …	Capture Group 2 = 2016 Changes Capture Group 1 = abcdefgh, abcdefgh_abcdefgh_1
5. _	.. underscore followed by …	Capture Group 2 = 2016_
6. \d{2}	.. 2 numeric digits (Month), followed by …	Capture Group 2 = 2016_09
7. _	.. underscore followed by	Capture Group 2 = 2016_09_
8. \d 2}	.. 2 numeric digits (Day), followed by …	Capture Group 2 = 2016_09_22
9. $	.. Last expression must have matched at EOL (must be at End of Line)	

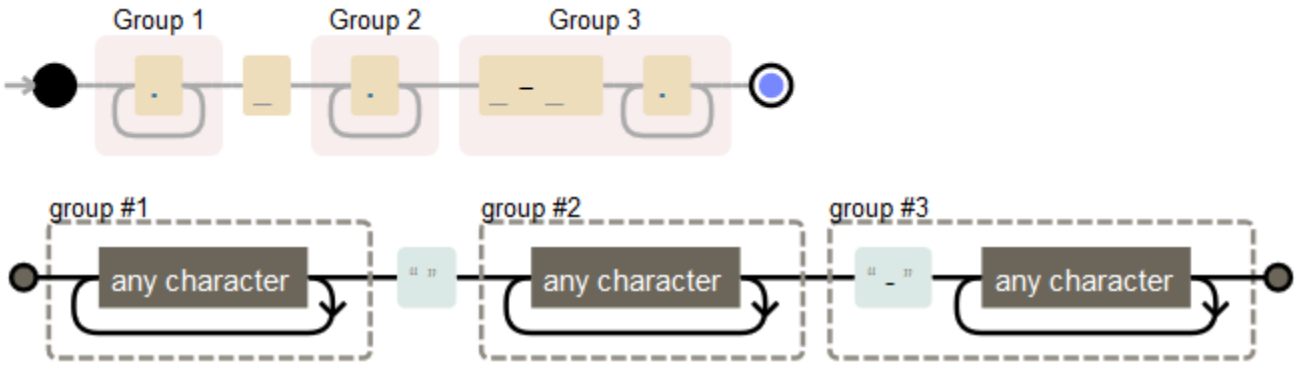
String Manipulation (Parsing)

Simple rearrange

FirstName LastName - songtitle.ext to LastName, Firstname - songtitle.ext

Match: (.+) (.+)(- .+)
Replace: \2, \1\3

Name	New Name
FirstName LastName - songtitle.ext	LastName, FirstName - songtitle.ext



Sample:

```
........................................
Match 1: FirstName LastName - songtitle
Group 1: FirstName
Group 2: LastName
Group 3:  - songtitle
```

Forum Contributor: Stefan

String Manipulation (Parsing)

Simple rearrange cont.

Using: FirstName LastName - songtitle.ext

Match: (.+) (.+)(- .+)

Where:

\1 = Firstname \2 = LastName \3 = <space> - <space> songtitle

Tim Analysis:

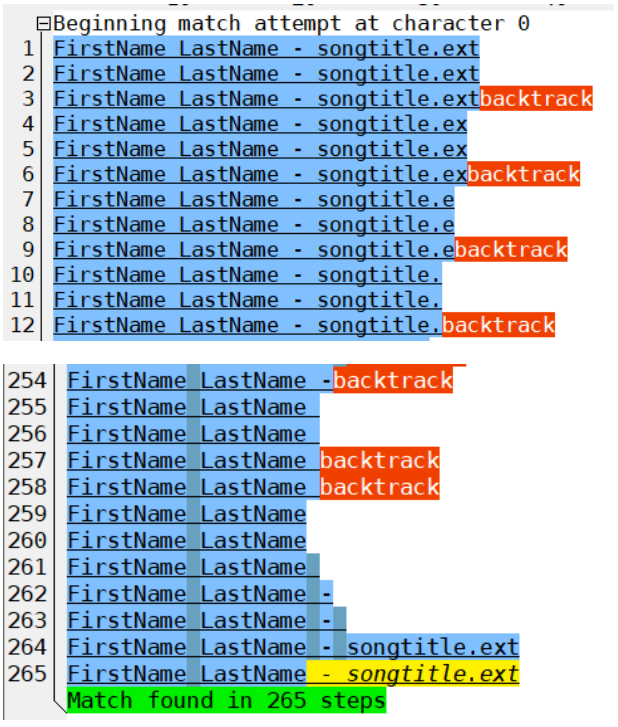
1..	Match against any character	Capture Group 1 =
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Lastname -'	Changes Capture Group 1 = Firstname LastName -
4..	Match against any character	Capture Group 2 = s
5. +	Make it Greedy Current position = EOL	Capture Group 2 = songtitle
6. <space>	Match against a <space> character This is the <space> after 'LastName -' Capture Group 2 changes to the <hyphen> after 'LastName <space>'	Capture Group 3 = <space> Changes Capture Group 2 = - Changes Capture Group 1 = Firstname Lastname
7. -	Match against <hyphen> This is the <hyphen> after "LastName <hyphen>	Capture Group 3 = <space> <hyphen> Changes Capture Group 2 = Lastname Changes Capture Group 1 = FirstName
8. .+	Match against remaining string	Capture Group 3 = <space> - <space> songtitle

Tim Notes:

1. The problem with what seems like a simple RegEx to parse strings, are actually the worse coded.

2. They are a mess to analyze (look at other examples in this book) and are full of extra steps the RegEx Engine has to make in order to satisfy the match.

3. This one is just one backtracking after another. To process this simple string took 265 steps! Multiply that by a few hundred and it will display a lag in Processing time.



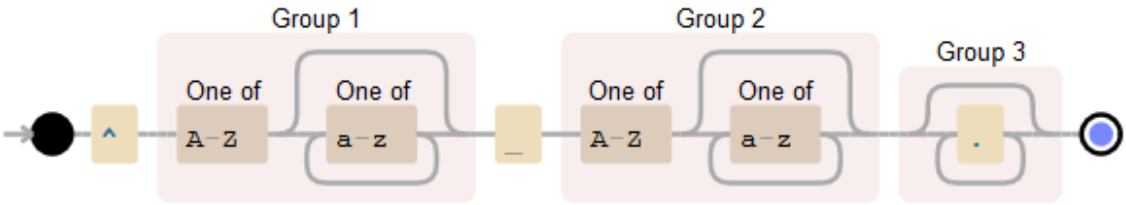
String Manipulation (Parsing)

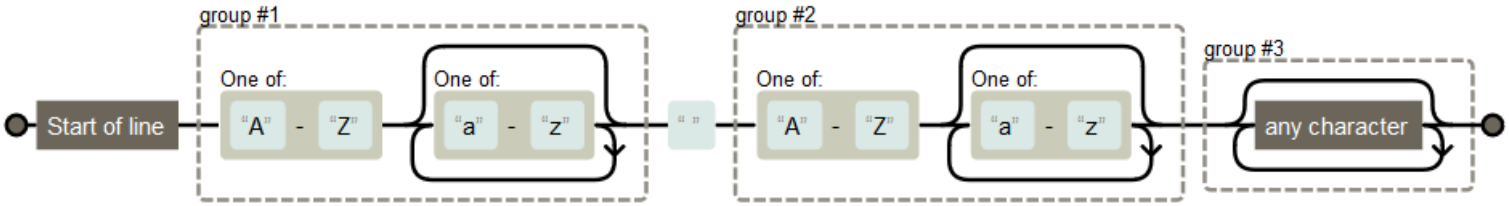
Simple rearrange - More difficult

Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3
to
Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3

Match: ^([A-Z][a-z]*) ([A-Z][a-z]*)(.*)
Replace: \2, \1\3

Name	New Name
Arthur Brown - The Crazy World Of Arthur Brown - 08 -...	Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3





Sample:

```
.....................................................................
Match 1: Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire
Group 1: Arthur
Group 2: Brown
Group 3:  - The Crazy World Of Arthur Brown - 08 - Fire
```

Forum Contributor: BRUlvr100

String Manipulation (Parsing)

Simple rearrange - More difficult cont.

Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3
to
Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3

Match: ^([A-Z][a-z]*) ([A-Z][a-z]*)(.*)

Where:
\1 = Arthur \2 = Brown

\3 = <space><hyphen><space>The Crazy World of Arthur Brown – 08 - Fire

Tim Analysis:

1. ^	Start at beginning of filename (Beginning of Line or BOL)	
2. [A-Z]	Match against first uppercase letter	Capture Group 1 = A
3. [a-z]	Match against lowercase letter	Capture Group 1 = Ar
4. *	Make it Greedy until…	Capture Group 1 = Arthur
5. <space>	… <space> (Not Captured) followed by…	
6. [A-Z]	Match against first uppercase letter	Capture Group 2 = B
7. [a-z]	Match against lowercase letter	Capture Group 2 = Br
8. *	Make it Greedy…	Capture Group 2 = Brown
9. .*	Capture remainder of string	Capture Group 3 = <space><hyphen><space> The Crazy World of Arthur Brown – 08 - Fire

Tim Notes:

1. Classes are defined as characters that are enclosd within brackets

String Manipulation (Parsing)

Another Switch around

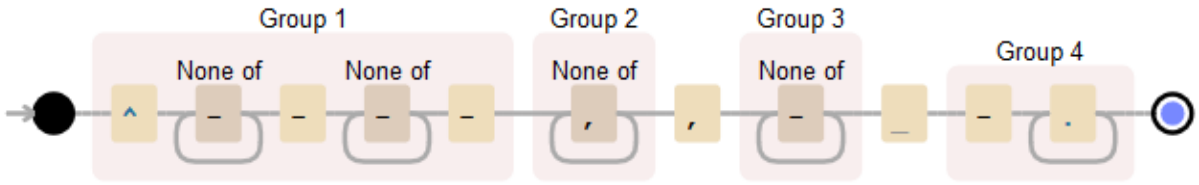
Tim Notes:

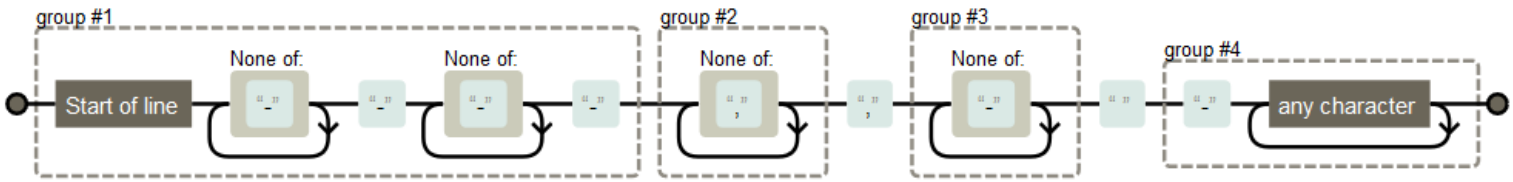
1. Illustrates using Negated Classes

AH2002-04 - Twain, Shania - Any Man Of Mine
to
AH2002-04 - Shania Twain - Any Man Of Mine

Match: (^[^-]+-[^-]+-)([^,]+),([^-]+) (-.+)
Replace: \1\3\2 \4

Name	New Name
AH2002-04 - Twain, Shania - Any Man Of Mine	AH2002-04 - Shania Twain - Any Man Of Mine





Sample:

```
.....................................................
Match 1: AH2002-04 - Twain, Shania - Any Man Of Mine
Group 1: AH2002-04 -
Group 2:  Twain
Group 3:  Shania
Group 4: - Any Man Of Mine
```

Contributor Analysis:

1. It selects the text between the <hyphens> by looking for non <hyphens> [^-]

Forum Contributor: Jane

Tim Notes:

1. A Negated character Class is the opposite of a 'normal' class. A Class is defined as inclusive of the characters within the brackets. A Negated Class, as expressed by the syntax of [^], is defined as exclusive of the characters within the brackets, or in other words, Do NOT match against these characters if found. This normally results in 'skipping' past these characters.

   Normally this would cause the Evaluation to fail, but because we are instructing the RegEx engine to, in a sense, ignore these characters, then if found = TRUE and the evaluation continues.

String Manipulation (Parsing)

Another Switch around cont.

Using: AH2002-04 - Twain, Shania - Any Man Of Mine

Match: (^[^-]+-[^-]+-)([^,]+),([^-]+) (-.+)

Where:
\1 = AH2002-04- \2 = <space>Twain \3 = <space>Shania \4 = <hyphen><space>Any Man Of Mine

Tim Analysis:

^	Start at beginning of string (Beginning of Line or BOL)	Capture Group 1
[^-]	Using a Negated Class ignore the <hyphen> if found	Capture Group 1 = A
+	Make it Greedy so it matches against all characters that are not <hyphens> until ...	Capture Group 1 = AH2002
-	.. <hyphen>	Capture Group 1 = AH2002-
[^-]	Using a Negated Class ignore the <hyphen> if found	Capture Group 1 = AH2002-0
+	Make it Greedy so it matches against all characters that are not <hyphens> until ...	Capture Group 1 = AH2002-04
-	.. <hyphen>	Capture Group 1 = AH2002-04-
[^,]	Using a Negated Class, ignore the <comma> if found	Capture Group 2 = <space><hyphen><space>
+	Make it Greedy so it matches against all characters until…	Capture Group 2 = <space><hyphen><space>Twain
,	… comma, Not Captured	
[^-]	Using a Negated Class ignore the <hyphen> if found	Capture Group 3 = <space>
+	Make it Greedy so it matches against all characters until…	Capture Group 3 = <space>Shania
<space>	<space> Not Captured, followd by … This will remove the extra <space> that would otherwise occur between 'Shania' and 'Twain'	
-	.. <hyphen>, followed by … This is the <hyphen> following 'Shania'	Capture Group 4 = <hyphen>
.	.. any single character In this case it is the <space> following the preceding <hyphen>	Capture Group 4 = <hyphen><space>
+	Capture remainder of string	Capture Group 4 = <hyphen><space>Any Man of Mine

String Manipulation (Parsing)

Not as Simple a rearrange

Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

Some Notes by Jane (past Contributor):

These notes can apply to many RegEx renaming operations -

The answer can be simple to complex depending on how how flexible you want it to be and how accurately your example represents the range of file names you want to rename.

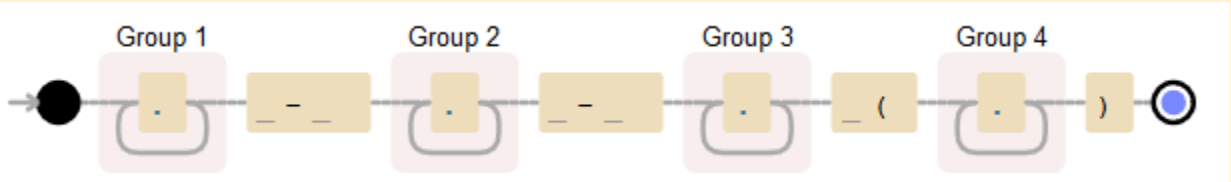
For example, in this sample -

Artist Name - Label Name - Track Name (Remix name)

Example #1. Look for a pattern.
a. A single <space> on both sides of each <hyphen>
.. and a <space> before the opening parenthesis.

If this is always the case you could match it with:
Match: (.+) - (.+) - (.+) \((.+)\)
Replace: \1 - \3 - \4 - \2

Name	New Name
Artist Name - Label Name - Track Name (Remix name)	Artist Name - Track Name - Remix name - Label Name



Sample:

```
.....................................................
Match 1: Artist Name - Label Name - Track Name (Remix name)
Group 1: Artist Name
Group 2: Label Name
Group 3: Track Name
Group 4: Remix name
```

However, if even one <space> is missing it will not match.

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Match: (.+) - (.+) - (.+) \((.+)\)

Where:

\1 = Artist Name \2 = Label Name \3 = Track Name \4 = Remix name

Tim Analysis: Example #1

1. .	Match against any character	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	<space> (Not Captured), followed by ..	Changes Capture Group 1 = Artist Name - Label Name - Track Name (Remix
4. -	.. <hyphen> (Not Captured), followed by ..	Changes Capture Group 1 = Artist Name - Label Name<space>
5. <space>	.. <space> (Not Captured)	Changes Capture Group 1 = Artist Name - Label Name
6. .	Match against any character	Capture Group 2 = T
7. +	Make it Greedy	Capture Group 2 = Track Name (Remix name)
8. <space>	<space> (Not Captured), followed by ..	Changes Capture Group 2 = Track Name (Remix
9. -	.. <hyphen> (Not Captured), followed by ..	Changes Capture Group 2 = Label Name <space>
		Changes Capture Group 1 = Artist Name
10. <space>	.. <space> (Not Captured)	Changes Capture Group 2 = Label Name
11. .	Match against any character	Capture Group 3 = T
12. +	Make it Greedy	Capture Group 3 = Track Name (Remix name)
13. <space>	<space> (Not Captured), followed by ..	Changes Capture Group 3 = Track Name (Remix
14. \(	.. Match against literal left parenthesis (Not Captured)	Changes Capture Group 3 = Track Name
15. .	Match against any character	Capture Group 4 = R
16 +	Make it Greedy	Capture Group 4 = Remix name)
17. \)	.. Match against literal right parenthesis (Not Captured)	Changes Capture Group 4 = Remix name

String Manipulation (Parsing)

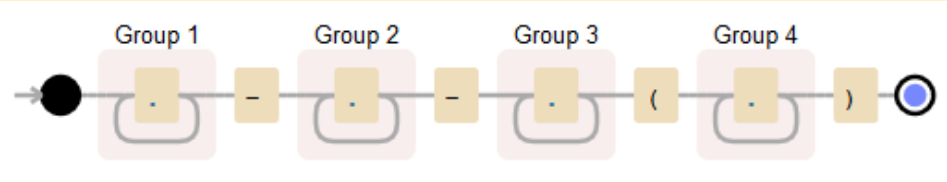
Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

Some Notes by Jane cont.

Example #2. If you weren't sure of the number of <spaces>, -
Match: (.+)-(.+)-(.+)\((.+)\)
Replace: \1 - \3 - \4 - \2

New Name
Artist Name - Track Name - Remix name - Label Name



Sample:

```
.....................................................
Match 1: Artist Name - Label Name - Track Name (Remix name)
Group 1: Artist Name
Group 2:  Label Name
Group 3:  Track Name
Group 4: Remix name
```

or

Replace: \1-\3-\4-\2

New Name
Artist Name - Track Name -Remix name- Label Name

depending on whether you wanted at least one <space> on each side of the <hyphen> in the renamed file name. This would be more flexible but you could end up with a variable number of <spaces> around the <hyphens> and before the parentheses.

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

Some Notes by Jane cont.

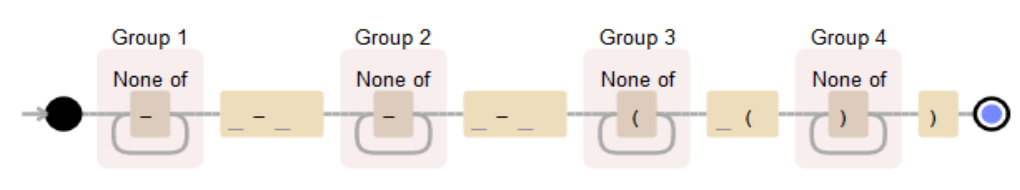
Example #3a & 3b. Backtracking.

Another problem with this type of RegEx (using .+ to get a match) is that in this case there is a huge amount of backtracking to get a match of a single file name.
a. The first Match above takes about 340 steps.
b. The second Match takes about 314 steps

This could be significant if you are naming thousands of files. A better RegEx would be:

Match: ([^-]+) - ([^-]+) - ([^(]+) \(([^)]+)\)
Replace: \1 - \3 - \4 - \2

New Name
Artist Name - Track Name - Remix name - Label Name



Sample:

```
.....................................................
Match 1: Artist Name - Label Name - Track Name (Remix name)
Group 1: Artist Name
Group 2: Label Name
Group 3: Track Name
Group 4: Remix name
```

or

Match: ([^-]+)-([^-]+)-([^(]+)\(([^)]+)\)
Replace: \1 - \3 - \4 - \2

New Name
Artist Name - Track Name - Remix name - Label Name

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Match: ([^-]+) - ([^-]+) - ([^(]+) \(([^)]+)\)

Where:

\1 = Artist Name \2 = Label Name \3 = Track Name \4 = Remix name

Tim Analysis: Example #3a

1. [^-]	Match against a Negated Class consisting of <hyphen>	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = Artist Name
3. <space>	<space> (Not Captured), followed by ..	
4. -	.. <hyphen> (Not Captured), followed by .	
5. <space>	.. <space> (Not Captured)	
6. [^-]	Match against a Negated Class consisting of <hyphen>	Capture Group 2 = L
7 +	Make it Greedy	Capture Group 2 = Label Name
8. <space>	<space> (Not Captured), followed by ..	
9. -	.. <hyphen> (Not Captured), followed by ..	
10. <space>	.. <space> (Not Captured)	
11. [^-]	Match against a Negated Class consisting of <hyphen>	Capture Group 3 = T
12. +	Make it Greedy	Capture Group 3 = Track Name (Remix name)
13. <space>	<space> (Not Captured), followed by ..	Changes Capture Group 3 = Track Name (Remix
14. \(	.. Match against literal left parenthesis (Not Captured)	Changes Capture Group 3 = Track Name
15. [^-]	Match against a Negated Class consisting of <hyphen>	Capture Group 4 = R
16. +	Make it Greedy	Capture Group 4 = Remix name)
17. \)	.. Match against literal right parenthesis (Not Captured)	Changes Capture Group 4 = Remix name

String Manipulation (Parsing)

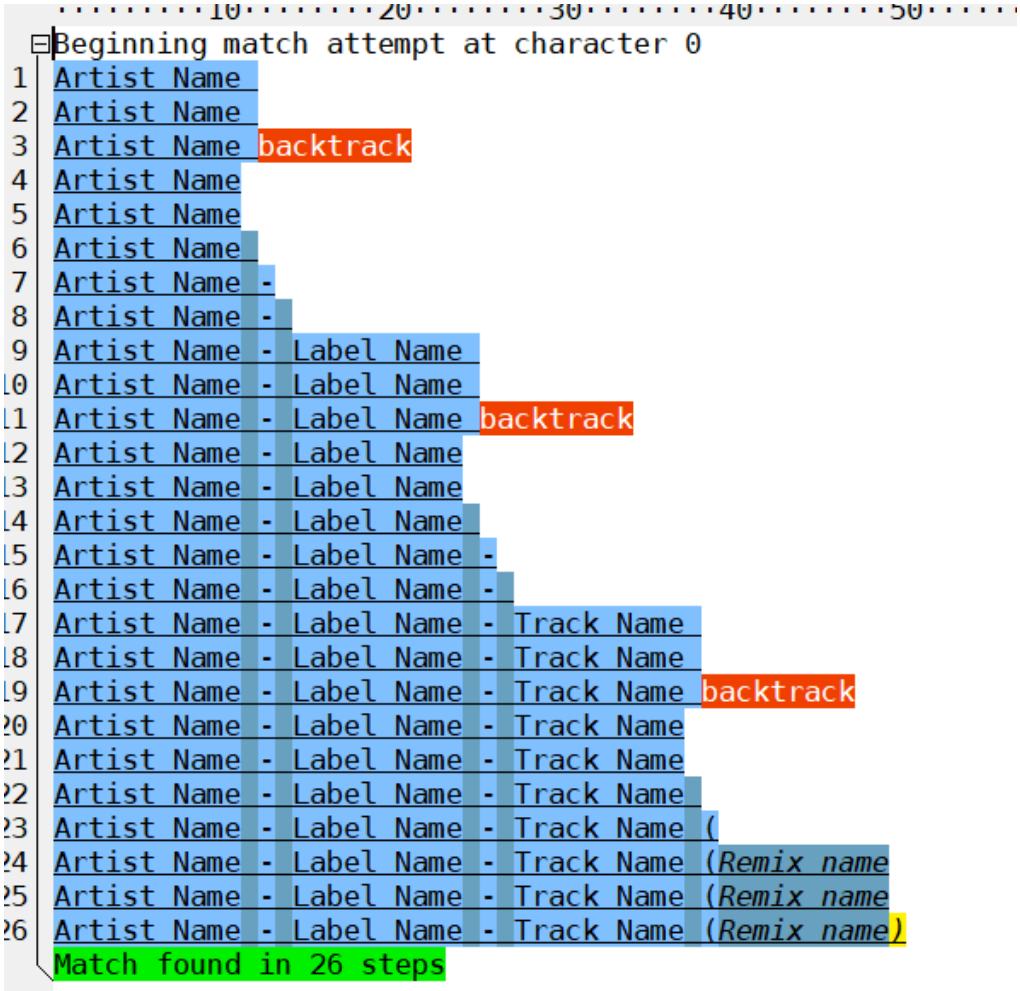
Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Tim Analysis: Example #3a cont.

Backtracking Comparison: Using Negated Classes (Example #3a)

([^-]+) - ([^-]+) - ([^(]+) \(([^)]+)\)



String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Tim Analysis: cont.

Backtracking Comparison: cont. Using just Wildcards (Example #1)

Match: (.+) - (.+) - (.+) \((.+)\)

```
   ·····10·····20·····30·····40·····50·····60···
  ⊟Beginning match attempt at character 0
1 Artist Name - Label Name - Track Name (Remix name)
2 Artist Name - Label Name - Track Name (Remix name)
3 Artist Name - Label Name - Track Name (Remix name)backtrack
4 Artist Name - Label Name - Track Name (Remix name
5 Artist Name - Label Name - Track Name (Remix name
6 Artist Name - Label Name - Track Name (Remix namebacktrack
7 Artist Name - Label Name - Track Name (Remix nam
8 Artist Name - Label Name - Track Name (Remix nam
9 Artist Name - Label Name - Track Name (Remix nambacktrack
10 Artist Name - Label Name - Track Name (Remix na
11 Artist Name - Label Name - Track Name (Remix na
12 Artist Name - Label Name - Track Name (Remix nabacktrack
13 Artist Name - Label Name - Track Name (Remix n
14 Artist Name - Label Name - Track Name (Remix n
15 Artist Name - Label Name - Track Name (Remix nbacktrack
16 Artist Name - Label Name - Track Name (Remix
17 Artist Name - Label Name - Track Name (Remix
18 Artist Name - Label Name - Track Name (Remix backtrack
19 Artist Name - Label Name - Track Name (Remix
20 Artist Name - Label Name - Track Name (Remix
21 Artist Name - Label Name - Track Name (Remix
22 Artist Name - Label Name - Track Name (Remix backtrack
```

This goes on for a total of: 340 steps!!

```
325 Artist Name - Label Name - Track Name (
326 Artist Name - Label Name - Track Name (
327 Artist Name - Label Name - Track Name (backtrack
328 Artist Name - Label Name - Track Name
329 Artist Name - Label Name - Track Name
330 Artist Name - Label Name - Track Name backtrack
331 Artist Name - Label Name - Track Name
332 Artist Name - Label Name - Track Name
333 Artist Name - Label Name - Track Name
334 Artist Name - Label Name - Track Name (
335 Artist Name - Label Name - Track Name (Remix name)
336 Artist Name - Label Name - Track Name (Remix name)
337 Artist Name - Label Name - Track Name (Remix name)backtrack
338 Artist Name - Label Name - Track Name (Remix name
339 Artist Name - Label Name - Track Name (Remix name
340 Artist Name - Label Name - Track Name (Remix name)
    Match found in 340 steps
```

Conclusion:

The simplest RegEx is not always the best solution when it comes to speed.

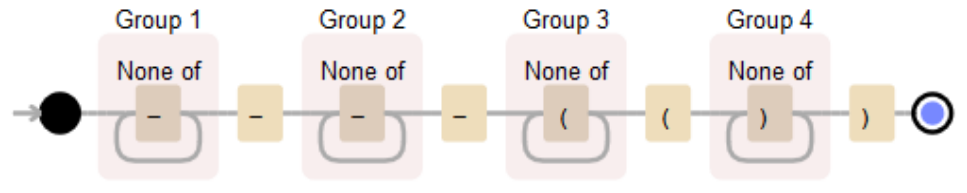
If you are only renaming a few dozen, then speed is really not a factor, but if your renaming thousands of files, then you may have to go get some lunch and return later.

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

Some Notes by Jane cont.

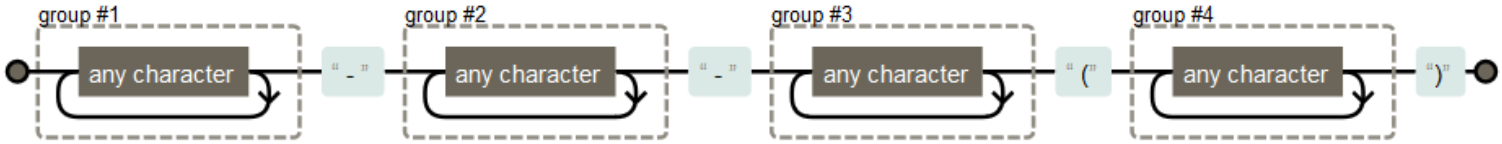


Sample:

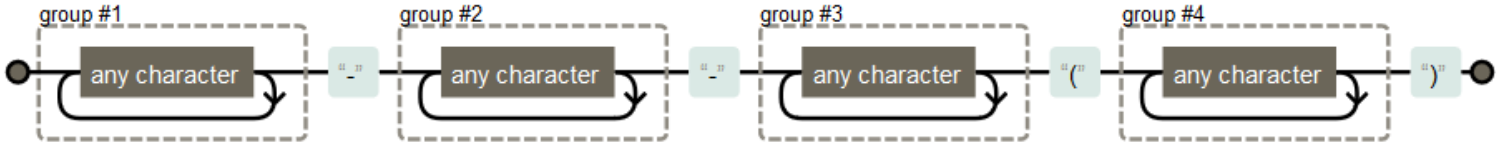
```
.....................................................................
Match 1: Artist Name - Label Name - Track Name (Remix name)
Group 1: Artist Name
Group 2:  Label Name
Group 3:  Track Name
Group 4: Remix name
```

Again, depending on whether you are sure of the <spaces> around the <hyphens> etc.
These do not backtrack and the first will match in 12 steps!.

Example #1: (.+) - (.+) - (.+) \((.+)\)



Example #2: (.+)-(.+)-(.+)\((.+)\)



String Manipulation (Parsing)

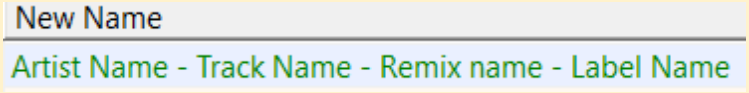
Not as Simple a rearrange cont.

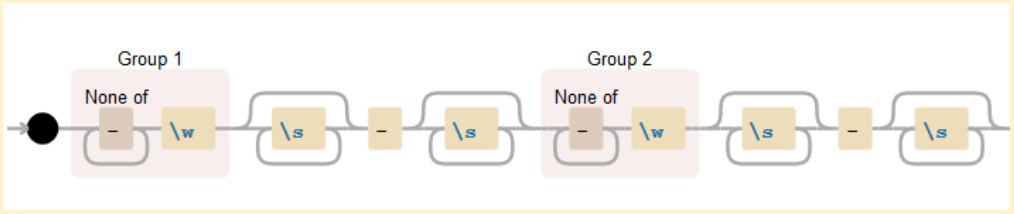
Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

Some Notes by Jane cont.

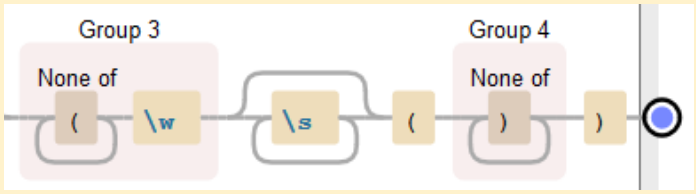
Example #4. If you wanted to be able to match variable spacing and were fussy that the replacement would have exactly the right number of <spaces> you could use:

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)
Replace: \1 - \3 - \4 - \2





…

…


Sample:

```
.....................................................
Match 1: Artist Name - Label Name - Track Name (Remix name)
Group 1: Artist Name
Group 2: Label Name
Group 3: Track Name
Group 4: Remix name
```

This captures no surrounding <spaces> and allows for a varying number of <spaces>, and does it in about 32 steps.

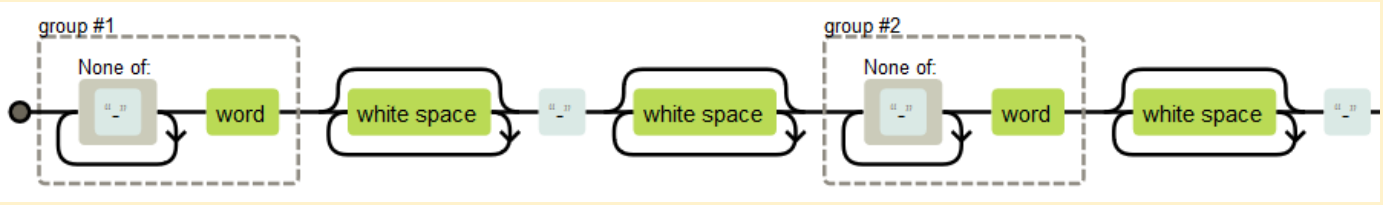
String Manipulation (Parsing)

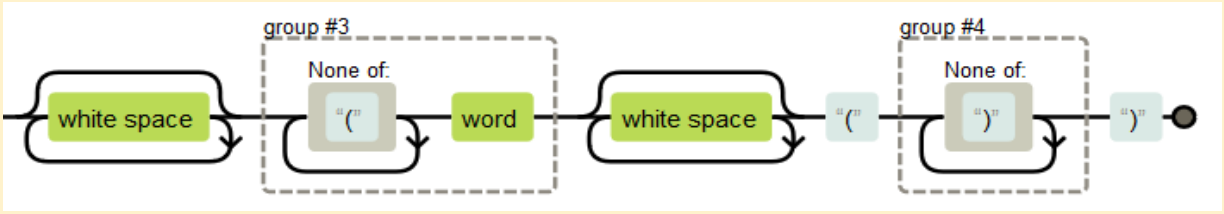
Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

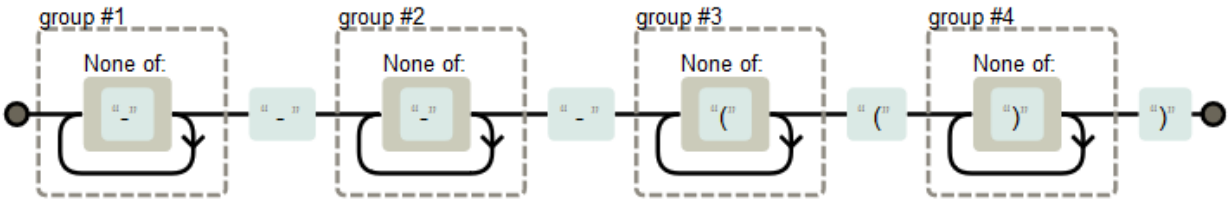
Some Notes by Jane cont.

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)

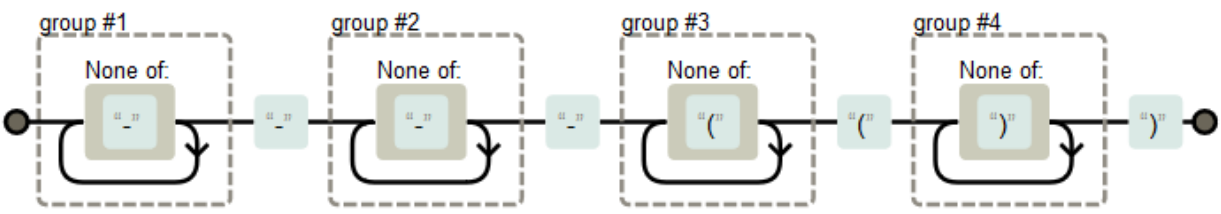




Example #3a: ([^-]+) - ([^-]+) - ([^(]+) \(([^)]+)\)



Example #3b: ([^-]+)-([^-]+)-([^(]+)\(([^)]+)\)



Forum Contributor: Jane

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)

Where:

\1 = Artist Name \2 = Label Name \3 = Track Name \4 = Remix name

Tim Analysis: Example #4

1. [^-]	Match against a Negated Class consisting of <hyphen>	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = Artist Name
3. \w	Word Metacharacter. Match against any single letter, number or underscore followed by …	
4. \s	.. <space>	
5. *	Make it Greedy Gathers up all <spaces> before the <hyphen>	
6. -	hyphen>, followed by …	
7. \s	.. <space>	
8. *	Make it Greedy Same as steps 4 nd 5	
9. [^-]	Match against a Negated Class consisting of <hyphen>	Capture Group 2 = L
10. +	Make it Greedy	Capture Group 2 = Label Name
11. \w	Word Metacharacter. Match against any single letter, number or underscore followed by …	
12. \s	.. <space>	
13. *	Make it Greedy	
14. -	hyphen>, followed by …	
15. \s	.. <space>	
16. *	Make it Greedy	
25. \)	Match against literal right parenthesis (Not Captured)	Changes Capture Group 4 = Remix name

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)

Where:

\1 = Artist Name \2 = Label Name \3 = Track Name \4 = Remix name

Tim Analysis: Example #4 cont.

17. [^\(]	Match against a Negated Class consisting of a left parenthesis	Capture Group 3 = T
18. +	Make it Greedy	Capture Group 3 = Track Name
19. \w	Word Metacharacter. Match against any single letter, number or underscore followed by …	
20. \s	.. <space>	
21. *	Make it Greedy	
22. \(	Match against literal left parenthesis (Not Captured)	
23. [^)]	Match against a Negated Class consisting of right parenthesis	Capture Group 4 = R
24. +	Make it Greedy	Capture Group 4 = Remix name)

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Artist Name - Label Name - Track Name (Remix name)

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)

Where:

\1 = Artist Name \2 = Label Name \3 = Track Name \4 = Remix name

Tim Analysis: Example #4 cont.

```
   ·········10·········20·········30·········40·········50·········60·
  ⊟Beginning match attempt at character 0
 1 Artist Name
 2 Artist Name backtrack
 3 Artist Name
 4 Artist Name backtrack
 5 Artist Name
 6 Artist Namebacktrack
 7 Artist Nam
 8 Artist Name
 9 Artist Name
10 Artist Name
11 Artist Name -
12 Artist Name -
13 Artist Name - Label Name
14 Artist Name - Label Name backtrack
15 Artist Name - Label Name
16 Artist Name - Label Name backtrack
17 Artist Name - Label Name
18 Artist Name - Label Name backtrack
19 Artist Name - Label Name
20 Artist Name - Label Namebacktrack
21 Artist Name - Label Nam
22 Artist Name - Label Name
23 Artist Name - Label Name
24 Artist Name - Label Name
25 Artist Name - Label Name -
26 Artist Name - Label Name -
27 Artist Name - Label Name - Track Name
28 Artist Name - Label Name - Track Name backtrack
29 Artist Name - Label Name - Track Name
30 Artist Name - Label Name - Track Namebacktrack
31 Artist Name - Label Name - Track Nam
32 Artist Name - Label Name - Track Name
33 Artist Name - Label Name - Track Name
34 Artist Name - Label Name - Track Name
35 Artist Name - Label Name - Track Name (
36 Artist Name - Label Name - Track Name (Remix name
37 Artist Name - Label Name - Track Name (Remix name
38 Artist Name - Label Name - Track Name (Remix name)
   Match found in 38 steps
```

String Manipulation (Parsing)

Not as Simple a rearrange cont.

Samples expanded:

Artist Name - Label Name - Track Name (Remix name)
Artist Name - Label Name   -   Track Name (Remix name)

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)

Tim Notes:

1. Steps 3, 11 and 19, in the sample used do nothing because \w would capture the <space> following each word but this is already accounted for by \s*. So why are they needed?

   For example:

   Match: ([^-]+...)
   Replace \1

Name	New Name
Artist Name - Label Name - Track N...	Artist Name - L
Artist Name - Label Name - Trac...	Artist Name -

   Adding 3 characters after the initial 'Artist Name' captured solely by ([^-]+), reveals the additional <spaces> that would need to be captured in the second sample, whereas the first only has one <space>.

   Match: ([^-]+)

```
Beginning match attempt at character 0
1 Artist Name
2 Artist Name
  Match found in 2 steps
```

   Adding \s* by itself:

   Match: ([^-]+\s*)

```
Beginning match attempt at character 0
1 Artist Name
2 Artist Name  ok
3 Artist Name
  Match found in 3 steps
```

String Manipulation (Parsing)

Not as Simple a rearrange cont.

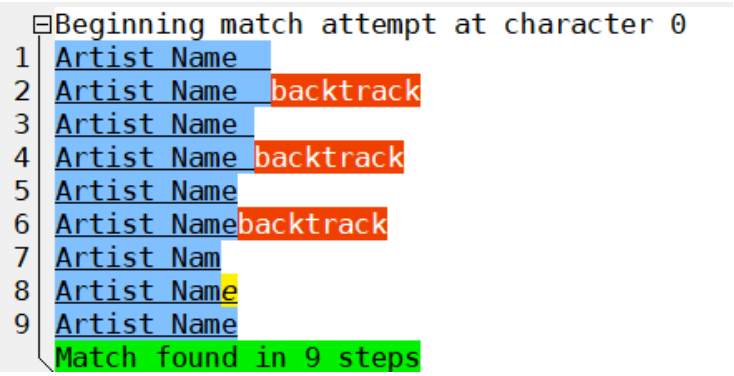
Artist Name - Label Name - Track Name (Remix name)

Match: ([^-]+\w)\s*-\s*([^-]+\w)\s*-\s*([^(]+\w)\s*\(([^)]+)\)

Tim Notes: cont.

Adding \w

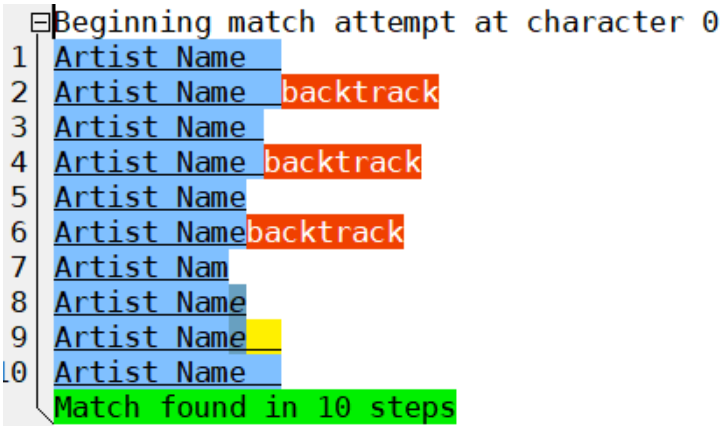
Match: ([^-]+\w)



The \w is a Word Boundary. After the [^-]+ gathered up, Artist name<space>s' , \w backtracks to match the 'e' of Name.

Adding \s*

Match: ([^-]+\w\s*)



After the 'e' is matched by \w in Capture Group 1, \s* matches any additional <spaces>. This would be redundant to [^-]\s* if not for the fact that \s* is NOT captured – meaning all additional <spaces> are accounted for and eliminated after 'Name' before the <hyphen>.

I only included \s* in Capture Group 1 in this test to illustrate what is affected. In the RegEx, ([^-]\w)\s*, using \w, backtracks to the 'e' of 'Name' allowing the <spaces> that follow to be isolated and dealt with separately.

2. Step 17, matching against a Negated Class of the left parenthesis, eliminates the capture of 'Track Name (Remix name)' that would otherwise occur in step 18, thus saving a backtracking step.

String Manipulation (Parsing)

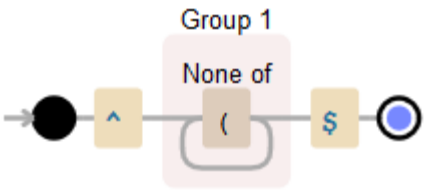
Ignore files with parentheses, otherwise append (Original Mix)

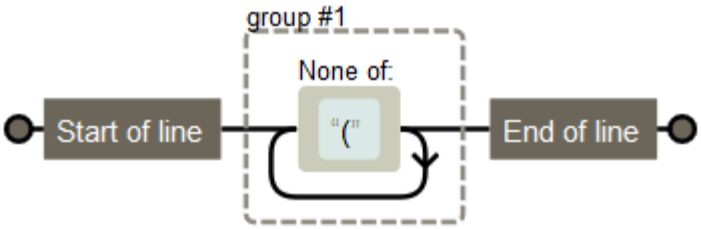
Artist Name 1 - Label Name 1 - Track Name 1 (Remix 1)
Ignore

Artist Name 2 - Label Name 2 - Track Name 2
to
Artist Name 2 - Label Name 2 - Track Name 2 (Original Mix)

Match: ^([^(]+)$
Replace: \1 (Original Mix)

Name	New Name
Artist Name 1 - Label Name 1 - Track Name 1 (Remix 1)	Artist Name 1 - Label Name 1 - Track Name 1 (Remix 1)
Artist Name 2 - Label Name 2 - Track Name 2	Artist Name 2 - Label Name 2 - Track Name 2 (Original Mix)





Sample:

```
.....................................................
Match 1: Artist Name 2 - Label Name 2 - Track Name 2
Group 1: Artist Name 2 - Label Name 2 - Track Name 2
```

Contributor and *Tim Analysis:*

1. ^ Match must begin at the start of the name (Beginning of Line or BOL)
2. [^(] Match only any character that is NOT a ' (' …. Capture Group 1
4. + .. one or more times Capture Group 1
6. $ .. until we hit the EOL (End of Line)

7. This will therefore only match lines in which, from start to finish, there are no opening parenthesis.
8. If it does match, the whole filename is captured and stored in Capture Group 1.
   a. We use \1 and append '(Original Mix)' for the replace.
9. If it doesn't match, it is ignored and left unchanged.

Forum Contributor: Jane

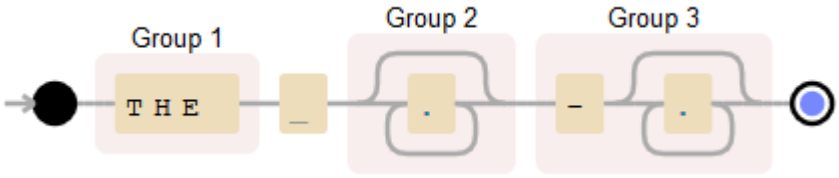
String Manipulation (Parsing)

Move 'The' to end of artist up to hyphen

THE BEATLES-ST.PEPPER to BEATLES,THE-ST.PEPPER
THE EAGLES-HOTEL CALIFORNIA to EAGLES,THE-HOTEL CALIFORNIA

Match: (THE) (.*)(-.*)
Replace: \2,\1\3

Name	New Name
THE BEATLES-ST.PEPPER	BEATLES,THE-ST.PEPPER
THE EAGLES-HOTEL CALIFORNIA	EAGLES,THE-HOTEL CALIFORNIA



Sample:

```
................................
Match 1: THE BEATLES-ST.PEPPER
Group 1: THE
Group 2: BEATLES
Group 3: -ST.PEPPER
```

Forum Contributor: KenP

String Manipulation (Parsing)

Move 'The' to end of artist up to hyphen cont.

Using: THE BEATLES-ST.PEPPER

Match: (THE) (.*)(-.*)

Where:

\1 = THE \2 = BEATLES \3 = -ST.PEPPER

Tim Analysis

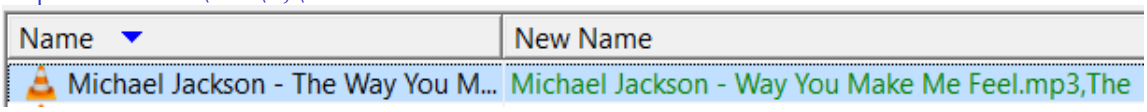
1. (THE)	Match against a string, 'THE'	Capture Group 1 = THE
2. <space>	Match against <space> character (Not Captured)	
3. (.*)	Match against remainder of string	Capture Group 2 = BEATLES-ST.PEPPER
4. -	Match against <hyphen>	Capture Group 3 = - Changes Capture Group 2 = BEATLES
5. .*	Match against remainder of string	Capture Group 3 = -ST.PEPPER

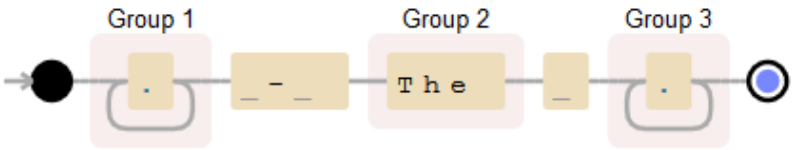
String Manipulation (Parsing)

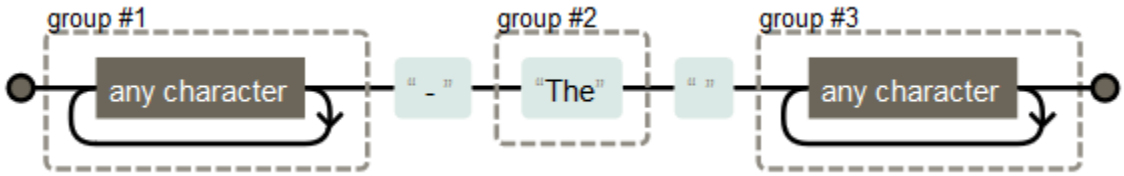
Move 'The, 'A', 'An' (insignificant words) to End of Filename

Michael Jackson - The Way You Make Me Feel.mp3 to Michael Jackson - Way You Make Me Feel, The.mp3

Match: (.+) - (The) (.+)
Replace: \1 - \3,\2







Sample:

```
.....................................................
Match 1: Michael Jackson - The Way You Make Me Feel
Group 1: Michael Jackson
Group 2: The
Group 3: Way You Make Me Feel
```

Where:

\1 = Michael Jackson \2 = The \3 = Way You Make Me Feel

Tim Analysis:

1. .	Match against any character	Capture Group 1 = M
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	<space> (Not Captured), followed by .. Backtracking.	Changes Capture Group 1 = Michael Jackson - The Way You Make Me<space>
4. -	.. Match against <hyphen> (Not Captured) followed by…	Changes Capture Group 1 = Michael Jackson
5. <space>	.. <space> (Not Captured), followed by ..	
6. (The)	.. Match against string, The' followed by …	Capture Group 2 = The
7. <space>	.. <space> (Not Captured)	
8. .+	Match against remaining string	Capture Group 3 = Way You Make Me Feel

String Manipulation (Parsing)

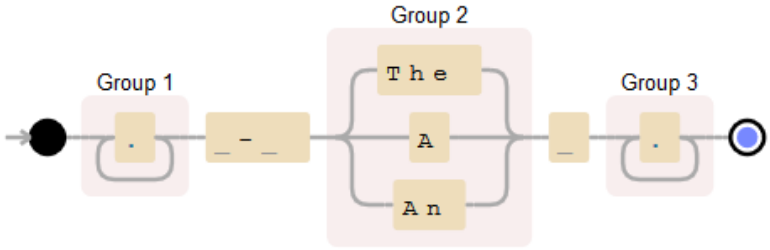
Move 'The, 'A', 'An' (insignificant words) to End of Filename cont.

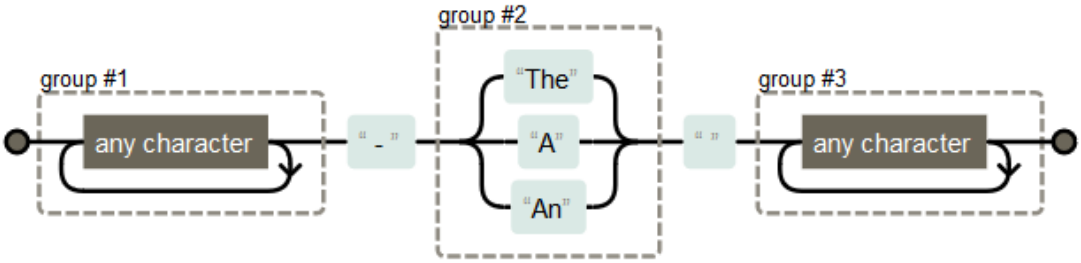
Michael Jackson - The Way You Make Me Feel.mp3 to Michael Jackson - Way You Make Me Feel, The.mp3

Or -

Match: (.+) - (The|A|An) (.+)
Replace: \1 - \3,\2

Name	New Name
Michael Jackson - The Way You M...	Michael Jackson - Way You Make Me Feel.mp3,The





Sample:

```
Match 1: Michael Jackson - The Way You Make Me Feel
Group 1: Michael Jackson
Group 2: The
Group 3: Way You Make Me Feel
```

Contributor Analysis:

1. Match either 'The' or 'A' or 'An' following <hyphen>.

Tim Notes:

1. A good example of using three alternates.
2. Another Similar one to 'Move "The" to end'

Forum Contributors: Narcissus, Admin

String Manipulation (Parsing)

Move 'The, 'A', 'An' (insignificant words) to End of Filename cont.

Michael Jackson - The Way You Make Me Feel.mp3 to Michael Jackson - Way You Make Me Feel, The.mp3

Match: (.+) - (The|A|An) (.+)

Where:

\1 = Michael Jackson \2 = The \3 = Way you Make Me Feel

This uses an Alternate for Capture Group 1:

This	OR	This	OR	This
The	|	A	|	An

Tim Analysis:

1. .	Match against any character	Capture Group 1 = M
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	<space> (Not Captured), followed by .. Backtracking.	Changes Capture Group 1 = Michael Jackson - The Way You Make Me<space>
4. -	.. Match against <hyphen> (Not Captured) followed by… RegEx Engine moving forward	Changes Capture Group 1 = Michael Jackson
5. <space>	.. <space> (Not Captured), followed by ..	
6. The | A |An	Tests for a Match for string, "The' or 'A' or 'An', whichever comes first	Capture Group 2 = The
7. <space>	.. <space> (Not Captured)	
8. .+	Match against remaining string	Capture Group 3 = Way You Make Me Feel

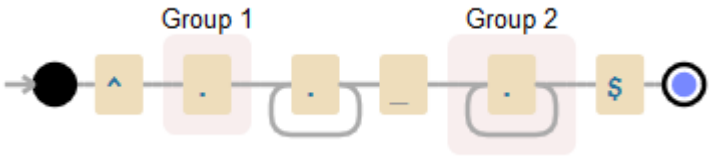
String Manipulation (Parsing)

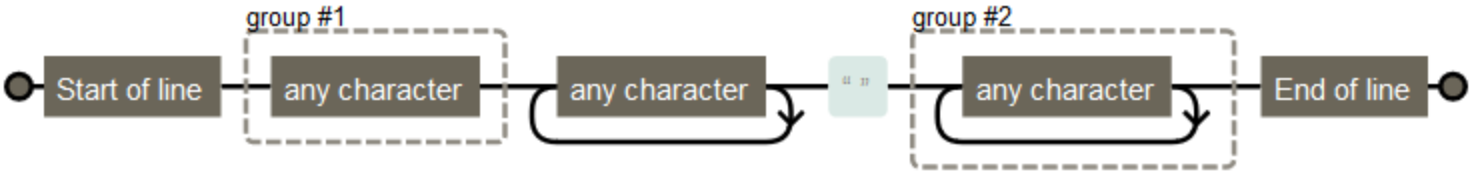
Convert <first name> <space> <surname> ~to~ <first Initial> <surname>

giles smith to gsmith

Match: ^(.).+ (.+)$
Replace: \1\2

Name	New Name
giles smith	gsmith





Sample:

```
.......................
Match 1: giles smith
Group 1: g
Group 2: smith
```

Where:

Forum Contributor: gilessmith \1 =g \2 = smith

Tim Analysis:

1.^	Start at beginning of string (Beginning of Line or BOL)		
2. .	Match against first character followed by …		Capture Group 1 = g
3. .	. another character (Not Captured)	= i	
4. +	Make it Greedy (Not Captured) until ….	= iles	
5. <space>	.. <space> (Not Captured) followed by…	= iles<space>	
6. .	.. a single character		Capture Group 2 = s
7. +	Make it Greedy		Capture Group 2 = smith
8. $	Must be at end of Line (EOL)		

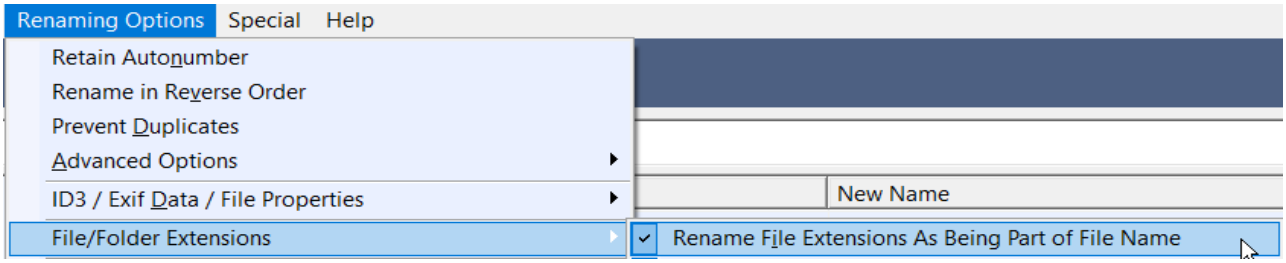
Tim Notes:

1. The $ is an anchor and here $ is used to indicate that after the .+, must be at the end of the string (End of Line or EOL) to be TRUE. The last expression must be matched at the and of the string.

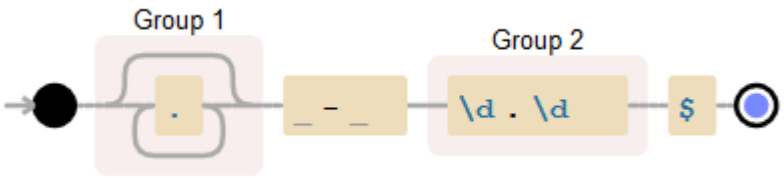
String Manipulation (Parsing)

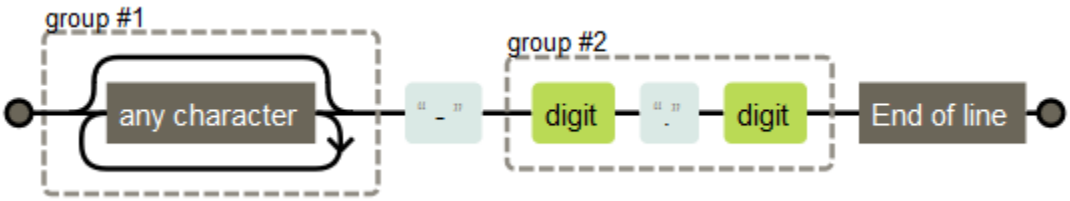
Move End to Beginning (Numeric Digits)

An Overview of DNS - CompTIA Network+ N10-006 - 1.3
to
1.3 - An Overview of DNS - CompTIA Network+ N10-006



Match: (.*) - (\d\.\d)$
Replace: \2 - \1





Sample:

```
.....................................................................
Match 1: An Overview of DNS - CompTIA Network+ N10-006 - 1.3
Group 1: An Overview of DNS - CompTIA Network+ N10-006
Group 2: 1.3
```

Forum Contributor: KenP

String Manipulation (Parsing)

Move End to Beginning (Numeric Digits) cont.

Using: An Overview of DNS - CompTIA Network+ N10-006 - 1.3

Match: (.*) - (\d\.\d)$

Where:

\1 = An Overview of DNS - CompTIA Network+ N10-006 \2 = 1.3

Tim Analysis:

1. .	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	<space> (Not Captured), followed by ..	Changes Capture Group 1 = An Overview of DNS - CompTIA Network + N10-006 <space>-<space>
4. -	Match against <hyphen> (Not Captured), followed by …	Changes Capture Group 1 = An Overview of DNS - CompTIA Network + N10-006 <space>
5.<space>	<space> (Not Captured), followed by ..	Changes Capture Group 1 = An Overview of DNS - CompTIA Network+ N10-006
6. \d	Match against Numeric Digit followed by	Capture Group 2 = 1
7. \.	Match against Literal <dot>	Capture Group 2 = 1.
8. \d	Match against Numeric Digit	Capture Group 2 = 1.3
9. $	Must be at EOL	

String Manipulation (Parsing)

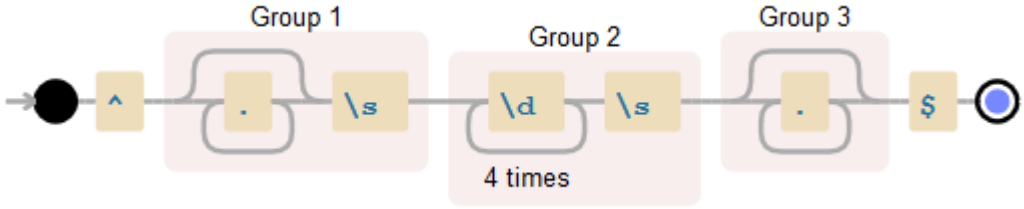
Move 4-digit year (within filename) to beginning of file

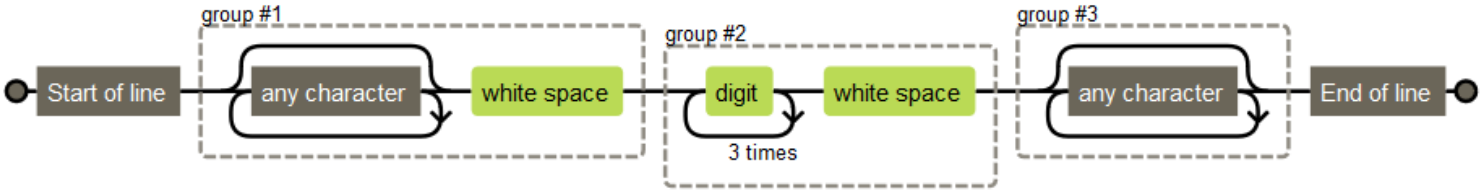
Escape from Alcatraz 1979 1080p
to
1979 Escape from Alcatraz 1080p

The Texas Chainsaw Massacre The Beginning 2006 1080p
to
2006 The Texas Chainsaw Massacre The Beginning 1080p

Match: ^(.*\s)(\d{4}\s)(.*)$
Replace: \2\1\3

Name	New Name
Escape from Alcatraz 1979 1080p	1979 Escape from Alcatraz 1080p
The Texas Chainsaw Massacre The Beginning 2006 1080p	2006 The Texas Chainsaw Massacre The Beginning 1080p





Sample:

```
.........................................
Match 1: Escape from Alcatraz 1979 1080p
Group 1: Escape from Alcatraz
Group 2: 1979
Group 3: 1080p
```

Forum Contributor: KenP

String Manipulation (Parsing)

Move 4-digit year (within filename) to beginning of file cont.

Using: Escape from Alcatraz 1979 1080p

Match: ^(.*\s)(\d{4}\s)(.*)$

Where:

\1 = Escape from Alcatraz <space>

\2 = 1979 <space>

\3 = 1080p

Tim Analysis:

Here is an example of using the Whitespace Metacharacter, \s. This is defined as any character that is a <space>, <tab> and <new line> (line break). Because BRU does not support tabs or new lines, this metacharacter represents a <space> character.

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .*	Match against all characters until…	Capture Group 1 = Escape from Alcatraz
3. \s	.. <space>, followed by …	Capture Group 1 = Escape from Alcatraz <space>
4. \d{4}	.. 4 numeric digits (Year), followed by …	Capture Group 2 = 1979
5. \s	.. <space>, followed by ..	Capture Group 2 = 1979<space>
6. .*	Capture remainder of string	Capture Group 3 = 1080p
7. $	Must be at End of Line (EOL)	

String Manipulation (Parsing)

Moving a Date (within filename) to the Front of the File Name

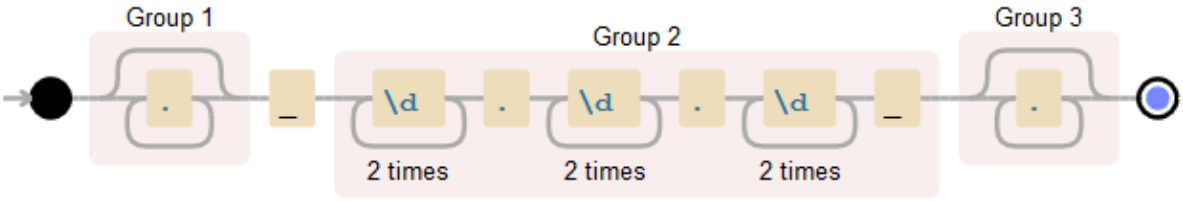
Client_2343421_04.21.17__12123.00.pdf to 04.21.17_Client_2343421_12123.00.pdf
SecondClient_1334_09.31.16__300.00.pdf to 09.31.16_SecondClient_1334_300.pdf

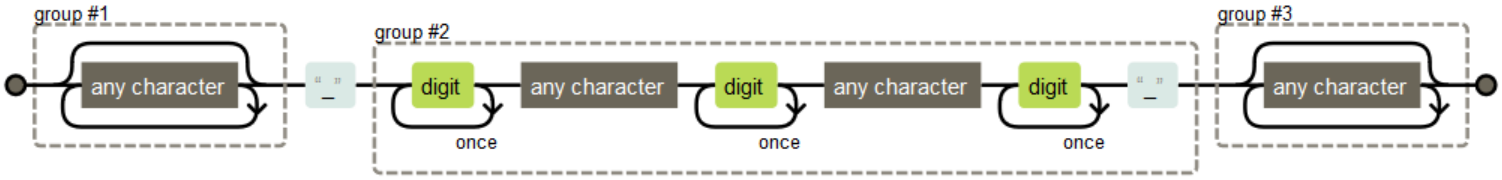
Observations:

1. The date format is as always MMDDYY, i.e., 09.27.17

Match: (.*)_(\d{2}.\d{2}.\d{2}_)(.*)
Replace: \2\1\3

Name	New Name
Client_2343421_04.21.17__12123.00.pdf	04.21.17_Client_2343421_12123.00.pdf
SecondClient_1334_09.31.16__300.00.pdf	09.31.16_SecondClient_1334_300.00.pdf



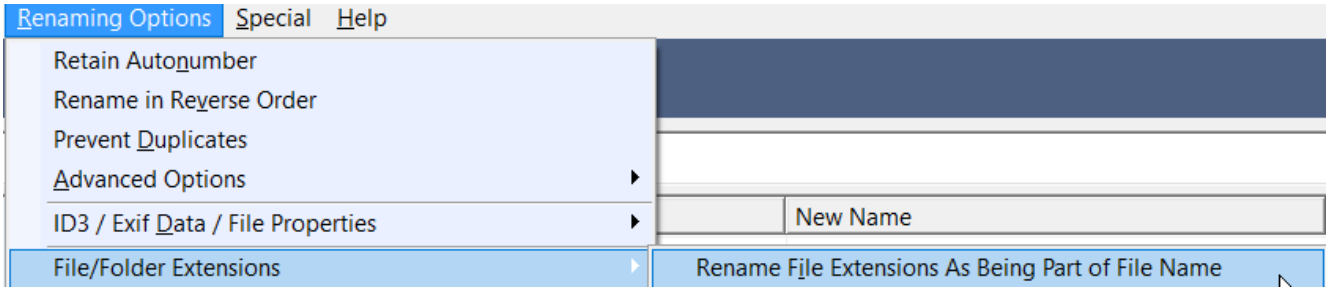


Sample:

```
Match 1: Client_2343421_04.21.17__12123.00
Group 1: Client_2343421
Group 2: 04.21.17_
Group 3: _12123.00
```

Tim Notes:

1. Although Samples have <dot> characters in filenames, do NOT enable File Extensions to achieve the desired outcome.



Forum Contributor: KenP

String Manipulation (Parsing)

Moving a Date (within filename) to the Front of the File Name cont.

Using: Client_2343421_04.21.17__12123.00.pdf

Match: (.*)_(\d{2}.\d{2}.\d{2}_)(.*)

Where:

\1 = Client_2343421 \2 = 04.21.17_ \3 = _12123.00

Tim Analysis:

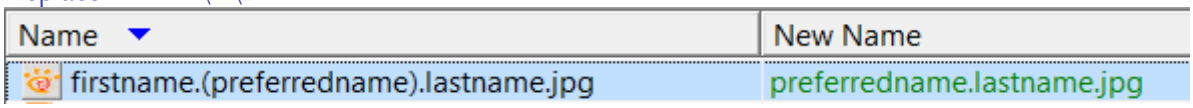
1. .	Match against any character	Capture Group 1 = C
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore character (Not Captured)	Changes Capture Group 1 = Client_2343421_04.21.17__
4. \d {2}	Match against 2 numeric digits	Capture Group 2 = 12
5. .	Match against any character	Capture Group 2 = 121
6. \d {2}	Match against 2 numeric digits	Capture Group 2 = 12123
7. .	Match against any character	Capture Group 2 = 12123.
8. \d {2}	Match against 2 numeric digits	Capture Group 2 = 12123.00
9. _	Match against underscore character	Capture Group 2 = 04.21.17_ Changes Capture Group 1 = Client_2343421
10. .*	Match against remaining string	Capture Group 3 = _12123.00

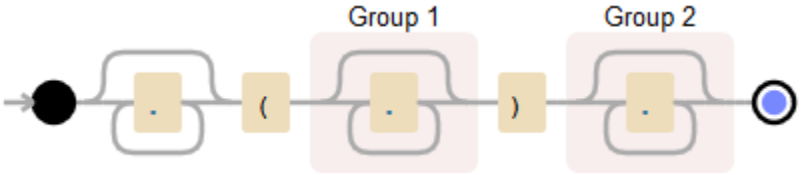
String Manipulation (Parsing)

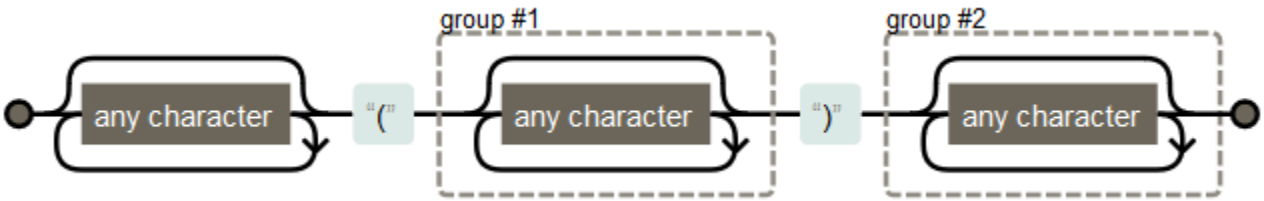
Rearrange Removing First Name and parentheses

firstname.(preferredname).lastname.jpg to preferredname.lastname.jpg

Match: .*\((.*)\)(.*)
Replace: \1\2





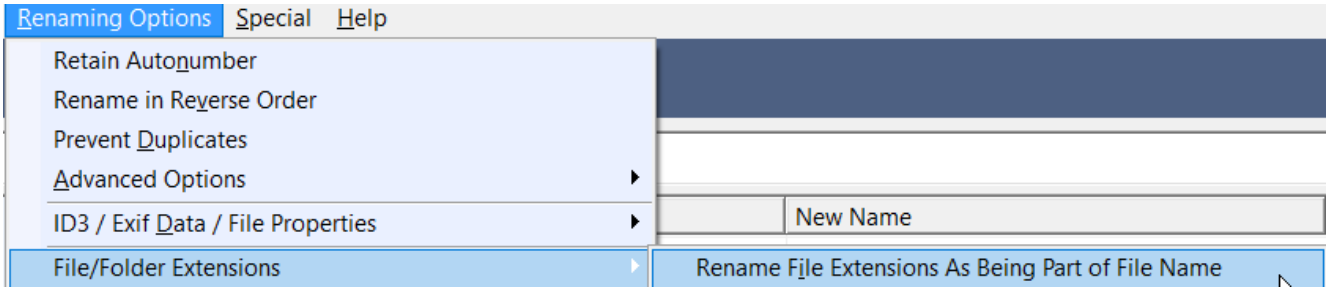


Sample:

```
Match 1: firstname.(preferredname).lastname
Group 1: preferredname
Group 2: .lastname
```

Tim Notes:

1. Although Samples have <dot> characters in filenames, do NOT enable File Extensions to achieve the desired outcome.



Forum Contributor: KenP

String Manipulation (Parsing)

Rearrange Removing First Name and parentheses cont.

Using: firstname.(preferredname).lastname.jpg

Match: .*\((.*)\)(.*)

Where:

\1 = preferredname \2 = .lastname

Tim Analysis:

1. .	Match against any character (Not Captured) = f	
2. *	Make it Greedy (Not Captured) = <entire string>	
2. \(	Match against a literal left parenthesis (Not Captured) = 'firstname.('	
3. .*	Match against remaining string	Capture Group 1 = (preferredname).lastname
4. \)	Match against a literal right parenthesis (Not Captured)	Changes Capture Group 1 = preferredname
5. .*	Match against remaining string	Capture Group 2 = .lastname

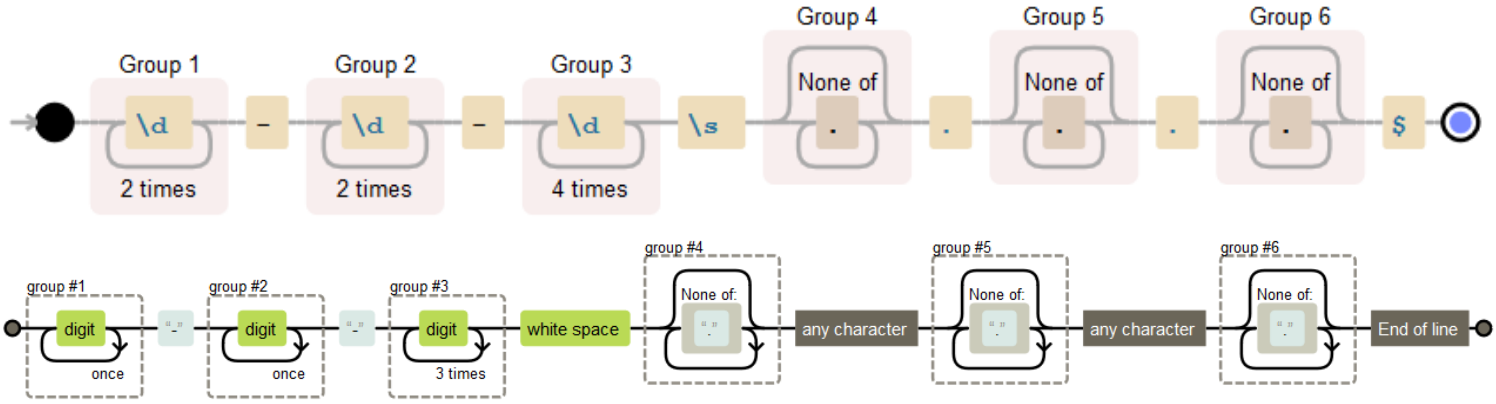
String Manipulation (Parsing)

Moving around digits in date for filename

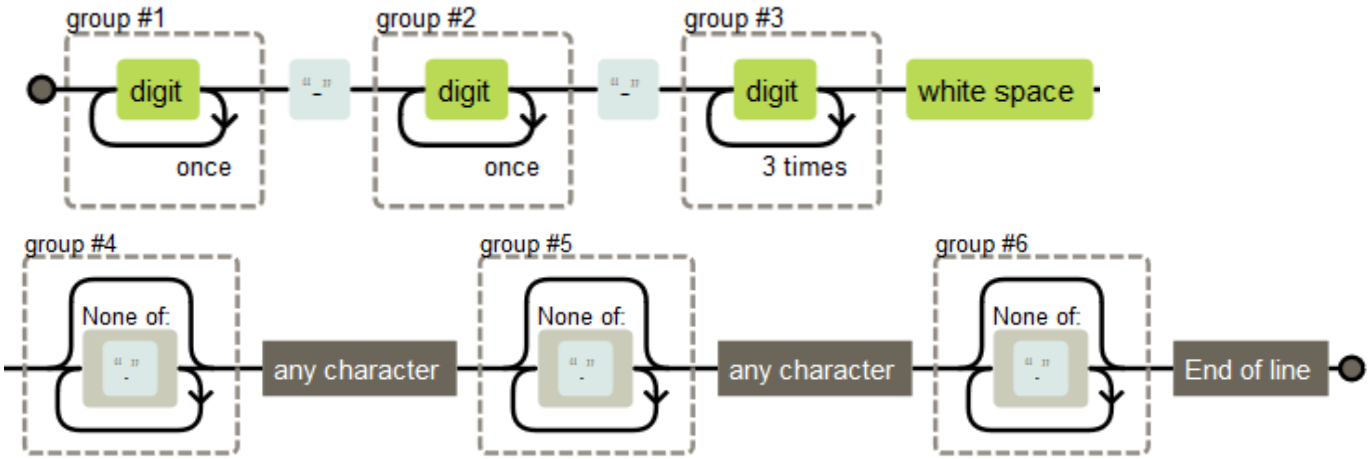
Video 10-08-2015 19.08.50 to 2015-08-10-190850

Match: (\d{2})-(\d{2})-(\d{4})\s([^.]*).([^.]*).([^.]*)$
Replace: \3-\2-\1-\4\5\6

Name	New Name
Video 10-08-2015 19.08.50	2015-08-10-190850



Expanded:



Sample:

```
..............................
Match 1: 10-08-2015 19.08.50
Group 1: 10
Group 2: 08
Group 3: 2015
Group 4: 19
Group 5: 08
Group 6: 50
```

Tim Notes:

1. I have previously explained Negated Classes so I won't repeat myself.
2. Here they are used to ignore the dot characters and move forward through the string to capture all of the text without including the dot characters.
3. Samples have <dot> characters in filenames. Enable File Extensions to achieve the desired outcome ('Rename File Extensions As Being Part of File Name' from Renaming Options menu).

Forum Contributor: KenP

String Manipulation (Parsing)

Moving around digits in date for filename cont.

Using: Video 10-08-2015 19.08.50

Match: (\d{2})-(\d{2})-(\d{4})\s([^.]*).([^.]*).([^.]*)$

Where:

\1 = 10 \2 = 08 \3 = 2015 \4 = 19 \5 = 08 \6 = 50

Tim Analysis:

1. \d {2}	Match against 2 numeric digits followed by…	Capture Group 1 = 10
2. -	.. Match against <hyphen> (Not Captured) followed by …	
3. \d {2}	.. Match against 2 numeric digits followed by…	Capture Group 2= 08
4. -	.. Match against <hyphen> (Not Captured) followed by …	
5. \d {4}	.. Match against 4 numeric digits followed by …	Capture Group 3 = 2015
6. \s	.. Match against <space> character (Not Captured)	
7. [^.]	Match against a Negated Class consisting of a <dot> Match anything but a literal <dot>	Capture Group 4 = 1
8. *	Make it Greedy	Capture Group 4 = 19
9. .	Match against any character (Not Captured)	
10. [^.]	Match against a Negated Class consisting of a <dot> Match anything but a literal <dot>	Capture Group 5 = 0
11. *	Make it Greedy (This is the '08' after '19.')	Capture Group 5 = 08
12. .	Match against any character (Not Captured)	
13. [^.]	Match against a Negated Class consisting of a <dot> Match anything but a literal <dot>	Capture Group 6 = 5
14. *	Make it Greedy	Capture Group 6 = 50
15. $	Must be at EOL	

String Manipulation (Parsing)

Rearrange Word and remove first underscore

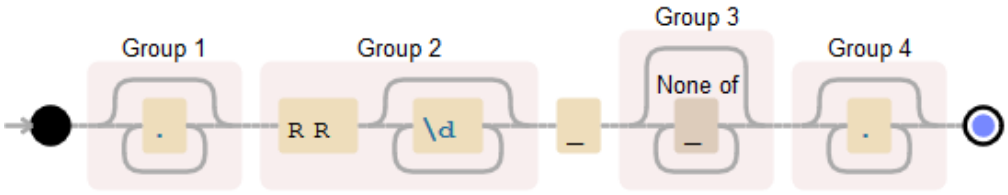
HiHatRR1_China_Choke_1_50_27 to HiHatChinaRR1_Choke_1_50_27

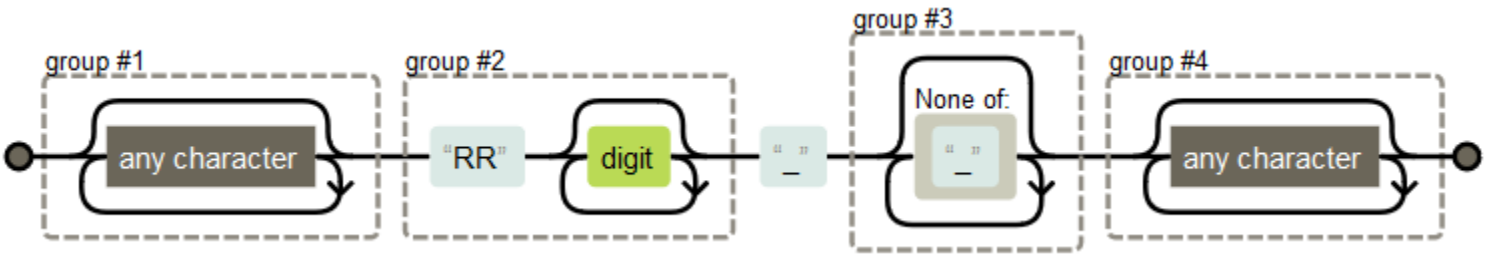
Observations:

1. Move the section after the first underscore and place it before 'RR' and delete the first underscore.
2. Filenames will always have the 'RR'.

Match: (.*)(RR\d*)_([^_]*)(.*)
Replace: \1\3\2\4

Name	New Name
HiHatRR1_China_Choke_1_50_27	HiHatChinaRR1_Choke_1_50_27





Sample:

```
.....................................
Match 1: HiHatRR1_China_Choke_1_50_27
Group 1: HiHat
Group 2: RR1
Group 3: China
Group 4: _Choke_1_50_27
```

Forum Contributor: KenP

String Manipulation (Parsing)

Rearrange Word and remove first underscore cont.

Using: HiHatRR1_China_Choke_1_50_27

Match: (.*)(RR\d*)_([^_]*)(.*)

Where:

\1 = HiHat \2 = RR1 \3 = China \4 = _Choke_1_50_27

Tim Analysis:

1. .	Match against any character	Capture Group 1 = H
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. RR	Match against a string 'RR'	Capture Group 2 = RR Changes Capture Group 1 = HiHat
4. \d	Match against Numeric Digit	Capture Group 2 =RR1
5. *	Make it Greedy	Capture Group 2 = No Change
6. _	Match against underscore (Not Captured)	
7. [^_]	Match against a Negated Class consisting of an underscore character	Capture Group 3 = C
8. *	Make it Greedy	Capture Group 3 = China
9. .*	Match against remaining string	Capture Group 4 = _Choke_1_50_27

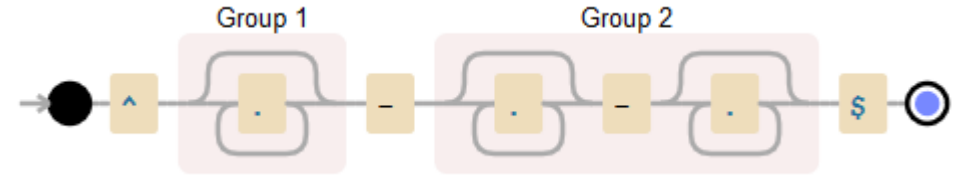
String Manipulation (Parsing)

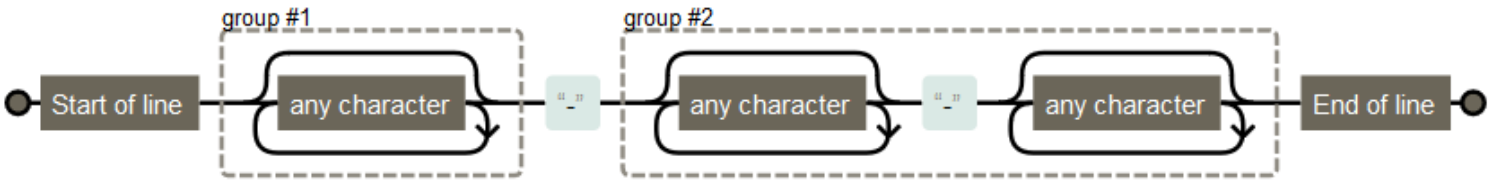
Change second hyphen to underscore

2016-07-23_(Textblock1)-(Textblock2)-(Textblock3)-(Textblockn)
to
2016-07-23_(Textblock1)-(Textblock2)_(Textblockn-3)-(Textblockn)

Match: ^(.*)-(.*-.*)$
Replace: \1_\2

Name	New Name
2016-07-23_(Textblock1)-(Textblock2)-(Textblock3)-(Textblockn)	2016-07-23_(Textblock1)-(Textblock2)_(Textblock3)-(Textblockn)





Sample:

```
.....................................................................
Match 1: 2016-07-23_(Textblock1)-(Textblock2)-(Textblock3)-(Textblockn)
Group 1: 2016-07-23_(Textblock1)-(Textblock2)
Group 2: (Textblock3)-(Textblockn)
```

Forum Contributor: KenP

String Manipulation (Parsing)

Change second hyphen to underscore cont.

Using: 2016-07-23_(Textblock1)-(Textblock2)-(Textblock3)-(Textblockn)

Match: ^(.*)-(.*-.*)$

Where:

\1 = 2016-07-23_(Textblock1)-(Textblock2) \2 = (Textblock3)-(Textblockn)

Tim Analysis:

1. ^	Start at Beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = 2
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. -	Match against <hyphen> (Not Captured)	Changes Capture Group 1 = 2016-07-23_(Textblock1)-(Textblock2)-(Textblock3)
2. .	Match against any character	Capture Group 2 = (
3. *	Make it Greedy	Capture Group 2 = (Textblockn)
4. -	Match against <hyphen>	Changes Capture Group 2 = (Textblock3)- Changes Capture Group 1 = 2016-07-23_(Textblock1)-(Textblock2)
2. .	Match against any character	Capture Group 2 = (Textblock3)-(
3. *	Make it Greedy	Capture Group 2 = (Textblock3)-(Textblockn)
4. $	Must be at EOL	

String Manipulation (Parsing)

Parsing at its simplest - Reformat Items with comma

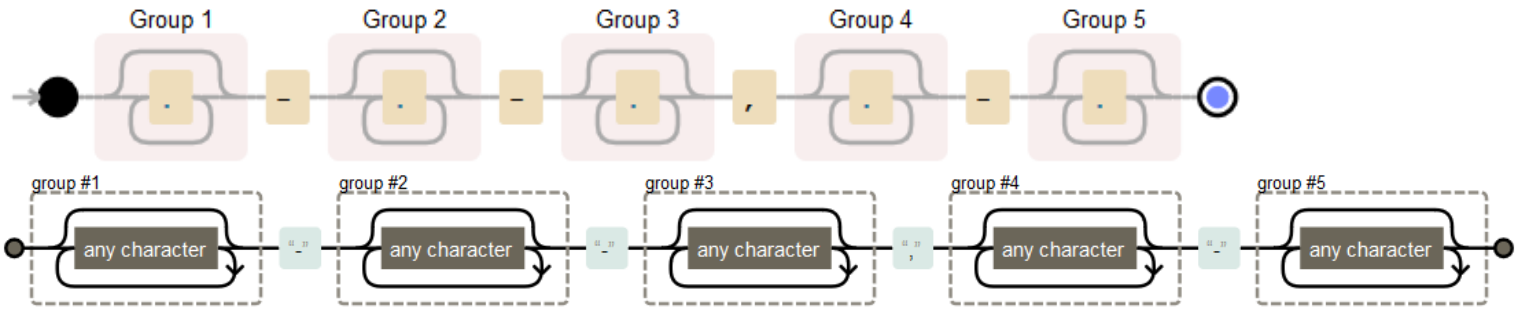
Tim Notes:

1. Here is a simple example of parsing a string:

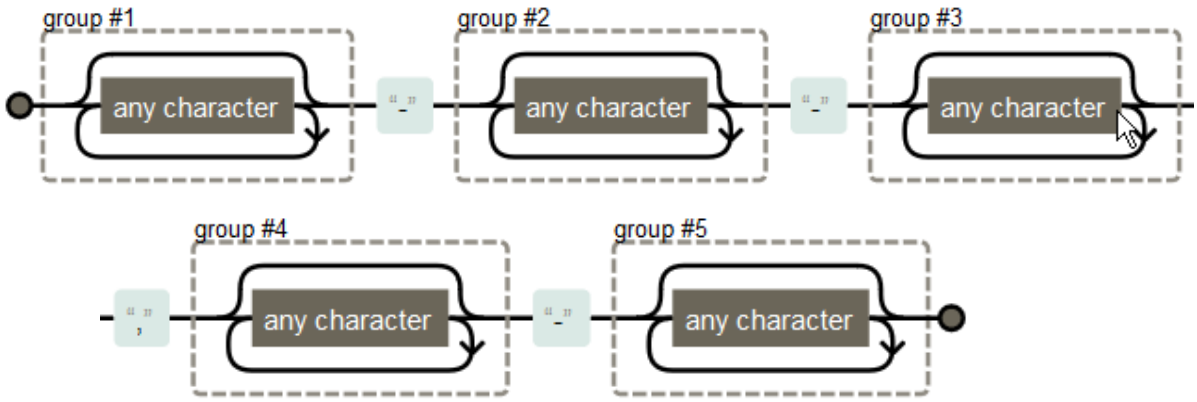
album-number-last,first-song name.zip to album-number-first last-song name.zip

Match: (.*)-(.*)-(.*),(.*)-(.*)
Replace: \1-\2-\4 \3-\5

Name	New Name
album-number-last,first-song name.zip	album-number-first last-song name.zip



Expanded:



Sample:

```
Match 1: album-number-last,first-song name
Group 1: album
Group 2: number
Group 3: last
Group 4: first
Group 5: song name
```

Forum Contributor: j.zulkeski

String Manipulation (Parsing)

Parsing at its simplest - Reformat Items with comma cont.

Using: album-number-last,first-song name.zip

Match: (.*)-(.*)-(.*),(.*)-(.*)

Where:

\1 = album \2 = number \3 = last \4 = first \5 = song name

Tim Analysis:

1..	Match against any character	Capture Group 1 = a
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'first'	Changes Capture Group 1 = album-number-last,first
4. .	Match against any character	Capture Group 2 = s
5. *	Make it Greedy Current position = EOL	Capture Group 2 = song name
6. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'first' Changes <hyphen> in step 3 to <hyphen> after 'number'	Changes Capture Group 2 = last,first Changes Capture Group 1 = album-number
7..	Match against any character	Capture Group 3 = s
8. *	Make it Greedy Current position = EOL	Capture Group 3 = song name
9. ,	Match against <comma> (Not Captured) Backtracks.	Changes Capture Group 3 = last Changes Capture Group 2 = number Changes Capture Group 1 = album
10..	Match against any character	Capture Group 4 = f
11. *	Make it Greedy Current position = EOL	Capture Group 4 = first-song name
12. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'first'	Changes Capture Group 4 = first
13. .*	Match against remaining string	Capture Group 5 = song name

Tim Notes:

1. Another example where a simple coded RegEx may not be the best solution.

String Manipulation (Parsing)

Parsing at its simplest - Reformat Items with comma cont.

Using: album-number-last,first-song name.zip

Match: (.*)-(.*)-(.*),(.*)-(.*)

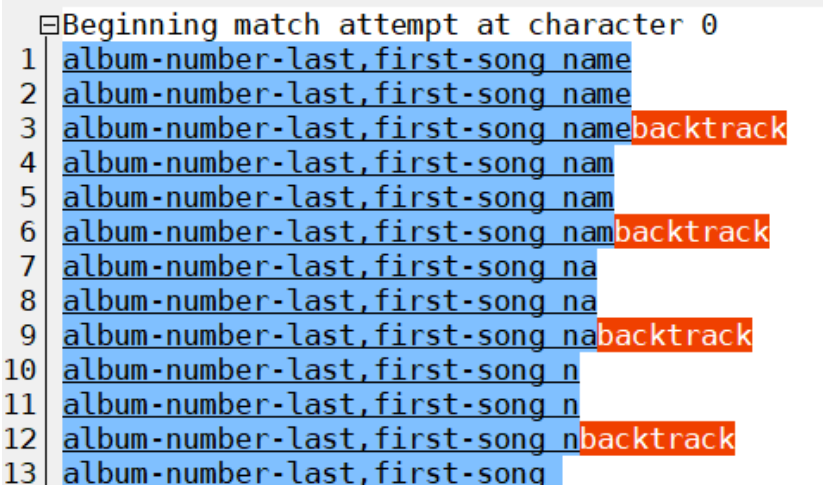
Where:

\1 = album \2 = number \3 = last \4 = first \5 = song name

Tim Analysis: cont.

Sample:

```
.........................................
Match 1: album-number-last,first-song name
Group 1: album
Group 2: number
Group 3: last
Group 4: first
Group 5: song name
```



Tim Notes:

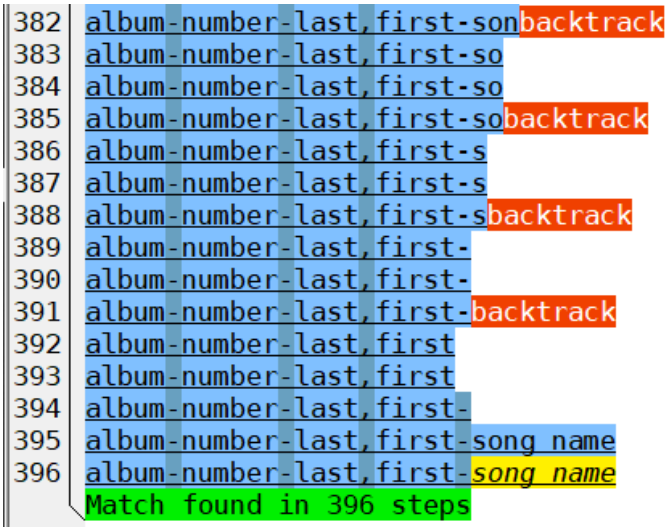
1. This seemingly simple solution took 396 steps!

2. Will create a lag time if processing a lot of files.

3. Simple solutions are not always the best – easy to code but hard to analyze and execute.

4. Anytime you use a wildcard with a Greedy Quantifier, you are going to have backtracking with constant changing values.

5. If you add multiple wildcard/Greedy combinations then you can see in the diagrams at the right the kind of mess this creates.



String Manipulation (Parsing)

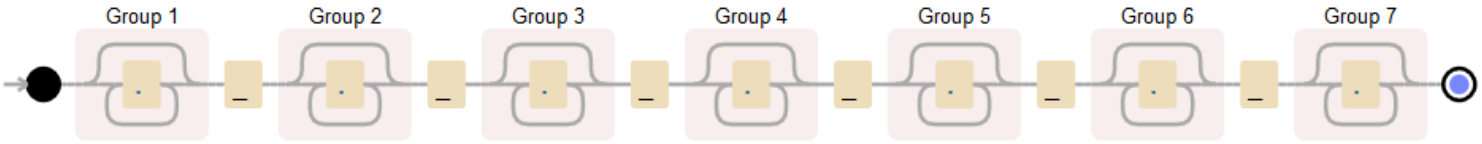
Another Real Simple Example of Re-arrangement

Spr16_51_5881_ELA_10_XXXXX1234X_SMITH-JOHN.pdf
to
Spr16_51_5881_ELA_10_SMITH-JOHN.pdf

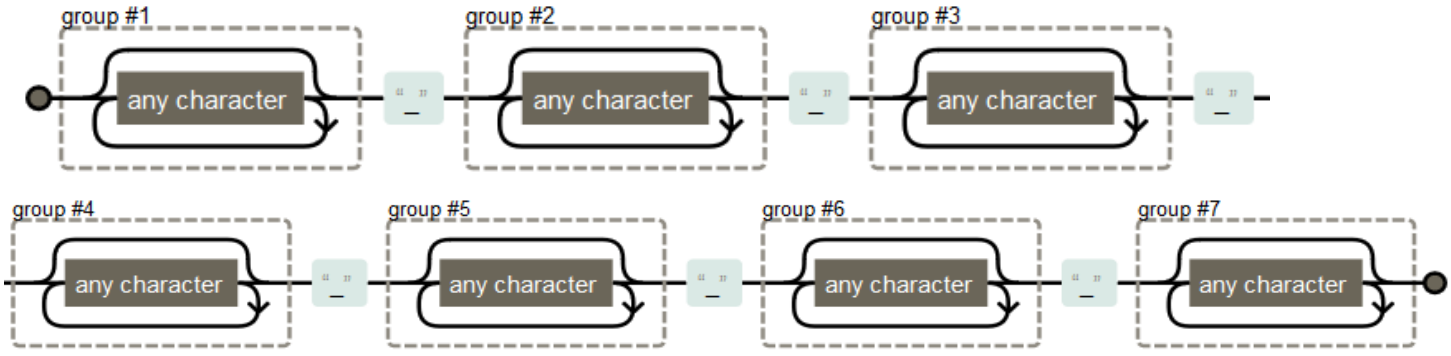
Spr16_51_0021_MAT_03_XXXXX4321X_JONES-LISA.pdf
to
Spr16_51_0021_MAT_03_JONES-LISA.pdf

Match: (.*)_(.*)_(.*)_(.*)_(.*)_(.*)_(.*)
Replace: \1_\2_\3_\4_\5_\7

Name	New Name
Spr16_51_5881_ELA_10_XXXXX1234X_SMITH-JOHN.pdf	Spr16_51_5881_ELA_10_SMITH-JOHN.pdf
Spr16_51_0021_MAT_03_XXXXX4321X_JONES-LISA.pdf	Spr16_51_0021_MAT_03_JONES-LISA.pdf



Expanded:



Sample:

```
Match 1: Spr16_51_5881_ELA_10_XXXXX1234X_SMITH-JOHN
Group 1: Spr16
Group 2: 51
Group 3: 5881
Group 4: ELA
Group 5: 10
Group 6: XXXXX1234X
Group 7: SMITH-JOHN
```

Forum Contributor: the rube

String Manipulation (Parsing)

Another Real Simple Example of Re-arrangement cont.

Using: Spr16_51_5881_ELA_10_XXXXX1234X_SMITH-JOHN.pdf

Match: (.*)_(.*)_(.*)_(.*)_(.*)_(.*)_(.*)

Where:

\1 = Spr16 \2 = 51 \3 = 5881 \4 = ELA \5 = 10

\6 = XXX1234X \7 = SMITH-JOHN

Tim Contribution –

Now that the string is parsed, it can be put back together using the Replace field in any order.

The original Poster also wanted this:

Spr16_51_5881_ELA_10_XXXXX1234X_SMITH-JOHN.pdf
to
SMITH-JOHN_Spr16_51_5881_ELA_10_XXXXX1234X.pdf

Spr16_51_0021_MAT_03_XXXXX4321X_JONES-LISA.pdf
to
JONES-LISA_Spr16_51_0021_MAT_03_XXXXX4321X.pdf

Just change around the Replace to:

Replace: \7_\1_\2_\3_\4_\5_\6

Name	New Name
Spr16_51_5881_ELA_10_XXXXX1234X_SMITH-JOHN.pdf	SMITH-JOHN_Spr16_51_5881_ELA_10_XXXXX1234X.pdf
Spr16_51_0021_MAT_03_XXXXX4321X_JONES-LISA.pdf	JONES-LISA_Spr16_51_0021_MAT_03_XXXXX4321X.pdf

Tim Notes:

1. Wow. Take a look at the previous example, 'Parsing at its simplest - Reformat Items with comma' to get a sense of what analyzing this would take. There isn't too much difference in coding so I will let stand the previous analysis and notations about using multiple wildcard/Greedy combinations.

2. Here's a perfect example for you to do a little self-analysis and in performing this task, it may teach you to be better Coders. But as I said at the beginning of the book, I am not trying to teach you to be efficient coders, that can come with time. I am trying to teach you the tools and how to use them. For now, as long as it does the task, it is good enough.

String Manipulation (Parsing)

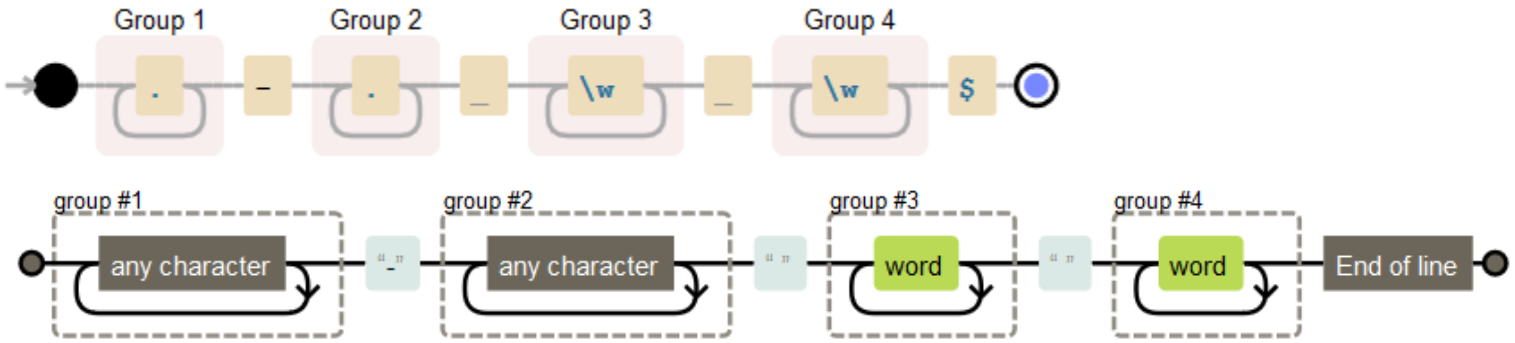
Moving the last 2 words of any phrase

2015-01-01 - Market Center Alpha San Jose.zip
to
2015-01-01 - San Jose - Market Center Alpha.zip

2015-01-10 - PRIMARY PUBLIC EDUCATION MATERIALS TWIN C.zip
to
2015-01-10 - TWIN C - PRIMARY PUBLIC EDUCATION MATERIALS.zip

Match: (.+)-(.+) (\w+) (\w+)$
Replace: \1- \3 \4 -\2

Name	New Name
2015-01-10 - PRIMARY PUBLIC EDUCATION ...	2015-01-10 - TWIN C - PRIMARY PUBLIC EDUCATION MATERIALS.zip
2015-01-01 - Market Center Alpha San Jose	2015-01-01 - San Jose - Market Center Alpha



Sample:

```
Match 1: 2015-01-10 - PRIMARY PUBLIC EDUCATION MATERIALS TWIN C
Group 1: 2015-01-10
Group 2:  PRIMARY PUBLIC EDUCATION MATERIALS
Group 3: TWIN
Group 4: C
```

Under Regex Buddy for this test, had to remove anchor $ (EOL) because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.zip' is just another part of the string of text.

Forum Contributor: RobsterUK

String Manipulation (Parsing)

Moving the last 2 words of any phrase cont.

Using: 2015-01-01 - Market Center Alpha San Jose.zip

Match: (.+)-(.+) (\w+) (\w+)$

Where:

\1 = 2015-01-01<space> \2 = <space>Market Center Alpha \3 = San \4 =Jose

Tim Analysis:

1. .	Match against first character	Capture Group 1 = 2
2. +	Make it Greedy.. until ..	Capture Group 1
3.-	.. <hyphen> (Not Captured)	

Tim Notes:

Because in step 2 it was made Greedy, Capture Group 1 changes to capture all before the last <hyphen>

		Changes Capture Group 1 = 2015-01-01<space>
4. .	Match against next character	Capture Group 2 = <space>
5. +	Make it Greedy until ….	Capture Group 2 = <space>Market Center Alpha San Jose
6. <space>	.. <space> (Not Captured), followed by …	Capture Group 2 = <space>Market Center Alpha San

Tim Notes:

Backtracks at this point to find the first <space> encountered which is right after before 'Jose'

7. \w	.. Word character	Capture Group 3 = J
8. +	Make it Greedy until …	Capture Group 3 =Jose

String Manipulation (Parsing)

Moving the last 2 words of any phrase cont.

Tim Analysis: cont.

9. <space>	.. <space> (Not Captured), followed by … a. Remember, it's now moving backwards through the string so the next <space> is right before 'Alpha' b. Because Capture Group 3 = San, Capture Group 2 changes to '<space>Market Center Alpha'. c. The RegEx engine will move forward again.	Capture Group 3 =San
10. \w	.. Word character a. The RegEx engine moves forward to encounter the first Word Character, 'J'	Capture Group 4 = J
11. +	Make it Greedy	Capture Group 4 = Jose
12. $	Indicates the end of the filename, so therefore forces the previous two matches (\w+) to only match the last two words. Or, in other words, the RegEx engine moves forward again through to the end of the string (End of Line or EOL).	

Tim Notes:

1. An example of using Word character, '\w'. The Word metacharacter is defined as any character that is used to form words. This includes the classes of:

[A-Z]	uppercase
[a-z]	lowercase
[0-9]	numeric digits
[_]	underscore

String Manipulation (Parsing)

Rearrange text in parentheses, remove parentheses, move hyphen

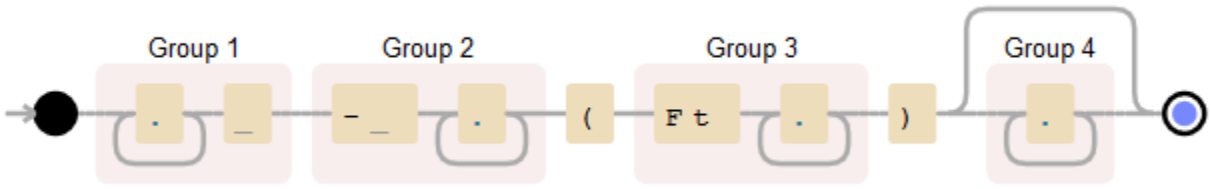
50 Cent - Peep Show (Ft Eminem)(Clean).MP3 to 50 Cent Ft Eminem - Peep Show (Clean).MP3
50 Cent - Outta Control (Ft Mobb Deep).MP3 to 50 Cent Ft Mobb Deep - Outta Control.MP3

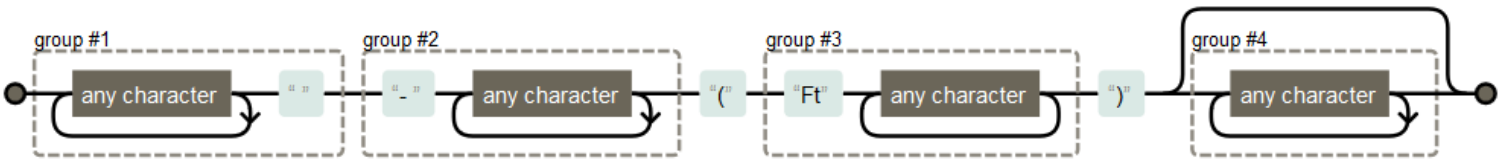
Observations:

1. Find "(Ft.+)" and move it before the <hyphen>, additional remove the parentheses.

Match: (.+)(- .+)\((Ft.+?)\)(.+)?
Replace: \1\3 \2\4

Name	New Name
50 Cent - Peep Show (Ft Eminem)(Clean).MP3	50 Cent Ft Eminem - Peep Show (Clean).MP3
50 Cent - Outta Control (Ft Mobb Deep).MP3	50 Cent Ft Mobb Deep - Outta Control \4.MP3





Sample:

```
.................................................
Match 1: 50 Cent - Peep Show (Ft Eminem)(Clean)
Group 1: 50 Cent
Group 2: - Peep Show
Group 3: Ft Eminem
Group 4: (Clean)
```

Tim Notes:

1. This one addresses an extra backreference problem for a RegEx that works with some samples but not others.

String Manipulation (Parsing)

Rearrange text in parentheses, remove parentheses, move hyphen cont.

Tim Notes:

1. The RegEx indicates by the red that it may match but there is something invalid. The backreferences are wrong.
2. The problem is that the RegEx addresses 4 Capture Groups:

50 Cent - Peep Show (Ft Eminem)(Clean).MP3

\1 = 50 Cent
\2 = - Peep Show
\3 = Ft Eminem
\4 = (Clean)

50 Cent - Outta Control (Ft Mobb Deep).MP3

\1 = 50 Cent
\2 = - Outta Control
\3 = Ft Mobb Deep
\4 = no defined value

The filename in 'red' only has 3 Capture Groups and the fourth is undefined. Therefore, the backreference, '\4', displays in New Name to indicate the problem (Red = Invalid).

3. This can be used to an advantage as the Contributor's technique below shows. The filename can be corrected with, 'Section #3: Replace', at the same time of the RegEx – it doesn't require an additional run. This removes the backreference and the RegEx now matches.

'Section #3: Replace' -

Replace: \4
With: null

Tim Notes:

1. What this does is remove any instance of the text '\4' for those samples that do not require a backreference 4 but does not interfere with those samples that do.

Name	New Name
50 Cent - Peep Show (Ft Eminem)(Clean).MP3	50 Cent Ft Eminem - Peep Show (Clean).MP3
50 Cent - Outta Control (Ft Mobb Deep).MP3	50 Cent Ft Mobb Deep - Outta Control .MP3

Forum Contributor: Stefan

Tim Notes:

1. I should point out that although \4 has been replaced with a null value (nothing) in the Replace criteria, it does not affect the back reference value of Capture Group 4 which retains an undefined value. This is because the Replace criteria only replaced the text '\4' and not the value.

String Manipulation (Parsing)

Rearrange text in parentheses, remove parentheses, move hyphen cont.

Using: 50 Cent - Peep Show (Ft Eminem)(Clean).MP3

Match: (.+)(- .+)\((Ft.+?)\)(.+)?

Where:

\1 = 50 Cent \2 = - Peep Show \3 = Ft Eminem \4 = (Clean)

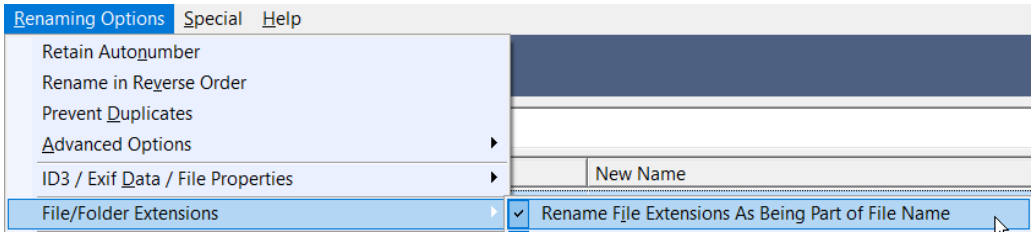
Tim Analysis:

1. .	Match against any character	Capture Group 1 = 5
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen>	Capture Group 2 = - Changes Capture Group 1 = 50 Cent
4. .	Match against any character	Capture Group 2 = -<space>
5. +	Make it Greedy	Capture Group 2 = - Peep Show (Ft Eminem)(Clean)
6. \(	Match against a literal left parenthesis (Not Captured)	Changes Capture Group 2 = - Peep Show (Ft Eminem)
7. Ft	Match against string 'Ft'	Capture Group 3 = Ft Changes Capture Group 2 = - Peep Show
8. .	Match against any character	Capture Group 3 = <space>
9. +	Make it Greedy	Capture Group 3 = Ft Eminem)(Clean)
10. ?	But not too Greedy	Capture Group 3 = Ft
11. \)	Match against a literal right parenthesis (Not Captured)	Changes Capture Group 3 = Ft Eminem
12. .	Match against any character	Capture Group 4 = (
13. +	Make it Greedy	Capture Group 4 = (Clean)
14. ?	But not too Greedy	Capture Group 4 = No Change

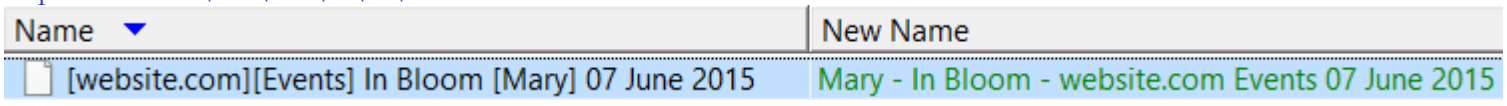
String Manipulation (Parsing)

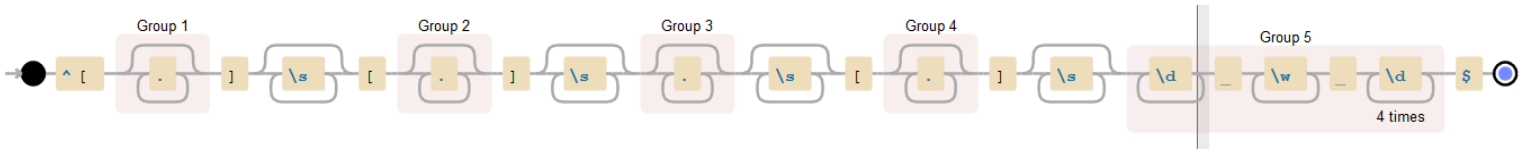
Parse and Rearrange Complex string

[website.com][Events] In Bloom [Mary] 07 June 2015 to Mary - In Bloom - website.com Events 07 June 2015

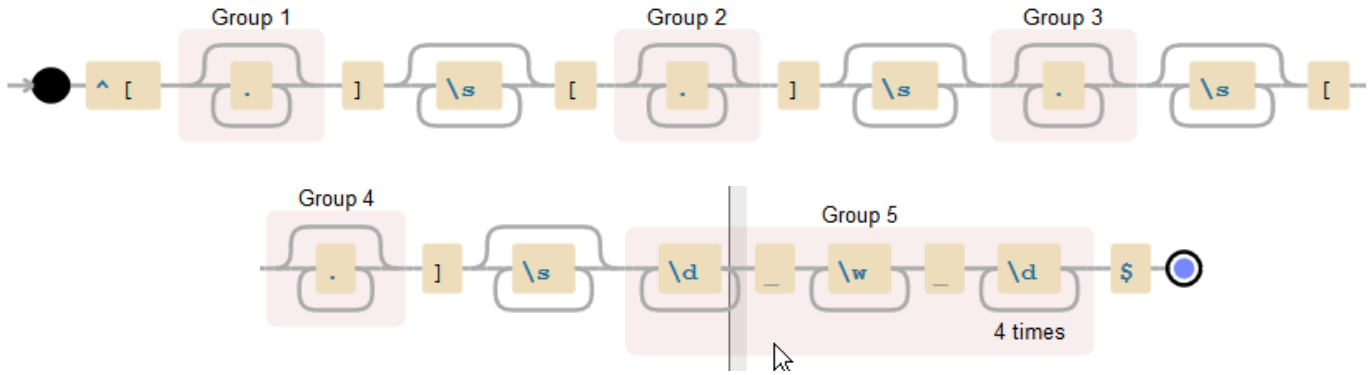


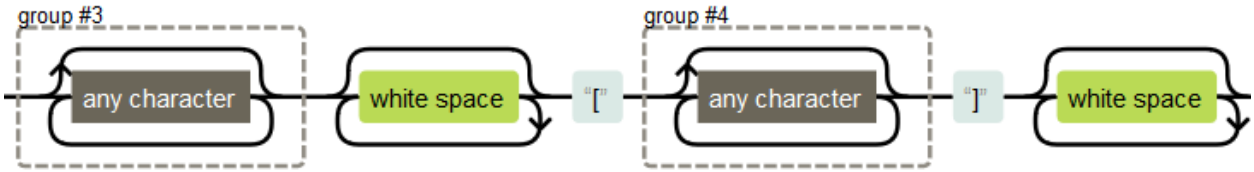
Match: ^\[(.*?)\]\s*\[(.*?)\]\s*(.*?)\s*\[(.*?)\]\s*(\d+ \w+ \d{4})$
Replace: \4 - \3 - \1 \2 \5

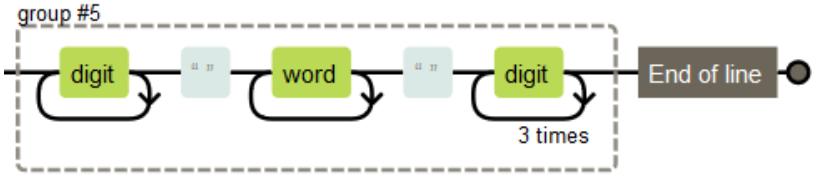




Enlarged:







String Manipulation (Parsing)

Parse and Rearrange Complex string cont.

[website.com][Events] In Bloom [Mary] 07 June 2015 to Mary - In Bloom - website.com Events 07 June 2015

Match: ^\[(.*?)\]\s*\[(.*?)\]\s*(.*?)\s*\[(.*?)\]\s*(\d+ \w+ \d{4})$

Sample:

```
.....................................................................
Match 1: [website.com][Events] In Bloom [Mary] 07 June 2015
Group 1: website.com
Group 2: Events
Group 3: In Bloom
Group 4: Mary
Group 5: 07 June 2015
```

Contributor Analysis:

1. Capture Group 1 - [website.com] uses \[(.*?)\]\s*
2. Capture Group 2 - [Events] uses \[(.*?)\]\s*
3. Capture Group 3 - In Bloom uses (.*?)\s*
4. Capture Group 4 - [Mary] uses \[(.*?)\]\s*
5. Capture Group 5 - 07 June 2015 uses (\d+ \w+ \d{4})

Tim Notes:

1. Uses literals i.e., ' \[' previously discussed
2. Uses Word metacharacter '\w' previously discussed
3. Uses Whitespace metacharacter '\s' previously discussed.

Forum Contributor: Stefan

String Manipulation (Parsing)

Parse and Rearrange Complex string cont.

Using: [website.com][Events] In Bloom [Mary] 07 June 2015

Match: ^\[(.*?)\]\s*\[(.*?)\]\s*(.*?)\s*\[(.*?)\]\s*(\d+ \w+ \d{4})$

Where:

\1 = website.com \2 = Events \3 = In Bloom \4 = Mary \5 = 07 June 2015

Whenever I get a complex looking RegEx, I like to Break it Down so I can see exactly what I am dealing with:

Breaks Down to:

BOL	C-1		C-2		C-3		C-4		C-5	EOL
^	\[(.*?) \]	\s*	\[(.*?) \]	\s*	(.*?)	\s*	\[(.*?) \]	\s*	(\d+ \w+ \d{4})	$
	[website.com]		[Events]		In Bloom		[Mary]		07 June 2015	

Not so intimidating after all. Just a lot of repetition. Also, those brackets are not Class Metacharacters but literal brackets. You couldn't have an expression, i.e., (.*?), like that inside a class anyway.

String Manipulation (Parsing)

Parse and Rearrange Complex string cont.

Using: [website.com][Events] In Bloom [Mary] 07 June 2015

Match: ^\[(.*?)\]\s*\[(.*?)\]\s*(.*?)\s*\[(.*?)\]\s*(\d+ \w+ \d{4})$

Where:

\1 = website.com \2 = Events \3 = In Bloom \4 = Mary \5 = 07 June 2015

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. \[	Match against literal left bracket (Not Captured) =' ['	
3. .	Match against any character	Capture Group 1 = w
4. *	Make it Greedy	Capture Group 1 = website.com][Events] In Bloom [Mary] 07 June 2015
5. ?	But not too Greedy	Capture Group 1 = null
6. \]	Match against literal right bracket (Not Captured)	Changes Capture Group 1 = website.com
7. \s	Match against <space> character (Not Captured)	Changes Capture Group 1 = website.com][Events
8. *	Make it Greedy	Changes Capture Group 1 = website.com
9. \[	Match against literal left bracket (Not Captured)	
10. .	Match against any character	Capture Group 2 = E
11. *	Make it Greedy	Capture Group 2 = Events] In Bloom [Mary] 07 June 2015
12. ?	But not too Greedy	Capture Group 2= null
13. \]	Match against literal right bracket (Not Captured)	Changes Capture Group 2 = Events
14. \s	Match against <space> character (Not Captured)	
15. *	Make it Greedy	
16. .	Match against any character	Capture Group 3 = I
17. *	Make it Greedy	Capture Group 3 = In Bloom [Mary] 07 June 2015
18. ?	But not too Greedy	Capture Group 3 = null

String Manipulation (Parsing)

Parse and Rearrange Complex string cont.

Using: [website.com][Events] In Bloom [Mary] 07 June 2015

Match: ^\[(.*?)\]\s*\[(.*?)\]\s*(.*?)\s*\[(.*?)\]\s*(\d+ \w+ \d{4})$

Where:

\1 = website.com \2 = Events \3 = In Bloom \4 = Mary \5 = 07 June 2015

Tim Analysis: cont.

19. \s	Match against <space> character (Not Captured)	Changes Capture Group 3 = In
20. *	Make it Greedy	Changes Capture Group 3 = null
21. \[	Match against literal left bracket (Not Captured)	Changes Capture Group 3 = In Bloom
22. .	Match against any character	Capture Group 4 = M
23. *	Make it Greedy	Capture Group 4 = Mary] 07 June 2015
24. ?	But not too Greedy	Capture Group 4 = null
25. \]	Match against literal right bracket (Not Captured)	Changes Capture Group 4 = Mary
26. \s	Match against <space> character (Not Captured)	
27. *	Make it Greedy	
28. \d	Match against Numeric Digit	Capture Group 5 = 0
29. +	Make it Greedy	Capture Group 5 = 07
30. <space>	Match against <space>	Capture Group 5 = 07<space>
31. \w	Word Metacharacter. Match against any single letter, number or underscore followed by …	Capture Group 5 = 07 J
32. +	Make it Greedy	Capture Group 5 = 07 June
33. <space>	Match against <space>	Capture Group 5 = 07 June <space>
31. \d {4}	Match against 4 numeric digits	Capture Group 5 = 07 June 2015
32. $	Must be at EOL	

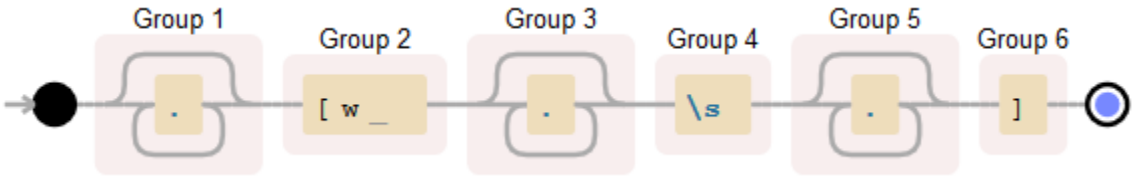
String Manipulation (Parsing)

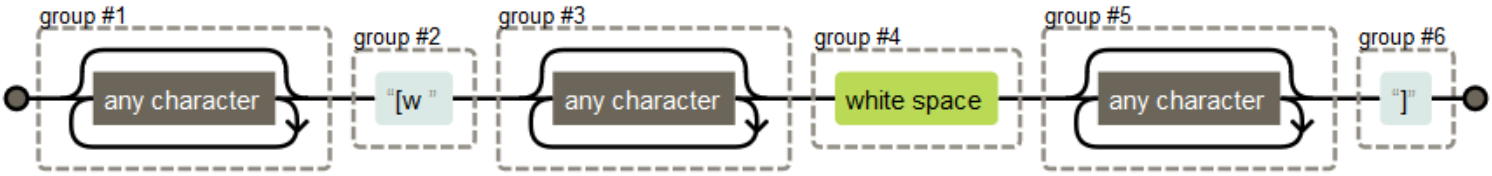
Rearrange Name in Brackets at end of filename

1987 - Chieftains in Ireland - [w James Galway] to 1987 - Chieftains in Ireland - [w Galway, James]

Match: (.*)(\[w)(.*)(\s)(.*)(\])
Replace: \1\2\5, \3\6

Name	New Name
1987 - Chieftains in Ireland - [w James Galway]	1987 - Chieftains in Ireland - [w Galway, James]





Sample:

```
Match 1: 1987 - Chieftains in Ireland - [w James Galway]
Group 1: 1987 - Chieftains in Ireland -
Group 2: [w
Group 3: James
Group 4:
Group 5: Galway
Group 6: ]
```

Forum Contributor: coyotewrw

Tim Notes:

1. Rather than give a full analysis, you should be able to do this on your own now. I will break it down for you:

Breaks Down to:

	literal	literal						
(.*)	(	\[	w	)	(.*)	(\s)	(.*)	(\])

String Manipulation (Parsing)

Change 'Date Taken' attribute of the photo ((YYYYMMDD_HH.MM.SS) to More Legible

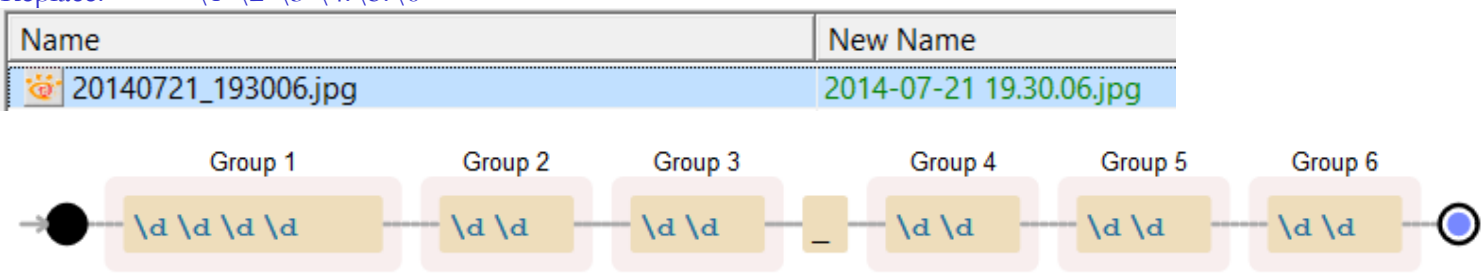
20140721_193006.jpg to 2014-07-21 19.30.06.jpg

Observations:

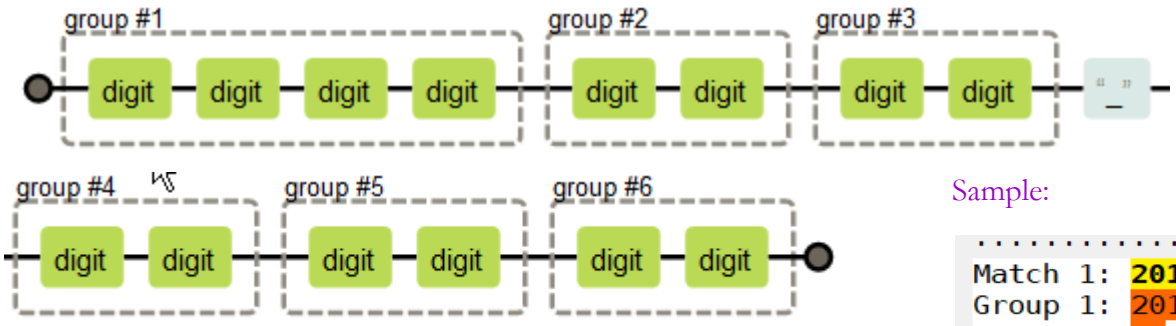
1. Separate date with <hyphens>, time with dot, and also replace the underscore by a <space>.
2. Pattern Established: <####> <##> <##> underscore <##> <##> <##>
3. Add the wanted delimiters in replacement string.

Try this …

Match: (\d\d\d\d)(\d\d)(\d\d)_(\d\d)(\d\d)(\d\d)
Replace: \1-\2-\3 \4.\5.\6



Expanded:



Sample:

```
Match 1: 20140721_193006
Group 1: 2014
Group 2: 07
Group 3: 21
Group 4: 19
Group 5: 30
Group 6: 06
```

Tim Notes:

1. Note the <space> between \3 and \4. in the Replacement String.
2. In all of the examples in the book, you should always be aware of <spaces> or other characters, especially in the Replacement String.

   Overlooked characters in the RegEx can usually be seen in the Visualization photo, but this does not depict the Replacement String. If you find that a RegEx is not working as it should, you should try copying the Replacement String directly. I have allowed the copying and printing of text from all of my books. You are not, however, allowed to make changes to any material in the books.

String Manipulation (Parsing)

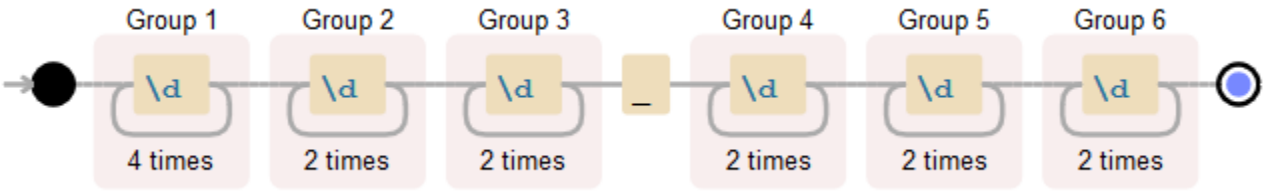
Change 'Date Taken' attribute of the photo ((YYYYMMDD_HH.MM.SS) to More Legible cont.

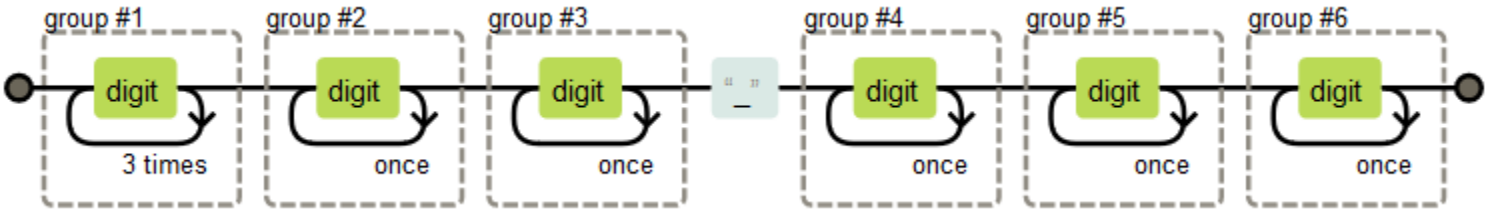
20140721_193006.jpg to 2014-07-21 19.30.06.jpg

.. Or this:

Match: (\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2})
Replace: \1-\2-\3 \4.\5.\6 (Note the <space> between \3 and \4.)

Name	New Name
20140721_193006.jpg	2014-07-21 19.30.06.jpg





Sample:

```
Match 1: 20140721_193006
Group 1: 2014
Group 2: 07
Group 3: 21
Group 4: 19
Group 5: 30
Group 6: 06
```

Contributor Analysis:

1. \1 represent what was matched with first parentheses (...) Capture Group 1
2. \2 represent what was matched with second parentheses (...) Capture Group 2
3. The <hyphen>, <space> and <dot> characters are entered literally in the Replacement string.

Tim Notes:

1. A good example of parsing dates with \d
2. Note the <space> between \3 and \4. in the Replacement String.
3. This last RegEx is the simplified version of the original RegEx used – substituting Quantifiers for the individual numeric digit metacharacters, nothing more.

Forum Contributor: Stefan

String Manipulation (Parsing)

Change 'Date Taken' attribute of the photo ((YYYYMMDD_HH.MM.SS) to More Legible cont.

Using: 20140721_193006.jpg

Match: (\d{4})(\d{2})(\d{2})_(\d{2})(\d{2})(\d{2})

Where:

\1 = 2014 \2 = 07 \3 = 21 \4 = 19 \5 = 30 \6 = 06

Tim Analysis:

1. \d{4}	Match against 4 numeric digits	Capture Group 1= 2014
2. \d {2}	Match against 2 numeric digits	Capture Group 2 = 07
3. \d {2}	Match against 2 numeric digits	Capture Group 3 = 21
4. _	Match against underscore (Not Captured)	
5. \d {2}	Match against 2 numeric digits	Capture Group 4 = 19
6. \d {2}	Match against 2 numeric digits	Capture Group 5 = 30
6. \d {2}	Match against 2 numeric digits	Capture Group 6 = 06

String Manipulation (Parsing)

Rearrange MP3 Title

Artist & whatever – Title.mp3 to Artist - Title (With whatever).mp3

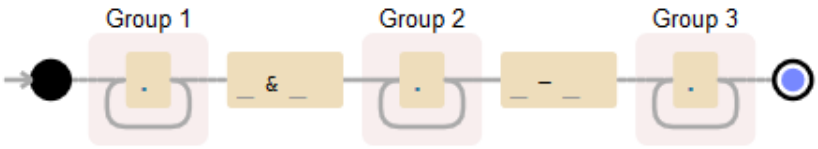
Samples:

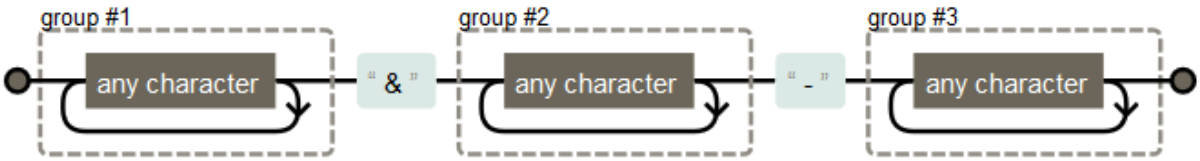
Dr. Dre & Eminem & Skylar Grey - I Need A Doctor
to
Dr. Dre - I Need A Doctor(With Eminem & Skylar Grey).mp3

Attempt by Poster:

Match: (.+) \& (.+) - (.+)
Replace: \1 - \3 (With \2)

Name	New Name
Dr. Dre & Eminem & Skylar Grey - I Need A Doctor	Dr. Dre & Eminem - I Need A Doctor (With Skylar Grey)
Artist & whatever – Title.mp3	Artist & whatever – Title.mp3





Sample:

```
.....................................................
Match 1: Dr. Dre & Eminem & Skylar Grey - I Need A Doctor
Group 1: Dr. Dre & Eminem
Group 2: Skylar Grey
Group 3: I Need A Doctor
```

Problem: Not Universal enough to handle complex titles

Forum Contributor: bitmonger

Tim Analysis:

1. The problem is the first (.+) is greedy and initially matches the whole filename.
2. Then it backtracks until it hits the first '\&' and stops.
3. To prevent this we make the + quantifier lazy by putting a ? after it. This will match the first '\&' no matter how many are in the filename.

String Manipulation (Parsing)

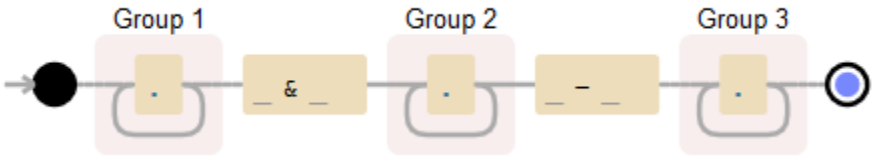
Rearrange MP3 Title cont.

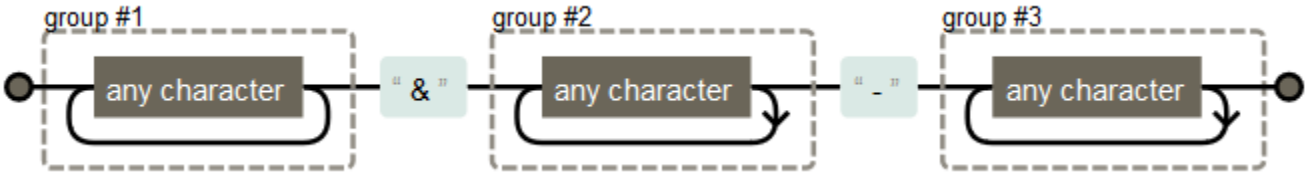
Tim Contribution –

Corrected RegEx:

Match: (.+?) \& (.+) - (.+)
Replace: \1 - \3 (With \2)

Name	New Name
Dr. Dre & Eminem & Skylar Grey - I Need A Doctor.mp...	Dr. Dre - I Need A Doctor (With Eminem & Skylar Grey).mp3
Artist & whatever - Title.mp3	Artist - Title (With whatever).mp3





Sample:

```
.....................................................
Match 1: Dr. Dre & Eminem & Skylar Grey - I Need A Doctor
Group 1: Dr. Dre
Group 2: Eminem & Skylar Grey
Group 3: I Need A Doctor
```

Where:

/1 = Dr. Dre \2 = Eminem & Skylar Grey \3 = I Need a Doctor

String Manipulation (Parsing)

Rearrange MP3 Title cont.

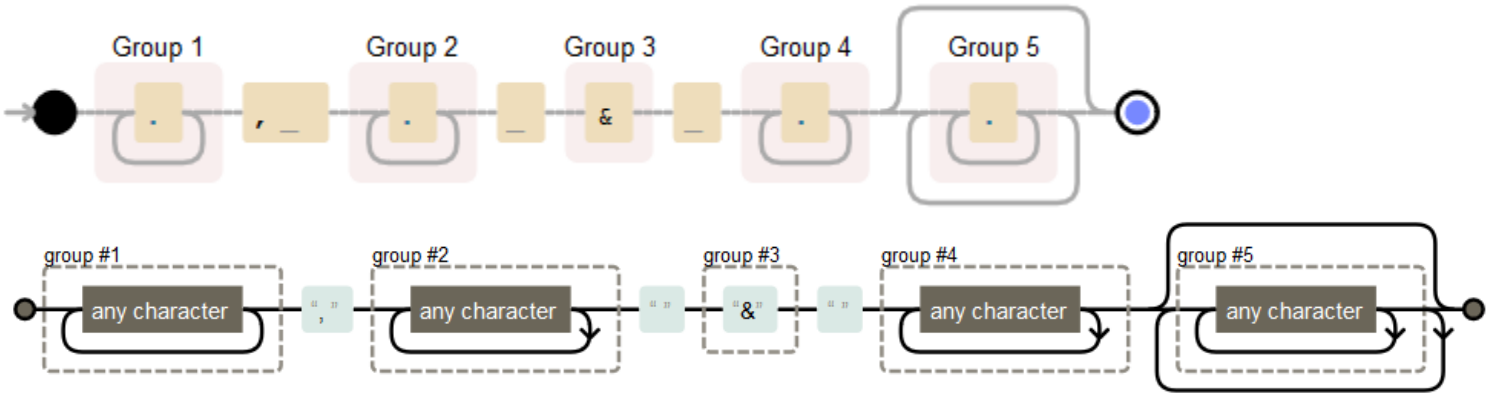
Tim Contribution – cont.

A variation to switch around Artists names and remove comma (suggested by poster but no RegEx originally shown) -

Michael, George & Queen to George Michael & Queen

Match: (.+?), (.+) (\&) (.+)(.+)*
Replace: \2 \1 \3 \4

Name	New Name
Michael, George & Queen.pdf	George Michael & Queen.pdf



Sample:

```
.....................................Start·Length····
Match 1: Michael, George & Queen          0     23
Group 1: Michael                          0      7
Group 2: George                           9      6
Group 3: &                               16      1
Group 4: Queen                           18      5
Group 5 did not participate in the match
```

Tim Notes:

1. Notice in the Sample run the 'Group 5 did not participate in the Match' message? This indicates a problem. Group 5 is not included in the Replacement String, AND it falls at the end of the RegEx – so chances are it is redundant.

   Let's find out -

String Manipulation (Parsing)

Rearrange MP3 Title cont.

Tim Contribution – cont.

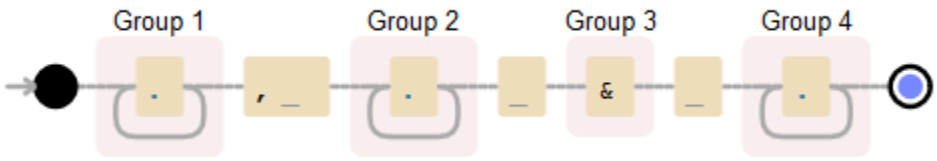
Using: Michael, George & Queen

Match: (.+?), (.+) (\&) (.+)(.+)*

Tim Analysis:

1. .	Match against any character	Capture Group 1 = M
2. .+	Make it Greedy. Matches against entire String	Capture Group 1 = <entire string>
3. ?	But not too Greedy. Makes the last match optional.	Capture Group 1 = M
4. ,	Match against <comma> (Not Captured)	
5. <space>	Match against <space> (Not Captured)	
6. .	Match against any character.	Capture Group 2 = G Changes Capture Group 1= Michael
7. +	Make it Greedy	Capture Group 2 = George & Queen
8. <space>	Match against <space> (Not Captured)	Changes Capture Group 2 = George &
9. \&	Match against literal '&'	Capture Group 3 = & Changes Capture Group 2 = George
10. <space>	Match against <space> (Not Captured)	
11. .	Match against any character	Capture Group 4 = Q
12. +	Make it Greedy	Capture Group 4 = Queen

At this point the string is parsed. I should have stopped. Concludes from previous that it *is* redundant.



Where:

\1 = Michael \2 = George \3 = & \4 = Queen

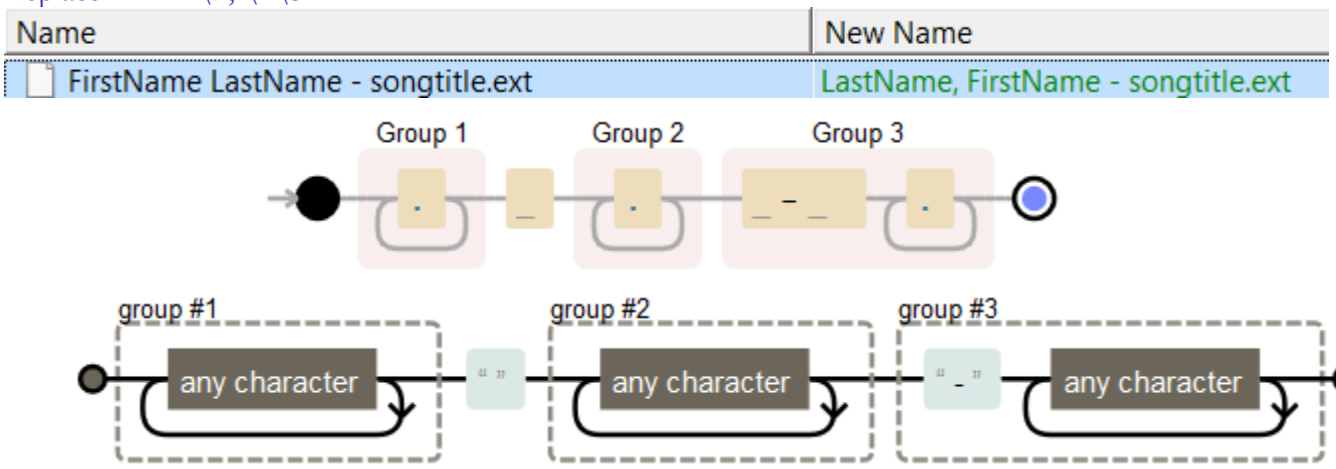
13. .	Match against any character It was at EOL, so it backtracked.	Capture Group 5 = n Changes Capture Group 4 = Quee
14. +	Make it Greedy Already back to EOL, nothing more to capture	Capture Group 5 = Unchanged
15. *	Make expression Greedy	Capture Group 5 = Invalid \5 Changes Capture Group 4 = Queen

String Manipulation (Parsing)

Simple Rearrange #1

FirstName LastName - songtitle.ext to LastName, Firstname - songtitle.ext

Match: (.+) (.+)(- .+)
Replace: \2, \1\3



Sample:

```
Match 1: FirstName LastName - songtitle
Group 1: FirstName
Group 2: LastName
Group 3:  - songtitle
```

Forum Contributor: Stefan

Where:
\1 = Firstname \2 = LastName \3 = <space>- songtitle

Tim Analysis:

1. .	Match against any character	Capture Group 1 = F
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against <space>	Changes Capture Group 1 = Firstname LastName - <space>
4. .	Match against any character	Capture Group 2 = s
5. +	Make it Greedy	Capture Group 2 = songtitle
6. <space>	Match against <space> This is the <space> before 'songtitle'	Capture Group 3 = <space> Changes Capture Group 2 = -<space>
7. -	Match against <hyphen> followed by …	Capture Group 3 = <space> <hyphen> Changes Capture Group 2 = LastName Changes Capture Group 1 = FirstName
8. .	Match against any character	Capture Group 3 = <space> <hyphen> <space>
9. +	Make it Greedy	Capture Group 3 = <space> <hyphen> <space> songtitle

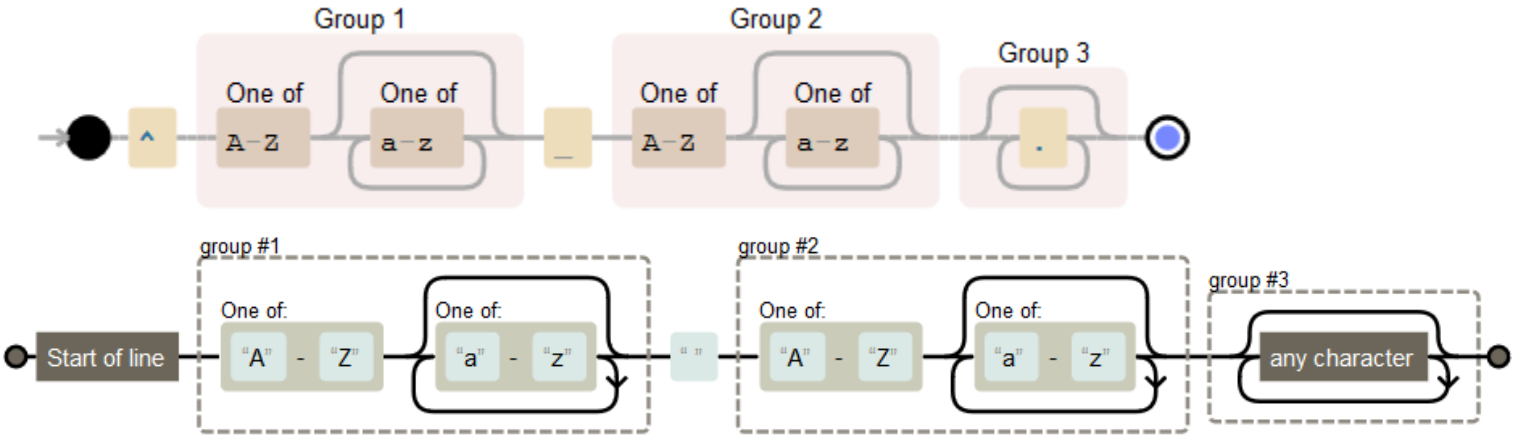
String Manipulation (Parsing)

Simple Rearrange #2

Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3
to
Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3

Match: ^([A-Z][a-z]*) ([A-Z][a-z]*)(.*)
Replace: \2, \1\3

Name	New Name
Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3	Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3



Sample:

```
Match 1: Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire
Group 1: Arthur
Group 2: Brown
Group 3:  - The Crazy World Of Arthur Brown - 08 - Fire
```

String Manipulation (Parsing)

Simple Rearrange #2 cont.

Using: Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3

Match: ^([A-Z][a-z]*) ([A-Z][a-z]*)(.*)

Where:

\1 = Arthur \2 = Brown \3 = <space>- The Crazy World Of Arthur Brown - 08 - Fire

Tim Analysis:

1. ^	Start at the beginning of the string (BOL)	
2. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 1 =A
3. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 1 =Ar
4. *	Make it Greedy	Capture Group 1 = Arthur
5. <space>	Match against a <space> character (Not Captured)	
6. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 =B
6. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 2 =Br
8. *	Make it Greedy	Capture Group 2 = Brown
9. .*	Match against remaining string	Capture Group 3 = <space>- The Crazy World Of Arthur <space> Brown - 08 - Fire

String Manipulation (Parsing)

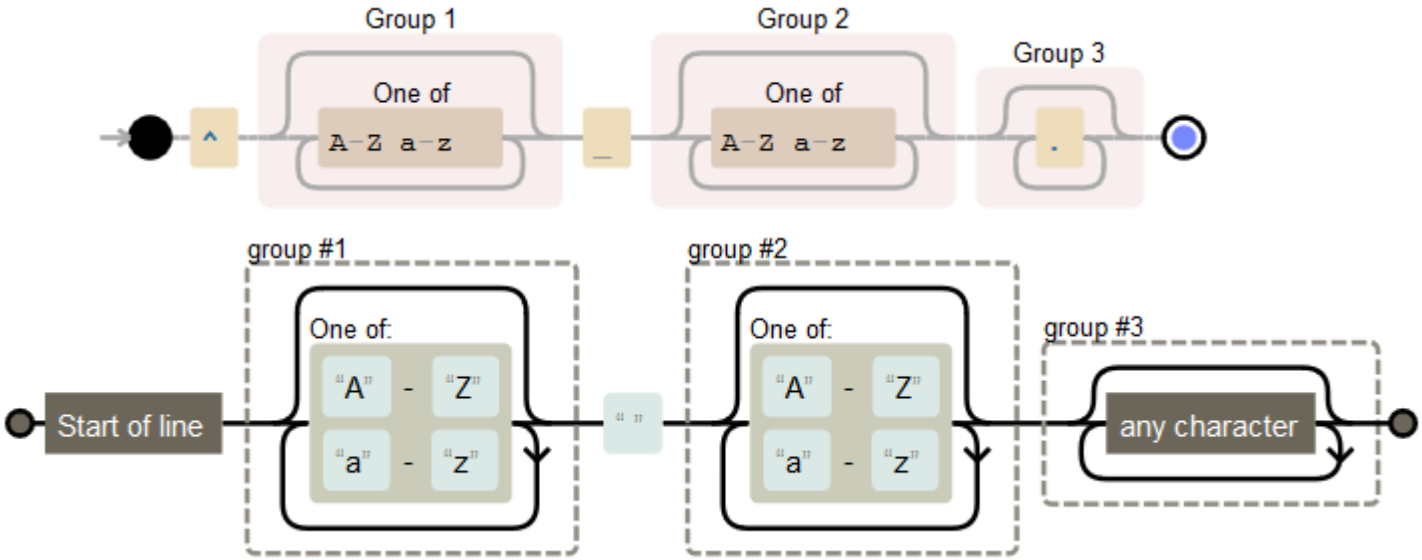
Simple Rearrange #2 cont.

Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3
to
Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3

Match: ^([A-Z][a-z]*) ([A-Z][a-z]*)(.*)

Tim Notes:

1. Classes previously discussed [A-Z] [a-z]
2. Could also have been written as ^([A-Za-z]*) ([A-Za-z]*)(.*) to obtain same results but visualization changes slightly:



Name	New Name
Arthur Brown - The Crazy World Of Arthur Brown - 08 - Fire.mp3	Brown, Arthur - The Crazy World Of Arthur Brown - 08 - Fire.mp3

Forum Contributor: BRUlvr100

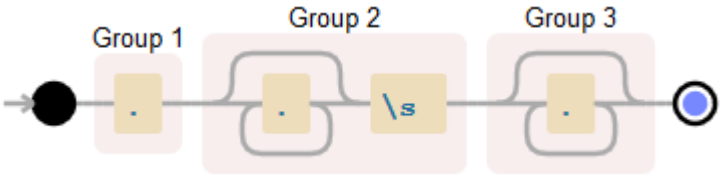
String Manipulation (Parsing)

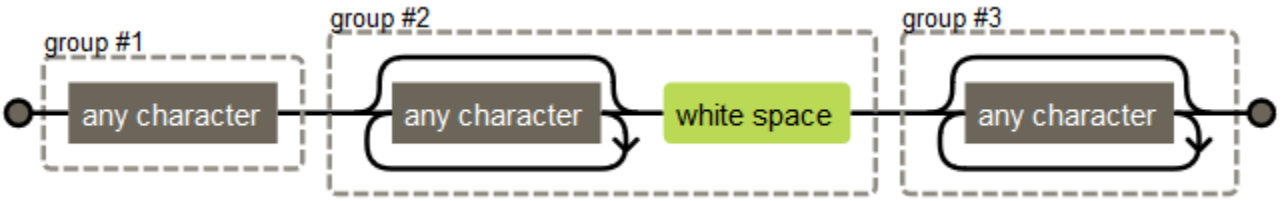
Rearrange names to LastNameFirstInitial

John Smith to SmithJ
Harry Jones to JonesH

Match: (.)(.*\s)(.*)
Replace: \3\1

Name	New Name
John Smith	SmithJ
Harry Jones	JonesH





Sample:

```
......................
Match 1: John Smith
Group 1: J
Group 2: ohn
Group 3: Smith
```

Forum Contributor: jimwillsher (Original Publisher of BRU)

String Manipulation (Parsing)

Rearrange names to LastNameFirstInitial cont.

Using: John Smith

Match: (.)(.*\s)(.*)

Where:

\1 = J \2 = ohn<space> \3 = Smith

Tim Analysis:

1. .	Match against any character	Capture Group 1 = J
2..	Match against any character	Capture Group 2 = o
3. *	Make it Greedy Current position = EOL	Capture Group 2 = ohn Smith
4. \s	Match against <space> character Backtracks. This is the <space> after 'John'	Capture Group 2 = ohn <space>
5. .*	Match against remaining string	Capture Group 3 = Smith

String Manipulation (Parsing)

Replace all the spaces and brackets in a filename by underscores

some name.ext to some_name.ext
some [name].ext to some_name_ext

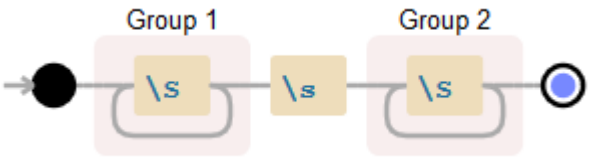
Observations:

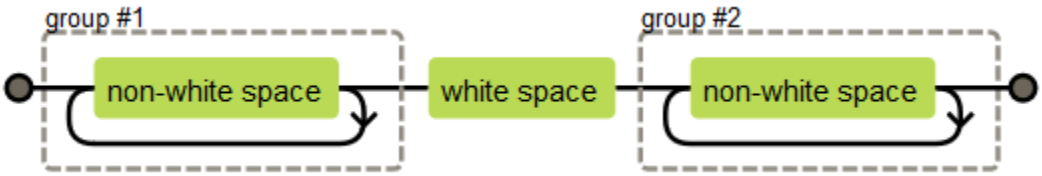
1. Replace all the <spaces> and brackets in a filename by underscores

Method #1:

Match: (\S+)\s(\S+)
Replace: \1_\2

Name	New Name
some name.ext	some_name.ext
some [name].ext	some_[name].ext





Sample:

```
Match 1: some name
Group 1: some
Group 2: name
```

Sample:

```
Match 1: some [name]
Group 1: some
Group 2: [name]
```

Doesn't work. Brackets still intact. Restrictive – only works with pattern – Text <space>Text

Tim Notes:

1. \S is a Non-Whitespace Metacharacter. It will match against any single non-whitespace character. If you recall, a whitespace character in BRU is a <space> character, so this means to match against any character *but* a <space>
2. \s is a Whitespace Metacharacter previously discussed.

Forum Contributor: KenP

String Manipulation (Parsing)

Replace all the spaces and brackets in a filename by underscores cont.

Using: some [name].ext

Method #1:

Match: (\S+)\s(\S+)

Where:

\1 = some \2 = [name]

Tim Analysis:

1. \S	Matches any Non-Whitespace Character	Capture Group 1 = s
2. +	Make it Greedy	Capture Group 1 = some
3. \s	Match against <space> character (Not Captured)	
4. \S	Matches any Non-Whitespace Character	Capture Group 2 =[
5. +	Make it Greedy	Capture Group 2 = [name]

Tim Notes:

1. \S refers to any character except <space>, the opposite of a <space> Metacharacter, \s

String Manipulation (Parsing)

Replace all the spaces and brackets in a filename by underscores cont.

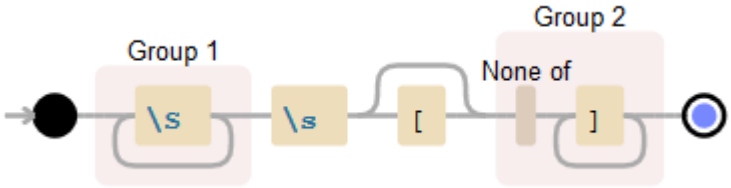
Tim Contribution –

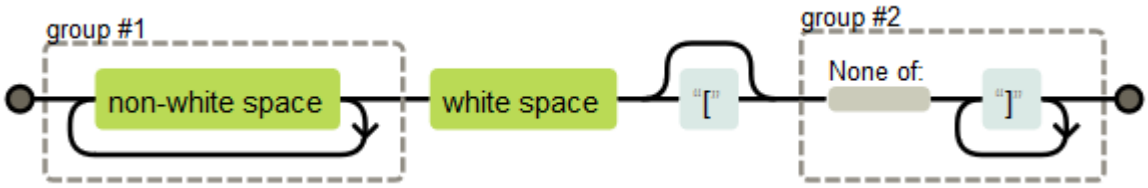
Method #1

This will work:

Match: (\S+)\s\[?([^]]+)
Replace: \1_\2

Name	
some name.ext	some_name.ext
some [name].ext	some_name.ext





Sample:

```
.....................
Match 1: some [name
Group 1: some
Group 2: name
```

Tim Notes:

1. Still restrictive.

String Manipulation (Parsing)

Replace all the spaces and brackets in a filename by underscores cont.

Using: some [name].ext

Tim Contribution – cont.

Method #1:

Match: (\S+)\s\[?([^]]+)

Where:

\1 = some \2 = name

Tim Analysis:

1. \S	Matches any Non-Whitespace Character	Capture Group 1 = s
2. +	Make it Greedy	Capture Group 1 = some
3. \s	Match against <space> character (Not Captured)	
4. \[	Match against a left bracket (Not Captured) = ' ['	
5. ?	Make it optional (if the bracket is not found)	
6. [^]]	Match against a Negated Class consisting of a right bracket. Match against anything but a right bracket.	Capture Group 2 = n
7. +	Make it Greedy	Capture Group 2 = name

String Manipulation (Parsing)

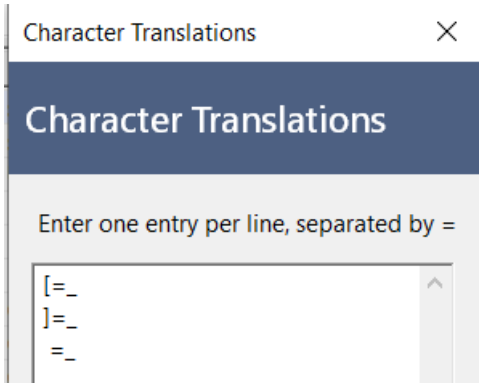
Replace all the spaces and brackets in a filename by underscores cont.

Expanded Samples:

some name.ext
some name again.ext
some name again and more.ext
some name [brackets1] again and [brackets 2] more.ext
some [name].ext
some [name] again.ext

Method #2:

'Section #14: Character Translations' –



Place each character translation on a separate line (no trailing spaces):
[=_
]=_
<space>=_

Name	New Name
some name.ext	some_name.ext
some name again.ext	some_name_again.ext
some name again and more.ext	some_name_again_and_more.ext
some name [brackets1] again and [brackets 2] more.ext	some_name__brackets1__again_and__brackets_2__more.ext
some [name].ext	some__name_.ext
some [name] again.ext	some__name__again.ext

Forum Contributor: KenP

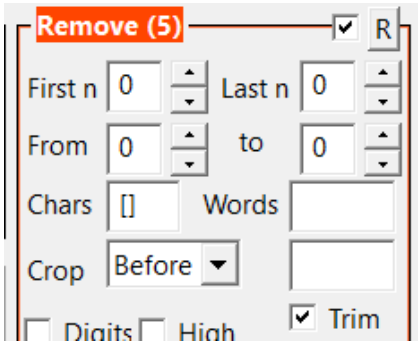
String Manipulation (Parsing)

Replace all the spaces and brackets in a filename by underscores cont.

Tim Contribution –

Method #3

1. 'Section #5: Remove' –



Remove:
Chars: [] Both left and right brackets - [ ]
Trim: Enabled. Just in case there are any leading or trailing <spaces>

2. 'Section #3: Replace' –

Replace <space> with underscore

Name	New Name
some name.ext	some_name.ext
some name again.ext	some_name_again.ext
some name again and more.ext	some_name_again_and_more.ext
some name [brackets1] again and [brackets 2] more.ext	some_name_brackets1_again_and_brackets_2_more.ext
some [name].ext	some_name.ext
some [name] again.ext	some_name_again.ext

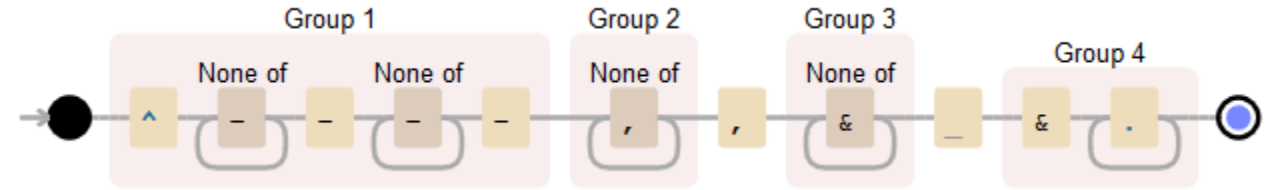
String Manipulation (Parsing)

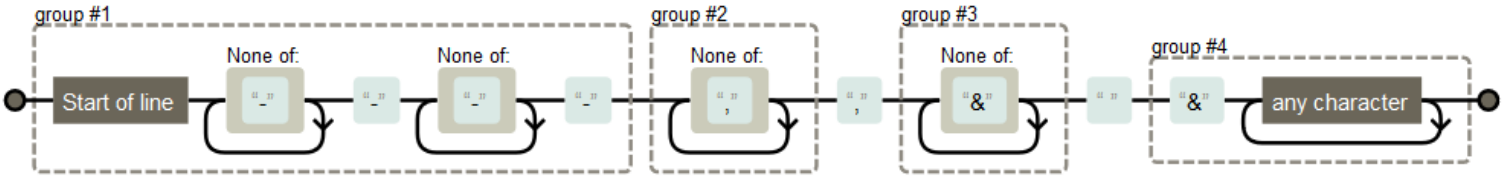
Simple Rearrange #3

BC40049-01 - Jackson, Michael & Paul McCartney - Ebony & Ivory
to
BC40049-01 - Michael Jackson & Paul McCartney - Ebony & Ivory

Match: (^[^-]+-[^-]+-)([^,]+),([^&]+) (&.+)
Replace: \1\3\2 \4

Name	New Name
BC40049-01 - Jackson, Michael & Paul McCartney - Ebony & Ivory	BC40049-01 - Michael Jackson & Paul McCartney - Ebony & Ivory





Sample:

```
.....................................................................
Match 1: BC40049-01 - Jackson, Michael & Paul McCartney - Ebony & Ivory
Group 1: BC40049-01 -
Group 2:  Jackson
Group 3:  Michael
Group 4: & Paul McCartney - Ebony & Ivory
```

Forum Contributor: Jane

String Manipulation (Parsing)

Simple Rearrange #3 cont.

Using: BC40049-01 - Jackson, Michael & Paul McCartney - Ebony & Ivory

Match: (^[^-]+-[^-]+-)([^,]+),([^&]+) (&.+)

Where: \1 = BC40049-01 – \2 = <space> Jackson \3 = <space> Michael
\4 = & Paul McCartney - Ebony & Ivory

Tim Analysis:

Capture Group 1

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. [^-]	Ignore first <hyphen> (Negated class)	
3. +	Make Greedy.	Capture Group 1 = BC40049
4. –	Include up to first <hyphen>.	Capture Group 1 = BC40049-
5. [^-]	Ignore second <hyphen> (Negated class)	Capture Group 1 = BC40049-0
6. +	Make Greedy.	Capture Group 1 = BC40049-01
7. -	Include up to second <hyphen>.	Capture Group 1 = BC400490-01 –

Capture Group 2

1. [^,]	Ignore <comma> (Negated class)	
2. +	Make Greedy.	Capture Group 2 = <space> Jackson

Not Captured

1. Match against <comma>. If matched = True, continue expression.

Capture Group 3

1. [^&]	Ignore <Ampersand>. (Negated class)	
2. +	Make Greedy.	Capture Group 3 = <space> Michael

Not Captured

1. Match against <space>. If matched = True, continue expression.

Capture Group 4

1. &	Capture <Ampersand>.	Capture Group 4 = &
2. .+	Capture character at current position and Make Greedy. Captures remainder of string.	Capture Group 4 = & Paul McCartney – Ebony & Ivory

String Manipulation (Parsing)

Re-arranging year placement w variable lengths

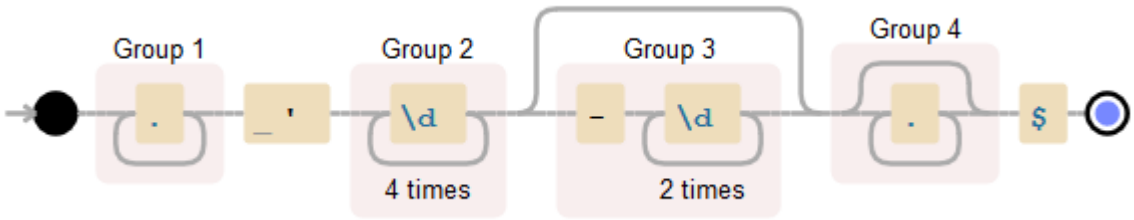
ford focus (107) '2010.jpg	to	2010 ford focus (107).jpg
mazda 626 (101) '1978-82.jpg	to	1978 mazda 626 (101).jpg
studebaker president (130) '1934-35 independent.jpg	to	1934 studebaker president (130) independent.jpg
alfa romeo giulia 1600 (101) '1962-65 by bertone.jpg	to	1962 alfa romeo giulia 1600 (101) by bertone.jpg

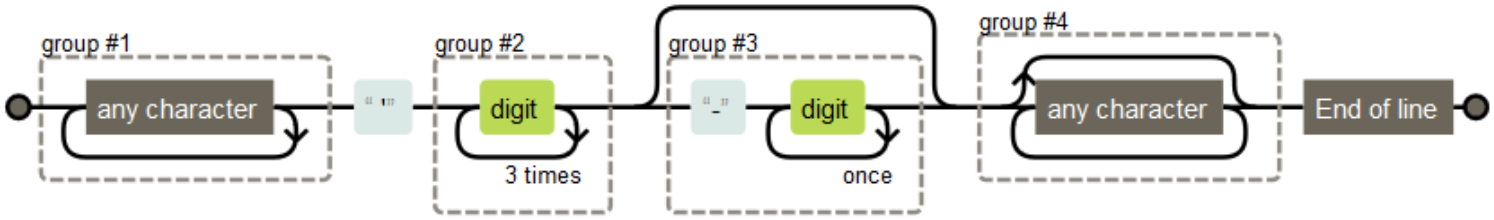
Observations:

1. The 4 numeric digits after the ' move to the beginning of the file name.
2. "-dd" needs to be dropped from some file names, i.e., '-82'
3. Text after 'dddd or 'dddd-dd need to be retained.

Match: (.+) '(\d{4})(-\d{2})?(.*?)$
Replace: \2 \1\4

Name	New Name
mazda 626 (101) '1978-82.jpg	1978 mazda 626 (101).jpg
alfa romeo giulia 1600 (101) '1962-65 by bertone.jpg	1962 alfa romeo giulia 1600 (101) by bertone.jpg
ford focus (107) '2010.jpg	2010 ford focus (107).jpg
studebaker president (130) '1934-35 independent.jpg	1934 studebaker president (130) independent.jpg





String Manipulation (Parsing)

Re-arranging year placement w variable lengths cont.

Samples used:

ford focus (107) '2010.jpg
mazda 626 (101) '1978-82.jpg
alfa romeo giulia 1600 (101) '1962-65 by bertone.jpg

Match: (.+) '(\d{4})(-\d{2})?(.*?)$

Sample:

```
·····························Start·Length··
Match 1: ford focus (107) '2010      0      22
Group 1: ford focus (107)            0      16
Group 2: 2010                       18       4
Group 3 did not participate in the match
Group 4:                            22       0
```

Sample:

```
·····························Start·Length····
Match 1: mazda 626 (101) '1978-82      0      24
Group 1: mazda 626 (101)               0      15
Group 2: 1978                         17       4
Group 3: -82                          21       3
Group 4:                              24       0
```

Sample:

```
·····························Start·Length····
Match 1: alfa romeo giulia 1600 (101) '1962-65 by bertone      0      34
Group 1: alfa romeo giulia 1600 (101)                          0      25
Group 2: 1962                                                 26       4
Group 3: -65                                                  30       3
Group 4:  by bertone                                          33       1
```

Where:

\1 = ford focus (107)	\2 = 2010	\3 = No Value	\4 = null
\1 = mazda 626 (101)	\2 = 1978	\3 = -82	\4 = null
\1 = alfa romeo giulia 1600 (101)	\2 = 1962	\3 = -65	\4 = <space> by bertone

String Manipulation (Parsing)

Re-arranging year placement w variable lengths cont.

Tim Notes:

1. This is one of those 'Group did not participate' messages again. Last time it was because there was a redundant expression at the end of the RegEx, but this is not the case here because it falls in the middle. What this is in fact is the result of an optional Quantifier.

2. The match for Capture Group 3 was made optional because of the ' ? ' quantifier that was used. It made the preceding expression optional which allowed it to match against several different types of samples.

   It matches against those samples that had the year range found in 3 of the samples (-\d{2})?

   – This fulfilled 'Observation #2' - "-dd" needs to be dropped from some file names, i.e., '-82'

   For those samples that didn't, the expression was optional and it is as if it didn't exist = No Value.

3. It also matched against samples that contained text after the year (.*?) for Capture Group 4

   – This fulfilled 'Observation #3 - Text after 'dddd or 'dddd-dd need to be retained.

   If there were no text remaining after the year because the current position is EOL, a null value was captured as a result of a Zero Occurrence Match in Capture Group 4. In other words, if a sample did not have, i.e., text after the year, the match would still occur but a null value would be captured.

3. Despite the range of the sample matches, I still find the pattern a little too restrictive. I think I can do better.

String Manipulation (Parsing)

Re-arranging year placement w variable lengths cont.

Using: ford focus (107) '2010.jpg

Match: (.+) '(\d{4})(-\d{2})?(.*?)$

Where:

\1 = ford focus (107) \2 = 2010 \3 = No Value \4 = null

Tim Analysis:

Certain Capture Groups are what I call 'reserved' for certain values. If a value is missing from a sample, then a Zero Occurrence Match may occur depending on the RegEx. If an Optional Quantifier is used as it is here, then the entire expression, (-\d{2}) is made optional as if it didn't exist resulting in No Value for Capture Group 3.

Capture Group 1 is reserved for the car model. Capture Group 2 is reserved for the car year. Capture Group 3 is reserved for the car year range and Capture Group 4 is reserved for any text that may follow after the car year range.

The Ford Focus sample above does not have the car year range resulting in No Value for Capture Group 3 because the expression is made optional, or any text following resulting in a null value for Capture Group 4 as a result of a Zero Occurrence Match.

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) followed by …	Changes Capture Group 1 = ford focus (107)
4. '	.. match against single apostrophe (Not Captured) followed by …	
5. \d{4}	.. Match against 4 numeric digits followed by …	Capture Group 2 = 2010
6. <hyphen>	.. Match against <hyphen> followed by …	Capture Group 3 = No Value
7. \d{2}	.. Match against 2 numeric digits	Capture Group 3 = No Value
8. ?	Make last expression optional	Capture Group 3 = No Value
9. .*	Match against remaining string	Capture Group 4 = EOL
10. ?	Make last expression optional	Capture Group 4 = null
11. $	Must be at EOL	

Tim Notes:

1. No Value is used to indicate 'No Match' as well as indicating the expression was never evaluated ('Capture Group x did not Participate in the Match'), usually as a result of an Optional Quantifier used – ' ? '.

2. The ' ? ' Metacharacter when used with a Greedy Quantifier, makes that Quantifier 'Lazy', i.e. .*? or .+?, but when used in conjunction with a sub-expression, it makes that sub-expression optional. Not too much difference technically speaking but in terms of learning I think using both the terms, 'optional' and 'lazy' better explains two different scenarios.

String Manipulation (Parsing)

Re-arranging year placement w variable lengths cont.

Variation:

To cover a 4 numeric digit year for 'YYYY-YYYY', i.e., Car name 697 (123) '1964-1966 by some person.jpg

Match: (.+) '(\d{4})(-\d+)?(.*?)$
Replace: \2 \1\4

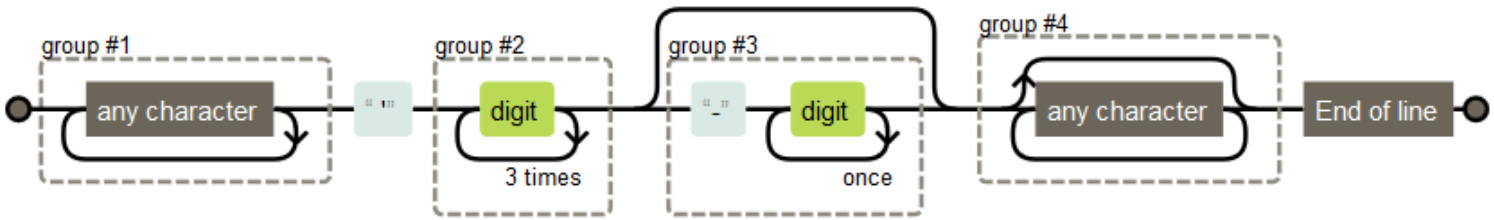
Name	New Name
Car name 697 (123) '1964-1966 by some person.jpg	1964 Car name 697 (123) by some person.jpg
mazda 626 (101) '1978-82.jpg	1978 mazda 626 (101).jpg
ford focus (107) '2010.jpg	2010 ford focus (107).jpg
alfa romeo giulia 1600 (101) '1962-65 by bertone.jpg	1962 alfa romeo giulia 1600 (101) by bertone.jpg

Under the original RegEx, it would not provide the expected results:

Name	New Name
Car name 697 (123) '1964-1966 by some person.jpg	1964 Car name 697 (123)66 by some person.jpg

Under the new RegEx, it does:

Name	New Name
Car name 697 (123) '1964-1966 by some person.jpg	1964 Car name 697 (123) by some person.jpg



The Visualization doesn't change

Sample:

```
.....................................................
Match 1: Car name 697 (123) '1964-1966 by some person
Group 1: Car name 697 (123)
Group 2: 1964
Group 3: -1966
Group 4:  by some person
```

Forum Contributor: TheRegExpMaster

String Manipulation (Parsing)

Re-arranging year placement w variable lengths cont.

Variation: cont.

Using: Car name 697 (123) '1964-1966 by some person.jpg

Match: (.+) '(\d{4})(-\d+)?(.*?)$

Where:

\1 = Car name 697 (123) \2 = 1964 \3 = -1966 \4 = <space> by some person

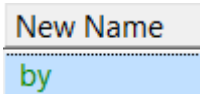
Tim Analysis:

1. .	Match against any character	Capture Group 1 = C
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) followed by …	Changes Capture Group 1 = Car name 697 (123)
4. ‘	.. match against single apostrophe (Not Captured) followed by …	
5. \d{4}	.. Match against 4 numeric digits followed by …	Capture Group 2 = 1964
6. <hyphen>	.. Match against <hyphen> followed by …	Capture Group 3 = -
7. \d	Match against Numeric Digit	Capture Group 3 = -1
8. +	Make it Greedy	Capture Group 3 = -1966
9. ?	Make last expression optional	Capture Group 3 = Unchanged
10. .*	Match against remaining string	Capture Group 4 = <space> by some person
11. ?	Make last expression optional	Changes Capture Group 4 = No Value
12. $	Must be at EOL	Changes Capture Group 4 = <space> by some person

Tim Notes:

1. The only difference between the modified and the original is one minor change from, ‘ \d{2} ’ to ‘ \d + ’
2. In RegEx a small change can mean a big difference.
3. In step 11, making Capture Group 4 optional removes the value ‘<space> by some person’, but by specifying that this last match must take place from the end of the string, restores the value This is because the ‘ ? ’ backtracks the engine to the position just between ‘1966’ and the <space> that follows. This is evidenced by:

(.*?...)



I added a few <dot> Metacharacters to provide positioning. The first dot revealed the current position as the <space> after ‘1966’, followed by the ‘b’ and ‘y’ to match against the second and third <dot> Metacharacters.

By including the ‘ $ ’, the RegEx engine moves forward and repeats the match against the remaining string.

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal

Tim Contribution –

I got this idea from a posting to which RegExpMaster replied with a RegEx, but it was way too restrictive. I found that it would make a great learning lesson though.

Samples:

RA-617.pdf
RA-617_11006175messen.pdf
RA-6175_.pdf
RA-61753_.pdf

Tim Reflections on Pattern Matching:

1. Look for pattern but don't limit yourself too much if you can help it.

    i.e. instead of matching RA-, try something more universal so it can apply to other types of files

  a. Pattern Established - <text><hyphen><numeric digits><rest of string>

2. Apply a RegEx that can match as many filenames without being too restrictive.

3. First try parsing the string of the elements that you want to pattern.

4. Once you have parsed the string, rearranging, adding and removing elements is simplified.

5. If that doesn't work, then you will have to change the pattern to become more restrictive.

6. Be conscientious that a 'universal' RegEx doesn't do more harm than good – rename with unintended consequences.

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

RA-617.pdf
RA-617_11006175messen.pdf
RA-6175_.pdf
RA-61753_.pdf

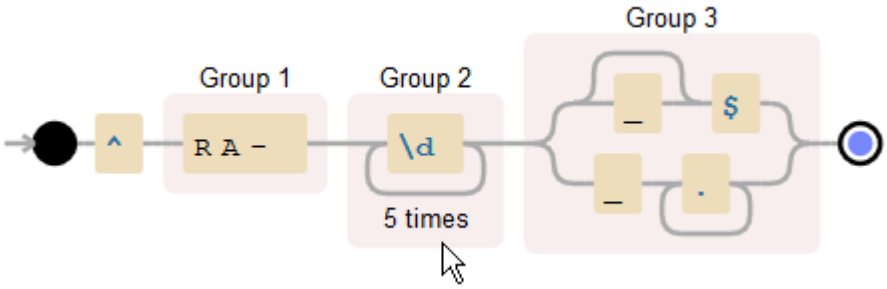
In this example, a restrictive form of the pattern would be:

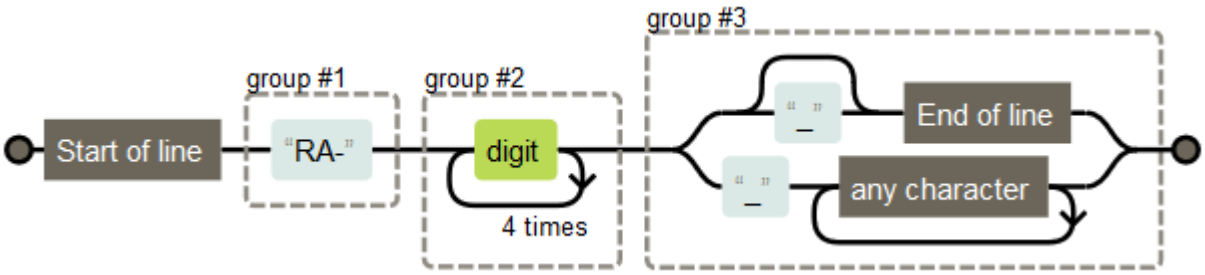
Match: ^(RA-)(\d{5})(_?$|_.+)
Replace: \1

Tim Notes:

1. using a Replace string of \1 only to illustrate the matched file -

Name	New Name
RA-617_11006175messen.pdf	RA-617_11006175messen.pdf
RA-6175_.pdf	RA-6175_.pdf
RA-61753_.pdf	RA-.pdf
RA-617.pdf	RA-617.pdf





Tim Notes:

1. The Reason the Second Railroad Diagram always reports 1 less repetition, i.e., '4 times', than what is specified, i.e. {5}, is because it counts the 'numeric digit' '\d' as the first match, so the number '4' represents the additional matches.

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

Using: RA-61753_.pdf

In this example, a restrictive form of the pattern would be: cont.

Match: ^(RA-)(\d{5})(_?$|_.+)

Where:

\1 = RA- \2 = 61753 \3 = _

This uses an Alternate for Capture Group 3:

This	OR	This
_?$	|	_. +

This matches:

First Alternate -

$ = a filename that ends with an underscore with no following text, i.e., RA-61753.pdf
_?$ = a filename that doesn't have an underscore and simply ends, i.e., RA-61753.pdf

Second Alternate:

_.+ = a filename that has an underscore followed by additional text, i.e., RA-61753_more text.pdf

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. RA	Match against string, 'RA'	Capture Group 1 = RA
3. -	Match against <hyphen> followed by …	Capture Group 1 = RA-
4. \d{5}	Match against 5 numeric digits	Capture Group 2 = 61753
5. _?$ | _.+	Match against either (see above)	Capture Group 3 = _

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

RA-617.pdf
RA-617_11006175messen.pdf
RA-6175_.pdf
RA-61753_.pdf

Match: ^(RA-)(\d{5})(_?$|_.+)

Sample:

```
.....................
Match 1: RA-61753_
Group 1: RA-
Group 2: 61753
Group 3: _
```

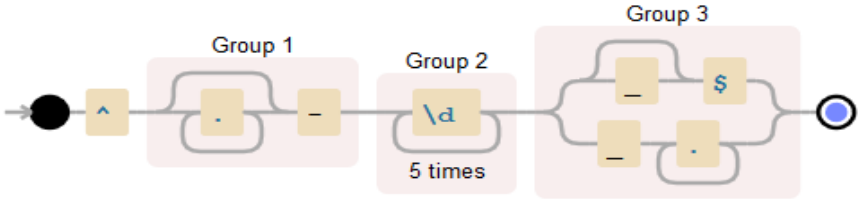
This matches only those filenames that have the following pattern: RA-##### (5 numeric digits only).
In the samples, this would only match against RA-61753_.pdf.

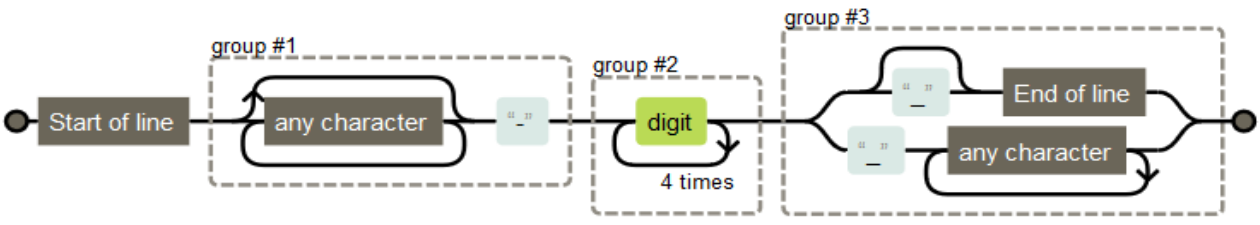
1. Change the 'RA' to match 'any' text (.*) at the beginning of the file ^ (BOL) up to the <hyphen> = ^(.*?-)

Match: ^(.*?-)(\d{5})(_?$|_.+)
Replace: \1

This works. Same result, but now you are not limited to RA. However, it still must precede a <hyphen>.

Name	New Name
RA-617_11006175messen.pdf	RA-617_11006175messen.pdf
RA-6175_.pdf	RA-6175_.pdf
RA-61753_.pdf	RA-.pdf
RA-617.pdf	RA-617.pdf





String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

Expanded Samples:

RA-61753_.pdf
GHT-61753_.pdf

1. Change the 'RA' to match 'any' text (.*) at the beginning of the file ^ (BOL) up to the <hyphen> = ^(.*?-) cont.

Match: ^(.*?-)(\d{5})(_?$|_.+)
Replace: \1

Name	New Name
GHT-61753_.pdf	GHT-
RA-61753_.pdf	RA-

Using: GHT-61753_.pdf

Where:

\1 = GHT- \2 = 61753 \3 = _

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = G
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. -	Match against <hyphen> followed by …	Capture Group 1 = GHT-
6. \d{5}	Match against 5 numeric digits	Capture Group 2 = 61753
7. _?$ | _.+	Match against either .. (see alternate explanation previous pages)	Capture Group 3 = _

Tim Notes:

1. Just a reminder that BOL is Beginning of Line or position zero. EOL is End of Line, or the position just after the string and before any file extension.

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

RA-617.pdf
RA-6175_.pdf
GHT-61753_.pdf
RA-617_11006175messen.pdf
RA-61753_.pdf
FG-61753_.pdf

Match: ^(.*?-)(\d{5})(_?$|_.+)

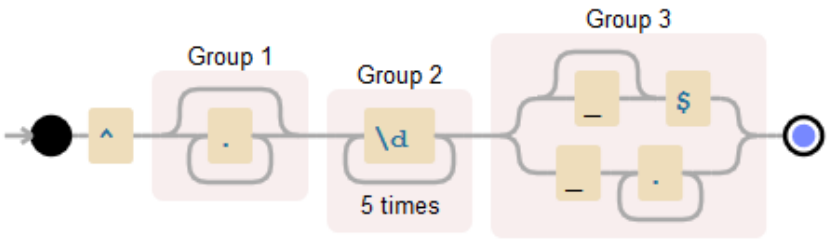
Sample: FG-61753_.pdf

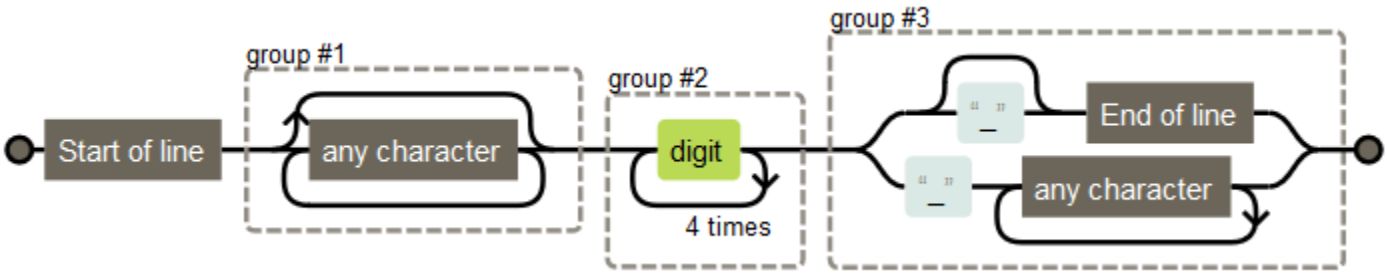
```
.....................
Match 1: FG-61753_
Group 1: FG-
Group 2: 61753
Group 3: _
```

2. Now let's eliminate the <hyphen> in ^(.*?-) . It is not needed because the (.*?) will select the text up to the numeric digits (\d{5}):

Match: ^(.*?)(\d{5})(_?$|_.+)
Replace: \1

Name	New Name
RA-617_11006175messen.pdf	RA-617_11006175messen.pdf
RA-6175_.pdf	RA-6175_.pdf
RA-61753_.pdf	RA-.pdf
RA-617.pdf	RA-617.pdf





String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

2. Now let's eliminate the <hyphen> in ^(.*?-) . It is not needed because the (.*?) will select the text up to the numeric digits (\d{5}): cont.

Match: ^(.*?)(\d{5})(_?$|_.+)

Using: GHT-61753_.pdf

Where:

\1 = GHT- \2 = 61753 \3 = _

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = G
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. \d{5}	Match against 5 numeric digits	Capture Group 2 = 61753
		Changes Capture Group 1= GHT-
6. _?$ | _.+	Match against either .. (see alternate explanation previous pages)	Capture Group 3 = _

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

RA-617.pdf
RA-6175_.pdf
GHT-61753_.pdf
RA-617_11006175messen.pdf
RA-61753_.pdf
FG-61753_.pdf

Sample:

```
.....................
Match 1: RA-61753_
Group 1: RA-
Group 2: 61753
Group 3: _
```

Sample:

```
.........................................
Match 1: FG-61753_RA-617_11006175messen
Group 1: FG-
Group 2: 61753
Group 3: _RA-617_11006175messen
```

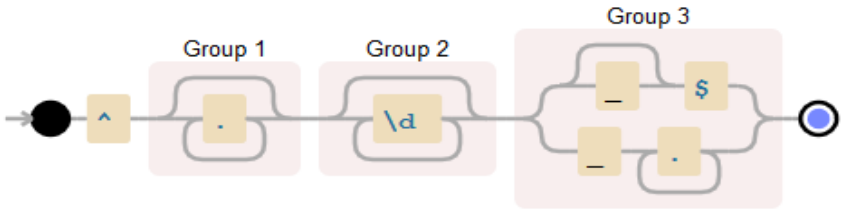
3. Finally let's get rid of the restriction on the number of numeric digits matched – we could be here all day otherwise running a different RegEx for every instance pertaining to the number of numeric digits:

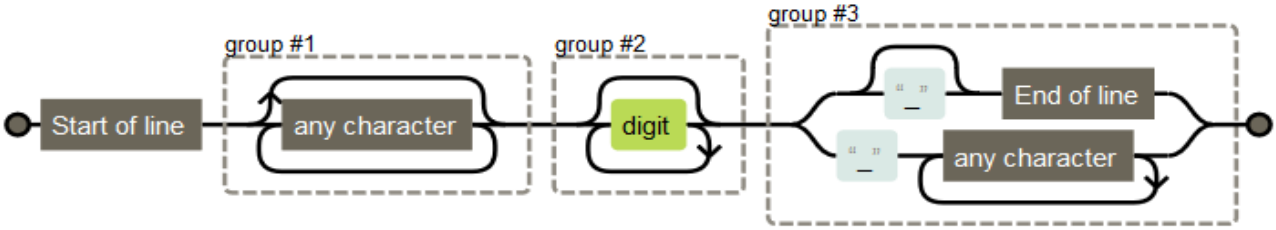
i.e.

^(.*?)(\d{5})(_?$|_.+)
^(.*?)(\d{4})(_?$|_.+)
^(.*?)(\d{3})(_?$|_.+)
.
etc.

Match: ^(.*?)(\d*)(_?$|_.+)
Replace: \1

Name	New Name
RA-617_11006175messen.pdf	RA-.pdf
RA-6175_.pdf	RA-.pdf
RA-61753_.pdf	RA-.pdf
RA-617.pdf	RA-.pdf





String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

Match: ^(.*?)(\d*)(_?$|_.+)

Sample:

```
....................
Match 1: RA-61753_
Group 1: RA-
Group 2: 61753
Group 3: _
```

All files matched.

Where:

\1 = RA-
\2 = numeric portion after the <hyphen>
\3 = remainder of text

\1 results:

Name	New Name
RA-617_11006175messen.pdf	RA-.pdf
RA-6175_.pdf	RA-.pdf
RA-61753_.pdf	RA-.pdf
RA-617.pdf	RA-.pdf

\2 results:

Name	New Name
RA-617_11006175messen.pdf	617.pdf
RA-6175_.pdf	6175.pdf
RA-61753_.pdf	61753.pdf
RA-617.pdf	617.pdf

\3 results:

Name	New Name
RA-617_11006175messen.pdf	_11006175messen.pdf
RA-6175_.pdf	_.pdf
RA-61753_.pdf	_.pdf
RA-617.pdf	.pdf

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

3. Finally let's get rid of the restriction on the number of numeric digits matched – we could be here all day otherwise running a different RegEx for every instance pertaining to the number of numeric digits: cont.

Using: RA-617_11006175messen.pdf

Match: ^(.*?)(\d*)(_?$|_.+)

Where:

\1 = RA- \2 = 617 \3 = _11006175messen

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = R
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
	Backtracking	
5. \d	Match against Numeric Digit	Capture Group 2 = 6
6. *	Make it Greedy	Capture Group 2 = BOL
	Remember, it was backtracking	Capture Group 1 = No Change
7. _?$ | _.+	Match against either ..	Capture Group 3 =
	(see alternate explanation previous pages)	_11006175messen
		Changes Capture Group 2= 617
		Changes Capture Group 1= RA-

Tim Notes:

1. In this sample, the second alternate is selected which matches against ' _.+ ' because there is additional text that follows after the underscore.

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

Now let's discuss the Alternate part of the RegEx, some of which I have already touched upon here and discussed in previous examples.

(_?$|_.+) - this is an Alternate, or in programmer's lingo, an OR statement. It instructs the evaluation of the expression to select either of two options. Those options being –

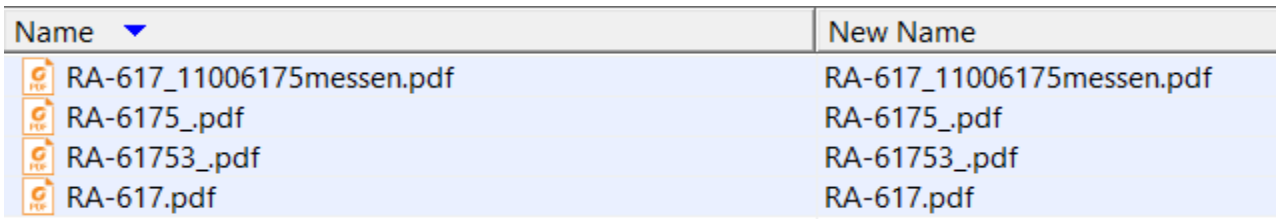
Option #1 - _?$
Expression is TRUE if there is an underscore at the end of the filename (EOL as indicated by using the $ anchor) BUT, the '?' makes this optional. In other words, it doesn't matter if there is an underscore at the end of the filename or not. The expression will still be True. As long as the EOL file is reached at this point. This matches, RA-617.pdf, RA-6175_.pdf and RA-61753_.pdf.

Option #2 - _.+
Expression is TRUE if there is (definitely) an underscore FOLLOWED by additional characters or text. The text/characters are represented by the single dot and the + makes it greedy. Greedy means to capture the remaining characters to EOL (End of Line). This matches, RA-617_11006175messen.pdf.

This is why when I defined the pattern established by the RegEx in the first part of this segment, I did not include anything other than RA-#####, because the RegEx was already taking in consideration the end section of the samples.

One more note of interest.

If I were to specify \1 \2 \3 in the replacement string (backreferences to Capture Groups 1-3), I would get this:

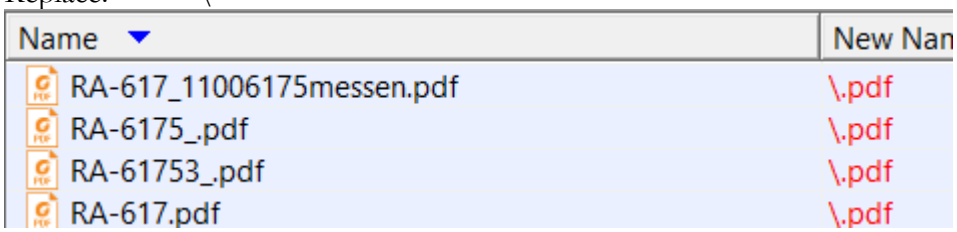


Name	New Name
RA-617_11006175messen.pdf	RA-617_11006175messen.pdf
RA-6175_.pdf	RA-6175_.pdf
RA-61753_.pdf	RA-61753_.pdf
RA-617.pdf	RA-617.pdf

No New Name Change. Does this mean the RegEx didn't work and matched nothing? No not necessarily. Here it is just the opposite. It matched everything but changed nothing – therefore no New Name.

You can see this clearly by using a replacement of \1 or \2 or \3 separately as we did before. Another indication is if you just place a single slash for the Replace string:

Replace: \



Name	New Nan
RA-617_11006175messen.pdf	\.pdf
RA-6175_.pdf	\.pdf
RA-61753_.pdf	\.pdf
RA-617.pdf	\.pdf

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

The backslash is displayed in red (followed by the file extension) \.pdf indicating that something is there (matches) but it comes up invalid. Something isn't quite right. If there was something wrong with the RegEx, and it didn't match, using just the '\' again for example, you would instead get:

Name	New Name
RA-617_11006175messen.pdf	RA-617_11006175messen.pdf
RA-6175_.pdf	RA-6175_.pdf
RA-61753_.pdf	RA-61753_.pdf
RA-617.pdf	RA-617.pdf

Meaning the RegEx in this case failed and didn't match, rather than the 'matched everything and changed nothing' scenario.

Conclusion: Sometimes you have to dig a little deeper to find a problem.

So if it matched everything and changed nothing, what's the point?

Just add *something* in the Replace string to see a change – here I added a zero forcing New Name:

i.e.,

Replace: \10\2\3

Name	New Name
RA-617_11006175messen.pdf	RA-0617_11006175messen.pdf
RA-6175_.pdf	RA-06175_.pdf
RA-61753_.pdf	RA-061753_.pdf
RA-617.pdf	RA-0617.pdf

i.e., Here I rearranged the string forcing New name

Replace: \2\1\3

Name	New Name
RA-617_11006175messen.pdf	617RA-_11006175messen.pdf
RA-6175_.pdf	6175RA-_.pdf
RA-61753_.pdf	61753RA-_.pdf
RA-617.pdf	617RA-.pdf

You get the idea.

String Manipulation (Parsing)

Exercise in parsing string containing numeric digits – Restrictive to Universal cont.

Tim Contribution – cont.

Finally, what about this?

Name	New Name
RA-617_11006175messen.pdf	617RA-_11006175messen\4.pdf
RA-6175_.pdf	6175RA-_\4.pdf
RA-61753_.pdf	61753RA-_\4.pdf
RA-617.pdf	617RA-\4.pdf

There indicates the expression is invalid like before, *but* the fact that a numbered backreference is displayed in New name identifies a backreference to an undefined Capture Group. In other words, you have 3 Capture Groups in your RegEx but you have a \4 representative in your Replace string: Remove the fourth capture group to correct the RegEx.

Match: ^(.*?)(\d*)(_?$|_.+)
Replace: \2\1\3\4

Tim Analysis:

1. (.*?) Capture Group 1
2. (\d*) Capture Group 2
3. (_?$|_.+) Capture Group 3

change to (Removing Capture Group 4) –

Replace: \2\1\3

and you get the successful New name:

Name	New Name
RA-617_11006175messen.pdf	617RA-_11006175messen.pdf
RA-6175_.pdf	6175RA-_.pdf
RA-61753_.pdf	61753RA-_.pdf
RA-617.pdf	617RA-.pdf

String Manipulation (Parsing)

A Parsing Lesson

Pink Floyd - DSOTM - 1973.ext to Pink Floyd - 1973 - DSOTM.ext

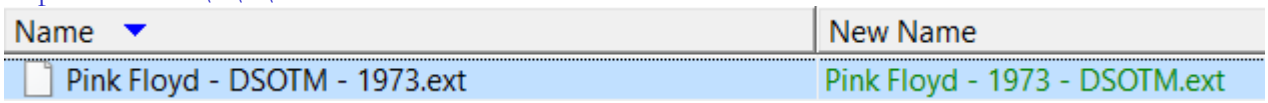
Observations:

1. Match everything until before first <space> <hyphen>, and store as Capture Group 1 = Pink Floyd
2. Match everything until before second <space> <hyphen>, and store as Capture Group 2 = - DSOTM
3. Match the rest, and store as Capture Group 3 = - 1973

Each of these performs the same result – parse the string:

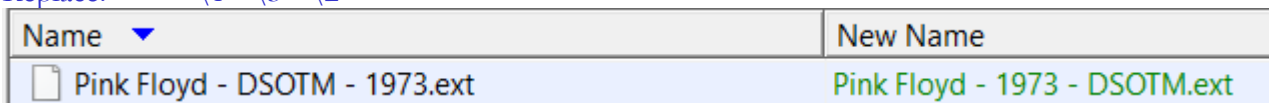
Example # 1

Match: (.+?)(- .+?)(- \d\d\d\d)
Replace: \1\3\2



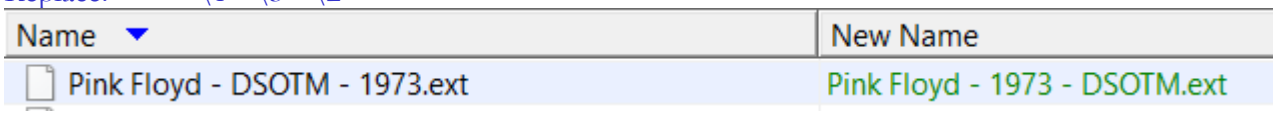
Example #2

Match: (.+?) - (.+?) - (\d\d\d\d)
Replace: \1 - \3 - \2



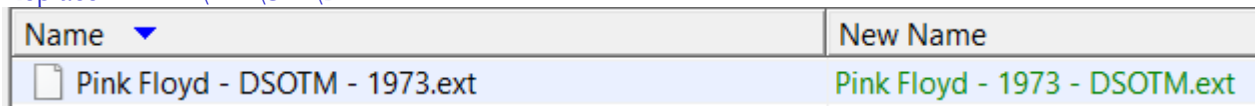
Example #3

Match: (.+?) - (.+?) - (\d{4})
Replace: \1 - \3 - \2



Example #4

Match: (.+) - (.+) - (.+)
Replace: \1 - \3 - \2



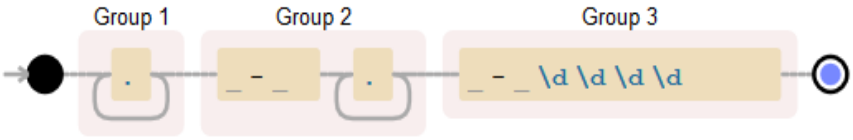
Where:
\1 = Pink Floyd \2 = DSOTM \3 = 1973

String Manipulation (Parsing)

A Parsing Lesson cont.

Pink Floyd - DSOTM - 1973.ext to Pink Floyd - 1973 - DSOTM.ext

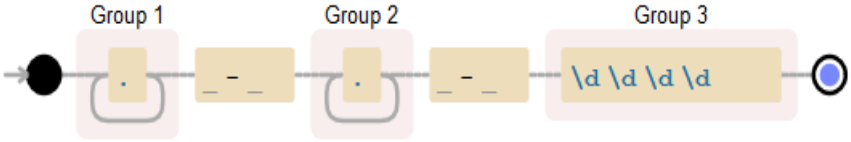
Example #1: (.+?)(- .+?)(- \d\d\d\d)



Sample:

```
.....................................
Match 1: Pink Floyd - DSOTM - 1973
Group 1: Pink Floyd
Group 2:  - DSOTM
Group 3:  - 1973
```

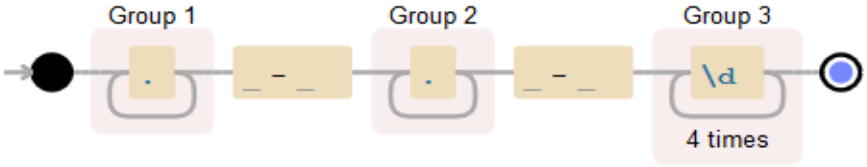
Example #2: (.+?) - (.+?) - (\d\d\d\d)



Sample:

```
.....................................
Match 1: Pink Floyd - DSOTM - 1973
Group 1: Pink Floyd
Group 2: DSOTM
Group 3: 1973
```

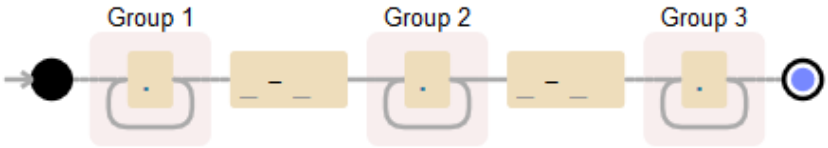
Example #3: (.+?) - (.+?) - (\d{4})



Sample: same output as #2

```
.....................................
Match 1: Pink Floyd - DSOTM - 1973
Group 1: Pink Floyd
Group 2: DSOTM
Group 3: 1973
```

Example #4: (.+) - (.+) - (.+)



Sample: same output as # 2

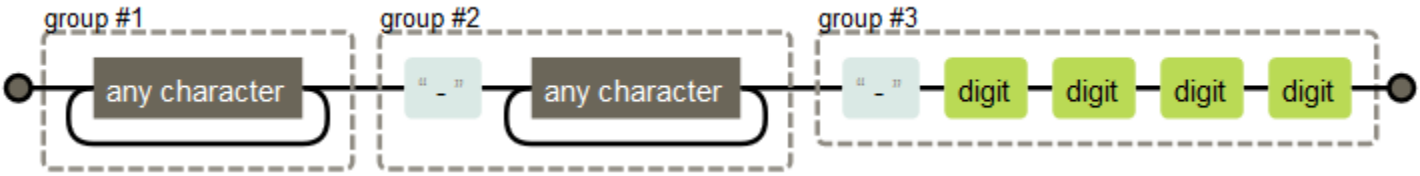
```
.....................................
Match 1: Pink Floyd - DSOTM - 1973
Group 1: Pink Floyd
Group 2: DSOTM
Group 3: 1973
```

String Manipulation (Parsing)

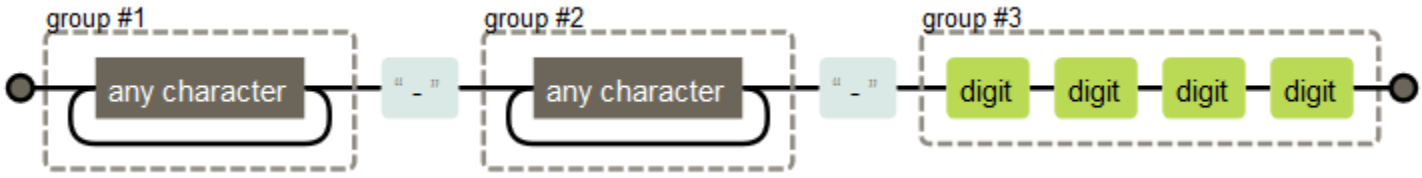
A Parsing Lesson cont.

Pink Floyd - DSOTM - 1973.ext to Pink Floyd - 1973 - DSOTM.ext

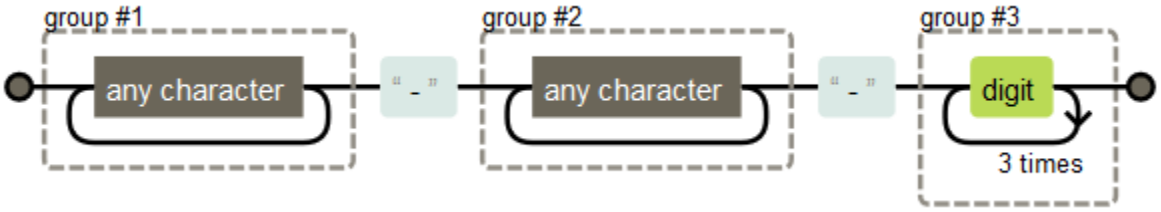
Example #1: (.+?)(- .+?)(- \d\d\d\d)



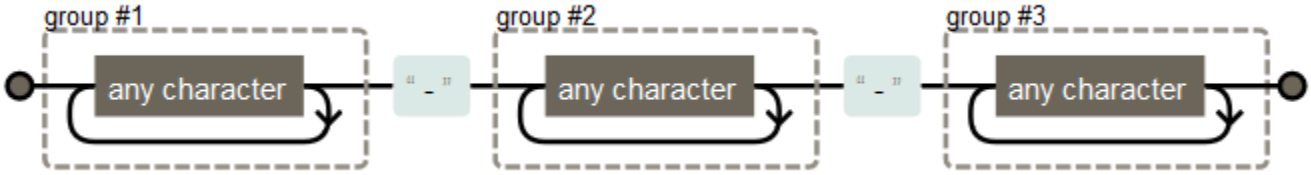
Example #2: (.+?) - (.+?) - (\d\d\d\d)



Example #3: (.+?) - (.+?) - (\d{4})



Example #4: (.+) - (.+) - (.+)



Forum Contributor: Stefan

String Manipulation (Parsing)

A Parsing Lesson cont.

Using: Pink Floyd - DSOTM - 1973.ext

Example #1: (.+?)(- .+?)(- \d\d\d\d)

Where:

\1 = Pink Floyd \2 = <space> - DSOTM \3 = <space> - 1973

Tim Analysis:

1..	Match against any character	Capture Group 1 = P
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = P
4. <space>	Match against a <space> character This is the <space> after 'Pink'	Capture Group 2 = <space> Changes Capture Group 1 = Pink
5. -	Match against <hyphen> This is the <hyphen> after 'Floyd <space>'	Capture Group 2 = <space> - Changes Capture Group 1 = Pink Floyd
6..	Match against any character This is the <space> after 'Floyd <space> <hyphen>'	Capture Group 2 = <space>-<space>
7. +	Make it Greedy	Capture Group 2 = <space> - DSOTM - 1973
8. ?	But not too Greedy	Capture Group 2 = <space> - <space>
9. <space>	Match against a <space> character This is the <space> after 'DSOTM'	Capture Group 3 = <space> Changes Capture Group 2 = <space> - DSOTM
10. -	Match against <hyphen> This is the <hyphen> after 'DOTSM <space>'	Capture Group 3 = <space> -
11. <space>	Match against a <space> character This is the <space> after 'DSOTM <space> <hyphen>	Capture Group 3 = <space> - <space>
12. \d\d\d\d	Match against 4 numeric digits	Capture Group 3 = <space> - 1973

String Manipulation (Parsing)

A Parsing Lesson cont.

Using: Pink Floyd - DSOTM - 1973.ext

Example #3: (.+?) - (.+?) - (\d{4})

Where:

\1 = Pink Floyd \2 = DSOTM \3 = 1973

Tim Analysis:

1..	Match against any character	Capture Group 1 = P
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = P
4. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Pink'	Changes Capture Group 1 = Pink
5. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'Floyd <space>'	Changes Capture Group 1 = Pink Floyd
6. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Floyd <space> <hyphen>'	
7..	Match against any character	Capture Group 2 = D
8 +	Make it Greedy	Capture Group 2 = DSOTM - 1973
9. ?	But not too Greedy Backtracks to position just before the 'D'	Capture Group 2 = null
10. <space>	Match against a <space> character (Not Captured) This is the <space> after 'DSOTM'	Changes Capture Group 2 = DSOTM
11. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'DOTSM <space>'	
12. <space>	Match against a <space> character (Not Captured) This is the <space> after 'DSOTM <space> <hyphen>	
13. \d{4}	Match against 4 numeric digits	Capture Group 3 = 1973

Tim Notes:

1. Example #2 and Example #3 are identical. Example #3 is the simplified version of Example #2 using a Quantifier of \d{4} instead of \d\d\d\d

String Manipulation (Parsing)

A Parsing Lesson cont.

Using: Pink Floyd - DSOTM - 1973.ext

Example #4: (.+) - (.+) - (.+)

Where:

\1 = Pink Floyd \2 = DSOTM \3 = 1973

Tim Analysis:

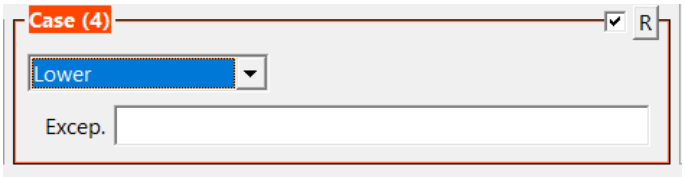
1..	Match against any character	Capture Group 1 = P
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'DSOTM -'	Changes Capture Group 1 = Pink Floyd - DSOTM -
5. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'DSOTM <space>'	Changes Capture Group 1 = Pink Floyd - DSOTM <space>
6. <space>	Match against a <space> character (Not Captured) This is the <space> after 'DSOTM'	Changes Capture Group 1 = Pink Floyd - DSOTM
7..	Match against any character	Capture Group 2 = 1
8 +	Make it Greedy Current position = EOL	Capture Group 2 = 1973
9. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'DSOTM -'	Changes Capture Group 2 = DSOTM - Changes Capture Group 1= Pink Floyd
10. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'DOTSM <space>' Changes step 10 to the <space> after 'DSOTM' Moving Forward.	Changes Capture Group 2 = DSOTM'
11. <space>	Match against a <space> character (Not Captured) This is the <space> after 'DSOTM <space> <hyphen>	
12..+	Match against remaining string	Capture Group 3 = 1973

String Manipulation (Parsing)

Rearrange – take first Initial, append to end and lowercase all

Jane Dog.png	to	dog_j.png
Carolyn Elizabet Walker.jpg	to	elizabet walker_c.jpg
Martin Fisher Miller.txt	to	fisher miller_m.txt
Sarah Maria Campbell.jpg	to	maria Campbell_s.jpg

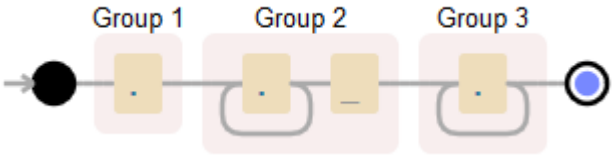
'Section #4: Case' –



Case: Lower(case)

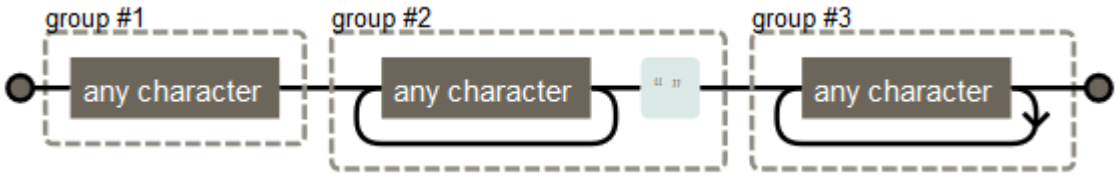
Match: (.)(.+?)(.+)
Replace: \3_\1

Name	New Name
Sarah Maria Campbell.jpg	maria campbell_s.jpg
Martin Fisher Miller.txt	fisher miller_m.txt
Carolyn Elizabet Walker.jpg	elizabet walker_c.jpg
Jane Dog.png	dog_j.png



Sample:

```
Match 1: Carolyn Elizabet Walker
Group 1: C
Group 2: arolyn
Group 3: Elizabet Walker
```



Contributor and *Tim Analysis:*

1. .	(dot) Capture first character of string	Capture Group 1
2. .+?<space>	Capture rest of first name with <space> – not used (.+?<space>)	Capture Group 2
3. .+	Capture remainder of string	Capture Group 3

Forum Contributor: Stefan

String Manipulation (Parsing)

Rearrange – take first Initial, append to end and lowercase all cont.

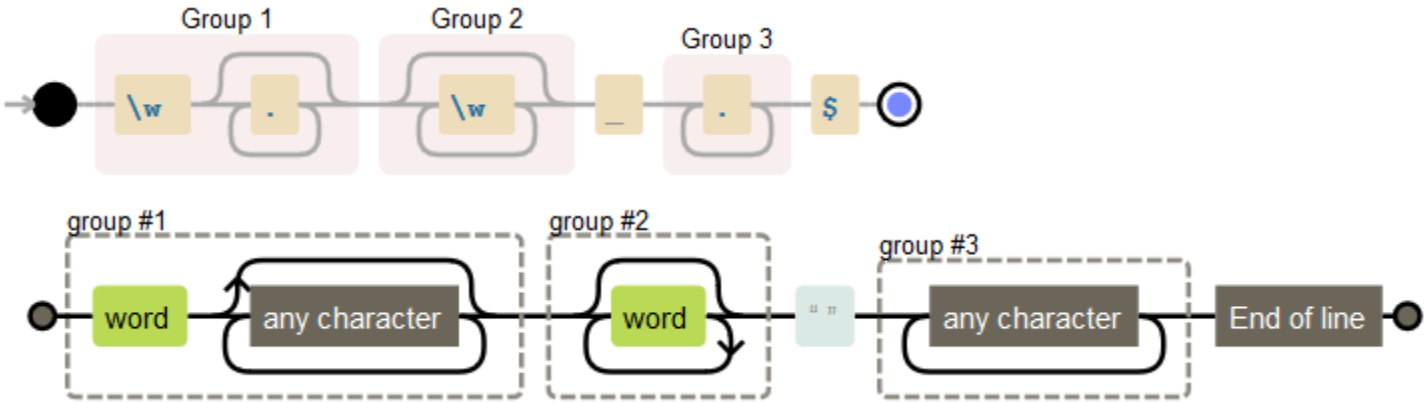
Jane Dog.png	to	dog_j.png
Carolyn Elizabet Walker.jpg	to	elizabet walker_c.jpg
Martin Fisher Miller.txt	to	fisher miller_m.txt
Sarah Maria Campbell.jpg	to	maria Campbell_s.jpg

Tim Contribution –

Another way – does the same thing but without capturing the <space> as well:

Match: (\w.*?)(\w*) (.+?)$
Replace: \3_\1

Name	New Name
Sarah Maria Campbell.jpg	maria campbell_s.jpg
Martin Fisher Miller.txt	fisher miller_m.txt
Carolyn Elizabet Walker.jpg	elizabet walker_c.jpg
Jane Dog.png	dog_j.png



Sample:

```
.....................................
Match 1: Carolyn Elizabet Walker
Group 1: C
Group 2: arolyn
Group 3: Elizabet Walker
```

Tim Analysis:

1. \w.*?	Capture first initial of word	Capture Group 1 = C
2. \w*	Capture rest of first name	Capture Group 2 = arolyn
3. <space>	<space> Not Captured	
4. .+?	Capture remainder of string	Capture Group 3 = Elizabet Walker
5. $	Must be at EOL (End of Line)	

String Manipulation (Parsing)

Demonstration of 3 Methods to Rearrange a not-so-simple string

DNS-24597-HTD.pdf	to	24597-DNS--HTD.pdf
PO-24567 VXI.pdf	to	24567-PO-VXI.pdf
POS-24655-XfD.pdf	to	24655-POS-XfD.pdf
PO 24604 EMM.pdf	to	24604-PO-EMM.pdf

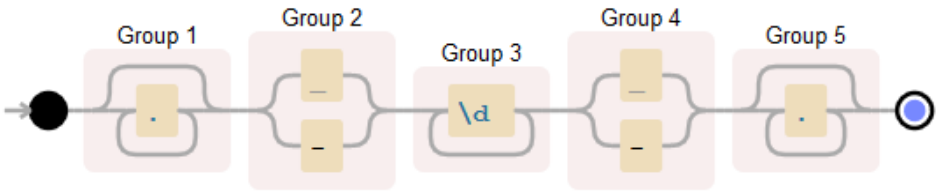
Observation #1:

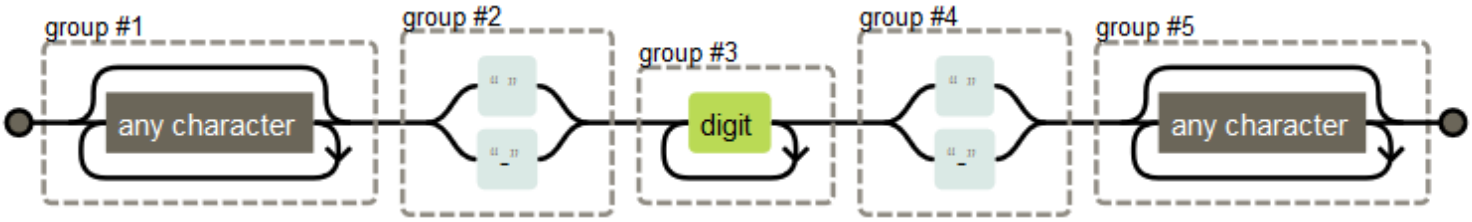
1. Looks simple, but not all of the filenames have <hyphens>.

Method #1: You need to use alternating groups (an OR) to match either the <space> or the <hyphen>:

Match: (.*)(|-)(\d+)(|-)(.*)
Replace: \3-\1-\5

Name	New Name
DNS-24597-HTD.pdf	24597-DNS-HTD.pdf
PO-24567 VXI.pdf	24567-PO-VXI.pdf
POS-24655-XfD.pdf	24655-POS-XfD.pdf
PO 24604 EMM.pdf	24604-PO-EMM.pdf





Sample:

```
.......................
Match 1: DNS-24597-HTD
Group 1: DNS
Group 2: -
Group 3: 24597
Group 4: -
Group 5: HTD
```

Forum Contributor: bitmonger

String Manipulation (Parsing)

Demonstration of 3 Methods to Rearrange a not-so-simple string cont.

DNS-24597-HTD.pdf	to	24597-DNS--HTD.pdf
PO-24567 VXI.pdf	to	24567-PO-VXI.pdf
POS-24655-XfD.pdf	to	24655-POS-XfD.pdf
PO 24604 EMM.pdf	to	24604-PO-EMM.pdf

Match: (.*)(|-)(\d+)(|-)(.*)

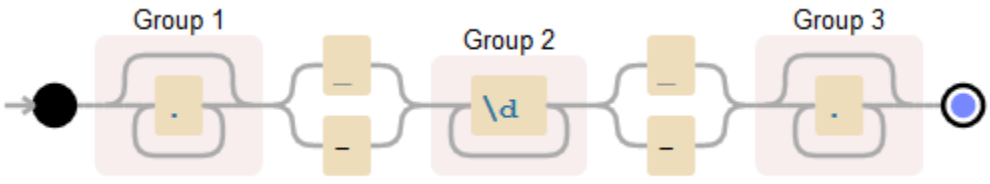
Contributor and *Tim Analysis:*

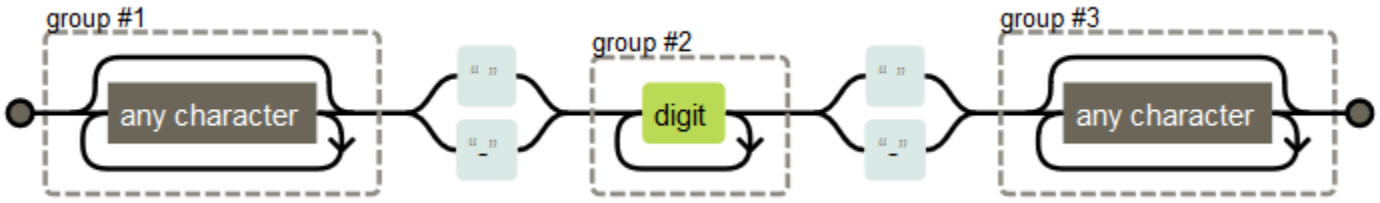
1. .*	Match all up until…	Capture Group 1
2. <space>|-	… Match either <space> OR the <hyphen>	Capture Group 2
3. \d	Match first numeric digit	Capture Group 3
4. +	Make it Greedy so it includes all of the numeric digits till…	Capture Group 3
5. <space>|-	… Match either <space> OR the <hyphen>	Capture Group 4
6. .*	Match against remainder of string	Capture Group 5

Method #2: If you wanted to make the alternating groups non-capturing you could use Non-Marking Groups:

Match: (.*)(?: |-)(\d+)(?: |-)(.*)
Replace: \2-\1-\3

Name	New Name
DNS-24597-HTD.pdf	24597-DNS-HTD.pdf
PO-24567 VXI.pdf	24567-PO-VXI.pdf
POS-24655-XfD.pdf	24655-POS-XfD.pdf
PO 24604 EMM.pdf	24604-PO-EMM.pdf





Tim Notes:

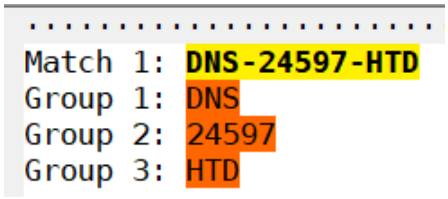
1. See the RegEx Manuals available in both Volume I and Volume II for more information regarding Non-Marking Groups.

String Manipulation (Parsing)

Demonstration of 3 Methods to Rearrange a not-so-simple string cont.

DNS-24597-HTD.pdf	to	24597-DNS--HTD.pdf
PO-24567 VXI.pdf	to	24567-PO-VXI.pdf
POS-24655-XfD.pdf	to	24655-POS-XfD.pdf
PO 24604 EMM.pdf	to	24604-PO-EMM.pdf

Sample:



Tim Notes:

1. (?: |-) The syntax (?:) used is a N**on-Marking Group** (refer to the Regular Expressions Manual found in the Appendix of Volume 1 Bulk Rename Utility Operations Manual). It will match but not capture. In this example it moves the RegEx engine and saves on Capture Groups – only allowed 9 total.

Forum Contributor: bitmonger

Observation #2:

1. The number in the center is bounded by 2 non-numeric digits, either <hyphens> or <space> characters.

-24597-
<space> 24604 <space>

a. A numeric digit is represented by the \d.

b A Non-Numeric Digit Metacharacter is represented by a \D.
1. This includes anything that is not a numeric digit as defined under the PCRE v7.5 specification., i.e., letters, characters, including a <space> and a <hyphen>.
2. This is the same as [^0-9]

String Manipulation (Parsing)

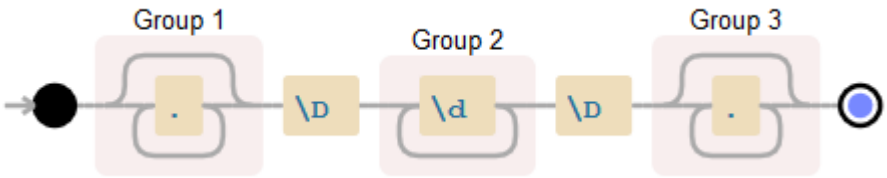
Demonstration of 3 Methods to Rearrange a not-so-simple string cont.

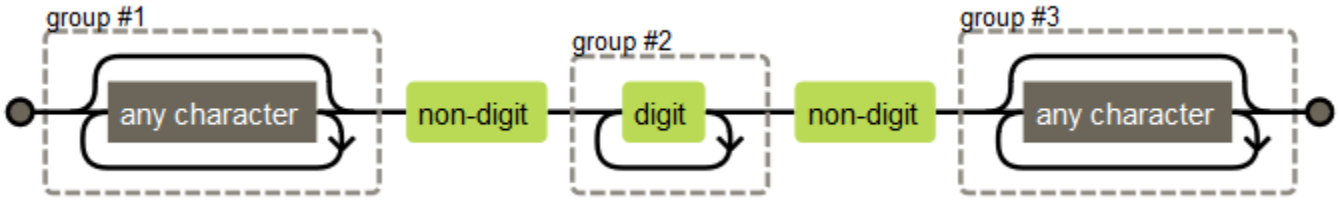
DNS-24597-HTD.pdf	to	24597-DNS--HTD.pdf
PO-24567 VXI.pdf	to	24567-PO-VXI.pdf
POS-24655-XfD.pdf	to	24655-POS-XfD.pdf
PO 24604 EMM.pdf	to	24604-PO-EMM.pdf

Method #3: Capture all characters at the beginning and end of the string and 'Bind' the Numeric digits in the center:

Match: (.*)\D(\d+)\D(.*)
Replace: \2-\1-\3

Name	New Name
DNS-24597-HTD.pdf	24597-DNS-HTD.pdf
PO-24567 VXI.pdf	24567-PO-VXI.pdf
POS-24655-XfD.pdf	24655-POS-XfD.pdf
PO 24604 EMM.pdf	24604-PO-EMM.pdf





Sample:

```
.........................
Match 1: DNS-24597-HTD
Group 1: DNS
Group 2: 24597
Group 3: HTD
```

Forum Contributor: bitmonger

String Manipulation (Parsing)

Demonstration of 3 Methods to Rearrange a not-so-simple string cont.

Using: DNS-24597-HTD.pdf

Method #3: cont.

Match: (.*)\D(\d+)\D(.*)

Where:

\1 = DNS \2 = 24597 \3 = HTD

Tim Analysis:

1. .	Match against any character	Capture Group 1 = D
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \D	Match against any Non-Numeric Digit (Not Captured) = 'D' This is the 'D' of 'DNS' (backtracking)	
4. \d	Match against Numeric Digit Changes step 3 Not Captured value = ' - ' This is the <hyphen> after 'DNS'	Capture Group 2 = 2 Changes Capture Group 1 = DNS
5. +	Make it Greedy	Capture Group 2 = 24597
6. \D	Match against any Non-Numeric Digit (Not Captured) = ' - '	
7. .*	Match against remaining string	Capture Group 3 = HTD

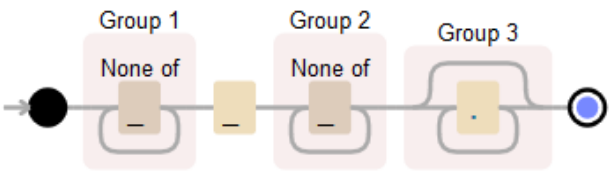
String Manipulation (Parsing)

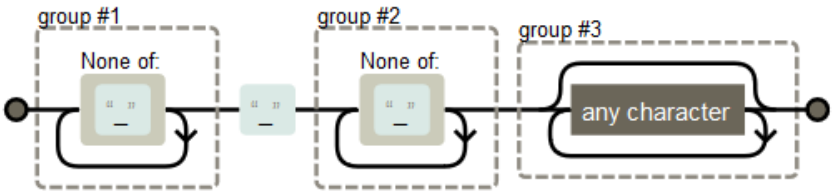
Rearrange Names and Remove Extraneous

Adam_Kennedy.jpg	to	Kennedy_Adam.jpg
Madison_James_LAA.jpg	to	James_Madison.jpg
Jones_Jack_LAA_2010-2011.jpg	to	Jack_Jones.jpg

Match: ([^_]+)_([^_]+)(.*)
Replace: \2_\1

Name	New Name
Adam_Kennedy.jpg	Kennedy_Adam.jpg
Madison_James_LAA.jpg	James_Madison.jpg
Jones_Jack_LAA_2010-2011.jpg	Jack_Jones.jpg





Sample:

```
.....................................
Match 1: Jones_Jack_LAA_2010-2011
Group 1: Jones
Group 2: Jack
Group 3: _LAA_2010-2011
```

Contributor Analysis:

Using Adam_Kennedy.jpg

1. ([^_]+)_	Match against one or more characters which are not an underscore, till an underscore is found = "Adam" + "_",	Capture Group 1
2. ([^_]+)	again the same, but till the end (or till an underscore is matched by next expression) = "Kennedy"	Capture Group 2
3. (.*)	Capture remainder of string	Capture Group 3

Forum Contributor: Stefan

String Manipulation (Parsing)

Change around names at end of filename

v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3
to
v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Patrick Hawes-Quanta Qualia.mp3

Observations:

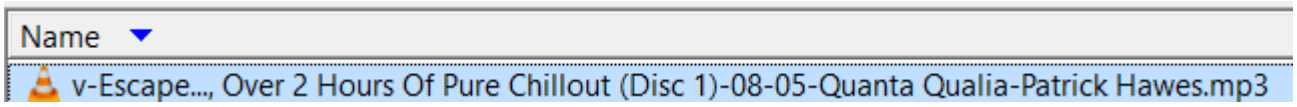
1. Swap the last 2 titles around, i.e. change the last part of this file from –

Quanta Qualia-Patrick Hawes to Patrick Hawes-Quanta Qualia

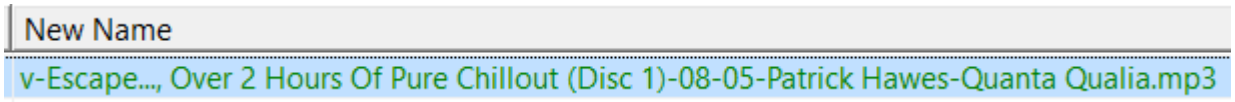
Parsed string:

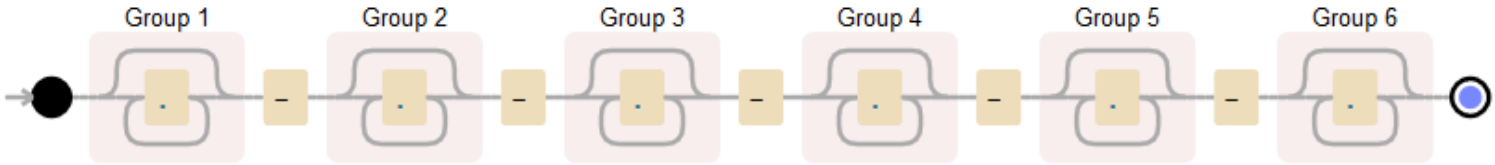
Match: (.*)-(.*)-(.*)-(.*)-(.*)-(.*)
Replace: \1-\2-\3-\4-\6-\5

Name

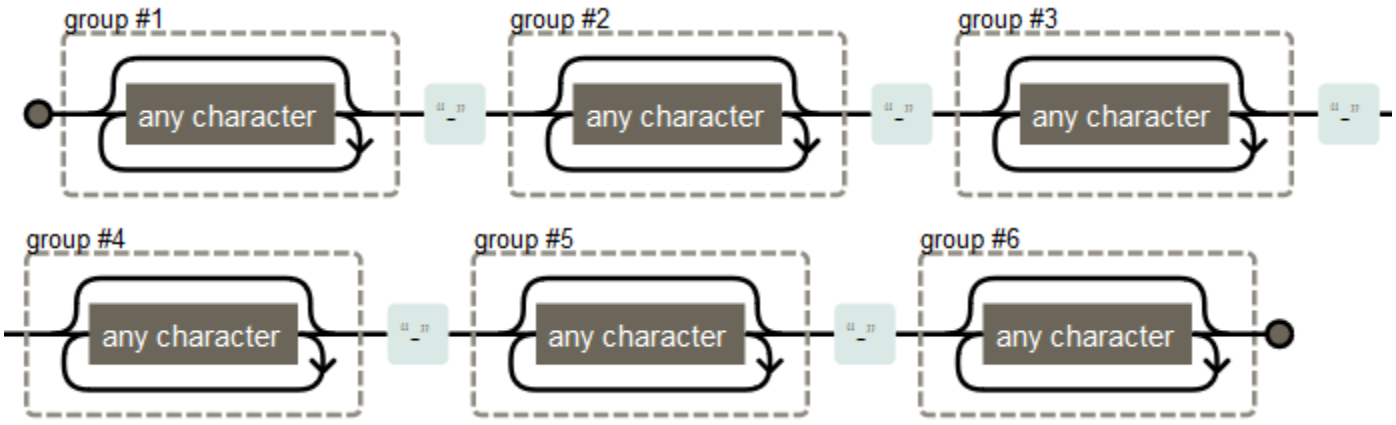


New Name





Expanded:



String Manipulation (Parsing)

Change around names at end of filename cont.

v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3
to
v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Patrick Hawes-Quanta Qualia.mp3

Match: (.*)-(.*)-(.*)-(.*)-(.*)-(.*)

Sample:

```
.....................................................................................
Match 1: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
Group 1: v
Group 2: Escape..., Over 2 Hours Of Pure Chillout (Disc 1)
Group 3: 08
Group 4: 05
Group 5: Quanta Qualia
Group 6: Patrick Hawes
```

Tim Notes:

1. Nice exercise in parsing but simplified - uses too many Capture Groups.
2. One of those, looks simple until you try to analyze it, examples I have spoken of previously. I have done enough of these to know that simple isn't always best – go ahead and do your own analysis. It is the only way to learn what works and what doesn't.

Forum Contributor: GMA

String Manipulation (Parsing)

Change around names at end of filename cont.

Using: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3

Match: (.*)-(.*)-(.*)-(.*)-(.*)-(.*)

Where:

\1 = v \2 = Escape..., Over 2 Hours Of Pure Chillout (Disc 1) \3 = 08 \4 = 05

\5 = Quanta Qualia \6 = Patrick Hayes

Tim Analysis:

(.*)	v	Capture Group 1
-		
(.*)	Escape..., Over 2 Hours Of Pure Chillout (Disc 1)	Capture Group 2
-		
(.*)	08	Capture Group 3
-		
(.*)	05	Capture Group 4
-		
(.*)	Quanta Qualia	Capture Group 5
-		
(.*)	Patrick Hawes	Capture Group 6

Looks simple enough – let's see.

String Manipulation (Parsing)

Change around names at end of filename cont.

Using: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3

Match: (.*)-(.*)-(.*)-(.*)-(.*)-(.*)

Where:

\1 = v \2 = Escape..., Over 2 Hours Of Pure Chillout (Disc 1) \3 = 08 \4 = 05

\5 = Quanta Qualia \6 = Patrick Hawes

Tim Analysis: cont.

1..	Match against any character	Capture Group 1 = v
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'Qualia'	Changes Capture Group 1= v-Escape..., Over 2 Hours Of Pure Chillout <space> (Disc 1)-08-05-Quanta Qualia
4..	Match against any character	Capture Group 2 = P
5. *	Make it Greedy Current position = EOL	Capture Group 2 = Patrick Hawes
6. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'Qualia' Changes step 3 to the <hyphen> after '05'	Changes Capture Group 2 = Quanta Qualia Changes Capture Group 1 = v-Escape..., Over 2 Hours Of Pure Chillout <space> (Disc 1)-08-05
7..	Match against any character	Capture Group 3 = P
8. *	Make it Greedy Current position = EOL	Capture Group 3 = Patrick Hawes
9. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'Qualia' Changes step 6 to the <hyphen> after '05' Changes step 3 to the <hyphen> after '08'	Changes Capture Group 3 = Quanta Qualia Changes Capture Group 2 = 05 Changes Capture Group 1 = v-Escape..., Over 2 Hours Of Pure Chillout <space> (Disc 1)-08
10..	Match against any character	Capture Group 4 = P
11. *	Make it Greedy Current position = EOL	Capture Group 4 = Patrick Hawes

Tim Notes:

1. Because of the sheer complexity of the analysis, I have decided to change the color coding I usually employ and retain the colors in the first part of the analysis on the previous page so hopefully these steps can be better understood.

String Manipulation (Parsing)

Change around names at end of filename cont.

Using: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3

Match: (.*)-(.*)-(.*)-(.*)-(.*)-(.*)

Where:

\1 = v \2 = Escape..., Over 2 Hours Of Pure Chillout (Disc 1) \3 = 08 \4 = 05

\5 = Quanta Qualia \6 = Patrick Hawes

Tim Analysis: cont.

12. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'Qualia' Changes step 9 to the <hyphen> after '05' Changes step 6 to the <hyphen> after '08' Changes step 3 to the <hyphen> after '(Disc 1)'	Changes Capture Group 4 = Quanta Qualia Changes Capture Group 3 = 05 Changes Capture Group 2 = 08 Changes Capture Group 1 = v-Escape..., Over 2 Hours Of Pure Chillout <space> (Disc 1)
13..	Match against any character	Capture Group 5 = P
14. *	Make it Greedy Current position = EOL	Capture Group 5 = Patrick Hawes
15. -	Match against <hyphen> (Not Captured) Backtracks. This is the <hyphen> after 'Qualia' Changes step 12 to the <hyphen> after '05' Changes step 9 to the <hyphen> after '08' Changes step 6 to the <hyphen> after '(Disc 1)' Changes step 3 to the <hyphen> after 'v'	Changes Capture Group 5 = Quanta Qualia Changes Capture Group 4 = 05 Changes Capture Group 3 = 08 Changes Capture Group 2 = Escape..., Over 2 Hours Of Pure Chillout <space> (Disc 1) Changes Capture Group 1 = v
16. .*	Match against remaining string	Capture Group 6 = Patrick Hawes

String Manipulation (Parsing)

Change around names at end of filename cont.

Using: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3

Match: (.*)-(.*)-(.*)-(.*)-(.*)-(.*)

Where:

\1 = v \2 = Escape..., Over 2 Hours Of Pure Chillout (Disc 1) \3 = 08 \4 = 05

\5 = Quanta Qualia \6 = Patrick Hawes

Tim Analysis: cont.

Sample:

```
.....................................................................................
Match 1: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
Group 1: v
Group 2: Escape..., Over 2 Hours Of Pure Chillout (Disc 1)
Group 3: 08
Group 4: 05
Group 5: Quanta Qualia
Group 6: Patrick Hawes
```

Tim Notes:

1. This seemingly simple solution took 2,215 steps!

2. Will create a lag time if processing a lot of files.

3. Simple solutions are not always the best – easy to code but hard to analyze and execute.

4. Anytime you use a wildcard with a Greedy Quantifier, you are going to have backtracking with constant changing values.

5. If you add multiple wildcard/Greedy combinations then you can see in the diagrams at the right the kind of mess this creates.

```
Step  ·····10·····20·····30·····40·····50·····60·····70·····80·····90·····
   ⊟Beginning match attempt at character 0
   1 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
   2 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
   3 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawesbacktrack
   4 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawe
   5 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawe
   6 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawebacktrack
   7 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Haw
   8 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Haw
   9 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawbacktrack
  10 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Ha
  11 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Ha
  12 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Habacktrack
  13 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick H
  14 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick H
  15 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hbacktrack
  16 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick
  17 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick
  18 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick backtrack
```

```
2189 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrickbacktrack
2190 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patric
2191 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patric
2192 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patricbacktrack
2193 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patri
2194 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patri
2195 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patribacktrack
2196 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patr
2197 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patr
2198 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrbacktrack
2199 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Pat
2200 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Pat
2201 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patbacktrack
2202 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Pa
2203 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Pa
2204 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Pabacktrack
2205 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-P
2206 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-P
2207 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Pbacktrack
2208 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-
2209 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-
2210 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-backtrack
2211 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia
2212 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia
2213 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-
2214 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
2215 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
     Match found in 2215 steps
```

String Manipulation (Parsing)

Change around names at end of filename cont.

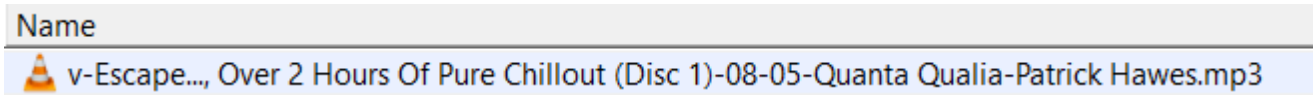
v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3
to
v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Patrick Hawes-Quanta Qualia.mp3

Better:

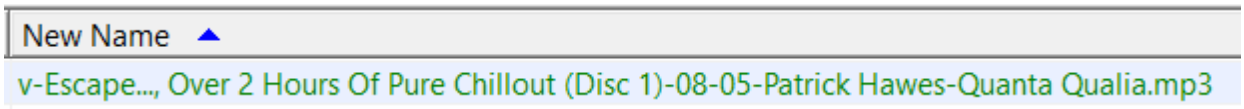
Only uses 3 Capture Groups with utilization of \w (Word Metacharacter) and \s (Whitespace Metacharacter)

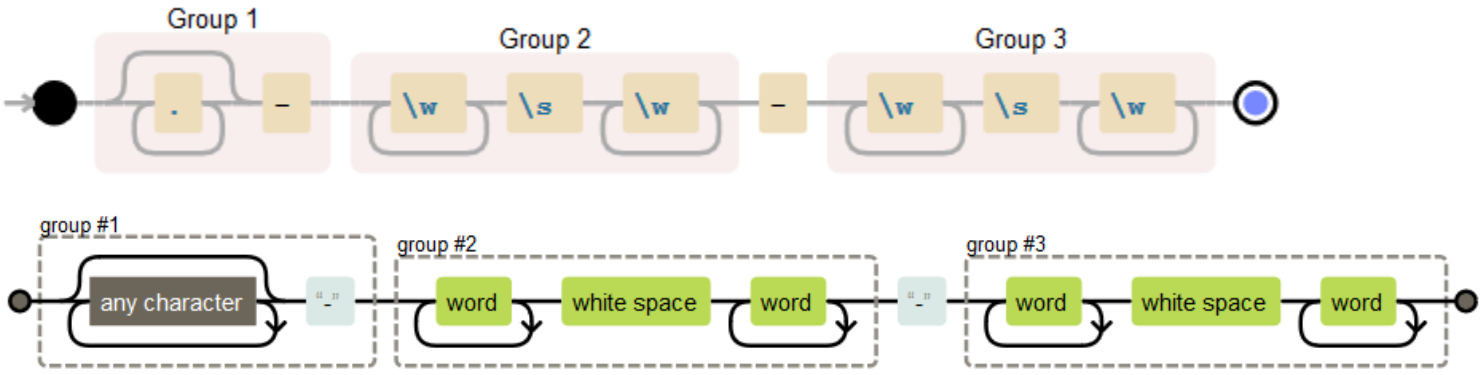
Match: (.*-)(\w+\s\w+)-(\w+\s\w+)
Replace: \1\3-\2

Name:



New Name:





Sample:

```
Match 1: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
Group 1: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-
Group 2: Quanta Qualia
Group 3: Patrick Hawes
```

Contributor Notes:

1. This uses the greediness of .* to capture everything, then give up only what it needs to in order to match the final (\w+\s\w+)-(\w+\s\w+)

Forum Contributor: Jane

String Manipulation (Parsing)

Change around names at end of filename cont.

Using: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3

Match: (.*-)(\w+\s\w+)-(\w+\s\w+)

Where:

\1 = v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05- \2 = Quanta Qualia \3 = Patrick Hawes

Tim Analysis:

1..	Match against any character	Capture Group 1 =
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. -	Match against <hyphen> followed by … This is the <hyphen> after 'Qualia'	Capture Group 1 = v-Escape..., Over 2 Hours Of Pure Chillout <space> (Disc 1)-08-05-Quanta Qualia-
4. \w	.. Word Metacharacter. Match against any single letter, number or underscore	Capture Group 2 = P
5. +	Make it Greedy	Capture Group 2 = Patrick
6. \s	Match against a <space> character followed by …	Capture Group 2 = Patrick <space>
7. \w	.. Word Metacharacter. Match against any single letter, number or underscore	Capture Group 2 = Patrick H
8. +	Make it Greedy Current position = EOL	Capture Group 2 = Patrick Hawes
9. -	Match against <hyphen> (Not Captured) followed by … This is the <hyphen> after 'Qualia'	Changes Capture Group 2 = Quanta Qualia
10. \w	.. Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = P
11. +	Make it Greedy	Capture Group 3 = Patrick
12. \s	Match against a <space> character followed by …	Capture Group 3 = Patrick <space>
13. \w	.. Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = Patrick H
14. +	Make it Greedy	Capture Group 3 = Patrick Hawes

String Manipulation (Parsing)

Change around names at end of filename cont.

Using: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes.mp3

Match: (.*-)(\w+\s\w+)-(\w+\s\w+)

Where:

\1 = v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05- \2 = Quanta Qualia \3 = Patrick Hawes

Tim Analysis: cont.

```
.....................................................................................
Match 1: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
Group 1: v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-
Group 2: Quanta Qualia
Group 3: Patrick Hawes
```

Tim Notes:

1. This solution took only 99 steps. A big difference from the 2,215 Steps of the last RegEx.

```
Beginning match attempt at character 0
 1 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawes
 2 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawesbacktrack
 3 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawe
 4 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawebacktrack
 5 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Haw
 6 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawbacktrack
 7 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Ha
 8 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Habacktrack
 9 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick H
10 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hbacktrack
11 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick
12 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick backtrack
13 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick
14 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrickbacktrack
15 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patric
16 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patricbacktrack

72 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta
73 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta backtrack
74 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta
75 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quantabacktrack
76 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quant
77 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quantbacktrack
78 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quan
79 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanbacktrack
80 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Qua
81 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quabacktrack
82 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Qu
83 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Qubacktrack
84 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Q
85 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Qbacktrack
86 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-
87 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-backtrack
88 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05
89 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-
90 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-
91 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta
92 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta
93 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia
94 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia
95 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-
96 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick
97 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick
98 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawe
99 v-Escape..., Over 2 Hours Of Pure Chillout (Disc 1)-08-05-Quanta Qualia-Patrick Hawe
Match found in 99 steps
```

String Manipulation (Parsing)

Simple Rearrange #4

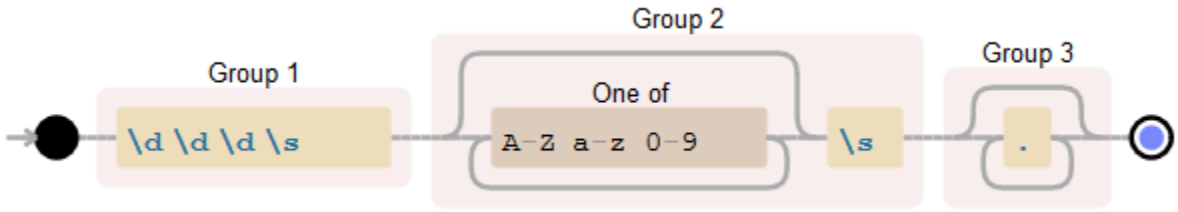
001 G61138 ElPaisanoHotel,MarfaTX20051221.jpg	to	001 ElPaisanoHotel,MarfaTX20051221 G61138.jpg
002 SD0337 PresiCourthouse,MarfaTX20051221.jpg	to	002 PresiCourthouse,MarfaTX20051221 SD0337.jpg
003 G14338 OldHouse,MarfaTX20051221.jpg	to	003 OldHouse,MarfaTX20051221 G14338.jpg

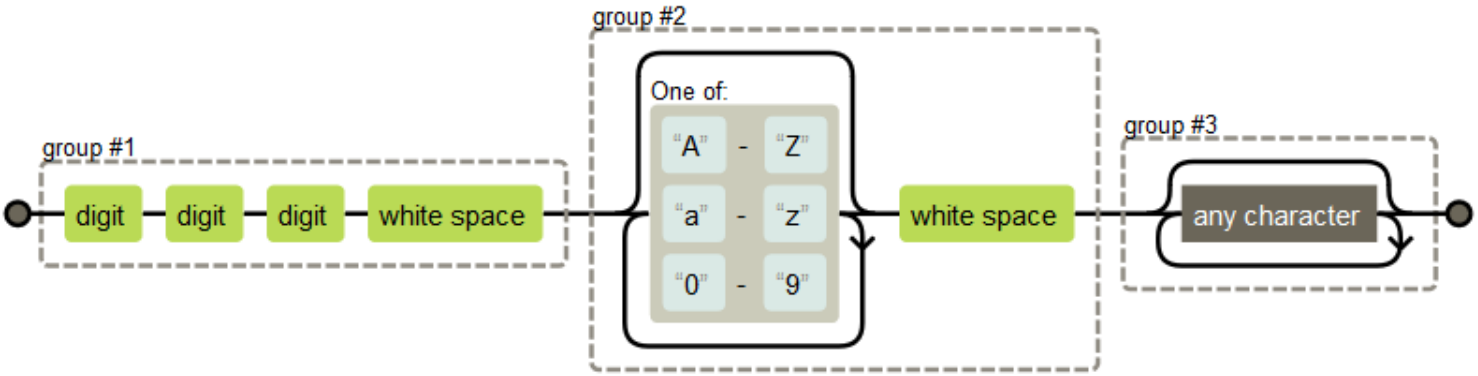
Observations:

1. Move the second group of alphanumeric characters (i.e., G61138, SD0337, G14338, etc.) to the end of the filename.

Match: (\d\d\d\s)([A-Za-z0-9]*\s)(.*)
Replace: \1\3 \2

Name	New Name
003 G14338 OldHouse,MarfaTX20051221.jpg	003 OldHouse,MarfaTX20051221 G14338 .jpg
002 SD0337 PresiCourthouse,MarfaTX20051221.jpg	002 PresiCourthouse,MarfaTX20051221 SD0337 .jpg
001 G61138 ElPaisanoHotel,MarfaTX20051221.jpg	001 ElPaisanoHotel,MarfaTX20051221 G61138 .jpg





Sample:

```
.....................................................
Match 1: 001 G61138 ElPaisanoHotel,MarfaTX20051221
Group 1: 001
Group 2: G61138
Group 3: ElPaisanoHotel,MarfaTX20051221
```

Forum Contributor: jimwillsher (Original Publisher BRU)

String Manipulation (Parsing)

Simple Rearrange #4 cont.

Using: 003 G14338 OldHouse,MarfaTX20051221.jpg

Match: (\d\d\d\s)([A-Za-z0-9]*\s)(.*)

Where:

\1 = 003 <space> \2 = G14338<space> \3 = OldHouse,MarfaTX20051221

Tim Analysis:

1. \d\d\d	Match against 3 numeric digits followed by …	Capture Group 1 = 003
2. <space>	.. Match against a <space> character	Capture Group 1 = 003<space>
3. [A-Za-z0-9]	Match against a class consisting of an uppercase letter, a lowercase letter or a numeric digit	Capture Group 2 = G
4. *	Make it Greedy	Capture Group 2 = G14338
5. \s	Match against <space> character	Capture Group 2 = G14338 <space>
6. .*	Match against remaining string	Capture Group 3 = OldHouse,MarfaTX20051221

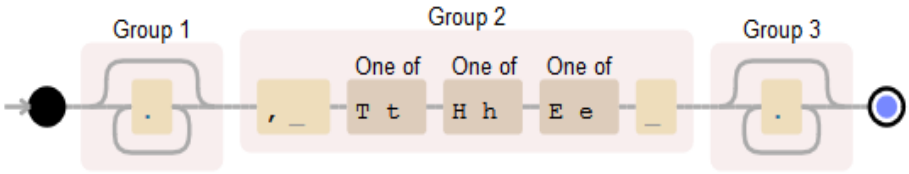
String Manipulation (Parsing)

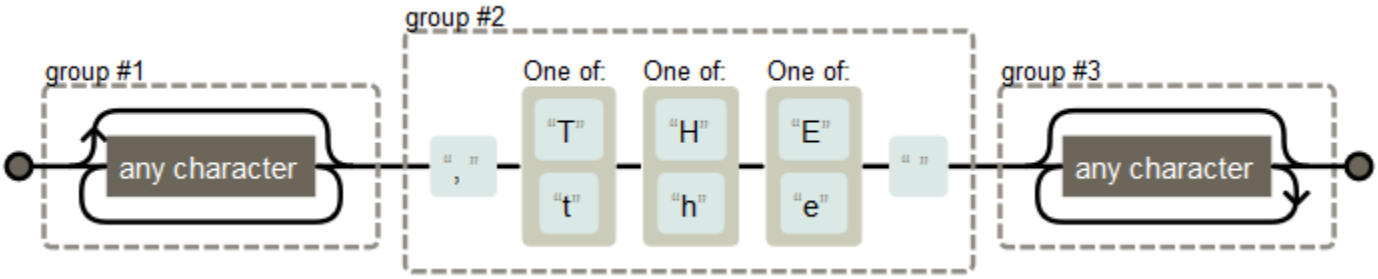
Rearrange word following comma to beginning of string

movie, the [1995] to the movie [1995]

Match: (.*?)(, [Tt][Hh][Ee])(.*)
Replace: The \1 \3

Name	New Name
movie, the [1995]	The movie [1995]





Sample:

```
.............................
Match 1: movie, the [1995]
Group 1: movie
Group 2: , the
Group 3: [1995]
```

Contributor Notes:

1. For finding files, there is nothing better than the 'Everything' search program from VoidTools.com:

", the "

And in particular this program is especially useful if the files are strewn about in different directories.

Forum Contributor: therube

String Manipulation (Parsing)

Rearrange word following comma to beginning of string cont.

Using: movie, the [1995]

Match: (.*?)(, [Tt][Hh][Ee])(.*)

Where:

\1 = movie \2 = , the \3 = [1995]

Tim Analysis:

1. .* 2. ,	Capture until… … the comma followed by …	Capture Group 1= <entire string> Capture Group 2 = <comma>
3. <space>	.. a <space> followed by …	Capture Group 2= <comma> <space> Changes Capture Group 1 = Movie
3. [Tt] [Hh] [E e]	.. an uppercase or lowercase 'The' followed by a <space>	Capture Group 2 = <comma> the <space>
4. .*	Capture the remainder of the string	Capture Group 3 = [1995]
5.	The Replace String prefixes with, 'The', then everything else, excluding '<comma><space>the<space>' in Capture Group 2	

String Manipulation (Parsing)

Example of parsing a group of numbers with no delimiters

Tim Notes:

1. This is a good demonstration of how a group of numbers with no delimiters (<spaces>, <hyphens> etc.) is parsed.

867220140415210328.jpg	to	041514 210328.jpg
866820140415202327.jpg	to	041514 202327.jpg
50320140502211456.jpg	to	050214 211456.jpg

Break it down:

867220	14	04	15	21	03	28.jpg
866820	14	04	15	20	23	27.jpg
50320	14	05	02	21	14	56.jpg

Further:

867220	14 = YY	04 = MM	15 = DD	21 = HH	03 = MM	28 = SS
Random Garbage	Year	Month	Day	Hours	Minutes	Seconds

Observations:

1. Remove extraneous prefix
2. Convert YMDHMS to MDYHMS
3. The YMDHMS consists of 12 numeric digits
4. The extraneous prefix are random numbers and have no fixed length
5. HHMMSS are a fixed length of 6 numeric digits and remain as is during the conversion.
6. Year gets moved, Month and Day remain together.

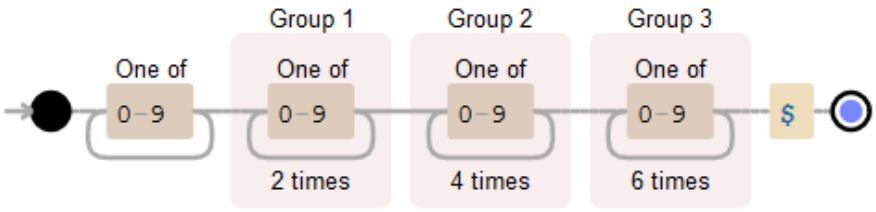
String Manipulation (Parsing)

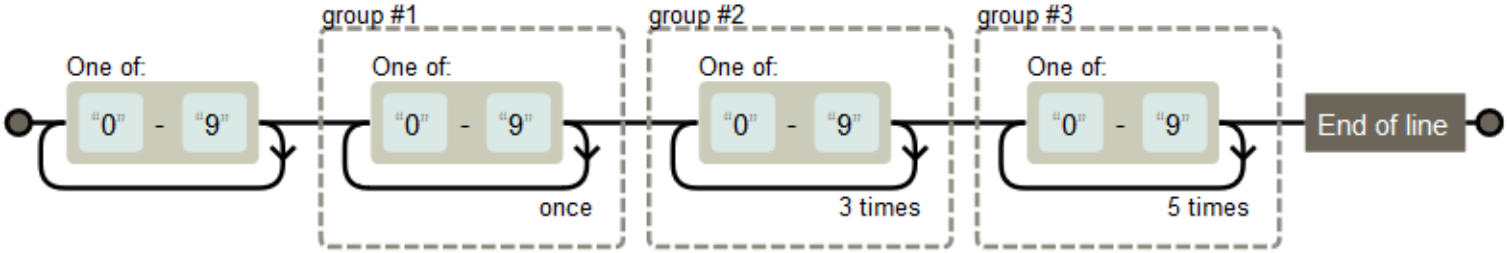
Example of parsing a group of numbers with no delimiters cont.

867220140415210328.jpg	to	041514 210328.jpg
866820140415202327.jpg	to	041514 202327.jpg
50320140502211456.jpg	to	050214 211456.jpg

Match: [0-9]+([0-9]{2})([0-9]{4})([0-9]{6})$
Replace: \2\1\3

	New Name
867220140415210328.jpg	041514210328.jpg
866820140415202327.jpg	041514202327.jpg
50320140502211456.jpg	050214211456.jpg





Sample:

```
.............................
Match 1: 867220140415210328
Group 1: 14
Group 2: 0415
Group 3: 210328
```

Under Regex Buddy for this test, had to remove anchor $ because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.jpg' is just another part of the string of text.

Forum Contributor: truth

String Manipulation (Parsing)

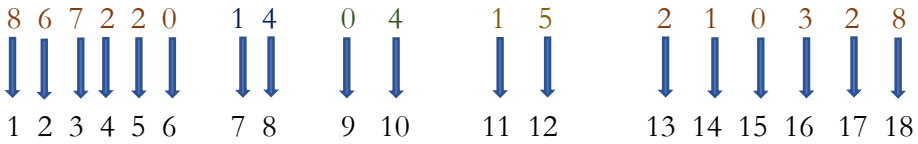
Example of parsing a group of numbers with no delimiters cont.

Using sample, 867220140415210328.jpg

Where:

\1 = 14 \2 = 0415 \3 = 210328

Tim Analysis: Match: [0-9]+([0-9]{2})([0-9]{4})([0-9]{6})$



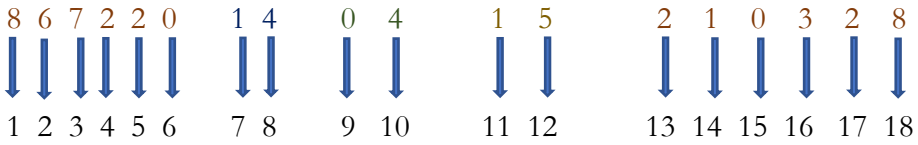
1. [0-9]	Match against a class consisting of a numeric digit (Not Captured) = '8' RegEx engine moving forward. The '8' is the first numeric digit of the string. Current position = 1.	
2. +	Make it Greedy (Not Captured) = entire string to EOL (End of Line) Current position = EOL	
3. [0-9]	Match against a class consisting of a numeric digit Changes previous value Not Captured to '86722014041521032' RegEx engine is backtracking. The '8' is the second numeric digit of 'SS' Current position = 18.	Capture Group 1 = 8
4. {2}	Limit the matching of the previous expression to two repetitions. What this does is capture the two numeric digits in sequence since the previous expression was \d. This is the same as \d\d Changes previous value Not Captured to '8672201404152103' RegEx engine still backtracking. The '2' is the first numeric digit of 'SS'. Current position = 17.	Capture Group 1 = 28
5. [0-9]	Match against a class consisting of a numeric digit Changes previous value Not Captured to '867220140415210' RegEx engine is moving forward. The '8' is the second numeric digit of 'SS'. Current position = 18.	Capture Group 2 = 8 Changes Capture Group 1 = 32
6. {4}	Capture the next group of 4 numeric digits (see step 4 for how this works). Because the evaluation could not be satisfied moving forward, the RegEx engine backtracks to capture the last 4 numeric digits of the string. Current position = 15. Changes previous value Not Captured to '867220140415'	Capture Group 2 = 0328 Changes Capture Group 1 = 21
7. [0-9]	Match against a class consisting of a numeric digit Changes previous value Not Captured to '86722014041' RegEx engine is moving forward. The '8' is the second numeric digit of 'SS'. Current position = 18.	Capture Group 3 = 8 Changes Capture Group 2 = 1032 Changes Capture Group 1 = 52
8. {6}	Capture the next group of 6 numeric digits (see step 4 for how this works). Because the evaluation could not be satisfied moving forward, the RegEx engine backtracks to capture the last 6 numeric digits of the string. Current position = 13. Changes previous value Not Captured to '867220'	Capture Group 3 = 210328 Changes Capture Group 2 = 0415 Changes Capture Group 1 = 14
9. $	Last expression must have matched at EOL	

String Manipulation (Parsing)

Example of parsing a group of numbers with no delimiters cont.

Using sample, 867220140415210328.jpg

Tim Analysis: cont. Match: [0-9]+([0-9]{2})([0-9]{4})([0-9]{6})$



Where:

\1 = 14 \2 = 0415 \3 = 210328

Tim Notes:

Throughout this book I have used in my analysis, where $ = Must be at EOL or something similar. EOL is the position after the last character of the string. It means End of Line or more appropriately here, End of Filename or Foldername, but to clarify $ also mean this:

e.g.

You have a string:

Now is the Time for All Good Men to Come to the Aid of Their Country

This will Match ^(Now.*Country)$ but in BRU, not very useful because it will match against the entire string

Something like this would be more useful:

Match: ^(.*)(Aid.*Country)$
Replace: \2 = Aid of their Country

or this:

Match: ^(.*)(\W)(Country)$
Replace: \3 = Country

The ^ indicates the string should start with 'Aid' and the $ indicates the string must end with 'Country' and there can be zero or more characters (any characters) in between using the .* after 'Aid'.

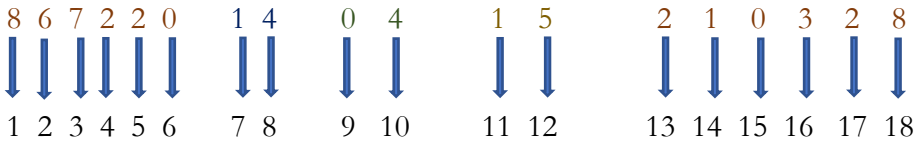
And, as used in step 9, it also indicates that the evaluation is finished – there are no more matches and RegEx engine moves forward to EOL. I have used the analogy of the RegEx engine like a train on a track. Think of the ^ as the train pulling out of the station, and the $ as the train pulling into the station.

String Manipulation (Parsing)

Example of parsing a group of numbers with no delimiters cont.

Using sample, 867220140415210328.jpg

Tim Analysis: cont. Match: [0-9]+([0-9]{2})([0-9]{4})([0-9]{6})$



Where:

\1 = 14 \2 = 0415 \3 = 210328

Tim Notes:

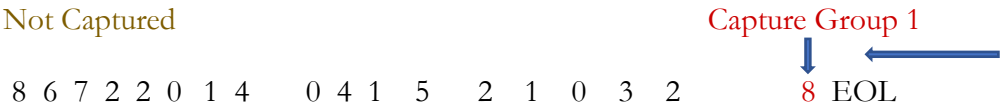
1. Changes to previous values occur because of forced reevaluations based on the RegEx engine's movement, typically as a result of backtracking. In some cases of a RegEx not matching at a certain point of the evaluation, the current Capture Group may give up the match (value is removed) forcing the reevaluation of other matches held by previous Capture Groups. A Range Quantifier or Lazy Quantifier may also be a cause.

Visualize:

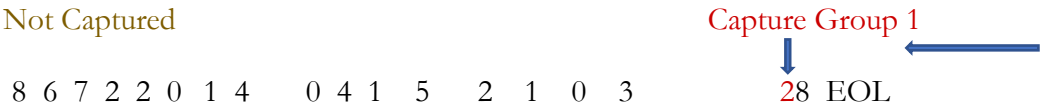
Step 1. Match against a class consisting of a numeric digit [0-9]

Step 2. Make it Greedy +

Step 3. Match against a class consisting of a numeric digit [0-9]



Step 4. Limit the matching of the previous expression to two repetitions {2}

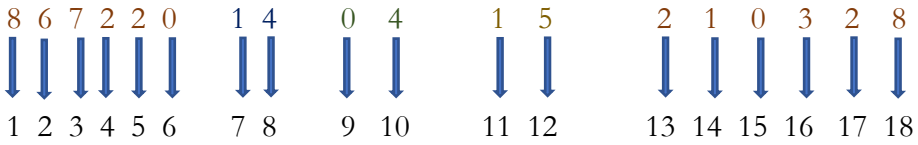


String Manipulation (Parsing)

Example of parsing a group of numbers with no delimiters cont.

Using sample, 867220140415210328.jpg

Tim Analysis: cont. Match: [0-9]+([0-9]{2})([0-9]{4})([0-9]{6})$

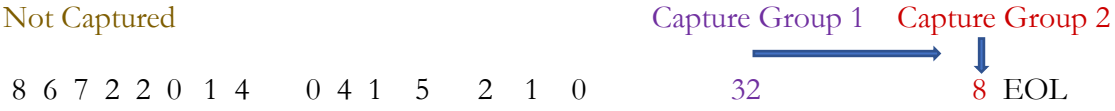


Where:

\1 = 14 \2 = 0415 \3 = 210328

Visualize:

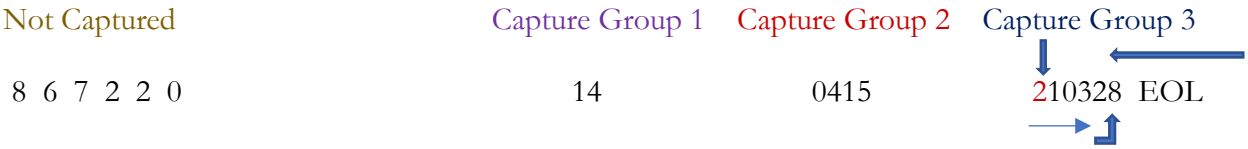
Step 5. Match against a class consisting of a numeric digit [0-9]



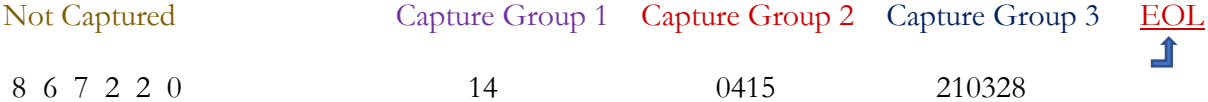
Step 6. Capture the next group of 4 numeric digits. Current position = last '8' {4}

Step 7. Match against a class consisting of a numeric digit. Reaches EOL. Backtracks to match the '8' [0-9]

Step 8. Capture the next group of 6 numeric digits. Current position = last '8' {6}



Step 9. $. Must be at EOL $



String Manipulation (Parsing)

Switch portions of filename strings

20071013_002_Come Away With Me_Norah Jones.mp3

to

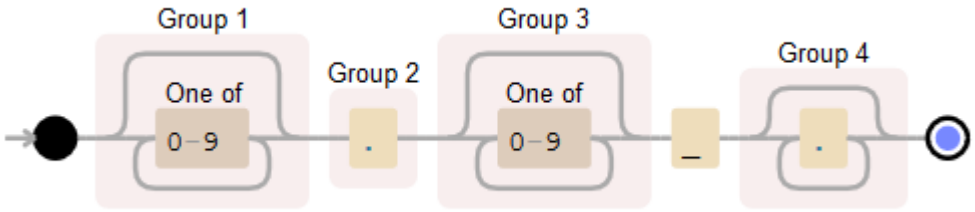
Come Away With Me_Norah Jones_20071013_002_.mp3

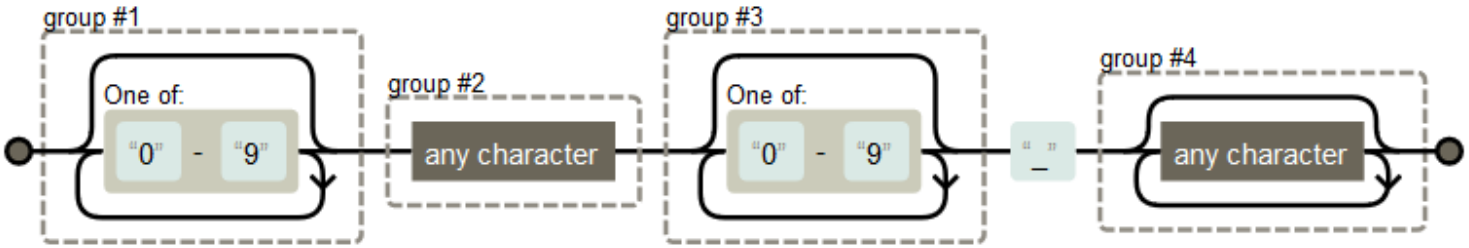
Observations:

1. Move the front of the file name with varying length to the end of the filename.

Match: ([0-9]*)(.)([0-9]*)_(.*)
Replace: \4_\1\2\3

Name	New Name
20071013_002_Come Away With Me_Norah Jones.mp3	Come Away With Me_Norah Jones_20071013_002.mp3





Sample:

```
.....................................................
Match 1: 20071013_002_Come Away With Me_Norah Jones
Group 1: 20071013
Group 2: _
Group 3: 002
Group 4: Come Away With Me_Norah Jones
```

Forum Contributor: GMA

String Manipulation (Parsing)

Switch portions of filename strings cont.

Using: 20071013_002_Come Away With Me_Norah Jones.mp3

Match: ([0-9]*)(.)([0-9]*)_(.*)

Where:

\1 = 20071013 \2 = _ \3 = 002 \4 = Come Away With Me_Norah Jones

Tim Analysis:

1. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1 = 2007103
3. .	Match against any character This is the underscore after '13'	Capture Group 2 = _
4. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 3 = 0
5. *	Make it Greedy	Capture Group 3 = 002
6. _	Match against underscore (Not Captured) This is the underscore after '002'	
7. .*	Match against remaining string	Capture Group 4 = Come Away With Me_Norah Jones

String Manipulation (Parsing)

Insertion or Deletion from End of Filename

Sample: 123456789abcdefghij

Character Position from End:

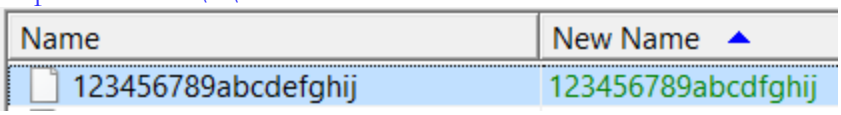
													6	5	4	3	2	1

Position:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2	3	4	5	6	7	8	9	a	b	c	d	**e**	**f**	**g**	h	i	j

To delete the sixth character from the end:

Match: (.*)(.{1})(.{5})$
Replace: \1\3



Isolates the 'e'

To delete the fifth character from the end:

Match: (.*)(.{1})(.{4})$
Replace: \1\3

Isolates the 'f'

To delete the fourth character from the end:

Match: (.*)(.{1})(.{3})$
Replace: \1\3

Isolates the 'g'

To delete the fifth and fourth characters from the end:

Match: (.*)(.{2})(.{3})$
Replace: \1\3

Isolates the 'f' and 'g'

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Sample: 123456789abcdefghij

Character Position from End:

												6	5	4	3	2	1	

Position:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2	3	4	5	6	7	8	9	a	b	c	d	**e**	**f**	**g**	h	i	j

To delete the sixth, fifth and fourth characters from the end:

Match: (.*)(.{3})(.{3})$
Replace: \1\3

Name	New Name
123456789abcdefghij	123456789abcdhij

Isolates the ‘e’, ‘f’ and ‘g’

To insert something at position 4 from the end:

Match: (.*)(.{3})$
Replace: \1 Insert text here \2

Name	New Name
123456789abcdefghij	123456789abcdefg insert text here hij

To insert something at position 5 from the end:

Match: (.*)(.{4})$
Replace: \1 Insert text here \2

Name	New Name
123456789abcdefghij	123456789abcdef insert text here ghij

In the last two examples, of course, you have to replace "insert text here" with whatever it is you need to insert.

Forum Contributor: GMA

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Using: 123456789abcdefghij

To delete the fourth character from the end: cont.

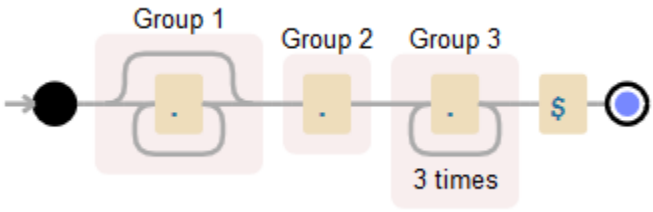
Match: (.*)(.{1})(.{3})$

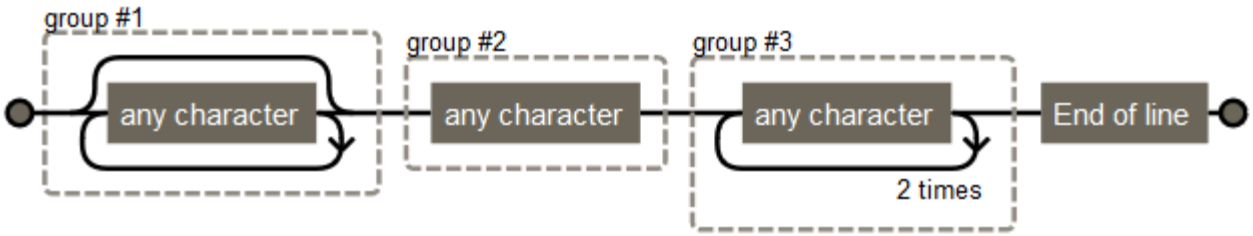
Sample:

```
Match 1: 123456789abcdefghij
Group 1: 123456789abcdef
Group 2: g
Group 3: hij
```

Where:

\1 = 123456789abcdef \2 = g]3 = hij





Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. .	Match against any character	Capture Group 2 = j Changes Capture Group 1 = 123456789abcdefghi
4. {1}	Repeat last expression to match 1 iteration.	Capture Group 2 = Unchanged
5. .	Match against any character	Capture Group 3 = j Changes Capture Group 2 = i Changes Capture Group 1 = 123456789abcdefgh
6. {3}	Repeat last expression to match 3 iterations.	Capture Group 3 = hij Changes Capture Group 2 = g Changes Capture Group 1 = 123456789abcdef
7. $	Must be at EOL (End of Line)	

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Using: 123456789abcdefghij

To delete the fifth and fourth characters from the end: cont.

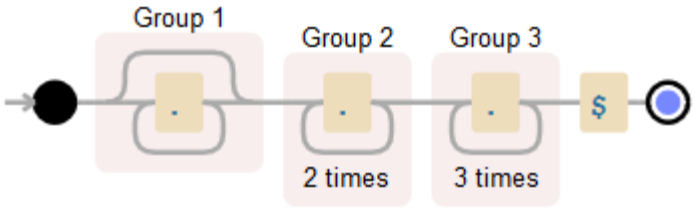
Match: (.*)(.{2})(.{3})$

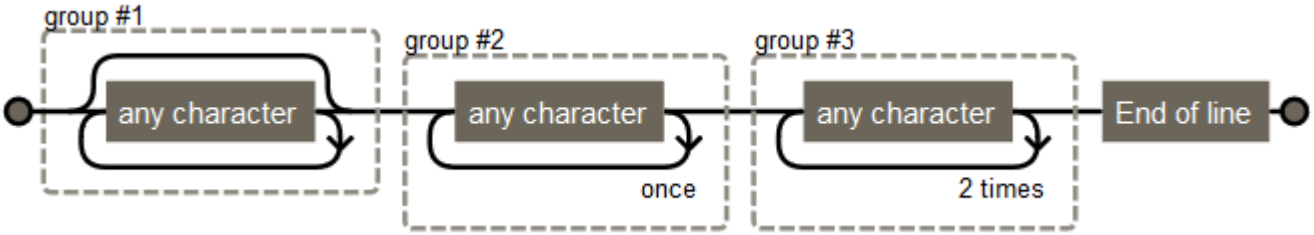
Sample:

```
..............................
Match 1: 123456789abcdefghij
Group 1: 123456789abcde
Group 2: fg
Group 3: hij
```

Where:

\1 = 123456789abcde \2 = fg \3 = hij





Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. .	Match against any character	Capture Group 2 = j Changes Capture Group 1 = 123456789abcdefghi
4. {2}	Repeat last expression to match 2 iterations.	Capture Group 2 = ij
5. .	Match against any character	Capture Group 3 = j Changes Capture Group 2 = hi Changes Capture Group 1 = 123456789abcdefg
6. {3}	Repeat last expression to match 3 iterations.	Capture Group 3 = hij Changes Capture Group 2 = fg Changes Capture Group 1 = 123456789abcde
7. $	Must be at EOL (End of Line)	

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Using: 123456789abcdefghij

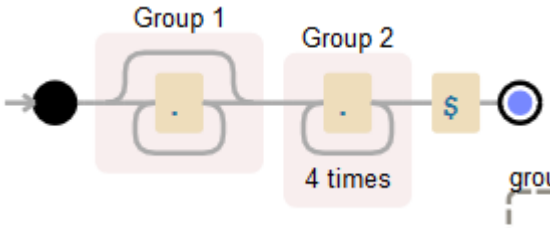
To insert something at position 5 from the end: cont.

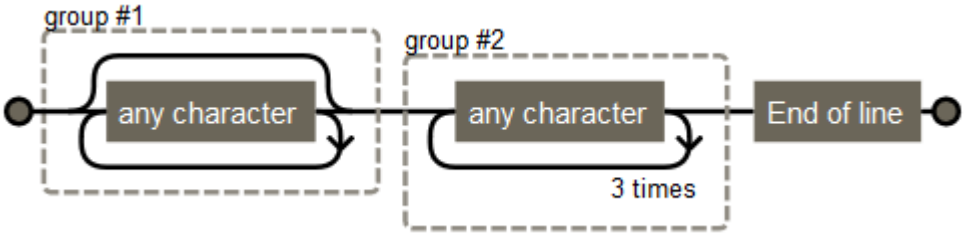
Match: (.*)(.{4})$

Where:

\1 = 123456789abcdef \2 = ghij

```
Match 1: 123456789abcdefghij
Group 1: 123456789abcdef
Group 2: ghij
```





Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. .	Match against any character	Capture Group 2 = j Changes Capture Group 1 = 123456789abcdefghi
4. {4}	Repeat last expression to match 4 iterations.	Capture Group 2 = ghij Changes Capture Group 1 = 123456789abcdef
5. $	Must be at EOL (End of Line)	

Tim Notes:

1. Performed analysis on just these samples, because the remainder of the RegEx not analyzed are just minor variations of these.

2. To add or change something at the end of the string, you start at the end and backtrack – using

(.*) <rest of expression>

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Sample: 123456789abcdefghij

Character Position from Beginning:

Position:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2	3	**4**	**5**	**6**	7	8	9	a	b	c	d	e	f	g	h	i	j

Tim Contribution:

To delete the sixth character from the beginning:

Match: ^(.{5})(.{1})(.*)$
Replace: \1\3

Isolates the '6'

To delete the fifth character from the beginning:

Match: ^(.{4})(.{1})(.*)$
Replace: \1\3

Isolates the '5'

To delete the fourth character from the beginning:

Match: ^(.{3})(.{1})(.*)$
Replace: \1\3

Isolates the '4'

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Sample: 123456789abcdefghij

Character Position from Beginning:

Position:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	2	3	**4**	**5**	**6**	7	8	9	a	b	c	d	e	f	g	h	i	j

Tim Contribution: cont.

To delete the fourth and fifth characters from the beginning:

Match: ^(.{3})(.{2})(.*)$
Replace: 1\3

Isolates the 4' and '5'

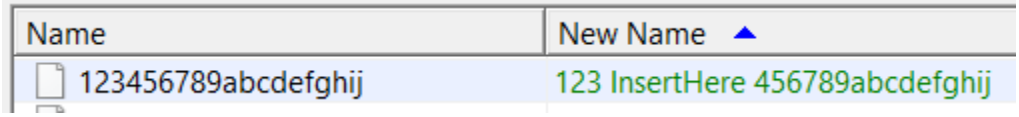
To delete the fourth, fifth and sixth, characters from the beginning:

Match: ^(.{3})(.{3})(.*)$
Replace: \1\3

Isolates the '4', '5' and '6'

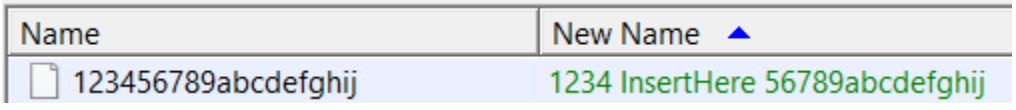
To insert something at position 4 from the beginning:

Match: ^(.{3})(.*)$
Replace: \1 InsertHere \2



To insert something at position 5 from the beginning:

Match: ^(.{4})(.*)$
Replace: \1 InsertHere \2



String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Using: 123456789abcdefghij

Tim Contribution: cont.

To delete the sixth character from the beginning: cont.

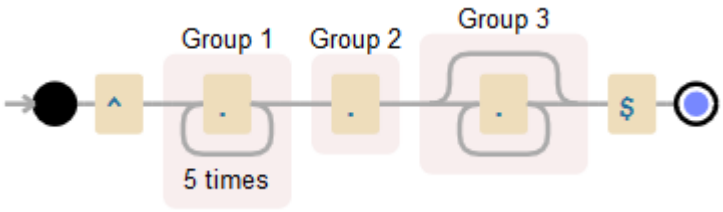
Match: ^(.{5})(.{1})(.*)$

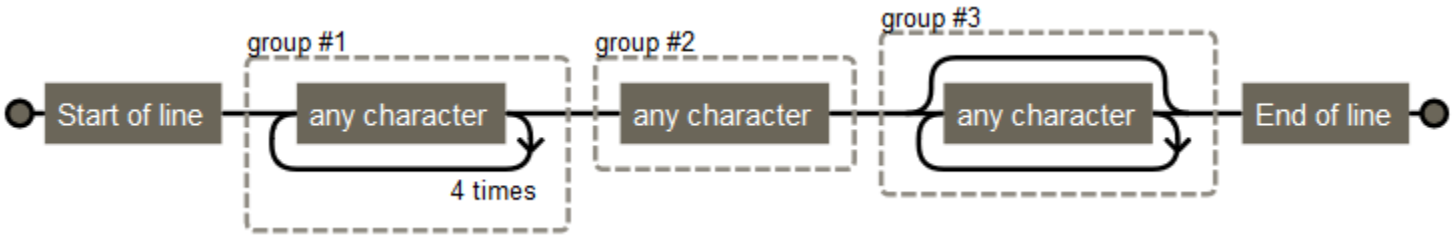
Where:

\1 = 12345 \2 = 6 \3 = 789abcdefghij

Sample:

```
Match 1: 123456789abcdefghij
Group 1: 12345
Group 2: 6
Group 3: 789abcdefghij
```





Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = 1
3. {5}	Repeat last expression to match 5 iterations.	Capture Group 1 = 12345
4. .	Match against any character	Capture Group 2 = 6
5. {1}	Repeat last expression to match 1 iteration.	All capture Groups = Unchanged
6. .*	Match against remaining string	Capture Group 3 = 789abcdefghij
7. $	Must be at EOL (End of Line)	

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Using: 123456789abcdefghij

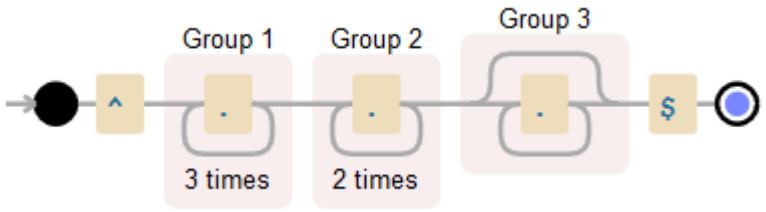
Tim Contribution: cont.

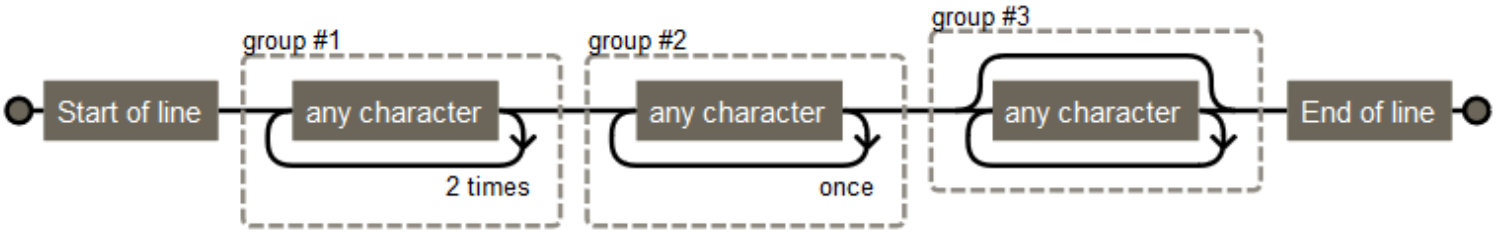
To delete the fourth and fifth characters from the beginning: cont.

Match: ^(.{3})(.{2})(.*)$

Sample:

```
.............................
Match 1: 123456789abcdefghij
Group 1: 123
Group 2: 45
Group 3: 6789abcdefghij
```





Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = 1
3. {3}	Repeat last expression to match 3 iteration.	Capture Group 1 = 123
4. .	Match against any character	Capture Group 2 = 4
5. {2}	Repeat last expression to match 2 iterations.	Capture Group 4 = 45
6. .*	Match against remaining string	Capture Group 3 = 6789abcdefghij
7. $	Must be at EOL (End of Line)	

String Manipulation (Parsing)

Insertion or Deletion from End of Filename cont.

Using: 123456789abcdefghij

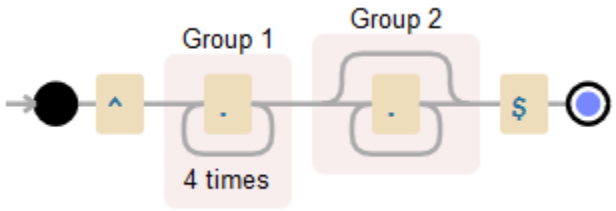
Tim Contribution: cont.

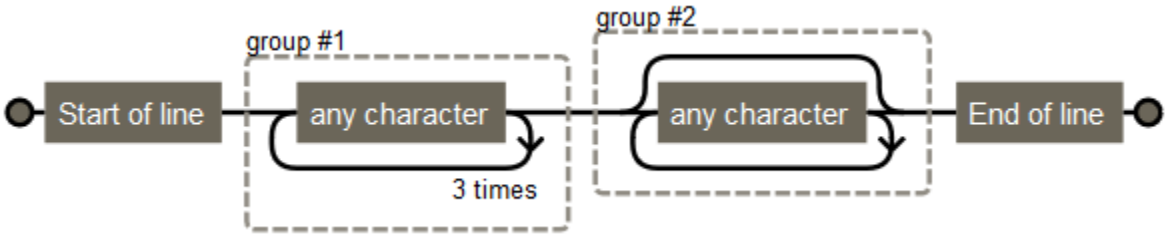
To insert something at position 5 from the beginning: cont.

Match: ^(.{4})(.*)$

Sample:

```
Match 1: 123456789abcdefghij
Group 1: 1234
Group 2: 56789abcdefghij
```





Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = 1
3. {4}	Repeat last expression to match 4 iterations.	Capture Group 1 = 1234
4. .*	Match against remaining string	Capture Group 3 = 56789abcdefghij
7. $	Must be at EOL (End of Line)	

Tim Notes:

1. Performed analysis on just these samples, because the remainder of the RegEx not analyzed are just minor variations of these.

2. To add or change something at the beginning of the string, you start at the beginning and work toward the end using –

<expression> (.*)

String Manipulation (Parsing)

More Run on Sentences

Sample Base:

CoffeeAndTea
ThisIsReallyAnnoying&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.whatever
CadAbeBad

Observations:

1. Each word is Capitalized

Method 1:

1. 'Section #14: Character Translation' –

A,a= ,A,a	B,a= ,B,a	C,a= ,C,a	D,a= ,D,a	E,a= ,E,a
A,b= ,A,b	B,b= ,B,b	C,b= ,C,b	D,b= ,D,b	E,b= ,E,b
A,c= ,A,c	B,c= ,B,c	C,c= ,C,c	D,c= ,D,c	E,c= ,E,c
A,d= ,A,d	B,d= ,B,d	C,d= ,C,d	D,d= ,D,d	E,d= ,E,d
A,e= ,A,e	B,e= ,B,e	C,e= ,C,e	D,e= ,D,e	E,e= ,E,e
A,f= ,A,f	B,f= ,B,f	C,f= ,C,f	D,f= ,D,f	E,f= ,E,f
A,g= ,A,g	B,g= ,B,g	C,g= ,C,g	D,g= ,D,g	E,g= ,E,g
A,h= ,A,h	B,h= ,B,h	C,h= ,C,h	D,h= ,D,h	E,h= ,E,h
A,i= ,A,i	B,i= ,B,i	C,i= ,C,i	D,i= ,D,i	E,i= ,E,i
A,j= ,A,j	B,j= ,B,j	C,j= ,C,j	D,j= ,D,j	E,j= ,E,j
A,k= ,A,k	B,k= ,B,k	C,k= ,C,k	D,k= ,D,k	E,k= ,E,k
A,l= ,A,l	B,l= ,B,l	C,l= ,C,l	D,l= ,D,l	E,l= ,E,l
A,m= ,A,m	B,m= ,B,m	C,m= ,C,m	D,m= ,D,m	E,m= ,E,m
A,n= ,A,n	B,n= ,B,n	C,n= ,C,n	D,n= ,D,n	E,n= ,E,n
A,o= ,A,o	B,o= ,B,o	C,o= ,C,o	D,o= ,D,o	E,o= ,E,o
A,p= ,A,p	B,p= ,B,p	C,p= ,C,p	D,p= ,D,p	E,p= ,E,p
A,q= ,A,q	B,q= ,B,q	C,q= ,C,q	D,q= ,D,q	E,q= ,E,q
A,r= ,A,r	B,r= ,B,r	C,r= ,C,r	D,r= ,D,r	E,r= ,E,r
A,s= ,A,s	B,s= ,B,s	C,s= ,C,s	D,s= ,D,s	E,s= ,E,s
A,t= ,A,t	B,t= ,B,t	C,t= ,C,t	D,t= ,D,t	E,t= ,E,t
A,u= ,A,u	B,u= ,B,u	C,u= ,C,u	D,u= ,D,u	E,u= ,E,u
A,v= ,A,v	B,v= ,B,v	C,v= ,C,v	D,v= ,D,v	E,v= ,E,v
A,w= ,A,w	B,w= ,B,w	C,w= ,C,w	D,w= ,D,w	E,w= ,E,w
A,x= ,A,x	B,x= ,B,x	C,x= ,C,x	D,x= ,D,x	E,x= ,E,x
A,y= ,A,y	B,y= ,B,y	C,y= ,C,y	D,y= ,D,y	E,y= ,E,y
A,z= ,A,z	B,z= ,B,z	C,z= ,C,z	D,z= ,D,z	E,z= ,E,z

String Manipulation (Parsing)

More Run on Sentences cont.

Sample Base:

CoffeeAndTea
ThisIsReallyAnnoying&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.whatever
CadAbeBad

Method 1: cont.

1. 'Section #14: Character Translation' –

F,a= ,F,a	G,a= ,G,a	H,a= ,H,a	I,a= ,I,a	J,a= ,J,a
F,b= ,F,b	G,b= ,G,b	H,b= ,H,b	I,b= ,I,b	J,b= ,J,b
F,c= ,F,c	G,c= ,G,c	H,c= ,H,c	I,c= ,I,c	J,c= ,J,c
F,d= ,F,d	G,d= ,G,d	H,d= ,H,d	I,d= ,I,d	J,d= ,J,d
F,e= ,F,e	G,e= ,G,e	H,e= ,H,e	I,e= ,I,e	J,e= ,J,e
F,f= ,F,f	G,f= ,G,f	H,f= ,H,f	I,f= ,I,f	J,f= ,J,f
F,g= ,F,g	G,g= ,G,g	H,g= ,H,g	I,g= ,I,g	J,g= ,J,g
F,h= ,F,h	G,h= ,G,h	H,h= ,H,h	I,h= ,I,h	J,h= ,J,h
F,i= ,F,i	G,i= ,G,i	H,i= ,H,i	I,i= ,I,i	J,i= ,J,i
F,j= ,F,j	G,j= ,G,j	H,j= ,H,j	I,j= ,I,j	J,j= ,J,j
F,k= ,F,k	G,k= ,G,k	H,k= ,H,k	I,k= ,I,k	J,k= ,J,k
F,l= ,F,l	G,l= ,G,l	H,l= ,H,l	I,l= ,I,l	J,l= ,J,l
F,m= ,F,m	G,m= ,G,m	H,m= ,H,m	I,m= ,I,m	J,m= ,J,m
F,n= ,F,n	G,n= ,G,n	H,n= ,H,n	I,n= ,I,n	J,n= ,J,n
F,o= ,F,o	G,o= ,G,o	H,o= ,H,o	I,o= ,I,o	J,o= ,J,o
F,p= ,F,p	G,p= ,G,p	H,p= ,H,p	I,p= ,I,p	J,p= ,J,p
F,q= ,F,q	G,q= ,G,q	H,q= ,H,q	I,q= ,I,q	J,q= ,J,q
F,r= ,F,r	G,r= ,G,r	H,r= ,H,r	I,r= ,I,r	J,r= ,J,r
F,s= ,F,s	G,s= ,G,s	H,s= ,H,s	I,s= ,I,s	J,s= ,J,s
F,t= ,F,t	G,t= ,G,t	H,t= ,H,t	I,t= ,I,t	J,t= ,J,t
F,u= ,F,u	G,u= ,G,u	H,u= ,H,u	I,u= ,I,u	J,u= ,J,u
F,v= ,F,v	G,v= ,G,v	H,v= ,H,v	I,v= ,I,v	J,v= ,J,v
F,w= ,F,w	G,w= ,G,w	H,w= ,H,w	I,w= ,I,w	J,w= ,J,w
F,x= ,F,x	G,x= ,G,x	H,x= ,H,x	I,x= ,I,x	J,x= ,J,x
F,y= ,F,y	G,y= ,G,y	H,y= ,H,y	I,y= ,I,y	J,y= ,J,y
F,z= ,F,z	G,z= ,G,z	H,z= ,H,z	I,z= ,I,z	J,z= ,J,z

Tim Notes:

1. I did the work for you (You're Welcome). All you have to do is copy and paste into Character Translation to test.

String Manipulation (Parsing)

More Run on Sentences cont.

Sample Base:

CoffeeAndTea
ThisIsReallyAnnoying&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.whatever
CadAbeBad

Method 1: cont.

1. 'Section #14: Character Translation' –

K,a= ,K,a	L,a= ,L,a	M,a= ,M,a	N,a= ,N,a	O,a= ,O,a
K,b= ,K,b	L,b= ,L,b	M,b= ,M,b	N,b= ,N,b	O,b= ,O,b
K,c= ,K,c	L,c= ,L,c	M,c= ,M,c	N,c= ,N,c	O,c= ,O,c
K,d= ,K,d	L,d= ,L,d	M,d= ,M,d	N,d= ,N,d	O,d= ,O,d
K,e= ,K,e	L,e= ,L,e	M,e= ,M,e	N,e= ,N,e	O,e= ,O,e
K,f= ,K,f	L,f= ,L,f	M,f= ,M,f	N,f= ,N,f	O,f= ,O,f
K,g= ,K,g	L,g= ,L,g	M,g= ,M,g	N,g= ,N,g	O,g= ,O,g
K,h= ,K,h	L,h= ,L,h	M,h= ,M,h	N,h= ,N,h	O,h= ,O,h
K,i= ,K,i	L,i= ,L,i	M,i= ,M,i	N,i= ,N,i	O,i= ,O,i
K,j= ,K,j	L,j= ,L,j	M,j= ,M,j	N,j= ,N,j	O,j= ,O,j
K,k= ,K,k	L,k= ,L,k	M,k= ,M,k	N,k= ,N,k	O,k= ,O,k
K,l= ,K,l	L,l= ,L,l	M,l= ,M,l	N,l= ,N,l	O,l= ,O,l
K,m= ,K,m	L,m= ,L,m	M,m= ,M,m	N,m= ,N,m	O,m= ,O,m
K,n= ,K,n	L,n= ,L,n	M,n= ,M,n	N,n= ,N,n	O,n= ,O,n
K,o= ,K,o	L,o= ,L,o	M,o= ,M,o	N,o= ,N,o	O,o= ,O,o
K,p= ,K,p	L,p= ,L,p	M,p= ,M,p	N,p= ,N,p	O,p= ,O,p
K,q= ,K,q	L,q= ,L,q	M,q= ,M,q	N,q= ,N,q	O,q= ,O,q
K,r= ,K,r	L,r= ,L,r	M,r= ,M,r	N,r= ,N,r	O,r= ,O,r
K,s= ,K,s	L,s= ,L,s	M,s= ,M,s	N,s= ,N,s	O,s= ,O,s
K,t= ,K,t	L,t= ,L,t	M,t= ,M,t	N,t= ,N,t	O,t= ,O,t
K,u= ,K,u	L,u= ,L,u	M,u= ,M,u	N,u= ,N,u	O,u= ,O,u
K,v= ,K,v	L,v= ,L,v	M,v= ,M,v	N,v= ,N,v	O,v= ,O,v
K,w= ,K,w	L,w= ,L,w	M,w= ,M,w	N,w= ,N,w	O,w= ,O,w
K,x= ,K,x	L,x= ,L,x	M,x= ,M,x	N,x= ,N,x	O,x= ,O,x
K,y= ,K,y	L,y= ,L,y	M,y= ,M,y	N,y= ,N,y	O,y= ,O,y
K,z= ,K,z	L,z= ,L,z	M,z= ,M,z	N,z= ,N,z	O,z= ,O,z

Tim Notes:

1. This format of, i.e., K,z=<space>,K,z instead of K= <space>,K is used because otherwise it would affect Capitalized names like BRU that would end up as B R U.

String Manipulation (Parsing)

More Run on Sentences cont.

Sample Base:

CoffeeAndTea
ThisIsReallyAnnoying&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.whatever
CadAbeBad

Method 1: cont.

1. 'Section #14: Character Translation' –

P,a= ,P,a	Q,a= ,Q,a	R,a= ,R,a	S,a= ,S,a	T,a= ,T,a
P,b= ,P,b	Q,b= ,Q,b	R,b= ,R,b	S,b= ,S,b	T,b= ,T,b
P,c= ,P,c	Q,c= ,Q,c	R,c= ,R,c	S,c= ,S,c	T,c= ,T,c
P,d= ,P,d	Q,d= ,Q,d	R,d= ,R,d	S,d= ,S,d	T,d= ,T,d
P,e= ,P,e	Q,e= ,Q,e	R,e= ,R,e	S,e= ,S,e	T,e= ,T,e
P,f= ,P,f	Q,f= ,Q,f	R,f= ,R,f	S,f= ,S,f	T,f= ,T,f
P,g= ,P,g	Q,g= ,Q,g	R,g= ,R,g	S,g= ,S,g	T,g= ,T,g
P,h= ,P,h	Q,h= ,Q,h	R,h= ,R,h	S,h= ,S,h	T,h= ,T,h
P,i= ,P,i	Q,i= ,Q,i	R,i= ,R,i	S,i= ,S,i	T,i= ,T,i
P,j= ,P,j	Q,j= ,Q,j	R,j= ,R,j	S,j= ,S,j	T,j= ,T,j
P,k= ,P,k	Q,k= ,Q,k	R,k= ,R,k	S,k= ,S,k	T,k= ,T,k
P,l= ,P,l	Q,l= ,Q,l	R,l= ,R,l	S,l= ,S,l	T,l= ,T,l
P,m= ,P,m	Q,m= ,Q,m	R,m= ,R,m	S,m= ,S,m	T,m= ,T,m
P,n= ,P,n	Q,n= ,Q,n	R,n= ,R,n	S,n= ,S,n	T,n= ,T,n
P,o= ,P,o	Q,o= ,Q,o	R,o= ,R,o	S,o= ,S,o	T,o= ,T,o
P,p= ,P,p	Q,p= ,Q,p	R,p= ,R,p	S,p= ,S,p	T,p= ,T,p
P,q= ,P,q	Q,q= ,Q,q	R,q= ,R,q	S,q= ,S,q	T,q= ,T,q
P,r= ,P,r	Q,r= ,Q,r	R,r= ,R,r	S,r= ,S,r	T,r= ,T,r
P,s= ,P,s	Q,s= ,Q,s	R,s= ,R,s	S,s= ,S,s	T,s= ,T,s
P,t= ,P,t	Q,t= ,Q,t	R,t= ,R,t	S,t= ,S,t	T,t= ,T,t
P,u= ,P,u	Q,u= ,Q,u	R,u= ,R,u	S,u= ,S,u	T,u= ,T,u
P,v= ,P,v	Q,v= ,Q,v	R,v= ,R,v	S,v= ,S,v	T,v= ,T,v
P,w= ,P,w	Q,w= ,Q,w	R,w= ,R,w	S,w= ,S,w	T,w= ,T,w
P,x= ,P,x	Q,x= ,Q,x	R,x= ,R,x	S,x= ,S,x	T,x= ,T,x
P,y= ,P,y	Q,y= ,Q,y	R,y= ,R,y	S,y= ,S,y	T,y= ,T,y
P,z= ,P,z	Q,z= ,Q,z	R,z= ,R,z	S,z= ,S,z	T,z= ,T,z

String Manipulation (Parsing)

More Run on Sentences cont.

Sample Base:

CoffeeAndTea
ThisIsReallyAnnoying&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.whatever
CadAbeBad

Method 1: cont.

1. 'Section #14: Character Translation' –

U,a= ,U,a	V,a= ,V,a	W,a= ,W,a	X,a= ,X,a	Y,a= ,Y,a
U,b= ,U,b	V,b= ,V,b	W,b= ,W,b	X,b= ,X,b	Y,b= ,Y,b
U,c= ,U,c	V,c= ,V,c	W,c= ,W,c	X,c= ,X,c	Y,c= ,Y,c
U,d= ,U,d	V,d= ,V,d	W,d= ,W,d	X,d= ,X,d	Y,d= ,Y,d
U,e= ,U,e	V,e= ,V,e	W,e= ,W,e	X,e= ,X,e	Y,e= ,Y,e
U,f= ,U,f	V,f= ,V,f	W,f= ,W,f	X,f= ,X,f	Y,f= ,Y,f
U,g= ,U,g	V,g= ,V,g	W,g= ,W,g	X,g= ,X,g	Y,g= ,Y,g
U,h= ,U,h	V,h= ,V,h	W,h= ,W,h	X,h= ,X,h	Y,h= ,Y,h
U,i= ,U,i	V,i= ,V,i	W,i= ,W,i	X,i= ,X,i	Y,i= ,Y,i
U,j= ,U,j	V,j= ,V,j	W,j= ,W,j	X,j= ,X,j	Y,j= ,Y,j
U,k= ,U,k	V,k= ,V,k	W,k= ,W,k	X,k= ,X,k	Y,k= ,Y,k
U,l= ,U,l	V,l= ,V,l	W,l= ,W,l	X,l= ,X,l	Y,l= ,Y,l
U,m= ,U,m	V,m= ,V,m	W,m= ,W,m	X,m= ,X,m	Y,m= ,Y,m
U,n= ,U,n	V,n= ,V,n	W,n= ,W,n	X,n= ,X,n	Y,n= ,Y,n
U,o= ,U,o	V,o= ,V,o	W,o= ,W,o	X,o= ,X,o	Y,o= ,Y,o
U,p= ,U,p	V,p= ,V,p	W,p= ,W,p	X,p= ,X,p	Y,p= ,Y,p
U,q= ,U,q	V,q= ,V,q	W,q= ,W,q	X,q= ,X,q	Y,q= ,Y,q
U,r= ,U,r	V,r= ,V,r	W,r= ,W,r	X,r= ,X,r	Y,r= ,Y,r
U,s= ,U,s	V,s= ,V,s	W,s= ,W,s	X,s= ,X,s	Y,s= ,Y,s
U,t= ,U,t	V,t= ,V,t	W,t= ,W,t	X,t= ,X,t	Y,t= ,Y,t
U,u= ,U,u	V,u= ,V,u	W,u= ,W,u	X,u= ,X,u	Y,u= ,Y,u
U,v= ,U,v	V,v= ,V,v	W,v= ,W,v	X,v= ,X,v	Y,v= ,Y,v
U,w= ,U,w	V,w= ,V,w	W,w= ,W,w	X,w= ,X,w	Y,w= ,Y,w
U,x= ,U,x	V,x= ,V,x	W,x= ,W,x	X,x= ,X,x	Y,x= ,Y,x
U,y= ,U,y	V,y= ,V,y	W,y= ,W,y	X,y= ,X,y	Y,y= ,Y,y
U,z= ,U,z	V,z= ,V,z	W,z= ,W,z	X,z= ,X,z	Y,z= ,Y,z

Tim Notes:

1. There must NOT be any trailing <spaces> when entering items into the Character Translation Data Entry form.
2. Use Ctrl + Enter to ensure no trailing <spaces>, i.e. – enter 'U,z=<space>,U,z Ctrl + Enter.

String Manipulation (Parsing)

More Run on Sentences cont.

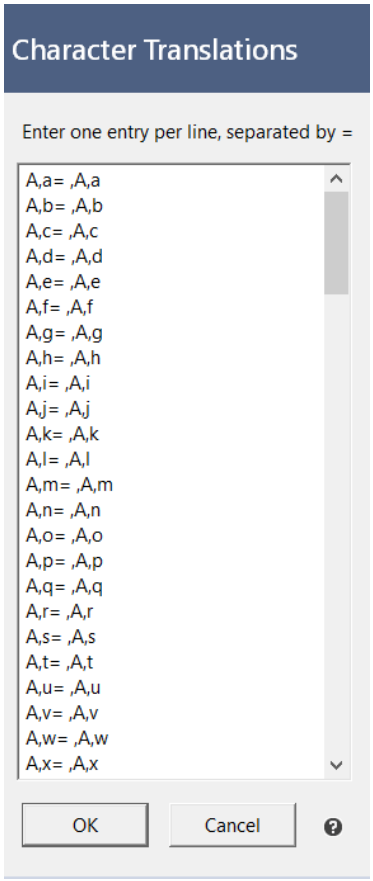
Sample Base:

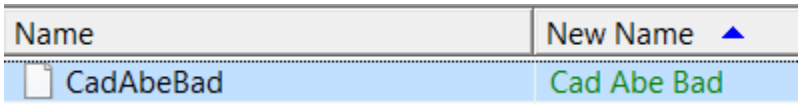
CoffeeAndTea
ThisIsReallyAnnoying&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.whatever
CadAbeBad

Method 1: cont.

1. 'Section #14: Character Translation' –

Z,a= ,Z,a
Z,b= ,Z,b
Z,c= ,Z,c
Z,d= ,Z,d
Z,e= ,Z,e
Z,f= ,Z,f
Z,g= ,Z,g
Z,h= ,Z,h
Z,i= ,Z,i
Z,j= ,Z,j
Z,k= ,Z,k
Z,l= ,Z,l
Z,m= ,Z,m
Z,n= ,Z,n
Z,o= ,Z,o
Z,p= ,Z,p
Z,q= ,Z,q
Z,r= ,Z,r
Z,s= ,Z,s
Z,t= ,Z,t
Z,u= ,Z,u
Z,v= ,Z,v
Z,w= ,Z,w
Z,x= ,Z,x
Z,y= ,Z,y
Z,z= ,Z,z





Tim Notes:

1. The same method can be used for numeric digits as well.

i.e., 4= <space>,4

String Manipulation (Parsing)

More Run on Sentences cont.

Sample Base:

CoffeeAndTea
ThisIsReally&Annoying$&.whatever
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.txt
CadAbeBad

Method #2:

RegEx

This is meant for multi-runs. each run will segment a different word based on the pattern established by the Lookarounds.

Match: (.+?)(?<=[a-z])(?=[A-Z])(.*)
Replace: \1 \2

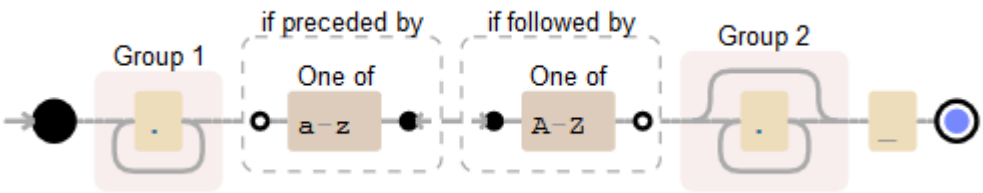
Name	New Name
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.txt	In The4HrsISpentResearchingThisICouldHaveTypedThemOut.txt
CadAbeBad	Cad AbeBad
CoffeeAndTea	Coffee AndTea
ThisIsReallyAnnoying&.whatever	This IsReallyAnnoying&.whatever

Sample: First Run

```
.........................
Match 1: CoffeeAndTea
Group 1: Coffee
Group 2: AndTea
```

Sample: Second Run

```
.........................
Match 1: Coffee AndTea
Group 1: Coffee And
Group 2: Tea
```



String Manipulation (Parsing)

More Run on Sentences cont.

Using: CoffeeAndTea

Method #2: cont.

Match: (.+?)(?<=[a-z])(?=[A-Z])(.*)

Where:

\1 = Coffee \2 = AndTea

Tim Analysis: First Run

1..	Match against any character	Capture Group 1 = C
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = C
4. (?<=[a-z])	Lookbehind. Look to the left. If character is a single lowercase letter then expression = True and evaluation continues.= BOL so False	Changes Capture Group 1 = Co

Tim Notes:

1. Since it failed the first test, the RegEx Engine will backtrack, then test the current character = 'C', and test the next character on down the line looking for a match until EOL. It satisfies the match on the second letter 'o'. This changes Capture Group 1 to 'Co'. Current position = first 'o' of 'Coffee'

5. (?=[A-Z])	Lookahead. Look to the right. If next character is a single uppercase letter then expression = True and evaluation continues.= 'f' so False	Changes Capture Group 1 = Coffee

Tim Notes:

1. Same as before. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on 'A' of 'And'. This changes Capture Group 1 to 'Coffee'. Current position = 'A' of 'And'

6. .*	Match against remaining string	Capture Group 2 = AndTea

String Manipulation (Parsing)

More Run on Sentences cont.

Using: Coffee AndTea

Method #2: cont.

Match: (.+?)(?<=[a-z])(?=[A-Z])(.*)

Where:

\1 = Coffee And \2 = Tea

Tim Analysis: Second Run

1..	Match against any character	Capture Group 1 = C
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = C
4. (?<=[a-z])	Lookbehind. Look to the left. If character is a single lowercase letter then expression = True and evaluation continues.= BOL so False	Changes Capture Group 1 = Co
5. (?=[A-Z])	Lookahead. Look to the right. If next character is a single uppercase letter then expression = True and evaluation continues.= 'f' so False	Changes Capture Group 1 = Coffee And

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on 'T' of 'Tea. This changes Capture Group 1 to 'Coffee And'

   Question: Why didn't it just match against the 'A' of 'And' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter preceded by a lowercase letter. This would be the 'T' preceded by the 'd' in the sample 'Coffee AndTea'

6. .*	Match against remaining string	Capture Group 2 = AndTea

String Exhausted – No More Matches

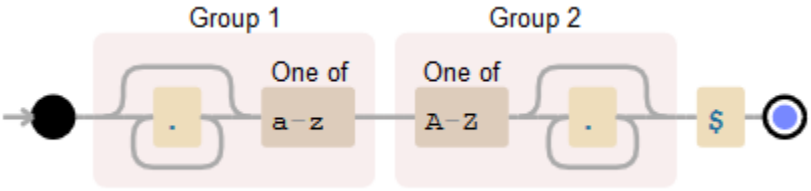
String Manipulation (Parsing)

More Run on Sentences cont.

Using: CoffeeAndTea

Method #2: cont.

A RegEx that doesn't use Lookarounds:

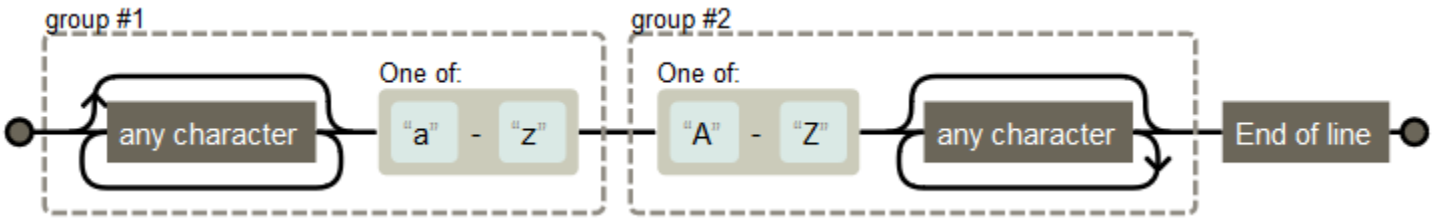


Match: (.*?[a-z])([A-Z].*)$
Replace: \1 \2

Name	New Name
CoffeeAndTea	Coffee AndTea

Sample: First Run

```
.....................
Match 1: CoffeeAndTea
Group 1: Coffee
Group 2: AndTea
```



Where:

\1 = Coffee \2 = AndTea

Tim Analysis: First Run

1..	Match against any character	Capture Group 1 = C
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. [a-z]	Match against a class consisting of a lowercase letter As soon as it matches the 'o' it includes the 'C' because of the previous (.*?)	Capture Group 1 = Co
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = A
6. .*	Match against remaining string	Capture Group 2 = AndTea
7. $	Must be at EOL (End of Line)	

Tim Notes:

1. Only showing first run, since subsequent runs have already been analyzed.

Forum Contributor: Glenn

String Manipulation (Parsing)

More Run on Sentences cont.

Using: ThisIsReally&Annoying$&.whatever

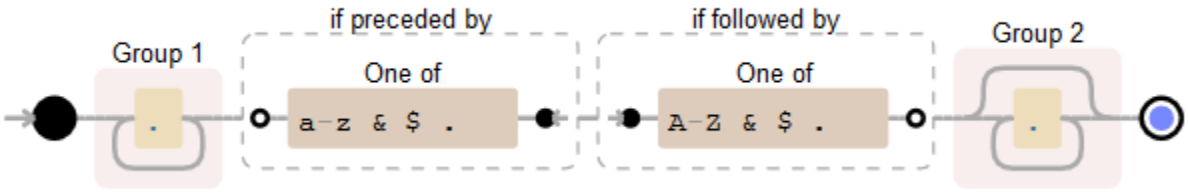
Method #2: cont.

Tim Contribution –

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)
Replace: \1 \2

Name	New Name
ThisIsReally&Annoying$&.whatever	This IsReally&Annoying$&.whatever



Breaks down:

(.+?)	Capture Group 1					
			Class Consisting of:			
(?<=[a-z&$\.])	Non-marking Group Class	Look Behind	a-z	&	$	\.
(?=[A-Z&$\.])	Non-marking Group Class	Look Ahead	A-Z	&	$	\.
(.*)	Capture Group 2					

Tim Notes: Observations only

1. The symbols appear in each of the Lookarounds because they have to be matched both going forward or back.
   a. The symbols can appear in the string together, preceding a lowercase letter or preceding an uppercase letter

String Manipulation (Parsing)

More Run on Sentences cont.

Using: ThisIsReallyAnnoying&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Using the Contributor's RegEx, this is how it performs on the string:

First Run: Is

```
.........................................
Match 1: ThisIsReallyAnnoying&.whatever
Group 1: This
Group 2: IsReallyAnnoying&.whatever
```

Second Run: Really

```
.........................................
Match 1: This IsReallyAnnoying&.whatever
Group 1: This Is
Group 2: ReallyAnnoying&.whatever
```

Third Run: Annoying

```
.........................................
Match 1: This Is ReallyAnnoying&.whatever
Group 1: This Is Really
Group 2: Annoying&.whatever
```

Fourth Run:

String exhausted. No more matches can be made.

Tim Notes: The Contributor's RegEx

1. It can't handle symbols so it treats the ampersand symbol as part of the word 'Annoying'.
2. It can't separate the word 'whatever' because if it is treated as a file extension, then of course, it would not be considered part of the string.
3. If it is treated as part of the string by enabling, 'Rename File Extension As Being part of File Name' from the 'File/ Folder Extensions' sub-menu of the 'Renaming Options Menu', then the <dot> would be treated as a symbol and therefore also be considered as part of the word 'Annoying'.
4. To the RegEx, the string ''Annoying&.whatever' is one word.

String Manipulation (Parsing)

More Run on Sentences cont.

Using: ThisIsReally&Annoying$&.whatever

Method #2: cont.

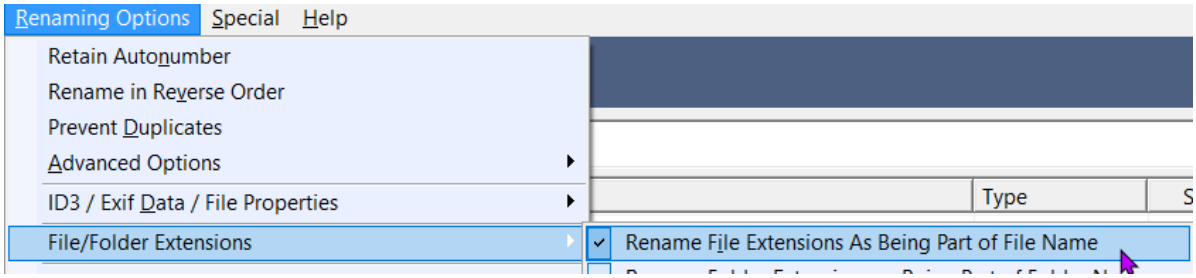
Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Continuing with a third run using the sample string containing all of the symbols –

To match against the <dot>, enable, 'Rename File Extension As Being part of File Name'



Third Run: First Ampersand

```
Match 1: This Is Really&Annoying$&.whatever
Group 1: This Is Really
Group 2: &Annoying$&.whatever
```

Fourth Run: Annoying

```
Match 1: This Is Really &Annoying$&.whatever
Group 1: This Is Really &
Group 2: Annoying$&.whatever
```

Fifth Run: Dollar Sign

```
Match 1: This Is Really & Annoying$&.whatever
Group 1: This Is Really & Annoying
Group 2: $&.whatever
```

Sixth Run: Second Ampersand

```
Match 1: This Is Really & Annoying $&.whatever
Group 1: This Is Really & Annoying $
Group 2: &.whatever
```

Seventh Run: <dot>

```
Match 1: This Is Really & Annoying $ &.whatever
Group 1: This Is Really & Annoying $ &
Group 2: .whatever
```

Eighth Run:

```
This Is Really & Annoying $ & .whatever
```

String Exhausted.

String Manipulation (Parsing)

More Run on Sentences cont.

Using: ThisIsReally&Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This \2 = IsReally&Annoying$&.whatever

Tim Analysis: First Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.])	Lookbehind.	Changes Capture Group 1 = Th
	Look to the left. If character is a:	
	a. single lowercase letter	
	b. ampersand	
	c. dollar sign	
	d. <dot>	
	then expression = True and evaluation	
	continues.= BOL so False.	
	Continues to search through string.	
	Matches against 'h' of 'This'	
5. (?=[A-Z&$\.])	Lookahead.	Changes Capture Group 1 = This
	Look to the right. If next character is a:	
	a. single uppercase letter	
	b. ampersand	
	c. dollar sign	
	d. <dot>	
	then expression = True and evaluation	
	continues.= 'f' so False.	
	Continues to search through string.	
	Matches against 'I' of 'Is'	

Class

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on 'I' of 'Is. This changes Capture Group 1 to 'This'

6. .*	Match against remaining string	Capture Group 2 = IsReally&Annoying$&.whatever

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This IsReally&Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is \2 = Really&Annoying$&.whatever

Tim Analysis: Second Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.])	Lookbehind.	Changes Capture Group 1 = Th
	Look to the left. If character is a:	
	a. single lowercase letter	
	b. ampersand	
	c. dollar sign	
	d. <dot>	
	then expression = True and evaluation	
	continues.= BOL so False.	
	Continues to search through string.	
	Matches against 'h' of 'This'	
5. (?=[A-Z&$\.])	Lookahead.	Changes Capture Group 1 = This Is
	Look to the right. If next character is a:	
	a. single uppercase letter	
	b. ampersand	
	c. dollar sign	
	d. <dot>	
	then expression = True and evaluation	
	continues.= 'i' so False.	
	Continues to search through string.	
	Matches against 'R' of 'Really'	

Class

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This IsReally&Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is \2 = Really&Annoying$&.whatever

Tim Analysis: Second Run cont.

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on 'I' of 'Is. This changes Capture Group 1 to 'This'

   Question: Why didn't it just match against the 'I' of 'Is' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter an ampersand, a dollar sign or a <dot> character . This would be the 'R' preceded by the 's' in the sample 'This IsReally&Annoying$&.whatever'

6. .* Match against remaining string Capture Group 2 = Really&Annoying$&.whatever

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really&Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really \2 = &Annoying$&.whatever

Tim Analysis: Third Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.]) Class	Lookbehind. Look to the left. If character is a: a. single lowercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= BOL so False. Continues to search through string. Matches against 'h' of 'This'	Changes Capture Group 1 = Th
5. (?=[A-Z&$\.])	Lookahead. Look to the right. If next character is a: a. single uppercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= 'i' so False. Continues to search through string. Matches against '&' of 'Really&'	Changes Capture Group 1 = This Is Really

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This IsReally&Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really \2 = &Annoying$&.whatever

Tim Analysis: Third Run cont.

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on '&' of 'Really&'. This changes Capture Group 1 to 'This Is Really'

   Question: Why didn't it just match against the 'I' of 'Is' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter an ampersand, a dollar sign or a <dot> character . This would be the '&' preceded by the 'y' in the sample 'This Is Really&Annoying$&.whatever'

6. .* Match against remaining string Capture Group 2 = &Annoying$&.whatever

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really &Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & \2 = Annoying$&.whatever

Tim Analysis: Fourth Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.]) Class	Lookbehind. Look to the left. If character is a: a. single lowercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= BOL so False. Continues to search through string. Matches against 'h' of 'This'	Changes Capture Group 1 = Th
5. (?=[A-Z&$\.])	Lookahead. Look to the right. If next character is a: a. single uppercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= 'i' so False. Continues to search through string. Matches against 'A' of 'Annoying'	Changes Capture Group 1 = This Is Really &

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really &Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & \2 = Annoying$&.whatever

Tim Analysis: Fourth Run cont.

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on 'A' of 'Annoying'. This changes Capture Group 1 to 'This Is Really &'

   Question: Why didn't it just match against the 'I' of 'Is' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter an ampersand, a dollar sign or a <dot> character . This would be the 'A' preceded by the '&' in the sample 'This Is Really &Annoying$&.whatever'

6. .* Match against remaining string Capture Group 2 = Annoying$&.whatever

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really & Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & Annoying \2 = $&.whatever

Tim Analysis: Fifth Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.]) Class	Lookbehind. Look to the left. If character is a: a. single lowercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= BOL so False. Continues to search through string. Matches against 'h' of 'This'	Changes Capture Group 1 = Th
5. (?=[A-Z&$\.])	Lookahead. Look to the right. If next character is a: a. single uppercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= 'i' so False. Continues to search through string. Matches against '$'	Changes Capture Group 1 = This Is Really & Annoying

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really & Annoying$&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & Annoying \2 = $&.whatever

Tim Analysis: Fifth Run cont.

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on '$' of '$&". This changes Capture Group 1 to 'This Is Really & Annoying'

   Question: Why didn't it just match against the 'I' of 'Is' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter an ampersand, a dollar sign or a <dot> character . This would be the '$' preceded by the 'g' in the sample 'This Is Really & Annoying$&.whatever'

6. .* Match against remaining string Capture Group 2 = $&.whatever

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really & Annoying $&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & Annoying $ \2 = &.whatever

Tim Analysis: Sixth Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.]) Class	Lookbehind. Look to the left. If character is a: a. single lowercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= BOL so False. Continues to search through string. Matches against 'h' of 'This'	Changes Capture Group 1 = Th
5. (?=[A-Z&$\.])	Lookahead. Look to the right. If next character is a: a. single uppercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= 'i' so False. Continues to search through string. Matches against '$' of '$&.whatever'	Changes Capture Group 1 = This Is Really & Annoying $

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really & Annoying $&.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & Annoying $ \2 = &.whatever

Tim Analysis: Sixth Run cont.

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on '&' of '&.whatever'. This changes Capture Group 1 to 'This Is Really & Annoying $'

   Question: Why didn't it just match against the 'I' of 'Is' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter an ampersand, a dollar sign or a <dot> character . This would be the '&' preceded by the '$' in the sample 'This Is Really & Annoying $&.whatever'

6. .* Match against remaining string Capture Group 2 = &.whatever

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really & Annoying $ &.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & Annoying $ & \2 = .whatever

Tim Analysis: Seventh Run

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = T
4. (?<=[a-z&$\.]) Class	Lookbehind. Look to the left. If character is a: a. single lowercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= BOL so False. Continues to search through string. Matches against 'h' of 'This'	Changes Capture Group 1 = Th
5. (?=[A-Z&$\.])	Lookahead. Look to the right. If next character is a: a. single uppercase letter b. ampersand c. dollar sign d. <dot> then expression = True and evaluation continues.= 'i' so False. Continues to search through string. Matches against '.' of '&.whatever'	Changes Capture Group 1 = This Is Really & Annoying $ &

String Manipulation (Parsing)

More Run on Sentences cont.

Using: This Is Really & Annoying $ &.whatever

Method #2: cont.

Tim Contribution – cont.

This will handle the symbols $, &, \. (more symbols can be added – just add them to both sides of the Lookarounds).

Match: (.+?)(?<=[a-z&$\.])(?=[A-Z&$\.])(.*)

Where:

\1 = This Is Really & Annoying $ & \2 = .whatever

Tim Analysis: Seventh Run cont.

Tim Notes:

1. If it tests false it will continue searching the string looking for a match until EOL. It satisfies the match on '.' of '.whatever'. This changes Capture Group 1 to 'This Is Really & Annoying $ &'

   Question: Why didn't it just match against the 'I' of 'Is' again as it did in the first run?

   Because as I see it, it is also using the Lookbehind at the same time so it is looking for a match of an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter an ampersand, a dollar sign or a <dot> character . This would be the '.' preceded by the '&' in the sample 'This Is Really & Annoying $ &.whatever'

6. .* Match against remaining string Capture Group 2 = .whatever

Tim Analysis: Eighth Run

Using: This Is Really & Annoying $ & .whatever

String Exhausted – No More Matches can be made.

Tim Notes:

1. Why? Because the RegEx will fail when it tries to match up an uppercase letter, an ampersand, a dollar sign or a <dot> character preceded by either a lowercase letter, an ampersand, a dollar sign or a <dot> character.

2. When the Lookahead tries to match against the <dot>, it will fail because the preceding character is a <space>, so the Lookbehind Fails. The next characters are all lowercase letters. The RegEx Engine will reach the EOL and then the entire RegEx Fails at that point.

String Manipulation (Parsing)

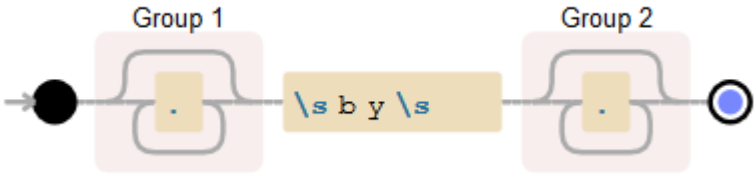
Move all text after "keyword" to front of file name

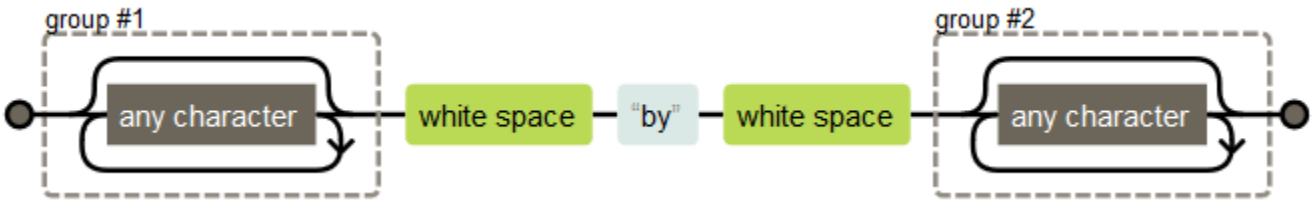
iPhone by Apple

Match: (.*)\sby\s(.*)
Replace: \2 - \1

Name	New Name
iPhone by Apple	Apple - iPhone

```
.........................
Match 1: iPhone by Apple
Group 1: iPhone
Group 2: Apple
```





Forum Contributor: therube

Where:

\1 = iPhone \2 = Apple

Tim Analysis:

1..	Match against any character	Capture Group 1 = i
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \s	Match against <space> (Not Captured)	Changes Capture Group 1 = iPhone by
4. by	Match against string 'by' (Not Captured)	Changes Capture Group 1= iPhone <space>
5. \s	Match against <space> (Not Captured)	Changes Capture Group 1 = iPhone
6. .*	Match against remaining string	Capture Group 2 = Apple

String Manipulation (Parsing)

Move all text after "keyword" to front of file name cont.

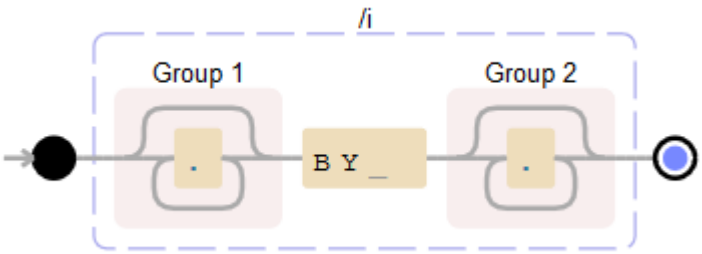
iPhone by Apple

The Case for Case Insensitivity

BRU does not directly support the Case Sensitive/Insensitive modifier. As introduced in the RegEx Manual Volume I, I found a work-around for Case Insensitivity using (?i). This modifier turns on Case Insensitivity on demand. The opposite of this is (?-i) that turns off Case Insensitivity on demand.

Match: (?i)(.*)BY (.*)
Replace: \2 - \1

Name	New Name
iPhone by Apple	Apple - iPhone



Sample:

```
.........................
Match 1: iPhone by Apple
Group 1: iPhone
Group 2: Apple
```

Tim Analysis:

1. (?i)	Turn on case Insensitivity	
2. .	Match against any character	Capture Group 1 = i
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. BY	Match against string 'BY,By,bY,by' (Not Captured) Case Insensitive.	Changes Capture Group 1 = iPhone
5. <space>	Match against <space> (Not Captured)	
6. .*	Match against remaining string	Capture Group 2 = Apple

Forum Contributor: Admin

String Manipulation (Parsing)

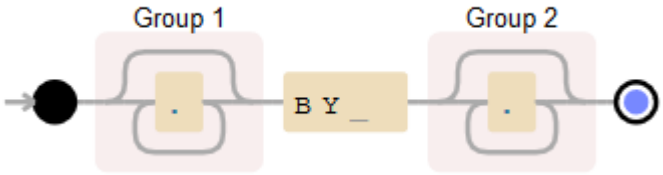
Move all text after "keyword" to front of file name cont.

iPhone by Apple

The Case for Case Insensitivity cont.

This will not match:

Match: (.*)BY (.*)

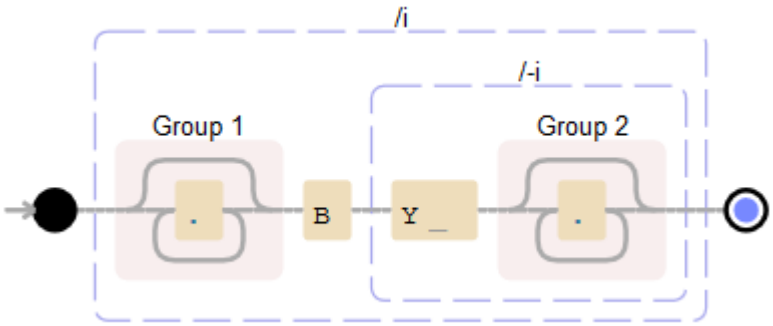


Tim Analysis:

1. .	Match against any character	Capture Group 1 = i
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. BY	Match against string 'BY' (Not Captured) Fails = 'by'	

Neither will this:

Match: (?i)(.*)B(?-i)Y (.*)



Tim Analysis:

1. (?i)	Turn on case Insensitivity	
2. .	Match against any character	Capture Group 1 = i
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. B	Match against string 'B,b' (Not Captured) Case Insensitive.	Changes Capture Group 1 = iPhone
5. (?-i)	Turn off case Insensitivity	
6. Y	Match against string 'Y' (Not Captured) Fails = 'y'	

Tim Notes:

1. By enabling Case Insensitivity at the beginning of the RegEx, it was able to match against the 'B', but then it was turned off making the RegEx revert back to Case Sensitive so it was unable to match against the 'Y'.

String Manipulation (Parsing)

Suffix Year and add parentheses

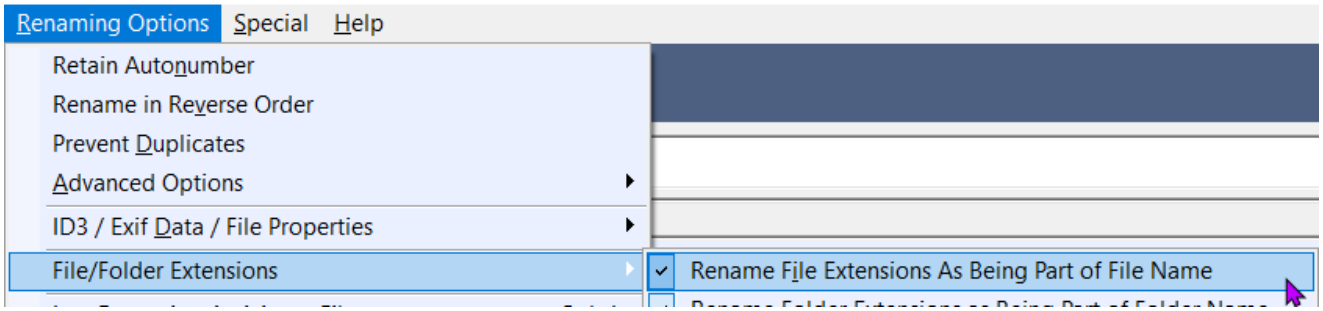
2011 6.6]	to	(2011)
abc 1990 11.11 .xyz	to	abc (1990).xyz
abc[2011 6.6].xyz	to	abc (2011).xyz
abc[1234 6.6].xyz	to	Ignore (1234 is not a year)
This is a test [1970 1.1].ext	to	This is a test (1970).ext

Observations:

1. Source filenames can have a year within brackets or not
2. All have a decimal portion that must be removed
3. Suffix the year in parentheses substituted for brackets

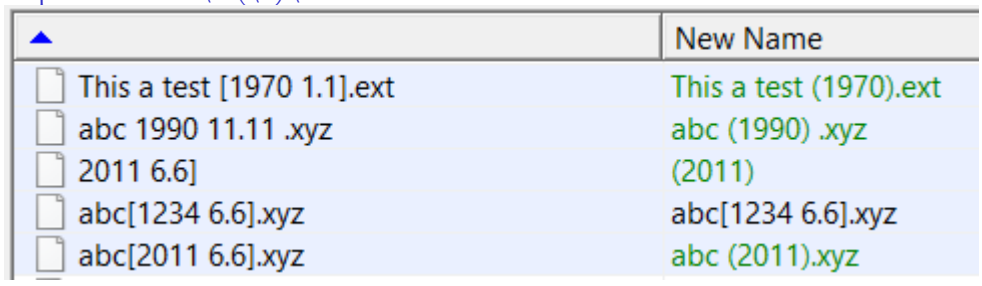
Since some of the strings use the <dot> character, you have to enable, 'Rename File Extensions As Being Part of File Name' from the File/ Folder Extensions sub menu of the Renaming Options Menu.

.



1. RegEx

Match: (.*?)\[*((19|20)\d{2}) \d{1,2}\.\d{1,2}\]*(.*)
Replace: \1 (\2)\4

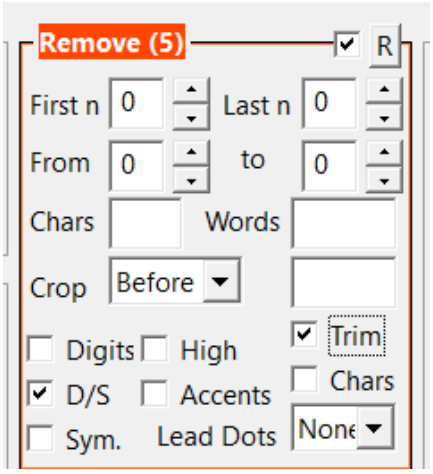


String Manipulation (Parsing)

Suffix Year and add parentheses cont.

2011 6.6]	to	(2011)
abc 1990 11.11 .xyz	to	abc (1990).xyz
abc[2011 6.6].xyz	to	abc (2011).xyz
abc[1234 6.6].xyz	to	Ignore (1234 is not a year)
This is a test [1970 1.1].ext	to	This is a test (1970).ext

2. 'Section #5: Remove' –



D/S: Enable
Trim: Enable

Sample:

		Start	Length
Match 1:	abc[2011 6.6]	0	13
Group 1:	abc	0	3
Group 2:	2011	4	4
Group 3:	20	4	2
Group 4:		13	0

Sample:

		Start	Length
Match 1:	This a test [1970 1.1]	0	22
Group 1:	This a test	0	12
Group 2:	1970	13	4
Group 3:	19	13	2
Group 4:		22	0

Sample:

		Start	Length
Match 1:	2011 6.6]	0	9
Group 1:		0	0
Group 2:	2011	0	4
Group 3:	20	0	2
Group 4:		9	0

What is Group 4? It will be reserved for any leftover text. Because the extension is treated as part of the filename, the extension will also appear in Group 4:

Sample: Without extension

abc 1990 11.11

		Start	Length
Match 1:	abc 1990 11.11	0	14
Group 1:	abc	0	4
Group 2:	1990	4	4
Group 3:	19	4	2
Group 4:		14	0

Sample: With Extension

abc 1990 11.11.xyz

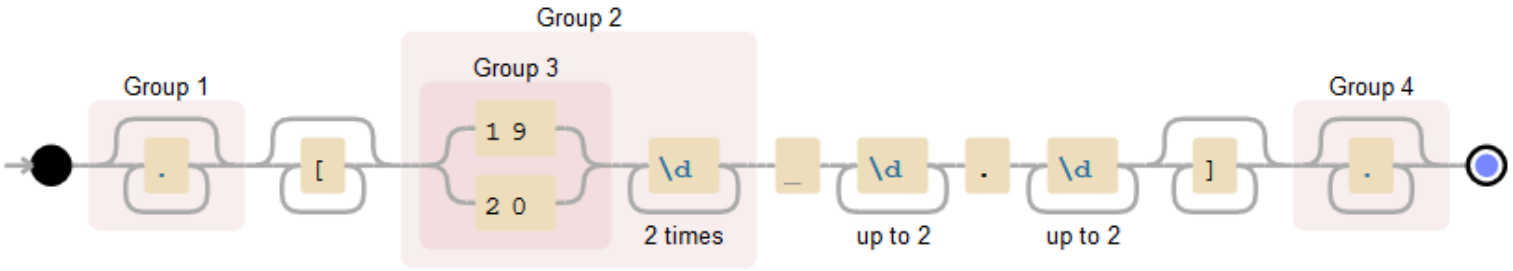
		Start	Length
Match 1:	abc 1990 11.11.xyz	0	18
Group 1:	abc	0	4
Group 2:	1990	4	4
Group 3:	19	4	2
Group 4:	.xyz	14	4

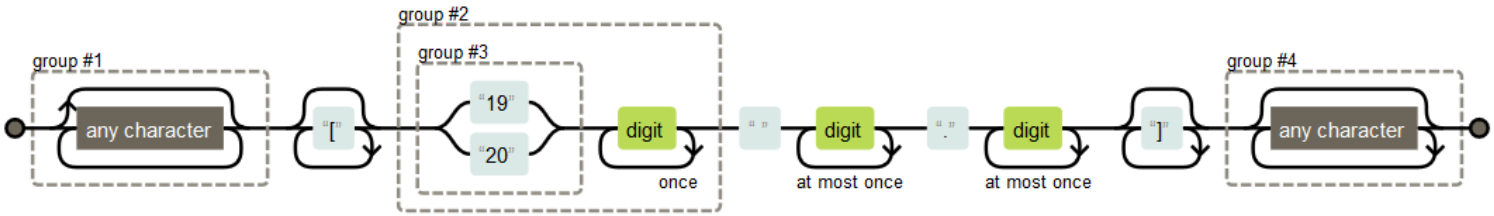
String Manipulation (Parsing)

Suffix Year and add parentheses cont.

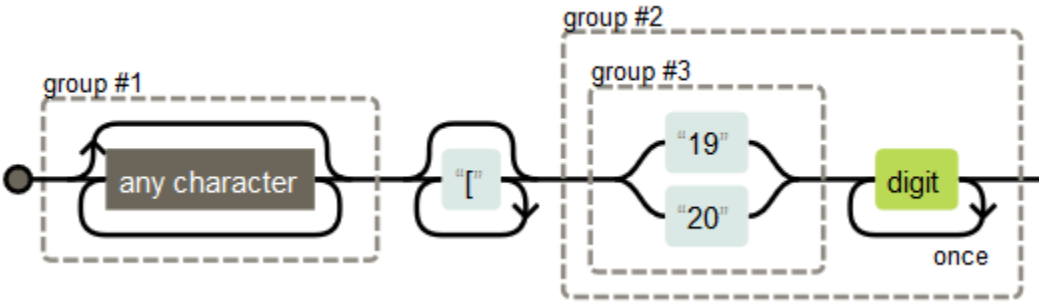
2011 6.6]	to	(2011)
abc 1990 11.11 .xyz	to	abc (1990).xyz
abc[2011 6.6].xyz	to	abc (2011).xyz
abc[1234 6.6].xyz	to	Ignore (1234 is not a year)
This is a test [1970 1.1].ext	to	This is a test (1970).ext

Match: (.*?)\[*((19|20)\d{2}) \d{1,2}\.\d{1,2}\]*(.*)





Expanded:

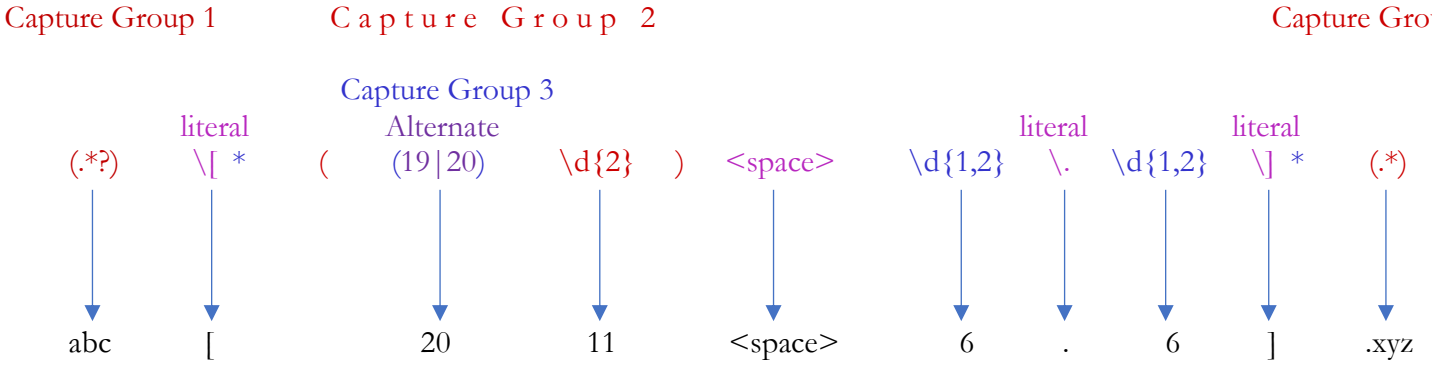


String Manipulation (Parsing)

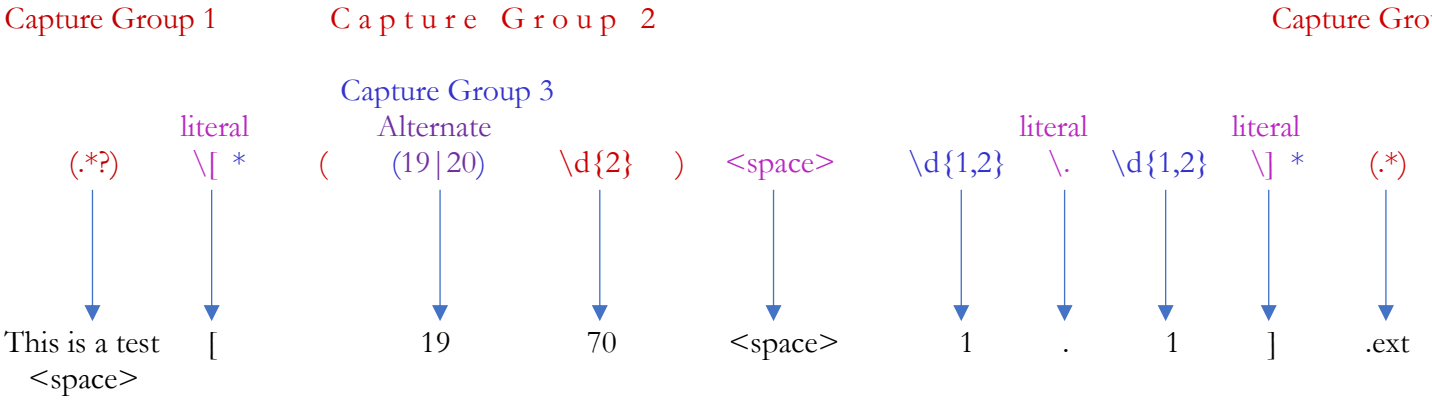
Suffix Year and add parentheses cont.

Breaks down:

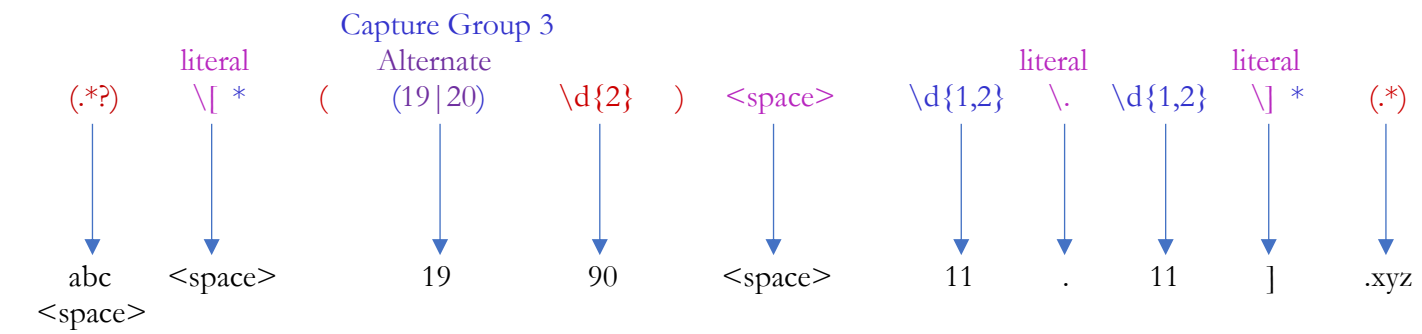
Using: abc[2011 6.6].xyz



Using: This a test [1970 1.1].ext



Using: abc 1990 11.11 .xyz



String Manipulation (Parsing)

Suffix Year and add parentheses cont.

Using: This a test [1970 1.1].ext

Match: (.*?)\[*((19|20)\d{2}) \d{1,2}\.\d{1,2}\]*(.*)

Where:

\1 = This is a test \2 = 1970 \3 = 19 \4 = .ext

This uses an Alternate for Capture Group 3:

This	OR	This
19	|	20

Tim Analysis:

1..	Match against any character		Capture Group 1 = T
2. *	Make it Greedy		Capture Group 1 = <entire string>
3. ?	But not too Greedy		Capture Group 1 = BOL
4. \[	Match against left bracket (Not Captured)		Changes Capture Group 1 = This is a test <space>
5. *	Make it Greedy Current position = EOL		Changes Capture Group 1 = <entire string>
6. 19|20	Alternate.		
	a. 19	Match against string '19' Backtracks = '19' = True	Capture Group 3 = 19 Capture Group 2 = 19
	or		
	a. 20	Match against string '20' Backtracks = '19' = False	

Tim Notes:

1. Alternate is used to capture the Century part of the year. Capture Group 2 is 'nested'. It contains Capture Group 3. When a Capture Group is nested, the value is the accumulation of all values captured. Thus far Capture Group 3 = '19', so Capture Group 2 = '19'.

7. \d{2}	Match against 2 numeric digits. This is the decade part of the year.	Capture Group 2 = 1970 Capture Group 3 = Unchanged

Tim Notes:

1. Here the capture of the decade is part of Capture Group 2 but does not fall within Capture Group 3. So Capture Group 2 has the value of Capture Group 3 – the '19' in addition to the new value '70' – so Capture Group 2 = '1970' while Capture Group 3's value, the '19', doesn't change.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

Using: This a test [1970 1.1].ext

Match: (.*?)\[*((19|20)\d{2}) \d{1,2}\.\d{1,2}\]*(.*)

Where:

\1 = This is a test \2 = 1970 \3 = 19 \4 = .ext

Tim Analysis: cont.

8. <space>	Match against a <space> character (Not Captured) This is the <space> following the year.	All Capture Groups = Unchanged
9. \d	Match against Numeric Digit (Not Captured) This is the 'whole' number portion of any decimal number = '1'	
10. {1,2}	Specifies a minimum of 1 and maximum of 2 match repetitions. This is to cover the 'whole' number portion of any decimal number from 1 to two numeric digits in length. There is only one, so the minimum iteration of 1 is used.	
11. \.	Match against a literal <dot> character. (Not Captured) This would be the decimal dot.	
12. \d	Match against Numeric Digit (Not Captured) This is the first numeric digit of the decimal portion. = '1'	
13. {1,2}	Specifies a minimum of 1 and maximum of 2 match repetitions. This is to cover the decimal portion from 1 to two numeric digits in length. There is only one, so the minimum iteration of 1 is used.	
14. \]	Match against right bracket (Not Captured)	
15. *	Make it Greedy	All Capture Groups = Unchanged
16. .*	Match against remaining string	Capture Group 4 = .ext

Tim Notes:

1. Steps 9 through 15 are meant to isolate any decimal numeric portion that appears after the year. Since the resulting value is not captured, this will remove it from the finished New name because there will not be any backreference to represent a Capture Group with that value.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

2011 6.6]	to	(2011)
abc 1990 11.11 .xyz	to	abc (1990).xyz
abc[2011 6.6].xyz	to	abc (2011).xyz
abc[1234 6.6].xyz	to	Ignore (1234 is not a year)
This is a test [1970 1.1].ext	to	This is a test (1970).ext

I needed to include those samples that would have text following the numeric portion.

So new additions to the Sample Group are:

abc [2011 7.6] the.xyz	to	abc the (2011).xyz	Text following decimal with year in brackets
def 2010 try.xyz	to	def try (2010).xyz	Text following decimal with no brackets
2009 have 123.12]	to	have (2009)	3 numeric digit whole number of decimal
again 1978 456.xyz	to	again 456 (1978)	No decimal portion. Treat '456' as text

1. RegEx

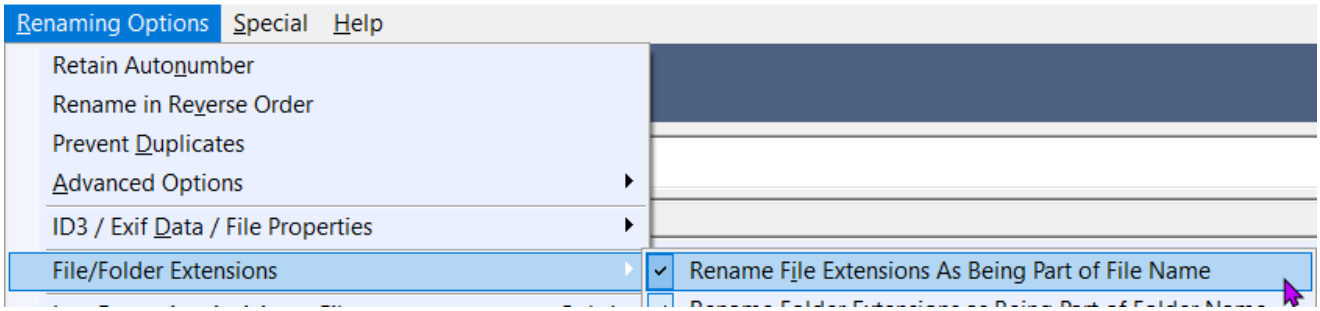
Match: ^(.*?)[[ ]*(19\d\d|20\d\d)(?: \d{1,2}\.\d{1,2} *]*)*([^\.]*)(\.[a-z]{2,3})*
Replace: \1 \3 (\2)\4

	New Name
This a test [1970 1.1].ext	This a test (1970).ext
def 2010 try.xyz	def try (2010).xyz
2009 have 123.12]	have 123 (2009)
abc[1234 6.6].xyz	abc[1234 6.6].xyz
2011 6.6]	(2011)
abc 1990 11.11 .xyz	abc (1990).xyz
abc [2011 7.6] the.xyz	abc the (2011).xyz
abc[2011 6.6] abc.xyz	abc abc (2011).xyz
abc[2011 6.6].xyz	abc (2011).xyz
again 1978 456.xyz	again 456 (1978).xyz

Caveat - use a filter or be careful of file selection because this will inadvertently rename unintended files.
Also a good idea to have 'Prevent Duplicates' enabled as well from the 'Renaming Options Menu'.

Since some of the strings use the <dot> character, you have to enable, 'Rename File Extensions As Being Part of File Name' from the File/ Folder Extensions sub menu of the Renaming Options Menu.

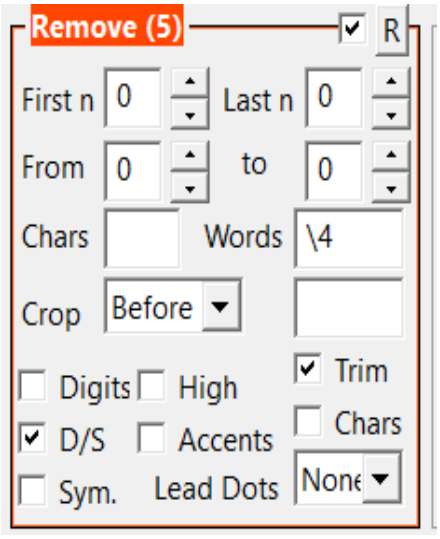
.



String Manipulation (Parsing)

Suffix Year and add parentheses cont.

2. 'Section #5: Remove' –



D/S: Enable
Trim: Enable
Words: \4

Tim Notes:

1. This uses that trick of preventing an Invalid '\4' appearing in New Name for those samples that do not have an extension. If you didn't have, 'Words' remove ' \4 ' –

	New Name
This a test [1970 1.1].ext	This a test (1970).ext
def 2010 try.xyz	def try (2010).xyz
2009 have 123.12]	have 123 (2009)\4
abc[1234 6.6].xyz	abc[1234 6.6].xyz
2011 6.6]	(2011)\4

Sample:

```
Match 1: abc [2011 7.6] the.xyz
Group 1: abc
Group 2: 2011
Group 3:  the
Group 4: .xyz
```

Sample: 2009 have 123.12]

```
                        Start  Length
Match 1: 2009 have 123      0      13
Group 1:                    0       0
Group 2: 2009               0       4
Group 3:  have 123          4       9
Group 4 did not participate in the match
```

Sample:

```
                                    Start  Length
Match 1: This a test [1970 1.1].ext     0      26
Group 1: This a test                    0      11
Group 2: 1970                          13       4
Group 3:                               22       0
Group 4: .ext                          22       4
```

Sample:

```
                        Start  Length
Match 1: abc 1990 11.11     0      14
Group 1: abc                0       3
Group 2: 1990               4       4
Group 3:                   14       0
Group 4 did not participate in the match
```

Where:

Capture Group 1 is 'reserved' for any preceding text otherwise will be null (length = 0 in the sample output above)
Capture Group 2 is 'reserved' for the year
Capture Group 3 is 'reserved' for any following text otherwise will be null (length = 0 in the sample output above)
Capture Group 4 is 'reserved' for any extension, otherwise, Invalid (Group 4 did not participate in the match)

Tim Notes:

1. The Trick referenced above, allows filenames to be renamed even though not all of the backreferences in the Replacement String apply (this has been previously discussed on page 290).

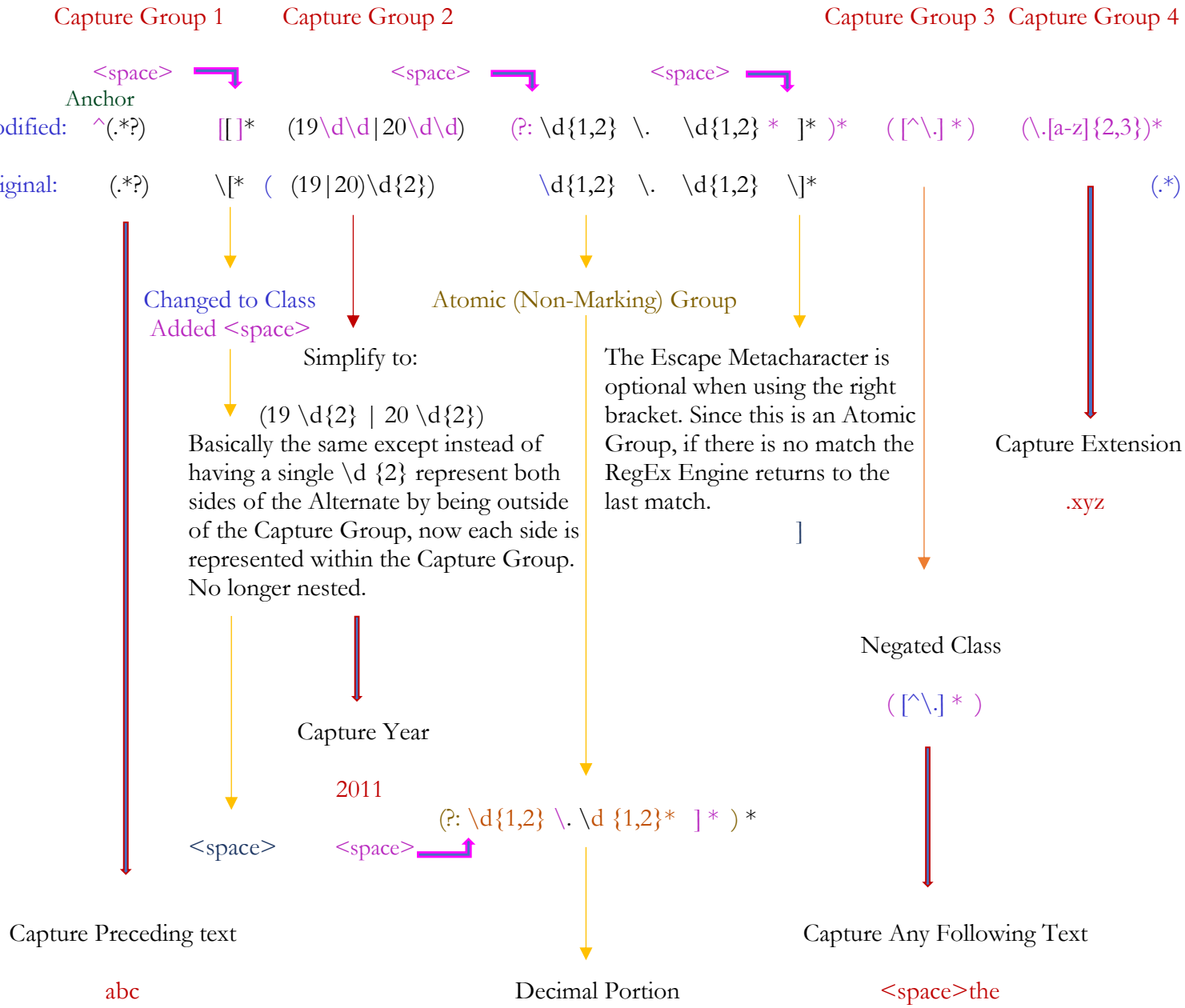
String Manipulation (Parsing)

Suffix Year and add parentheses cont.

Using: abc [2011 7.6] the.xyz

Match: `^(.*?)[[ ]*(19\d\d|20\d\d)(?: \d{1,2}\.\d{1,2} *]*)*([^\.]*)(\.[a-z]{2,3})*`

Compare:



Main Color Key (as it applies to the Compare diagram only):

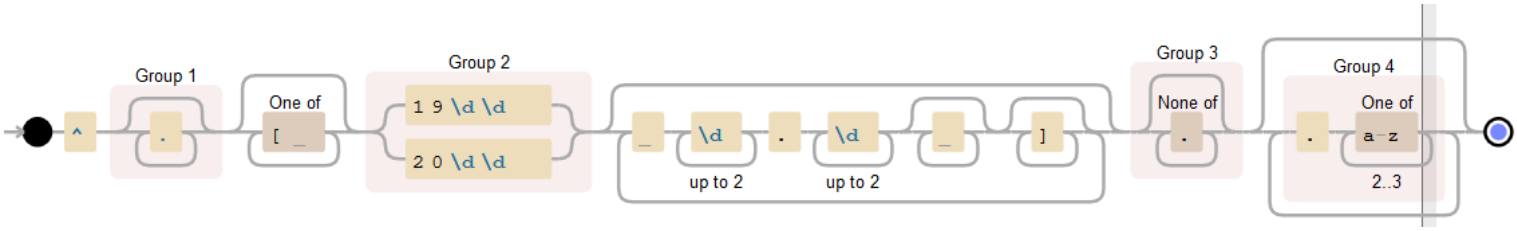
Yellow = Not Captured
Brown = Captured
Changed = Purple
Not Changed = Ocean Blue

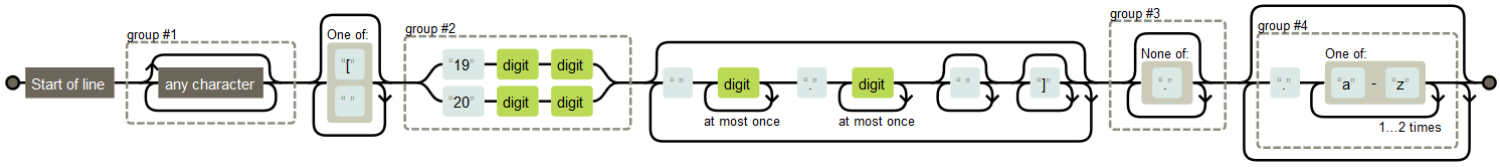
String Manipulation (Parsing)

Suffix Year and add parentheses cont.

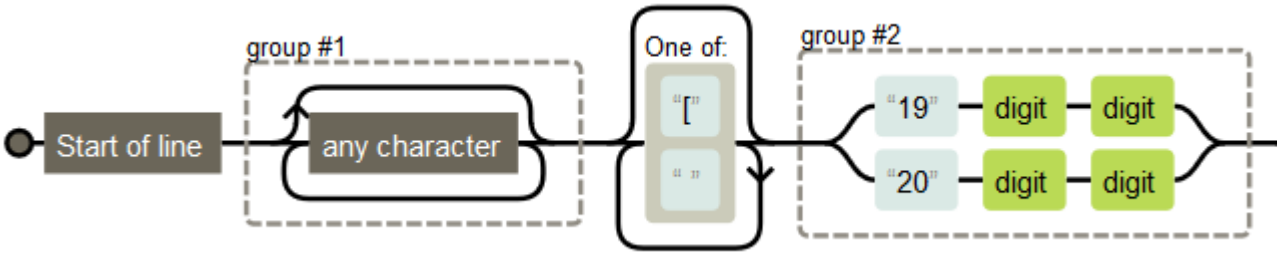
I needed to include those samples that would have text following the numeric portion. cont.

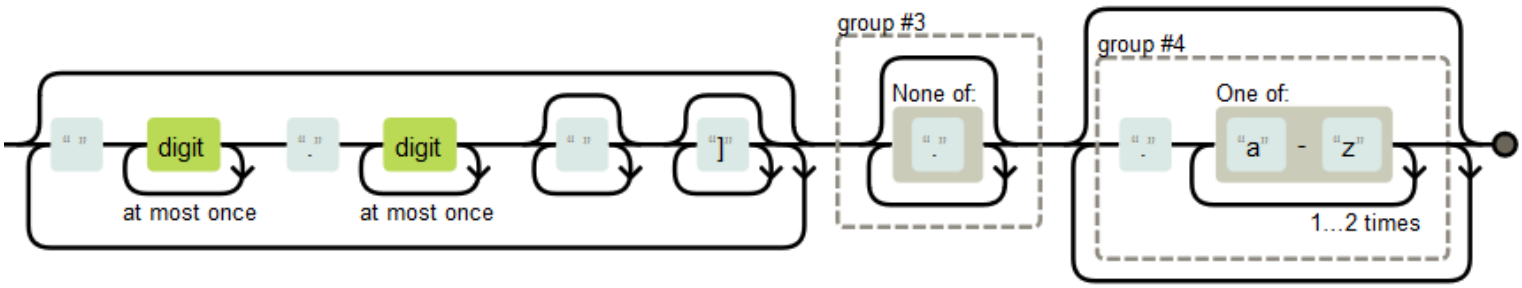
Match: ^(.*?)[[ ]*(19\d\d|20\d\d)(?: \d{1,2}\.\d{1,2} *]*)*([^\.]*)(\.[a-z]{2,3})*





Expanded:





String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Using: abc [2011 7.6] the.xyz

Match: ^(.*?)[[ ]*(19\d\d|20\d\d)(?: \d{1,2}\.\d{1,2} *]*)*([^\.]*)(\.[a-z]{2,3})*

Where:

\1 = abc \2 = 2011 \3 = <space>the \4 = .xyz

This uses an Alternate for Capture Group 2:

This	OR	This
19	|	20

Tim Analysis:

Get any Previous Text

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = a
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [[ ]	Match against a class consisting of a. a left bracket (Not Captured) b. a <space> = <space>	Changes Capture Group 1 = abc
6. *	Make it Greedy Current position = EOL	Changes Capture Group 1 = <entire string>

Tim Notes:

1. Because the 'Ignore File Extension' is enabled through the Rename Options Menu, the EOL is after the 'z' of '.xyz'

Get Year

7. 19|20	Alternate.		
	a. 19	Match against string '19' Backtracks = '20' = False	
	b. \d{2}	Match against 2 numeric digits. This is the decade part of the year.	
	or		
	a. 20	Match against string '20' Backtracks = '20' = True	Capture Group 2 = 2011
	b. \d{2}	Match against 2 numeric digits. This is the decade part of the year.	

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Using: abc [2011 7.6] the.xyz

Match: ^(.*?)[[ ]*(19\d\d|20\d\d)(?: \d{1,2}\.\d{1,2} *]*)*([^\.]*)(\.[a-z]{2,3})*

Where:

\1 = abc \2 = 2011 \3 = <space>the \4 = .xyz

Tim Analysis: cont.

Isolate Decimal Portion in a Non-Marking Group

8. <space>	Match against a <space> character (Not Captured) This is the <space> after '2011'	
9. \d	Match against Numeric Digit (Not Captured) This is the 'whole' number portion of any decimal number = '7'	
10. {1,2}	Specifies a minimum of 1 and maximum of 2 match repetitions. This is to cover the 'whole' number portion of any decimal number from 1 to two numeric digits in length. There is only one, so the minimum iteration of 1 is used.	
11. \.	Match against a literal <dot> character. (Not Captured) This would be the decimal dot.	
12. \d	Match against Numeric Digit (Not Captured) This is the first numeric digit of the decimal portion. = '6'	
13. {1,2}	Specifies a minimum of 1 and maximum of 2 match repetitions. This is to cover the decimal portion from 1 to two numeric digits in length. There is only one, so the minimum iteration of 1 is used.	
14. <space>	Match against a <space> character (Not Captured) This accounts for any <space> after the decimal portion. None in this sample.	Changes all Capture Groups = No Value
15. *	Make it Greedy	All Capture Groups = Restored
16. \]	Match against right bracket (Not Captured)	
17. *	Make it Greedy	All Capture Groups = Unchanged
18. *	Make the Non-Marking Expression Greedy	All Capture Groups = Unchanged

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Using: abc [2011 7.6] the.xyz

Match: ^(.*?)[[ ]*(19\d\d|20\d\d)(?: \d{1,2}\.\d{1,2} *]*)*([^\.]*)(\.[a-z]{2,3})*

Where:

\1 = abc \2 = 2011 \3 = <space>the \4 = .xyz

Tim Analysis: cont.

Capture any following text

19. [^\.]	Match against a Negated Class consisting of a literal <dot>. Match against anything but a literal <dot>	Capture Group 3 = No Value All Other Capture Groups = Unchanged
20. *	Make it Greedy	Capture Group 3 = <space>the All Other Capture Groups = Unchanged

Tim Notes:

1. Because the 'Ignore File Extensions' (previous name for the 'Rename File Extensions As Being Part of File Name') has been enabled through the Rename Options Menu, the <dot> is considered just another piece of text. This tests for any text that may appear after the decimal portion by capturing (the ' * ') up to the point of the extension <dot>.

Capture any extension

21. \.	Match against literal <dot>	Capture Group 4 = .
22. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 4 = .x
23. {2,3}	Specifies a minimum of 2 and maximum of 3 match repetitions. This will allow a file extension from 2 to 3 characters in length.	Capture Group 4 = .xyz
24. *	Make the Expression Greedy	All Capture Groups = Unchanged

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

Expanded Sample Base:

2011 6.6]	to	(2011)	Single bracket following decimal portion
abc 1990 11.11 .xyz	to	abc (1990).xyz	No brackets
abc[2011 6.6].xyz	to	abc (2011).xyz	Decimal portion following year in brackets
abc[1234 6.6].xyz	to	Ignore (1234 is not a year)	
This is a test [1970 1.1].ext	to	This is a test (1970).ext	Text phrase preceding year and decimal portion
abc [2011 7.6] the.xyz	to	abc the (2011).xyz	Text following decimal with year in brackets
def 2010 try.xyz	to	def try (2010).xyz	Text following decimal with no brackets
2009 have 123.12]	to	have (2009)	3 numeric digit whole number of decimal
again 1978 456.xyz	to	again 456 (1978)	No decimal portion. Treat '456' as text

Okay, one more sample added:

2009 have 123 1.12]	to	have 123 (2009)	Numeric text preceding decimal portion

	New Name
2009 have 123 1.12]	have 123 1 (2009)

Oops. That won't do. Also I notice an error in the file, '2009 have 123.12' . The Modified RegEx resulted in 'have 123 (2009)' when it should have been 'have (2009)'. Both of these issues are corrected with the next RegEx.

I needed to include those samples that would have text following the numeric portion. cont.

1. RegEx

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$
Replace: \1 \3 (\2)\4

	New Name
2009 have 123 1.12]	have 123 (2009)
2009 have 123.12]	have (2009)
2009 have 999.12].txt	have (2009).txt
2009 have [99.12].txt	have (2009).txt
2011 6.6]	(2011)
abc 1990 11.11 .xyz	abc (1990).xyz
abc [2011 7.6] the.xyz	abc the (2011).xyz
abc[1234 6.6].xyz	abc[1234 6.6].xyz
abc[2011 6.6] abc.xyz	abc abc (2011).xyz
abc[2011 6.6].txt	abc (2011).txt
abc[2011 6.6].xyz	abc (2011).xyz
abc[[[2011 [[6.6]].txt	abc (2011).txt
again 1978 456.xyz	again 456 (1978).xyz
def 2010 try.xyz	def try (2010).xyz
john 1990 abc.abc.abc.txt	john abc.abc.abc (1990).txt
This a test [1970 1.1].ext	This a test (1970).ext
This is a test [1970 1.1].txt	This is a test (1970).txt
TitleB - January 8, 2019.pdf	TitleB - January 8, (2019).pdf

Caveat –

1. use a filter or be careful of file selection because this will inadvertently rename unintended files.

2. Also a good idea to have 'Prevent Duplicates' enabled as well from the 'Renaming Options Menu'.

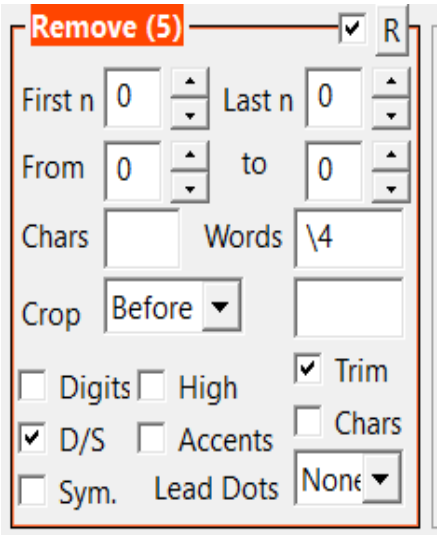
Tim Notes:

1. Sample Group has been expanded even more. See the Expanded Samples Page.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

2. 'Section #5: Remove' –



D/S: Enable
Trim: Enable
Words: \4

Tim Notes:

1. This uses that trick of preventing an Invalid '\4' appearing in New Name for those samples that do not have an extension. If you didn't have 'Words' remove ' \4 ' –

	New Name
2009 have 123 1.12]	have 123 (2009)\4
This a test [1970 1.1].ext	This a test (1970).ext
def 2010 try.xyz	def try (2010).xyz
2009 have 123.12]	have (2009)\4
abc[1234 6.6].xyz	abc[1234 6.6].xyz
2011 6.6]	(2011)\4

Sample:

```
Match 1: abc [2011 7.6] the.xyz
Group 1: abc
Group 2: 2011
Group 3:  the
Group 4: .xyz
```

Sample:

```
                          Start  Length
Match 1: 2009 have 123.12]    0      17
Group 1:                      0       0
Group 2: 2009                 0       4
Group 3:  have                4       5
Group 4 did not participate in the match
```

Sample:

```
                                    Start  Length
Match 1: This a test [1970 1.1].ext     0      26
Group 1: This a test                    0      11
Group 2: 1970                          13       4
Group 3:                               22       0
Group 4: .ext                          22       4
```

Sample:

```
                          Start  Length
Match 1: 2009 have 123 1.12]    0      19
Group 1:                        0       0
Group 2: 2009                   0       4
Group 3:  have 123              4       9
Group 4 did not participate in the match
```

Where:

Capture Group 1 is 'reserved' for any preceding text otherwise will be null (length = 0 in the sample output above)
Capture Group 2 is 'reserved' for the year
Capture Group 3 is 'reserved' for any following text otherwise will be null (length = 0 in the sample output above)
Capture Group 4 is 'reserved' for any extension, otherwise, Invalid (Group 4 did not participate in the match)

Tim Notes:

1. The Trick referenced above, allows filenames to be renamed even though not all of the backreferences in the Replacement String apply (this has been previously discussed on page 290).

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Expanded Samples

Due to the complexity of the RegEx, additional samples were added.

2009 have 123
2011 6
abc[2011 6.6]
again 1978 456
This is a test [1970 1.1]

2009 have 123 1
abc 1990 11.11
abc[2011 6.6]
def 2010 try
TitleB - January 8, 2019

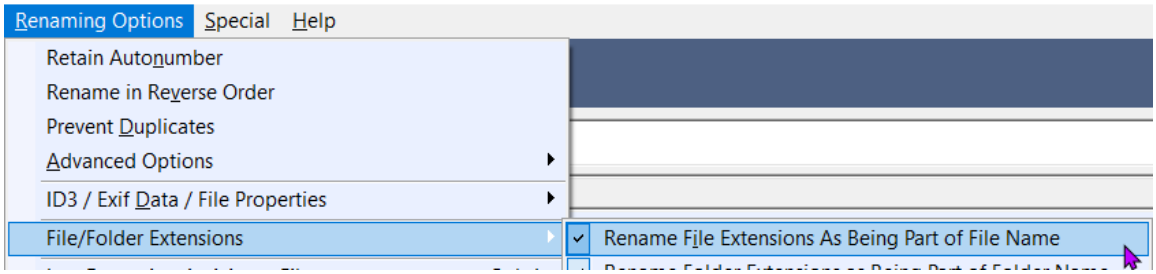
2009 have 999.12]
abc [2011 7.6] the
abc[2011 6.6] abc
john 1990 abc.abc.abc

2009 have [99.12]
abc[1234 6.6]
abc[[[2011 [[6.6]]
This a test [1970 1.1]

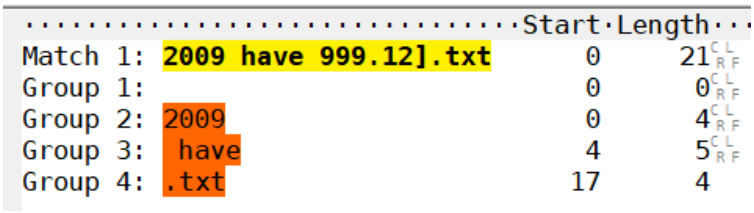
Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

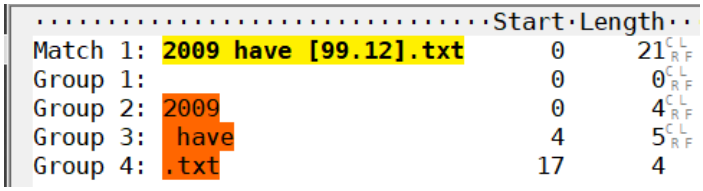
Since some of the strings use the <dot> character, you have to enable, 'Rename File Extensions As Being Part of File Name' from the File/ Folder Extensions sub menu of the Renaming Options Menu.

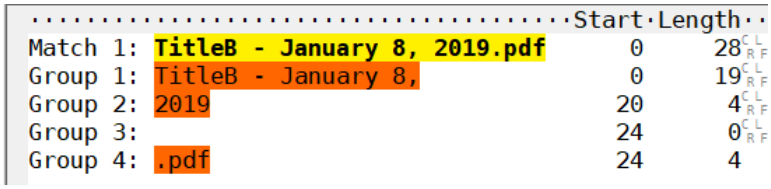
.

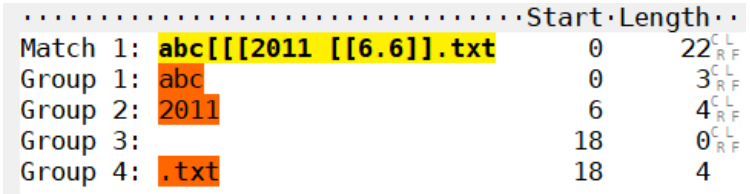


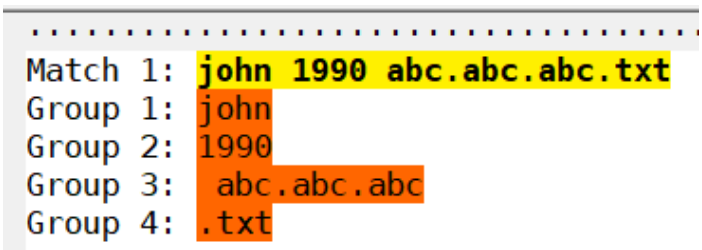
Samples











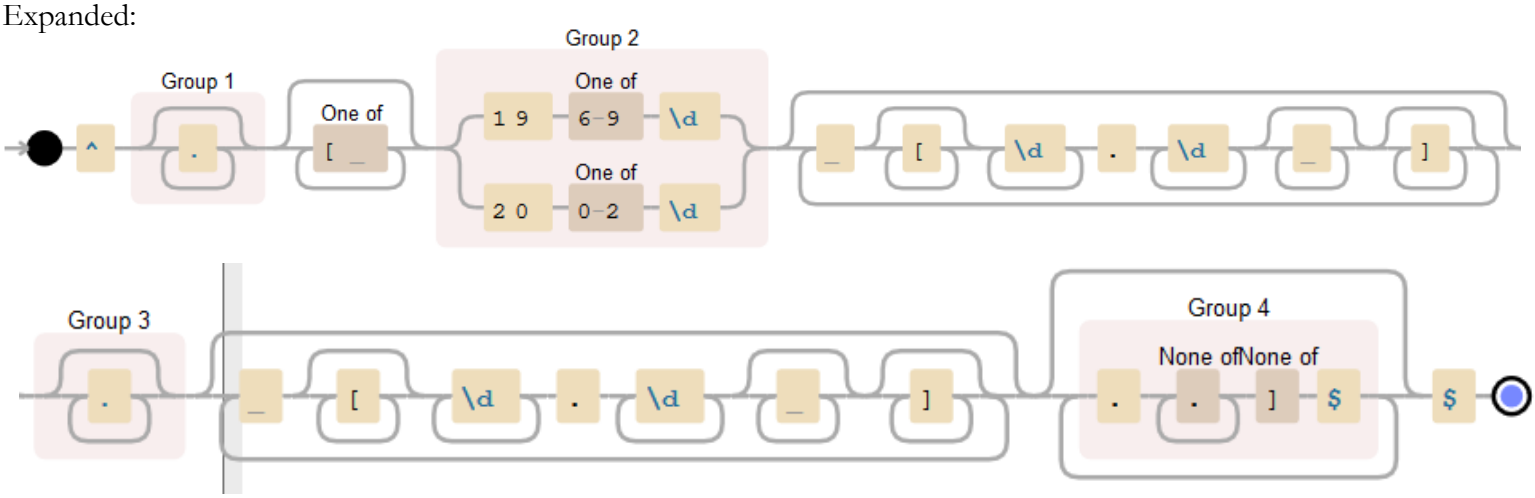
String Manipulation (Parsing)

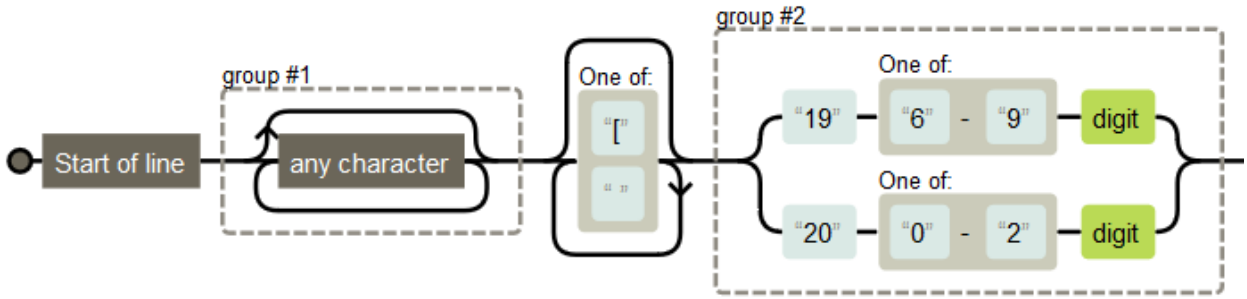
Suffix Year and add parentheses cont.

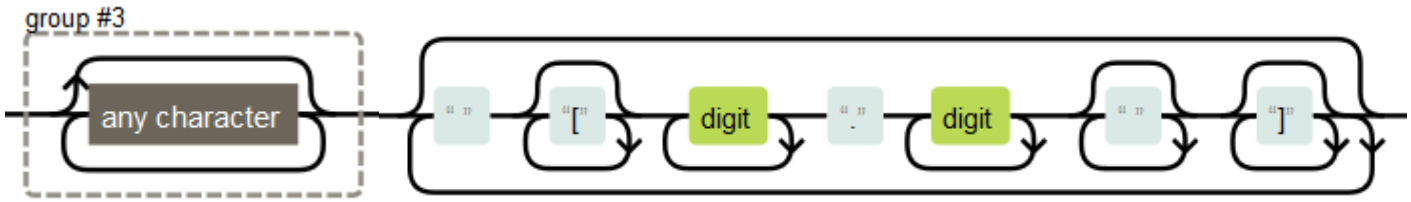
I needed to include those samples that would have text following the numeric portion. cont.

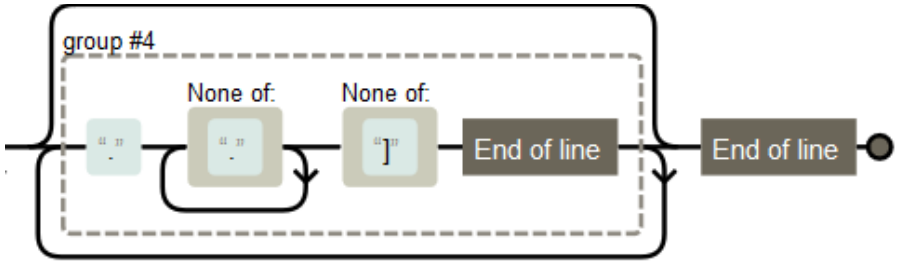
Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Expanded:









String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: abc [2011 7.6] the.xyz

Where:

\1 = abc \2 = 2011 \3 = <space> the \4 = .xyz

Sample

```
.................................
Match 1: abc [2011 7.6] the.xyz
Group 1: abc
Group 2: 2011
Group 3:  the
Group 4: .xyz
```

Breaks down:

Get Preceding text

^ (.*?) → Capture Group 1 → abc [[<space>] *

Get Year

Capture Group 2

This (19[6-9]\d → Alternate or → 2011 → This 20[0-2]\d)

```
   .........10........20........30.......
 Beginning match attempt at character 0
1  ok
2  ok
3  ok
4  ok
5  backtrack
6  backtrack
7  backtrack
8  a
9  a
10 aok
11 abacktrack
12 abacktrack
13 abacktrack
14 ab
15 ab
16 abok
17 abbacktrack
18 abbacktrack
19 abbacktrack
20 abc
21 abc
22 abc [
23 abc [backtrack
24 abc [2
25 abc [20
26 abc [201
27 abc [2011
28 abc [2011
29 abc [2011
30 abc [2011 ok
```

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: abc [2011 7.6] the.xyz

Where:

\1 = abc \2 = 2011 \3 = <space> the \4 = .xyz

Sample

```
.................................
Match 1: abc [2011 7.6] the.xyz
Group 1: abc
Group 2: 2011
Group 3:  the
Group 4: .xyz
```

```
31 abc [2011 7
32 abc [2011 7.
33 abc [2011 7.6
34 abc [2011 7.6ok
35 abc [2011 7.6]
36 abc [2011 7.6]
37 abc [2011 7.6]
38 abc [2011 7.6] ok
39 abc [2011 7.6] backtrack
40 abc [2011 7.6]ok
41 abc [2011 7.6]ok
```

Breaks down: cont.

Get Decimal Portion

Atomic Group
Non-Marking Group

(?:	<space>	→	\[*	→	\d+	\.	\d+	→	<space> *	→	]*	)*
	<space>		No Match		7	.	6		No Match		]	

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: abc [2011 7.6] the.xyz

Where:

\1 = abc \2 = 2011 \3 = <space> the \4 = .xyz

Sample

```
.................................
Match 1: abc [2011 7.6] the.xyz
Group 1: abc
Group 2: 2011
Group 3:  the
Group 4: .xyz
```

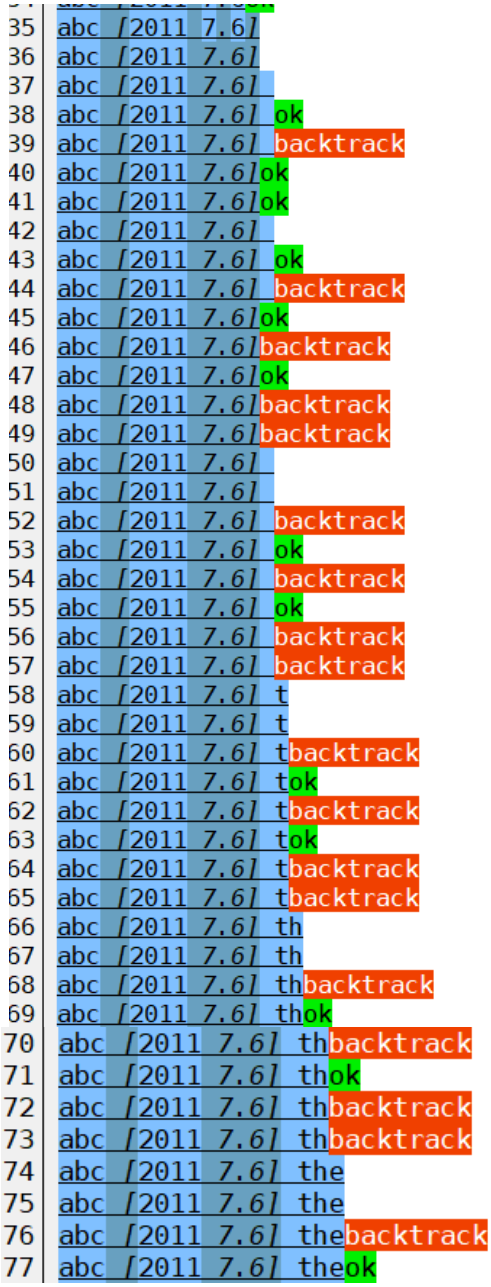
Breaks down: cont.

Get any Following Text

Capture Group 3

(.*? ⟶)

<space> the



String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: abc [2011 7.6] the.xyz

Where:

\1 = abc \2 = 2011 \3 = <space> the \4 = .xyz

Sample

```
.................................
Match 1: abc [2011 7.6] the.xyz
Group 1: abc
Group 2: 2011
Group 3:  the
Group 4: .xyz
```

Breaks down: cont.

Get any following numeric or decimal portion

Atomic Group
Non-Marking Group

(?: <space> → \[* → \d+ \. \d+ → <space> * →]*)*
No Match No Match No Match No Match No Match

Get Extension

Capture Group 4

(\. → [^\.]+ → [^\]] → $)*

. xyz No Match

$

Forum Contributor: RegexNinja

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

This uses an Alternate for Capture Group 2:

This	OR	This
19[6-9]\d	|	20[0-2]\d)

Tim Analysis:

Get any Previous Text

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = j
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [[ ]	Match against a class consisting of (Not Captured) a. a left bracket b. a <space> = <space>	Changes Capture Group 1 = john
6. *	Make it Greedy Current position = EOL	Changes Capture Group 1 = <entire string>

Tim Notes:

1. Because the 'Ignore File Extension' is enabled through the Rename Options Menu, the EOL is after the last 't' of '.txt'

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Get Year

19[6-9]	\d		20[0-2]	\d	
196	0-9	- covers years 1960 – 1969	20	0-9	- covers years 2000 – 2009
197	0-9	- covers years 1970 – 1979	21	0-9	- covers years 2010 – 2019
198	0-9	- covers years 1980 – 1989	22	0-9	- covers years 2020 – 2029
199	0-9	- covers years 1990 – 1999			

7. 19[6-9]\d|20[0-2]\d

Alternate.

a. 19	Match against string '19' Backtracks = '19' = True This is the century.	Capture Group 2 = 19
b. [6-9]	Match against a class consisting of a numeric digit in the range of '6' to '9' This is the decade.	Capture Group 2 = 199
c. \d	Match against a numeric digit. This is the individual year of the decade.	Capture Group 2 = 1990

or

a. 20	Match against string '20' Backtracks = '19' = False This is the century.	
b. [0-2]	Match against a class consisting of a numeric digit in the range of '0' to '2' This is the decade.	
c. \d	Match against a numeric digit. This is the individual year of the decade.	

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Isolate Decimal Portion in a Non-Marking Group

8. <space>	Match against a <space> character (Not Captured) This is the <space> after '1990'	
9. \[	Match against left bracket (Not Captured) = No Value There are no brackets in the sample.	
10. *	Make it Greedy	All Capture Groups = Unchanged
11. \d	Match against Numeric Digit (Not Captured) = No Value There are no numeric digits remaining in the sample.	
12. +	Make it Greedy	All Capture Groups = No Value
13. \.	Match against a literal <dot> character. (Not Captured) = No Value. There is not a literal dot following any numeric digits in this sample	
14. \d	Match against Numeric Digit (Not Captured) = No Value There are no numeric digits remaining in the sample.	
15. +	Make it Greedy	All Capture Groups = No Value
16. <space>	Match against a <space> character (Not Captured) = No Value. There is not a <space> character following any numeric digits after '1990'	
17. *	Make it Greedy	All Capture Groups = No Value
18. \]	Match against right bracket (Not Captured) = No Value There are no brackets in the sample.	
19. *	Make it Greedy	All Capture Groups = No value
20. *	Make the Non-Marking Expression Greedy Current position is just after the '0' of '1990'	All Capture Groups = Restored

Tim Notes:

1. The purpose of this is to identify any strings that have a numeric decimal portion immediately following the Year. This sample does not, so it doesn't match. Because it is Atomic, it either matches or not. It doesn't backtrack. At the end when the entire sub-expression Fails, the RegEx Engine returns to the last match.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Isolate Decimal Portion in a Non-Marking Group cont.

Tim Notes:

1. This whole section was to isolate any decimal portion. In this sample there isn't any, so there was no match in most of the sub-expressions resulting in No Value. At the end, it reached the EOL at step 19, but at step 20, by making the entire expression Greedy (the wildcard is outside of the Non-Marking Group), it returned back to the last match, the '0' of '1990'. (step 7)

   This is because the Non-Marking Group is Atomic. An Atomic Group does not backtrack – it either matches or it doesn't as it moves forward evaluating the sub-expressions within. Because there was no match, the entire Group failed to match. This caused the RegEx Engine to return to the previous match before the Non-Marking Group.

   This reflects the RegEx up to the current section only.

```
^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*
```

```
john 1990 abc.abc.abc.txt
```

```
                    Start  Length
Match 1: john 1990      0       9
Group 1: john           0       4
Group 2: 1990           5       4
Group 3 did not participate in the match
```

```
                .........10........20.....
1  15 | jo
1  16 | jook
1  17 | jobacktrack
1  18 | jobacktrack
1  19 | jobacktrack
1  20 | joh
1  21 | joh
1  22 | johok
1  23 | johbacktrack
1  24 | johbacktrack
1  25 | johbacktrack
1  26 | john
1  27 | john
1  28 | john
1  29 | john 1
1  30 | john 19
1  31 | john 199
1  32 | john 1990
1  33 | john 1990
1  34 | john 1990
1  35 | john 1990 ok
1  36 | john 1990 backtrack
1  37 | john 1990backtrack
1  38 | john 1990ok
1     | Match found in 38 steps
```

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Get any following Text

21..	Match against any character This is the <space> after '1990	Capture Group 3 = <space>
22. *	Make it Greedy	Capture Group 3 = <space> abc.abc.abc
23. ?	But not too Greedy Just after the '0' of '1990'	Capture Group 3 = null

Isolate a Second Decimal Portion in a Non-Marking Group

24. <space>	Match against a <space> character (Not Captured) This is the <space> after '1990'	
25. \[	Match against left bracket (Not Captured) = No Value There are no brackets in the sample.	
26. *	Make it Greedy	All Capture Groups = Unchanged
27. \d	Match against Numeric Digit (Not Captured) = No Value There are no numeric digits remaining in the sample.	
28. +	Make it Greedy	All Capture Groups = No Value
29. \.	Match against a literal <dot> character. (Not Captured) = No Value. There is not a literal dot following any numeric digits in this sample	
30. \d	Match against Numeric Digit (Not Captured) – No Value There are no numeric digits remaining in the sample.	
31. +	Make it Greedy	All Capture Groups = No Value
32. <space>	Match against a <space> character (Not Captured) = No Value. There is not a <space> character following any numeric digits after '1990'	
33. *	Make it Greedy	All Capture Groups = No Value
34. \]	Match against right bracket (Not Captured) = No Value There are no brackets in the sample.	
35. *	Make it Greedy	All Capture Groups = No value
36. *	Make the Non-Marking Expression Greedy Current position is just after the '0' of '1990'	All Capture Groups = Restored

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Isolate Second Decimal Portion in a Non-Marking Group cont.

Tim Notes:

1. Déjà vu. This is for those strings that may have an additional <dot> numerals– it could be another decimal number, or part of an extension.

2. Same as before. Evaluates same because in this sample, there are no numeric digits followed by a literal <dot>character.

3. Atomic Group same as before. It either matches or it doesn't. It didn't, so it returns back to previous match – current position remains just after the '0' of '1990' (step 23).

4. In addition to isolating any second decimal portion of a sample, this also serves to isolate any numeric portion that follows any text following the year. This sample does not qualify in either respect.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Capture any Extension

37. \.	Match against literal <dot> This is the <dot> after the first string of 'abc.'	Capture Group 4 = . Changes Capture Group 3 = <space> abc
38. [^\.]	Match against a Negated Class consisting of literal <dot>. Match against anything but a literal <dot>. This is the '.a' of the second string of 'abc.abc'.	Capture Group 4 = .a
39. +	Make it Greedy	Capture Group 4 = .abc Capture Group 3 = Unchanged
40. [^\]]	Match against a Negated Class consisting of a literal right bracket (also called a 'closed' bracket). Match against anything but a literal right bracket. This is the <dot> following the second string of 'abc.abc.'	Capture Group 4 = .abc.
41. $	Must be at EOL	Capture Group 4 = .txt Changes Capture Group 3 = <space> abc.abc.abc All other Capture Groups = Unchanged

Tim Notes:

1. This means that the last match must take place from the end of the string. so the [^\]] matches against the '.txt' instead of the previous value of '.abc.'. This in turn changes Capture Group 3.

42. *	Make the entire Expression Greedy Returns to last match, just after the '0' of '1990'. Because it returns to last match, Capture Group 3 results in a Zero Occurrence Match	Capture Group 4 = No Value Changes Capture Group 3 = null
43. $	Must be at EOL	All Capture Groups = Restored

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$
Replace: \1 \3 (\2)\4

Using: john 1990 abc.abc.abc.txt

Where:

\1 = john \2 = 1990 \3 = <space> abc.abc.abc \4 = .txt

Tim Analysis: cont.

Capture any Extension cont.

Tim Notes:

1. Is the final *$ redundant?

Apparently not. If you take away these two Metacharacters from the end of the RegEx,

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)

it results in this:

	New Name
2009 have 123 1.12]	2009 have 123 1.12]
2009 have 123.12]	2009 have 123.12]
2009 have 999.12].txt	have (2009).txt
2009 have [99.12].txt	have (2009).txt
2011 6.6]	2011 6.6]
abc 1990 11.11 .xyz	abc (1990).xyz
abc [2011 7.6] the.xyz	abc the (2011).xyz
abc[1234 6.6].xyz	abc[1234 6.6].xyz
abc[2011 6.6] abc.xyz	abc abc (2011).xyz
abc[2011 6.6].txt	abc (2011).txt
abc[2011 6.6].xyz	abc (2011).xyz
abc[[[2011 [[6.6]].txt	abc (2011).txt
again 1978 456.xyz	again 456 (1978).xyz
def 2010 try.xyz	def try (2010).xyz
john 1990 abc.abc.abc.txt	john abc.abc.abc (1990).txt
This a test [1970 1.1].ext	This a test (1970).ext
This is a test [1970 1.1].txt	This is a test (1970).txt
TitleB - January 8, 2019.pdf	TitleB - January 8, (2019).pdf

Three samples that previously matched do not match. let's find out why.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

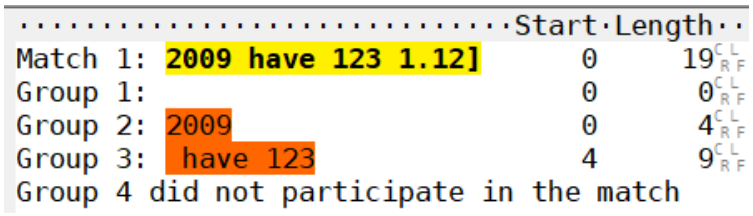
I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value



This uses an Alternate for Capture Group 2:

This	OR	This
19[6-9]\d	|	20[0-2]\d)

Tim Analysis:

Get any Previous Text

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = 2
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [[ ]	Match against a class consisting of (Not Captured) a. a left bracket b. a <space> = <space> This is the <space after '2009'	Changes Capture Group 1 = 2009
6. *	Make it Greedy Current position = EOL. Gives up the value of the previous <space> (Not Captured)	Changes Capture Group 1 = <entire string>

Tim Notes:

1. Because the 'Ignore File Extension' is enabled through the Rename Options Menu, the EOL is after the ']'

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value

Tim Analysis: cont.

Get Year

19[6-9]	\d		20[0-2]	\d	
196	0-9	- covers years 1960 – 1969	20	0-9	- covers years 2000 – 2009
197	0-9	- covers years 1970 – 1979	21	0-9	- covers years 2010 – 2019
198	0-9	- covers years 1980 – 1989	22	0-9	- covers years 2020 – 2029
199	0-9	- covers years 1990 – 1999			

7. 19[6-9]\d|20[0-2]\d

Alternate.

a. 19	Match against string '19' Backtracks = '20' = False This is the century.	
b. [6-9]	Match against a class consisting of a numeric digit in the range of '6' to '9' This is the decade.	
c. \d	Match against a numeric digit. This is the individual year of the decade.	

or

a. 20	Match against string '20' Backtracks = '20' = True This is the century.	Capture Group 2 = 20 Changes Capture Group 1 = null
b. [0-2]	Match against a class consisting of a numeric digit in the range of '0' to '2' This is the decade.	Capture Group 2 = 200
c. \d	Match against a numeric digit. This is the individual year of the decade.	Capture Group 2 = 2009

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value

Tim Analysis: cont.

Isolate Decimal Portion in a Non-Marking Group

8. <space>	Match against a <space> character (Not Captured) This is the <space> after '2009'	
9. \[	Match against left bracket (Not Captured) = False There is no left bracket in the sample.	All Capture Groups = No Value
10. *	Make it Greedy	All Capture Groups = Restored
11. \d	Match against Numeric Digit (Not Captured) = 'h' = 'False'	All Capture Groups = No Value
12. +	Make it Greedy	All Capture Groups = No Value
13. \.	Match against a literal <dot> character. (Not Captured) = 'h' = 'False'	All Capture Groups = No Value
14. \d	Match against Numeric Digit (Not Captured) = 'h' = 'False'	All Capture Groups = No Value
15. +	Make it Greedy	All Capture Groups = No Value
16. <space>	Match against a <space> character (Not Captured) = 'h' = 'False'	All Capture Groups = No Value
17. *	Make it Greedy	All Capture Groups = No Value
18. \]	Match against right bracket (Not Captured) = 'h' = 'False'	All Capture Groups = No Value
19. *	Make it Greedy	All Capture Groups = No Value
20. *	Make the Non-Marking Expression Greedy	All Capture Groups = Restored

Tim Notes:

1. The purpose of this is to identify any strings that have a numeric decimal portion immediately following the Year. This sample does not. It has the string, 'have', so it doesn't match. Because it is Atomic, it either matches or not. It doesn't backtrack. So at the end when the entire sub-expression Fails, the RegEx Engine returns to the last match. At the end, it reached the EOL at step 19, but at step 20, by making the entire expression Greedy (the wildcard is outside of the Non-marking Group), it returned back to the last match, the '9' of '2009'. (step 7)

   This, again, is because the Non-Marking Group is Atomic. An Atomic Group does not backtrack – it either matches or it doesn't as it moves forward evaluating the sub-expressions within. Because there was no match, the entire Group failed to match. This caused the RegEx Engine to return to the previous match before the Non-Marking Group.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value

Tim Analysis: cont.

Get any following Text

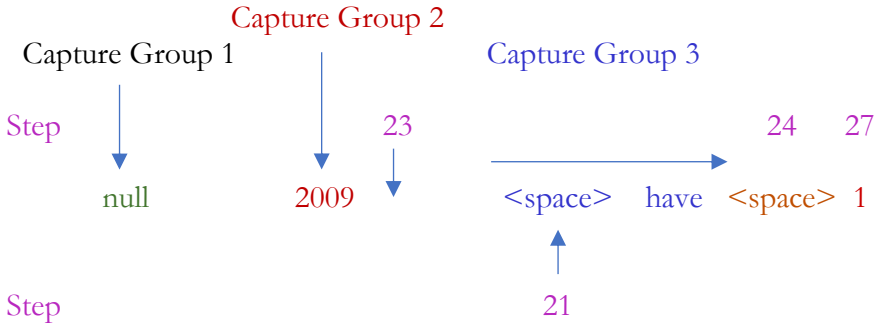
21..	Match against any character This is the <space> following '2009'	Capture Group 3 = <space>
22. *	Make it Greedy	Capture Group 3 = <space> have 123 1.12]
23. ?	But not too Greedy Current position = just before the <space> after '2009'	Capture Group 3 = null

Isolate a Second Decimal Portion in a Non-Marking Group

24. <space>	Match against a <space> character (Not Captured) = 'h' = False	All Capture Groups = No Value
25. \[	Match against left bracket (Not Captured) = False there is no left bracket in sample	All Capture Groups = No Value
26. *	Make it Greedy	All Capture Groups = No Value
27. \d	Match against Numeric Digit (Not Captured) = '<space>1'	Changes Capture Group 3 = <space> have All Other Capture Groups = Restored

Tim Notes:

1. Matches against the '1' but restores the value of the <space> in Capture Group 3 from step 21 and 22. This in turn includes all characters inclusive and leading up to the '1'. Capture Group 3's value changes to reflect this = '<space> have'. The <space> of step 24 (Not Captured) that didn't match previously now matches against the <space> after 'have',



String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value

Tim Analysis: cont.

Isolate a Second Decimal Portion in a Non-Marking Group cont.

28. +	Make it Greedy = '<space> 123' (Not Captured)	All Capture Groups = Unchanged
29. \.	Match against a literal <dot> character. (Not Captured) This matched against the <dot> after '1' of '1.12'. Not Captured = '<space>.1'	Changes Capture Group 3 = <space> have 123

Tim Notes:

1. Matches against the <dot> of '1.12' Changes Capture Group 3 to include the preceding characters '<space> have'. This in turn changes the Not Captured value from '<space> 123' step 28 to the <space> after '123' and including the '.1' just matched.

30. \d	Match against Numeric Digit (Not Captured) This matched against the second '1' of '1.12'. Not Captured = '<space>.1.1'	All capture Groups = Unchanged
31. +	Make it Greedy This matched against the '2' of '1.12'. Not Captured = '<space>1.12'	All capture Groups = Unchanged
32. <space>	Match against a <space> character (Not Captured) No Match. There is not a <space> character left in the string.	All Capture Groups = No Value
33. *	Make it Greedy Restores Not Captured = '<space>1.12'	All Capture Groups = Restored
34. \]	Match against right bracket (Not Captured) Matches against ']'. Not Captured = '<space> 1.12]'	All Capture Groups = Unchanged
35. *	Make it Greedy	All Capture Groups = Unchanged
36. *	Make the Non-Marking Expression Greedy Capture Group 3 value is null because it returns to the position just before the <space> after '2009' (prior to the Non-Marking Group – step 22). Not Captured loses all of its values.	Capture Groups 1,2 = Unchanged Capture Group 3 = null

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value

Tim Analysis: cont.

Capture any Extension

37. \.	Match against literal <dot> This is the <dot> of 1.12]	Capture Group 4 = . Changes Capture Group 3 = <space> have 123 1 Capture Group 2 = Restored = 2009 Capture Group 1 = Restored = null
38. [^\.]	Match against a Negated Class consisting of literal <dot>. Match against anything but a literal <dot>.	Capture Group 4 = .1 All other Capture Groups = Unchanged
39. +	Make it Greedy Current position = EOL	Capture Group 4 = .12] All other Capture Groups = Unchanged
40. [^\]]	Match against a Negated Class consisting of a literal right bracket (also called a 'closed' bracket). Match against anything but a literal right bracket. Current Position = the '2' of '1.12'	Capture Group 4 = .12 All other Capture Groups = Unchanged
41. $	Must be at EOL Not at EOL – No Match.	All Capture Groups = No Value
42. *	Make the entire Expression Greedy Changes Capture Group 3 to null, because it returns the current position to just before the <space> after '2009'. This removes Capture Group 4's match as if it never happened making it Invalid.	Capture Group 4 = Invalid Changes Capture Group 3 = null Capture Group 2,1 = Unchanged
43. $	Must be at EOL This also restores the Not Captured = '<space> 1.12]'. This in turn changes Capture Group 3 to '<space> have 123'	Capture Group 4 = Invalid Changes Capture Group 3 = <space> have 123 Capture Group 2,1 = Unchanged

Tim Notes:

1. This is kind of strange. The nearest I can come to a conclusion is because Capture Group 4 is Invalid, it removes all of the sub-expression from the evaluation ' (\.[^\.]+[^\]]$)* '. This is the same as reverting everything back to step 29. All of this is supported by the fact that these values from step 29 match up to the final values. Not Captured is '<space> 1.12]' , and Capture Group 3 in turn changes to '<space> have 123'. Capture Groups 1 and 2 are unaffected.

String Manipulation (Parsing)

Suffix Year and add parentheses cont.

I needed to include those samples that would have text following the numeric portion. cont.

Match: ^(.*?)[[ ]*(19[6-9]\d|20[0-2]\d)(?: \[*\d+\.\d+ *]*)*(.*?)(?: \[*\d+\.\d+ *]*)*(\.[^\.]+[^\]]$)*$

Using: 2009 have 123 1.12]

Where:

\1 = null \2 = 2009 \3 = <space> have 123 \4 = No Value

Tim Conclusion:

In answer to the question posed on the last example:

Why isn't the ' * $ ' redundant that follows the sub-expression, ' [^\]] $) ' ?

1. It is the EOL Anchor ' $ ' in the sub-expression, ' (\.[^\.]+[^\]]$) ' that is redundant and can be removed. The RegEx should be: ' (\.[^\.]+[^\]])*$ '.

2. This was a very complex RegEx and I feel I may have failed in fully understanding why it works, but nevertheless, as I explained at the beginning, the values are accurate even if my analysis is a bit off. You are encouraged to do your own self analysis to examine where I may be wrong.

3. If Capture Group 4 is 'reserved' for those strings that have extensions, but yet I set BRU to ignore file extensions, then how does this work? It relies on the last <dot> character in the string to isolate and capture the value for Capture Group 4. It captures from the <dot> character to the EOL.

   If this is true, then why, in the current example, did Capture Group 4's match become invalid? Because of the negated match for the right bracket. The right bracket for some samples falls just before the <dot> character that would signify the file extension, but in the sample, ' 2009 have 123 1.12] ', the right bracket is actually a part of the file extension.

   Therefore when the negated match encounters the right bracket it changes Capture Group 4's value to '1.12' instead of the previous '1.12]'. Next, when it tests for EOL using ' $ ', it Fails, invalidating Capture Group 4 and reverting everything back to step 29 before the sub-expression, ' (\.[^\.]+[^\]]$)* ' , of Capture Group 4 was evaluated.

4. 'Section #5: Remove', cleans up any excess <spaces> within the string as well as any leading or trailing <spaces>. It also removes any 'Invalid' backreference to Capture Group 4 that would otherwise prevent the renaming process from being performed for any affected samples.

Find Occurrences

Find Occurrences

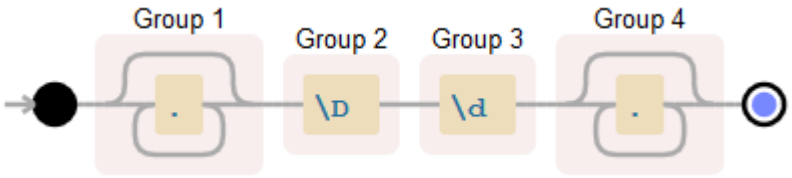
Find any First Occurrence of a Single Digit Number in a filename and add a preceding zero

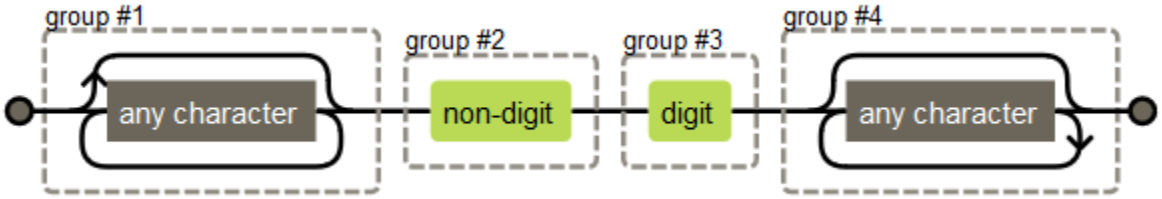
Lesson 1 to Lesson 01
Lesson 1 DNA to Lesson 01 DNA

My Contribution:–

Match: (.*?)(\D)(\d)(.*)
Replace: \1\20\3\4

Name	New Name
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA





Sample:

```
.......................
Match 1: Lesson 1 DNA
Group 1: Lesson
Group 2:
Group 3: 1
Group 4:  DNA
```

Using: Lesson 1 DNA

Where:
\1 = Lesson \2 = <space> \3 = 1 \4 = <space> DNA

Tim Analysis:

1. (.*?) Doesn't work with 1,2, 10, etc. Text, followed by a <space>, must precede the numeric digit (.*?)
2. \D Matches against <space> using a Non-Numeric Digit metacharacter.
3. \d Matches against the numeric digit.
4. .* Matches against remainder of string

Find Occurrences

Pad with Zero at Beginning, Middle and End of String

Pad Zeros in Middle of String -

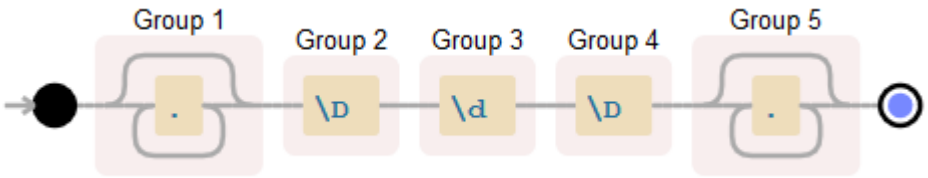
To find single numeric digits in the middle of a filename regardless of what comes before or after, and add a leading zero

String = 0 123 2 abc 3 456 7.txt

Middle: Isolate numeric digit then pad with zero in Replacement String.

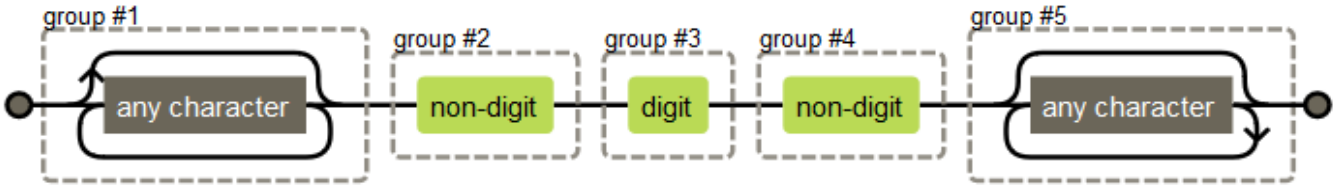
Match: (.*?)(\D)(\d)(\D)(.*)
Replace: \1\20\3\4\5

Name	New Name
0 123 2 abc 3 456 7.txt	0 123 02 abc 3 456 7.txt



Tim Notes:

1. I recognize this from before, if you recall the discussion of 'Binding' numeric digits using Non-Numeric Digits Boundaries, \D.



Sample:

```
                              Start  Length
Match 1: 0 123 2 abc 3 456 7     0      19
Group 1: 0 123                   0       5
Group 2:                         5       1
Group 3: 2                       6       1
Group 4:                         7       1
Group 5: abc 3 456 7             8      11
```

If you wish to pad 1st numeric then change Replacement String to: 0\1\20\3\4\5

Tim Notes:

1. See 'Demonstration of 3 Methods (complex to Simple) to Rearrange string' for the reference to 'binding'
2. You will have to remove the first zero, '0\1', on subsequent runs, otherwise it will just keep padding.

Find Occurrences

Pad with Zero at Beginning, Middle and End of String cont.

Pad Zeros in Middle of String – cont.

Using: 0 123 2 abc 3 456 7.txt

Match: (.*?)(\D)(\d)(\D)(.*)

Where:

\1 = 0 123 \2 = <space> \3 = 2 \4 = <space> \5 = abc 3 456 7

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy (Zero Occurrence Match)	Capture Group 1 = null
4. \D	Match against Non-Numeric Digit	Capture Group 2 = <space>
	Position = <space> after '0'	Changes Capture Group 1 = 0
5. \d	Matches against numeric digit	Capture Group 3 = 1
	This is the '1' of '123'	
6. \D	Match against Non-Numeric Digit	Capture Group 4 = <space>
	Position = '2' following '123'<space>	Changes Capture Group 1 = 0 123
		Capture Group 2 = Unchanged
		Changes Capture Group 3 = 2
7. .*	Match against remainder of string	Capture Group 5 = abc 3 456 7

Find Occurrences

Pad with Zero at Beginning, Middle and End of String cont.

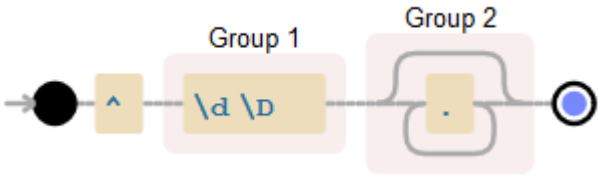
Pad Zeros at Beginning of String -

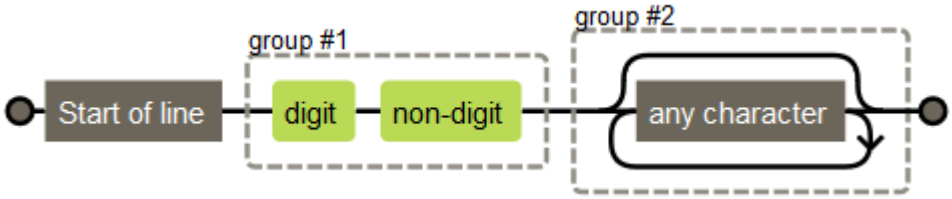
Using: 0 123 2 abc 3 456 7.txt

First: Isolate numeric digit at beginning of string then pad with zero in Replacement String.

Match: ^(\d\D)(.*)
Replace: 0\1\2

Name	New Name
0 123 2 abc 3 456 7.TXT	00 123 2 abc 3 456 7.TXT





Sample:

```
..............................
Match 1: 0 123 2 abc 3 456 7
Group 1: 0
Group 2: 123 2 abc 3 456 7
```

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. \d	Match against first numeric digit from the beginning of string	Capture Group 1 = 0
3. \D	Match against Non-Numeric Digit Position = <space> after '0'	Changes Capture Group 1 = <space>
4. .*	Match against remainder of string	Capture Group 2 = 123 2 abc 3 456 7

Find Occurrences

Pad with Zero at Beginning, Middle and End of String cont.

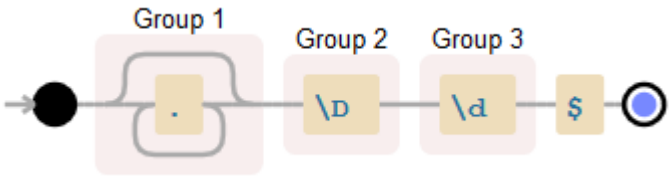
Pad Zeros at End of String -

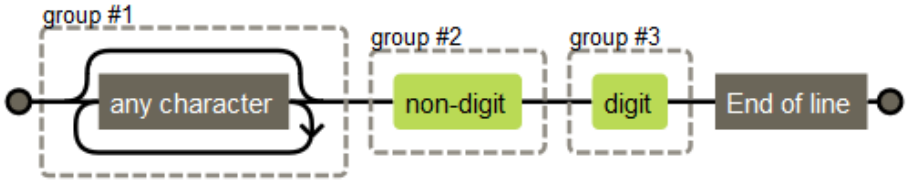
Using: 0 123 2 abc 3 456 7.txt

End: Isolate numeric digit at end of string then pad with zero in Replacement String.

Match: (.*)(\D)(\d)$
Replace: \1\20\3

Name	New Name
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 07.txt





Sample:

```
Match 1: 0 123 2 abc 3 456 7
Group 1: 0 123 2 abc 3 456
Group 2:
Group 3: 7
```

Under Regex Buddy for this test, had to remove anchor $ because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.zip' is just another part of the string of text.

Tim Analysis:

1. .*	Match against entire string This is done to move the RegEx Engine to EOL	Capture Group 1 = <entire string.
2. \D	Match against Non-Numeric Digit Position = <space> before '7' Backtracked	Capture Group 2 = <space> Changes Capture Group 1 = 123 2 abc 3 456
3. \d	Match against numeric digit Moving forward. Current Position = EOL	Capture Group 3 = 7
4. $	Must be at EOL	

Forum Contributors: Saintoctopus, therube

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1

Example #1

String = 0 123 2 abc 3 456 7.txt

Match: (.*?)(\D)(\d)(\D)(.*)
Replace: 0\1\20\3\4\5

1st Time run:

Name	New Name
0 123 2 abc 3 456 7.txt	00 123 02 abc 3 456 7.txt

The Replacement String pads the zero onto the first zero using 0\1. If '00' is not desired then just remove it.
The Replacment String pads the zero onto the '2' using 0\3.

Where:

\1 = 0 123 \2 = <space> \3 = 2 \4 = <space> \5 = abc 3 456 7

2nd Time run:

Use this Replacement String: \1\20\3\4\5 - remove the unnecessary extra pad for the first zero

Name	New Name
00 123 02 abc 3 456 7.txt	00 123 02 abc 03 456 7.txt

Where:

\1 = 00 123 02 abc \2 = <space> \3 = 3 \4 = <space> \5 = 456 7

RegEx is finished (exhausted) – cannot find any more matches:

Name	New Name
00 123 02 abc 03 456 7.TXT	00 123 02 abc 03 456 7.TXT

.. But not complete – the final single numeric digit, '7' remains.

Tim Notes:

1. This RegEx is the 'Middle' taken from 'Pad with Zero at Beginning, Middle and End of String'

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1 cont.

Example #2

String = 0 123 2 abc 3 456 7.txt

Match: (.*)(\D)(\d)$
Replace: \1\20\3

3rd Time run: Replace RegEx with the above

Name	New Name
00 123 02 abc 3 456 7.TXT	00 123 02 abc 03 456 7.TXT

Where:

\1 = 00 123 02 abc 03 456 \2 = <space> \3 = 7

Tim Notes:

1. This RegEx is the 'End' taken from 'Pad with Zero at Beginning, Middle and End of String'. Refer to this for further information and analysis.

Sample: Run #1 –

```
                              Start  Length
Match 1: 0 123 2 abc 3 456 7     0      19
Group 1: 0 123                   0       5
Group 2:                         5       1
Group 3: 2                       6       1
Group 4:                         7       1
Group 5: abc 3 456 7             8      11
```

Sample: Run #2 –

```
Match 1: 00 123 02 abc 3 456 7.txt
Group 1: 00 123 02 abc
Group 2:
Group 3: 3
Group 4:
Group 5: 456 7.txt
```

Sample: Run #3 – uses New RegEx

```
Match 1: 00 123 02 abc 3 456 7
Group 1: 00 123 02 abc 3 456
Group 2:
Group 3: 7
```

Forum Contributors: Saintoctopus, therube

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1 cont.

My Contribution:

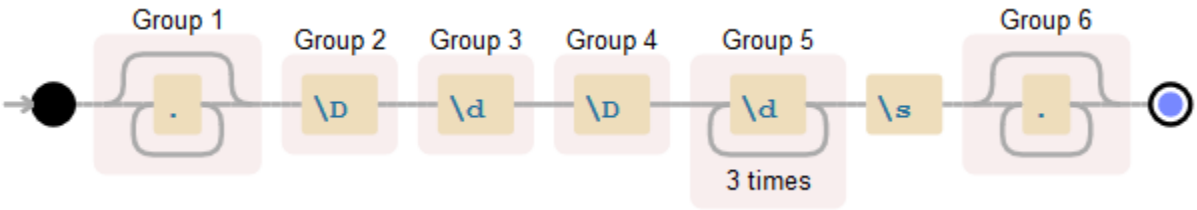
Example #3

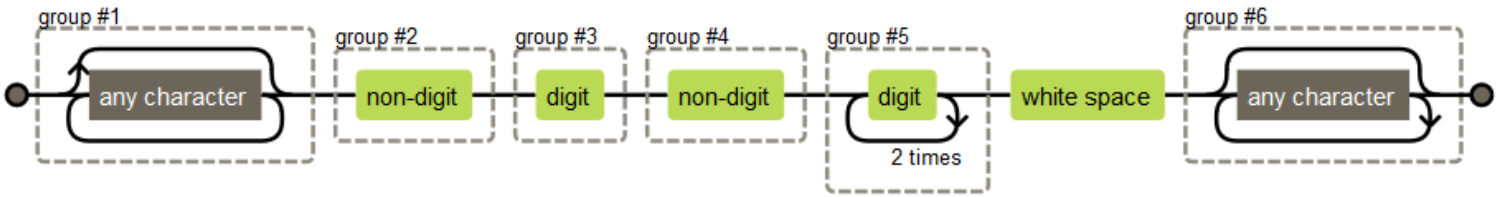
First Attempt:

so restrictive it will only match one pattern: 0 123 2 abc 3 456 7.txt

Match: (.*?)(\D)(\d)(\D)(\d{3})\s(.*)
Replace: 0\1\20\3\4\5 0\6

Name	New Name
0 123 2 abc 3 456 7.txt	00 123 2 abc 03 456 07.txt





Tim Notes:

1. Capture Groups 2 and 4 are not needed. These only capture <space> characters. The parentheses can be removed without affecting the result. This would reduce the Capturing Groups from 6 down to 4. The Replacement String would need to be adjusted to reflect the change.

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1 cont.

My Contribution: cont.

Example #3 cont.

First Attempt: cont.

Using: 0 123 2 abc 3 456 7.txt

Match: (.*?)(\D)(\d)(\D)(\d{3})\s(.*)

Where:

\1 = 0 123 2 abc \2 = <space> \3 = 3 \4 = <space> \5 = 456 \6 = 7

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy (Zero Occurrence Match)	Capture Group 1 = null
4. \D	Match against Non-Numeric Digit	Capture Group 2 = <space>
	Position = <space> after '0'	Changes Capture Group 1 = 0
5. \d	Matches against numeric digit	Capture Group 3 = 1
	This is the '1' of '123'	
6. \D	Match against Non-Numeric Digit	Capture Group 4 = <space>
	Position = '2' following '123'<space>	Changes Capture Group 1 = 0 123
		Capture Group 2 = Unchanged
		Changes Capture Group 3 = 2
7. \d	Match against numeric digit	Capture Group 5 = 4
8. {3}	Repeat the match 3 iterations	Capture Group 5 = 456
9.	Match against <space> (Not Captured)	
10. .*	Match against remainder of string	Capture Group 6 = 7

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1 cont.

My Contribution: cont.

Example #4

Second Attempt: Better but still restrictive -

Very slight modification to first RegEx (not satisfied). Pads 0's at beginning and end leaving middle. Since the Contributor has already provided for a RegEx to handle the Middle, I will use that for the second run.

Two runs.

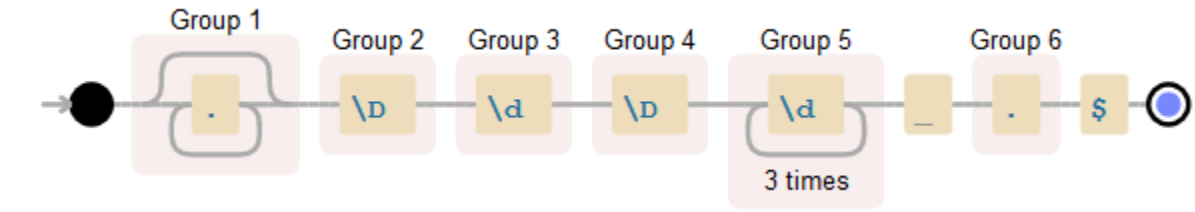
Match: (.*?)(\D)(\d)(\D)(\d{3}) (.)$
Replace: 0\1\20\3\4\5 0\6

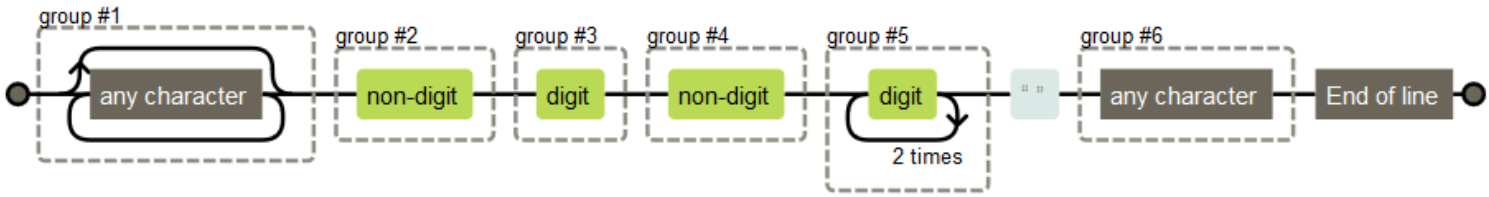
Sample:

	Start	Length
Match 1: 0 123 2 abc 3 456 7	0	19
Group 1: 0 123 2 abc	0	11
Group 2:	11	1
Group 3: 3	12	1
Group 4:	13	1
Group 5: 456	14	3
Group 6: 7	18	1

First Run: Using my RegEx

Name	New Name
0 123 2 abc 3 456 7.txt	00 123 2 abc 03 456 07.txt





Tim Notes:

1. Under Regex Buddy for this test, had to remove anchor $ (EOL) because Regex Buddy always includes extension as part of filename – because it deals with text not filenames generally, so to the program, '.zip' is just another part of the string of text.

Where:

\1 = 0 123 2 abc.txt \2 = <space> \3 = 3 \4 = <space>
\5 = 456 \6 = 7

Tim Analysis:

1. Isolates the '0' of \1, the '3', the '456' and '7'. This allows the Replace String to pad the zeros for the '0', '3', '456' and '7' leaving only the '2' which could not be isolated in this first run and is currently part of Capture Group 1.

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1 cont.

My Contribution: cont.

Example #4 cont.

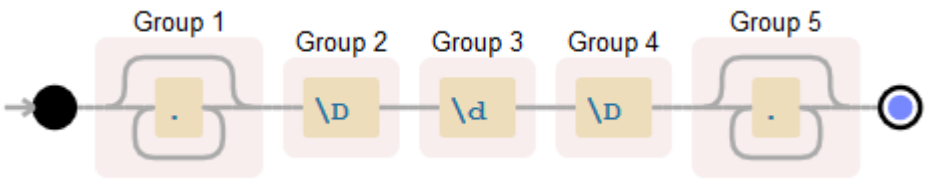
Second Attempt: Better but still restrictive – cont.

Second Run: Contributor RegEx – Middle taken from 'Pad with Zero at Beginning, Middle and End of String'

Match: (.*?)(\D)(\d)(\D)(.*)
Replace: \1\20\3\4\5

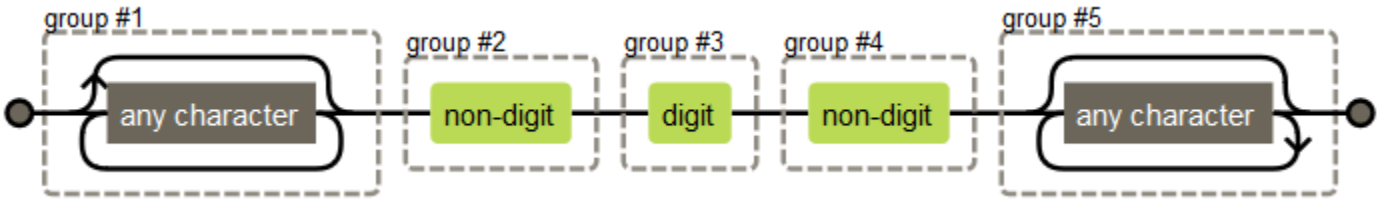
Name	New Name
00 123 2 abc 03 456 07.txt	00 123 02 abc 03 456 07.txt

Completed.



Tim Notes:

1. I recognize this from before, if you recall the discussion of 'Binding' numeric digits using Non-Numeric Digits Boundaries, \D.



Where:

\1 = 00 123 \2 = <space> \3 = 2 \4 = <space> \5 = abc 03 456 07

Tim Analysis:

1. Isolates the '2' allowing the Replacement String to pad the final zero.

Tim Notes:

1. See 'Demonstration of 3 Methods (complex to Simple) to Rearrange string' for the reference to 'binding'

Find Occurrences

Find any Single Digit Number anywhere in a filename and add a preceding zero #1 cont.

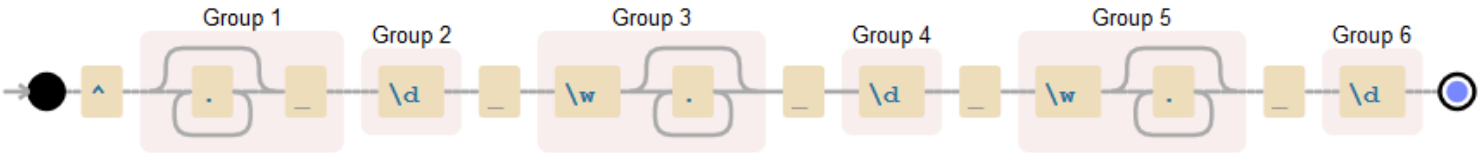
My Contribution: cont.

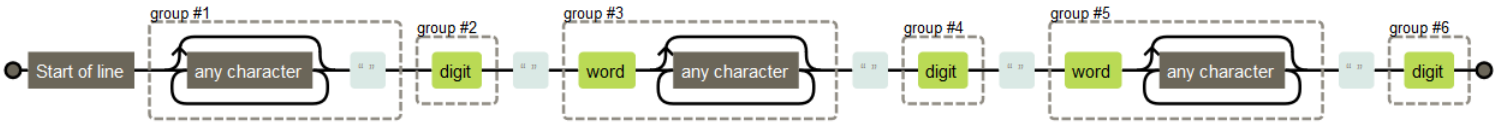
Example #5

Another one of mine is directly taken from another example in this book – this will complete the entire renaming in one run:

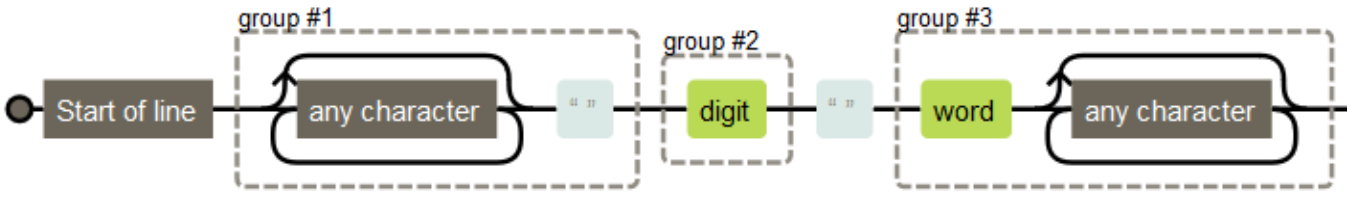
Match: ^(.*?)(\d{1}) (\w.*?) (\d{1}) (\w.*?) (\d{1})
Replace: \10\2 \3 0\4 \5 0\6

Name	New Name
0 123 2 abc 3 456 7.txt	00 123 02 abc 03 456 07.txt





Expanded:



Sample:

```
.....................................
Match 1: 0 123 2 abc 3 456 7
Group 1: 0 123
Group 2: 2
Group 3: abc
Group 4: 3
Group 5: 456
Group 6: 7
```

Tim Notes:

1. The RegEx can handle any three single numeric digits anywhere in the string – but no more or less than three.
2. See 'Prefix final digit with zero if preceded by word-separator (or null)' for further information and analysis.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2

This is important at least to me because there are not too many examples I have found that can match against both a string that has preceding text and those that don't. I have my own RegEx that I have created – see 'Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!)' – but it can only handle strings that have preceding text before the numeric digit. The following RegEx can handle both but right now this is limited to a single occurrence.

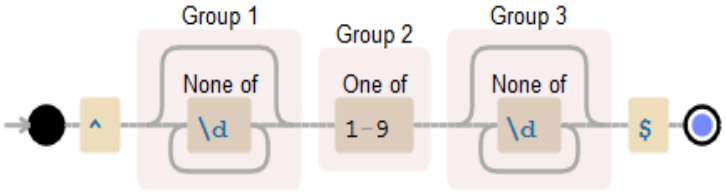
1	to	01
2	to	02
10	to	Ignore
Lesson 1 DNA	to	Lesson 01 DNA
Lesson 1	to	Lesson 01
05	to	Ignore
Lesson 25 DNA	to	Ignore
Lesson 0	to	Choice to either Ignore or change to Lesson 00[1]

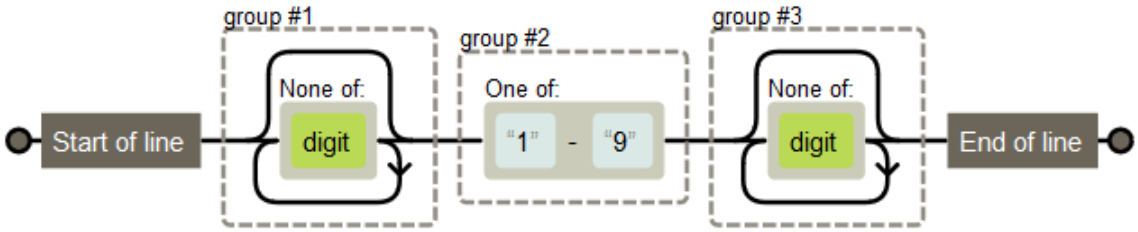
Example #1

Contributor RegEx #1

Match: ^([^\d]*)([1-9])([^\d]*)$
Replace: \10\2\3

Name	New Name
1	01
2	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Lesson 25 DNA	Lesson 25 DNA





Sample:

```
............Start·Length··
Match 1: 1       0       1
Group 1:         0       0
Group 2: 1       0       1
Group 3:         1       0
```

Sample:

```
....................
Match 1: Lesson 1 DNA
Group 1: Lesson
Group 2: 1
Group 3:  DNA
```

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #1 cont.

Contributor Notes:

1. Just change '[1-9]' to '[0-9]' to have it include 'Lesson 0'
2. Restrictive. Mandates that ONLY one [1-9] numeric digit exists 'anywhere' in filename –

Names like: 'Chapter1 Lesson2 DNA' will be ignored

Contributor Analysis:

1. Replaces one lone numeric digit [1-9] in filename

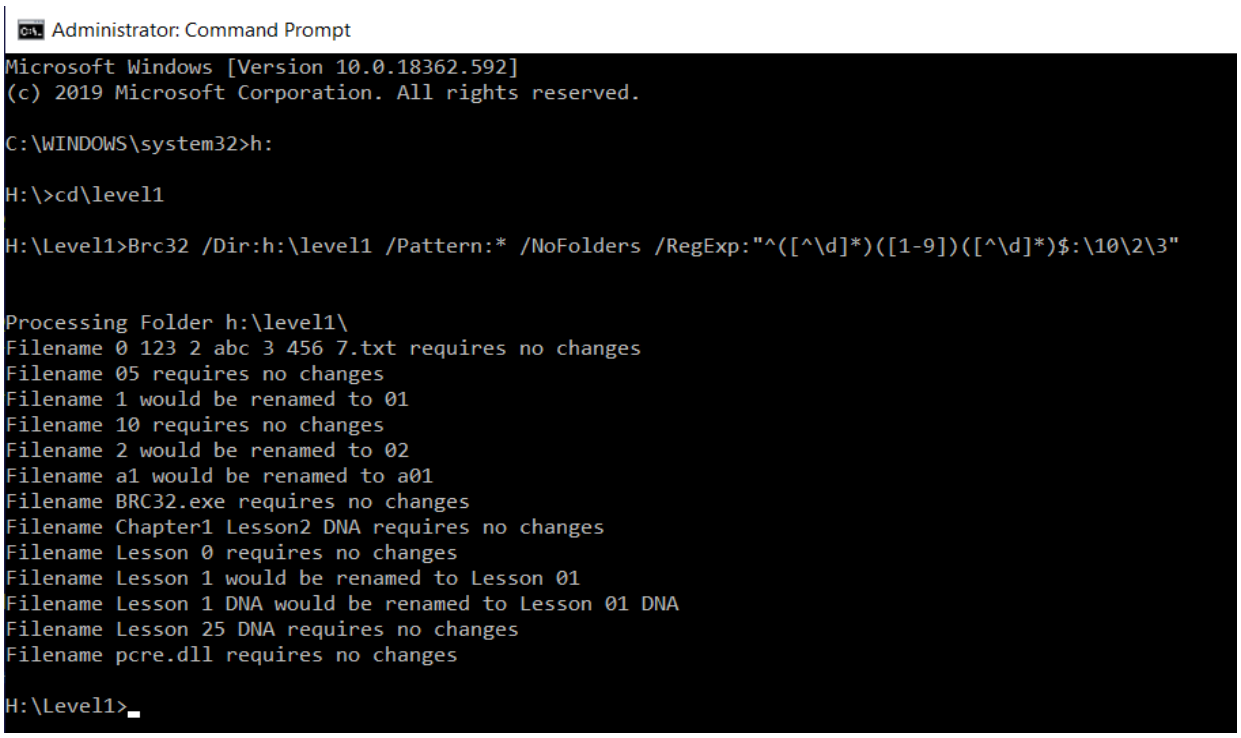
GROUP 1	GROUP2	GROUP3
Zero-or-more Non-Digits	1 Digit ranging from 1 thru 9	Zero-or-more Non-Digits
([^\d]*)	([1-9])	([^\d]*)$

BRC Command line Equivalent:

Brc32 /Dir:YourFolderPath /Pattern:whatever.ext /NoFolders /RegExp:"^([^\d]*)([1-9])([^\d]*)$:\10\2\3" /Execute

i.e,.

Brc32 /Dir:h:\level1 /Pattern:* /NoFolders /RegExp:"^([^\d]*)([1-9])([^\d]*)$:\10\2\3"



Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Tim Notes:

1. Specifying * alone in the pattern will process all files in the directory, regardless if there is a file extension.
   a. This is the same as not specifying any pattern, '/Pattern:' or eliminating the parameter all together.
2. Specifying *.* will process only those files that have a file extension.
3. Specifying *. will *not work* to process only those files that do not have a file extension (I can't find one that does)
4. Specifying Lesson* will process any file that begins with Lesson – case sensitive
5. Specifying Lesson*.* will only process those files that begin with Lesson and have a file extension.
6. Remove '/Execute' to receive a preview as on previous page.

JavaScript Equivalent:

```
newName=name.replace(/(^([^\d]*)([1-9])([^\d]*)$)/g,'$2'+'0'+'$3'+'$4')
```

Name	New Name
1	01
2	02
05	05
10	10
a1	a01
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Lesson 25 DNA	Lesson 25 DNA
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt

Contributor Notes:

1. Not a good candidate for BRC-batch because only the 1st regex-run does anything.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example # 1 cont.

Using: 1

Contributor RegEx #1

Match: ^([^\d]*)([1-9])([^\d]*)$
Replace: \10\2\3

Tim First Analysis using sample above:

1. ^ Start at beginning of string (Beginning of Line or BOL)
2. [^d] Match against a Negated Class consisting of a numeric digit. Ignore first numeric digit encountered. Capture Group = see Tim Notes

Tim Notes:

1. For a string that has text preceding this would capture the first alpha character, i.e. 'L' in 'Lesson'
2. For a string that had a numeric as the first character, it would not match and therefore ignore and move forward in the string.

3. * Make it Greedy Capture Group 1 = ""

Tim Notes:

1. For a string that has text preceding this would capture the rest of the first alpha word, i.e., 'Lesson'
2. For a string that had a numeric as the first character, it would be a Zero Occurrence Match resulting in a null value.

4. [1-9] Match against a class consisting of a numeric digit, 1 through 9. Capture Group 2 = 1

Tim Notes:

1. This will match against the first numeric in the string but will not show any Name Change unless you use a Replacement String, i.e., for testing purposes - \2 <space> so you can see the value. Will not match against a numeric digit of zero as the first character because of the [1-9]. To include zero, use [0-9] instead.

2. For our example, Capture Group 2 = 1

5. [^\d] Ignore if there is a second numeric that follows the first.

Tim Notes:

1. This disqualifies, 'Lesson 25 DNA', '10'.
2. '05' was previously disqualified in step 4.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Using: 1

Match: ^([^\d]*)([1-9])([^\d]*)$
Replace: \10\2\3

Where:

\1 = null \2 = 1 \3 = null

Tim First Analysis: cont.

6. * Make the Negated Class expression Greedy Capture Group 3 = null
7. $ Must be EOL

Tim Notes:

1. For strings that have text that follows after a numeric digit, i.e. 'Lesson 1 DNA', the text after the numeric digit is captured, Capture Group 3 = <space> 'DNA'

2. The purpose of Capture Group 1 and 3 is to capture any text that precedes or follows after the single numeric match. Those that do not have any text, capture a null value through Zero Occurrence Match. This can be clearly seen in the Regex Buddy sample runs.

3. Since this can handle both with and without text at beginning (Beginning of Line or BOL) and end of string (End of Line or EOL), I am going to perform a second analysis using a different sample.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #1 cont.

Using: Lesson 1 DNA

Match: ^([^\d]*)([1-9])([^\d]*)$
Replace: \10\2\3

Tim Second Analysis using sample above:

1. ^ Start at beginning of string (Beginning of Line or BOL)
2. [^d] Match against a Negated Class consisting of a numeric digit. Ignore first numeric digit encountered. Capture Group = see Tim Notes

Tim Notes:

1. For a string that has text preceding this would capture the first alpha character, i.e. 'L' in 'Lesson'
2. For a string that had a numeric as the first character, it would not match and therefore ignore and move forward in the string.

3. * Make it Greedy Capture Group 1 = Lesson

Tim Notes:

1. For a string that has text preceding this would capture the rest of the first alpha word, i.e., 'Lesson'
2. For a string that had a numeric as the first character, it would be a Zero Occurrence Match resulting in a null value.

4. [1-9] Match against a class consisting of a numeric digit, 1 through 9. Capture Group 2 = 1

Tim Notes:

1. This will match against the first numeric in the string but will not show any Name Change unless you use a Replacement String, i.e., for testing purposes - \2 <space> so you can see the value. Will not match against a numeric digit of zero as the first character because of the [1-9]. To include zero, use [0-9] instead.

2. For our example, Capture Group 2 = 1

5. [^\d] Ignore if there is a second numeric that follows the first.

Tim Notes:

1. This disqualifies, 'Lesson 25 DNA', '10'.
2. '05' was previously disqualified in step 4.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #1 cont.

Match: ^([^\d]*)([1-9])([^\d]*)$
Replace: \10\2\3

Name	New Name
1	01
2	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

This is same RegEx – just tested against different samples

Observations:

1. As expected, disqualifies any strings containing two or more numeric digits
2. Disqualifies any strings that have a zero as the first numeric digit – uses [1-9] instead of [0-9]
3. Strings that do not have a delimiter separating the numeric digit from text are also disqualified from the match, i.e.'Chapter1 Lesson2 DNA'
4. This one is more restrictive. Mandates that only one numeric digit [1-9] exists 'anywhere' in filename.

Tim Second Analysis: cont.

Using: Lesson 1 DNA

6. * Make the Negated Class expression Greedy Capture Group 3 = <space> DNA
7. $ Must be at EOL

Tim Notes:

1. For strings that have text that follows after a numeric digit, i.e. 'Lesson 1 DNA', the text after the numeric digit is captured, Capture Group 3 = <space> 'DNA'

2. The purpose of Capture Group 1 and 3 is to capture any text that precedes or follows after the single numeric match. Those that do not have any text, capture a null value through Zero Occurrence Match. This can be clearly seen in the Regex Buddy sample runs.

Find Occurrences

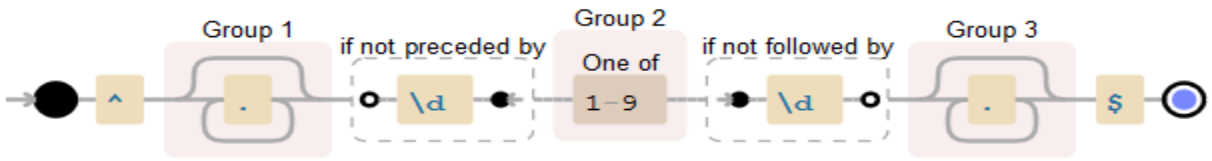
Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #2

This one uses a Lookaround combination of a Negative Lookbehind with a Negative Lookahead.

Match: ^(.*?)(?<!\d)([1-9])(?!\d)(.*)$
Replace: \10\2\3

Name	New Name
1	01
2	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Lesson 25 DNA	Lesson 25 DNA



Sample:

```
.......................
Match 1: lesson 1 dna
Group 1: lesson
Group 2: 1
Group 3:  dna
```

Sample:

```
...........Start·Length···
Match 1: 1       0       1
Group 1:         0       0
Group 2: 1       0       1
Group 3:         1       0
```

Zero Occurrence Match for Groups 1 and 3 (length = 0) that are meant to capture any text in the string.

Using: Lesson 1 DNA

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .	Match against any character	Capture Group 1 = L
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy (Lazy) Created Zero Occurrence Match. Current position = beginning of string (BOL).	Capture Group 1 = null
5. (?<!\d)	Negative Lookbehind. Look left to see if previous character is NOT a numeric digit. = True	
6. [1-9]	Match against a class consisting of a numeric digit, 1 through 9.	Capture Group 2 = 1 Changes Capture Group 1 = Lesson
7. (?!\d)	Negative Lookahead. Look right to see if following character is NOTa numeric digit. This disallows strings containing two or more numeric digits, i.e. 'Lesson 25 DNA' or '10'. = <space> = True	
8. .*	Match against remainder of string	Capture Group 3 = <space> DNA
9. $	Must be at EOL	

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

This one uses a Lookaround combination of a Negative Lookbehind with a Negative Lookahead cont.

Contributor Notes:

1. Just change '[1-9]' to '[0-9]' to have it include 'Lesson 0'
2. Replaces the 1st-occurrence a lone numeric digit [1-9]. Run multiple times if needed.
3. Restrictive. Mandates that ONLY one [1-9] numeric digit exists 'anywhere' in filename -

Names like: "Chapter1 Lesson2 DNA' will be ignored

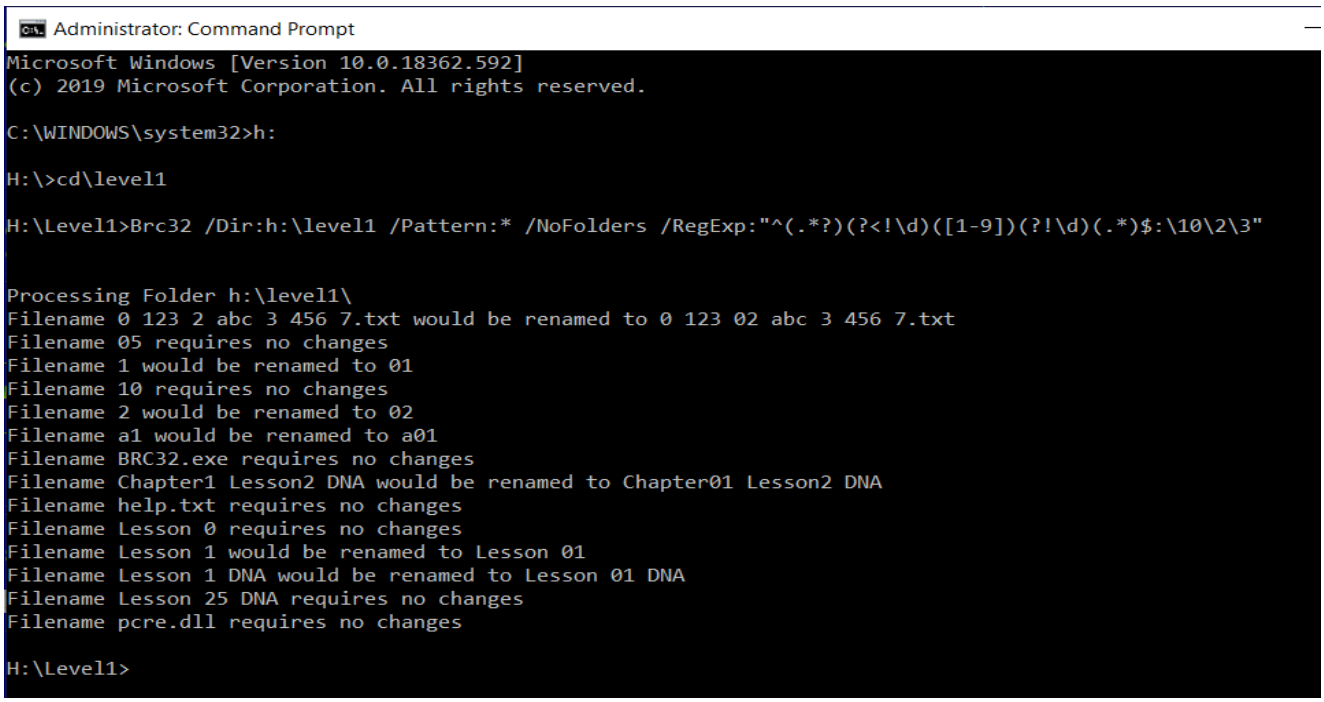
Contributor Analysis:

GROUP 1	NegativeLookBEHIND	GROUP2	NegativeLookAHEAD	GROUP3
Anything Until 1st-Occurrence of Capture Group 2	Test-LEFT 1 character Last char in Group 1 must not be a digit	Any 1 Digit ranging from 1 thru 9	Test-RIGHT 1 character 1st-char in Group 3 must not be a digit	Anything
^(.*?)	(?<!\d)	([1-9])	(?!\d)	(.*)$

1. Capture Groups 1 & 3 can match nul, only Group 2 mandates the existence of a character.

BRC Command line Equivalent: Will match "Chapter1 Lesson2 DNA'

Brc32 /Dir:YourFolderPath /Pattern:whatever.ext /NoFolders /RegExp:"^(.*?)(?<!\d)([1-9])(?!\d)(.*)$:\10\2\3" /Execute



Find Occurrences

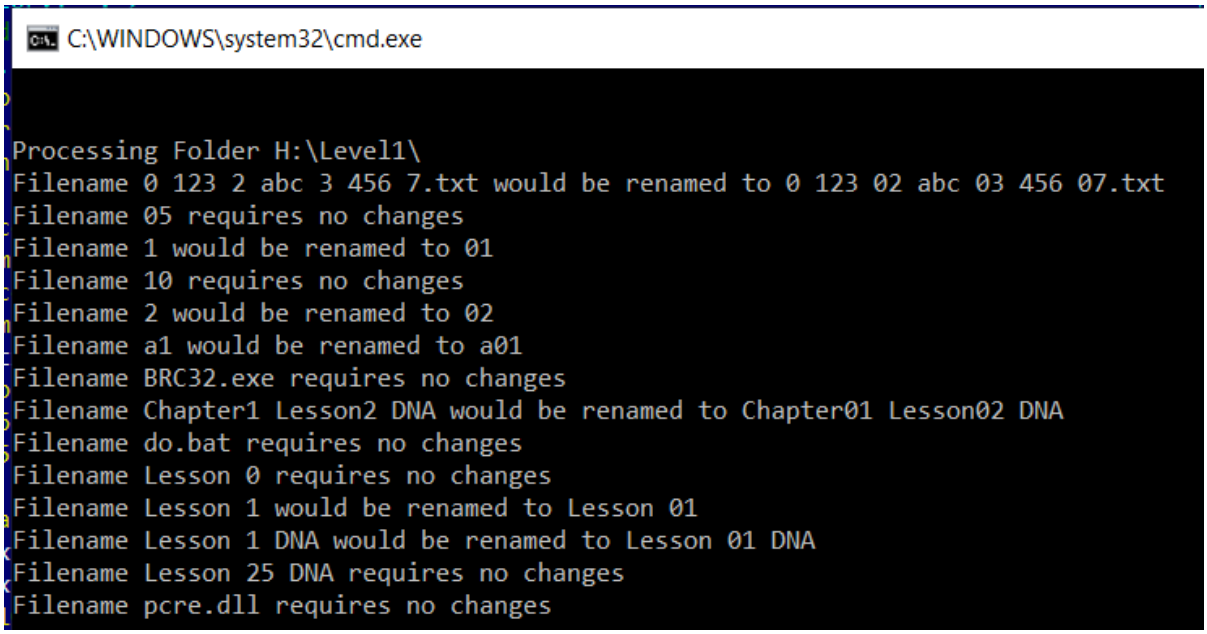
Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #2 cont.

This one uses a Lookaround combination of a Negative Lookbehind with a Negative Lookahead cont.

BRC BATDH equivalent for 10 runs: Will match 'Chapter1 Lesson2 DNA'

```
@echo off
cd/d "C:\YourFolderPath\"
Set reg1=/RegExp:"^(.*?)(?<!\d)([1-9])(?!\d)(.*)$:\10\2\3"
set ten=%reg1% %reg1% %reg1% %reg1% %reg1% %reg1% %reg1% %reg1% %reg1% %reg1% /Execute
brc32 /Pattern:whatever.ext /NoFolders %ten%
pause>nul
```



Contributor Notes:

1. '/Pattern' & '/NoFolders' prevent 'selecting' all files/folders for possible renaming.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

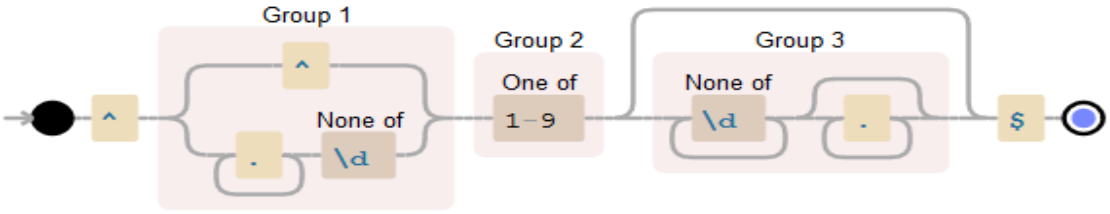
Example #3

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text.

Requires Multiple runs to match against all single numeric digits in a string. This will be demonstrated.

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$
Replace: \10\2\3

Name	New Name
1	01\3
2	02\3
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01\3
Lesson 1 DNA	Lesson 01 DNA
Chapter1 Lesson2 DNA	Chapter01 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 2 abc 3 456 7.txt	0 123 02 abc 3 456 7.txt



Sample:

```
.............................
Match 1: Chapter1 Lesson2 DNA
Group 1: Chapter
Group 2: 1
Group 3:  Lesson2 DNA
```

Sample:

```
.................Start·Length.........
Match 1: lesson 1      0      8
Group 1: lesson        0      7
Group 2: 1             7      1
Group 3 did not participate in the match
```

Sample:

```
........................
Match 1: Lesson 1 DNA
Group 1: Lesson
Group 2: 1
Group 3:  DNA
```

Sample:

```
...........Start·Length..............
Match 1: 1      0      1
Group 1:        0      0
Group 2: 1      0      1
Group 3 did not participate in the match
```

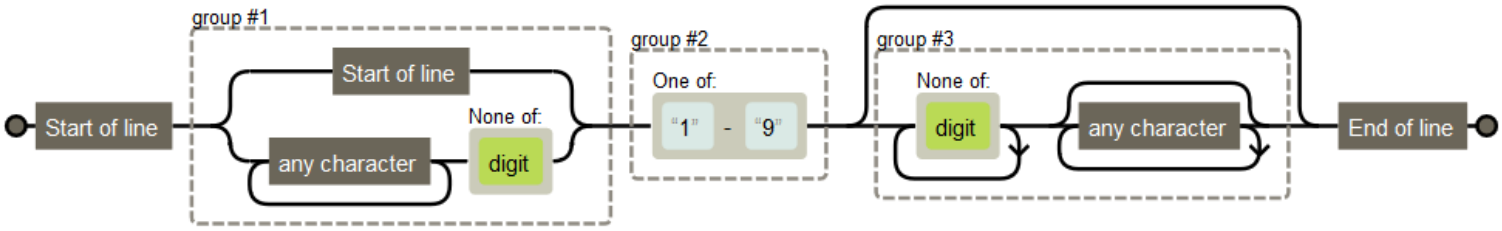
Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #3 cont.

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text. cont.

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$



Tim Notes:

1. You can clearly see where Group 1 in the 4th sample has a Zero Occurrence Match but for Group 3 in samples 2 and 4, Group 3 is invalidated \3 as in 'did not participate in the match'.

Forum Contributor: RegexNinja

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #3 cont.

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text. cont.

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$
Replace: \10\2\3

My Contribution:

Observations:

1. This one is interesting. Group 3, normally used to capture any text following the numeric digit, is invalidated - \3. In the previous two examples, if text was not present, a Zero Occurrence Match would result in a null value instead.
2. The invalidation does not invalidate the match – only Group 3. This is because of the {0,1} which is the same as ? making the expression optional if there is no match as if it doesn't exist, thus Capture Group 3 becomes Invalid.

By changing it to:

Example #4

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*)(.*)$

Name	New Name
1	1
2	2
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 1
Lesson 1 DNA	Lesson 01 DNA
Chapter1 Lesson2 DNA	Chapter01 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 2 abc 3 456 7.txt	0 123 02 abc 3 456 7.txt

This removes the invalid designation without affecting the previous matches – I added a (.*) after the ([^\d]+.*) expression to capture any remaining text and because it is zero or more matches, will capture to EOL regardles if at EOL. This results in a null value because of a Zero Occurrence Match for capture Group 4, but still, however, won't allow matches of '1', '2', and 'Lesson 1'

```
                              Start  Length
Match 1: Chapter1 Lesson1 DNA     0      20
Group 1: Chapter                  0       7
Group 2: 1                        7       1
Group 3:  Lesson1 DNA             8      12
Group 4:                         20       0
```

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #3 cont.

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text. cont.

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$
Replace: \10\2\3

My Contribution: cont.

This will: The Order of Evauation dictates the RegEx (actually 'Section #1') is performed prior to 'Section #3: Replace', therefore by using Replace *after* RegEx, this trick removes the invalid designation allowing the match in New Name. (discussed previously – see page 290).

Name	New Name
1	01
2	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Chapter1 Lesson2 DNA	Chapter01 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 2 abc 3 456 7.txt	0 123 02 abc 3 456 7.txt

'Section #3: Replace' –

Replace \3 with null

All matches are performed with no Invalid despite Group 3. This does not change the results of the previous samples – will still display as 'not participating' in Regex Buddy and 'Invalid' in BRU, but this allows the matches to take place.

Breaks down:

	Capture Group 1		Capture Group 2	Capture Group 3		
	Alternate					
Anchor	Anchor					Anchor
^	(^ | | .+ ? [^\d])	([1-9])	([^ \d])	+ .*)	{ 0,1 }	$

This uses an Alternate for Capture Group 1:

This	OR	This
^	|	. + ? [^ \d]

Typically, an alternate would provide a choice to match between this OR that, but in this case the choice is not made clear. This is because on the OR part, wildcards are used so of course it is always going to match. Simply put, it asks the question, at this moment is the current position at the Beginning of the string (Beginning of Line or BOL) or not?

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #3 cont.

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text. cont.

Using: Chapter1 Lesson1 DNA

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$

Where:
\1 = Chapter \2 = 1 \3 = <space> Lesson2 DNA

Tim Analysis:

1. ^ Start at beginninng (Beginning of Line or BOL)
2. ^|.+[^\d] ([1-9]) Alternate:
   a. Test if *either at beginning of string*. = True. Evaluate next expression.
   b. *Or not at beginning of string* in which case do the following –
      1. . Match against any character — Capture Group 1 = C
      2. + Make it Greedy — Capture Group 2 = <entire string>
      3. ? But not too Greedy (Lazy) — Capture Group 1 = C
      4. [^\d] Match against a Negated Class consisting of a numeric digit.= 'C' = True. This checks that a numeric digit does not immediately follow – won't match against 'a1' for example.
   c. ([1-9]) Match against a class consisting of a numeric. digit 1-9. = 'C' = False. This would also allow it to match against strings that start with a numeric digit and disallow zero.
   d. Move to next character = 'h' and repeat step 2 for each character until step 2c = True. When finished it will have tested each character of the word, 'Chapter' (including any following <space> character) until it encountered a numeric and matched it in Capture Group 2. — Capture Group 1 = Chapter; Capture Group 2 = 1

Tim Notes:

1. The ([1-9]) will match against the first numeric in the string but will not show any Name Change unless you use a Replacement String, i.e., for testing purposes - \2 <space> so you can see the value. Will not match against a numeric digit of zero as the first character because of the [1-9]. To include zero, use [0-9] instead.

2. For our example, Capture Group 2 = 1

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #3 cont.

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text. cont.

Using: Chapter1 Lesson1 DNA

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$

Where:
\1 = Chapter \2 = 1 \3 = <space> Lesson2 DNA

Tim Analysis: cont.

3. [^\d]	Ignore if there is a second numeric that follows the first.	Capture Group 3 = <space>

Tim Notes:

1. This disqualifies, 'Lesson 25 DNA', '10'.
2. '05' was previously disqualified in step 2c

4. +	Make the Negated Class expression Greedy	Capture Group 3 = <space> Lesson

Tim Notes:

1. For strings that have text that follows after a numeric digit, i.e. 'Lesson 1 DNA', the text after the numeric digit is captured, Capture Group 3 = <space> 'DNA'

2. The purpose of Capture Group 1 and 3 is to capture any text that precedes or follows after the single numeric match. Those that do not have any text, capture a null value through Zero Occurrence Match. This can be clearly seen in the Regex Buddy sample runs.

5. .	Match against any character	Capture Group 3 = <space>Lesson2
6. *	Make it Greedy	Capture Group 3 = <space>Lesson2 DNA
7. {0,1}	Repeat the previous expression zero or one iteration. This makes the ([^\d]+.*) optional in cases where there are no characters following, i.e., Lesson 1	

Tim Notes:

1. Could have used ' ? ' in place of {0,1}

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

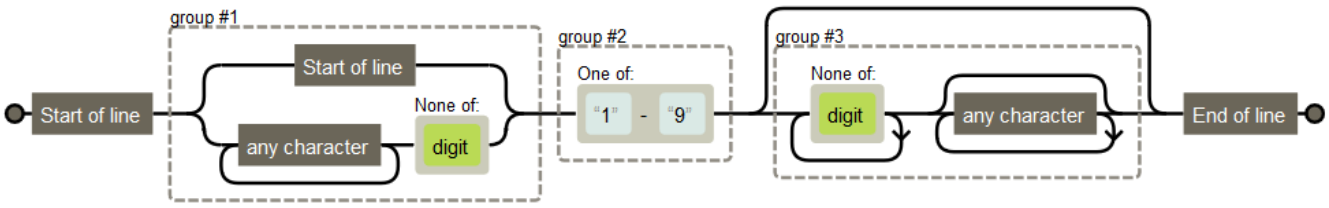
Example #3 cont.

Replaces the first occurrence of a single numeric digit and is not constrained by delimiters that separate the numeric digit from the text. cont.

Using: Chapter1 Lesson1 DNA

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$

Where:
\1 = Chapter \2 = 1 \3 = <space> Lesson2 DNA



Tim Analysis: cont.

Tim Notes:

1. By itself, the expression, ^(^|.+?[^\d])([1-9]) matches:

   '1', '2', 'Lesson 1' and '10' – although 10 is not wanted.

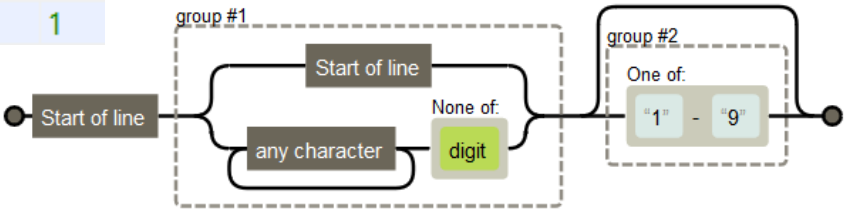
   Adding, ([^\d]+.*){0,1} disqualified those because now it only matched up against those strings that had following text after the match of the numeric digit.

2. By making the expression optional, strings, '1', '2', 'Lesson 1' will match but so will '10'.
   a. This is the equivalent of changing the RegEx to:

   Match: ^(^|.+?[^\d])([1-9]){0,1}
   Replace: \2

10	1

8. $ Must be at EOL



Tim Notes:

1. This takes care of '10' because prior to this in step 2d, Capture Group 2 contained a match for the '1' in '10'. By the inclusion of the $ anchor, since the match of the '1' is not at EOL, '10' and any similar two numeric digit or more string will fail and is disaqualified from the match.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #5

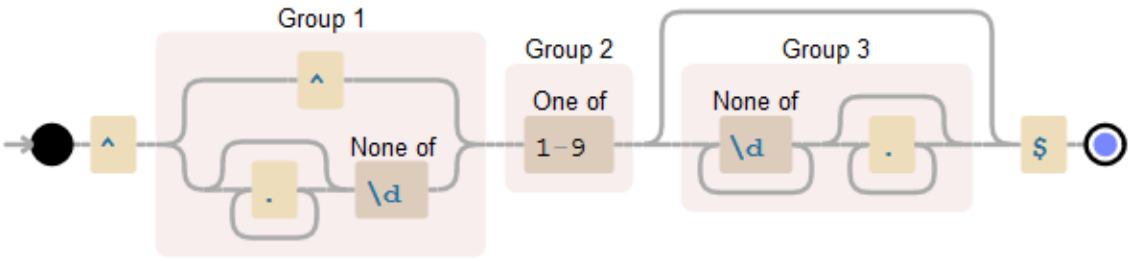
If you wanted to match against 'a1'

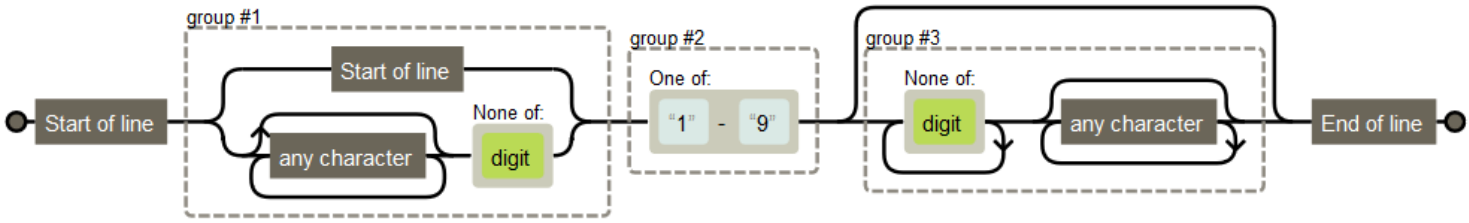
Modify + to * in Group 1 -

Match: ^(^|.*?[^\d])([1-9])([^\d]+.*){0,1}$
Result: \10\2\3

Name	New Name
1	01
2	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Chapter1 Lesson2 DNA	Chapter01 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA
a1	a01

Must include the trick using 'Section #3: Replace' to account for Group 3 invalid \3 in New Name.





Sample:

```
·············Start·Length·················
Match 1: a1       0       2
Group 1: a        0       1
Group 2: 1        1       1
Group 3 did not participate in the match
```

Further discussion on matching 'a1' follows.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

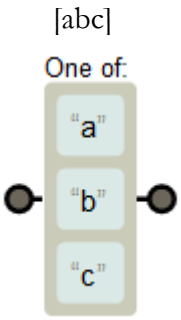
A Short Discussion on + *vs* *

Although more information can be found in both volumes in the RegEx Manuals of the Appendix, it might be noteworthy just to mention a few things and try to understand why + failed to match A1 when * did.

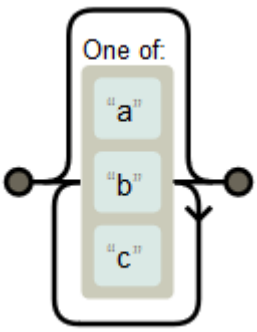
the * is known as the Kleene Star
the + is known as the Kleene Plus

Characteristics:

1. Both are considered Greedy. They take and take and take only 'giving back' as needed. Sounds like my Son in law.
2. Both typically follow a character or group containing an expression or character.
3. + matches one or more instances of the preceding pattern. There must be at least one instance of the pattern.
4. .* matches zero or more instances of the preceding pattern. It will still match even if there are no instances of it.
5. ? Lazy or Non- Greedy when applied against a Greedy Quantifier, matches the least amount of characters. Can also make the expression optional meaning if there is no match, it can backtrack and re-evaluate the previous expression. If this happens, the match may still take place, but nothing will be captured. This is called a Zero Occurrence Match. The Capture Group will contain "" or 'null' value. If applied against an expression, can make the entire expression optional. If this happens, it usually doesn't result in a Zero Occurrence Match.
6. For more information refer to the RegEx Manual in Volume I and the Supplemental in the Appendix here.
7. Zero Occurrence Match discussions can also be found throughout this book.

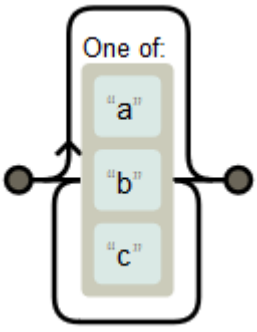


Greedy
[abc] *



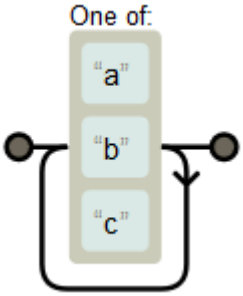
Zero or More

Non-Greedy
[abc]*?



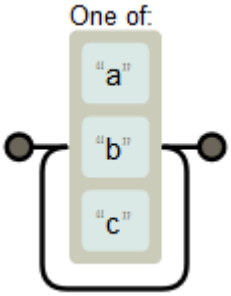
Least amount of matches
Optional

Greedy
[abc] +



One or More

Non-Greedy
[abc]+?



Least amount of matches
Optional

Find Occurrences

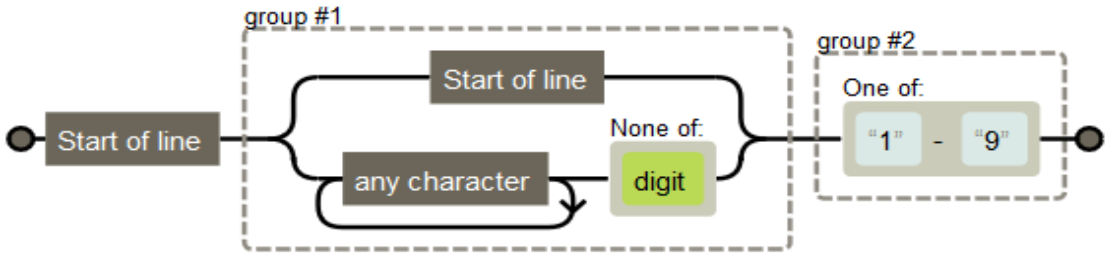
Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

I am only concentrating on the first half of the RegEx which comprises Group 1 and Group 2. It is these two expressions that determine a match on 'a1'. Group 3 comprises the second half.

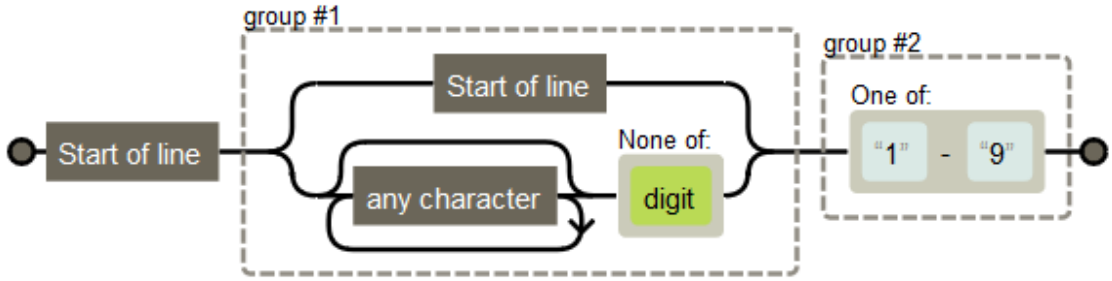
Here is how ^(^|.*?[^\d])([1-9]) looks visually:

String = a1

Using: .+



Using .*



Sample:

Using .+ `^(^|.+[^\d])([1-9])`

```
Beginning match attempt at character 0
1  ok
2  ok
3  ok
4  backtrack
5  backtrack
6  a1
7  a1backtrack
8  a
9  abacktrack
10 backtrack
   Match attempt failed after 10 steps
```

Sample:

Using .* `^(^|.*[^\d])([1-9])`

```
Beginning match attempt at character 0
1  ok
2  ok
3  ok
4  backtrack
5  backtrack
6  a1
7  a1backtrack
8  a
9  abacktrack
10 ok
11 a
12 a
13 a1
14 a1
   Match found in 14 steps
```

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #6

Here's a RegEx that doesn't require the 'Replacement' trick.

Match: ^(^|.*?[^\d])([1-9]([^\d]+.*){0,1})$
Replace: \10\2

Name	New Name
1	01
2	02
05	05
0 123 2 abc 3 456 7.txt	0 123 02 abc 3 456 7.txt
10	10
a1	a01
2016.10.06 - Thursday – 1-Enviro...	2016.10.06 - Thursday – 01-Environmental costs
2016.11.29 - Monday - 5. Physics-...	2016.11.29 - Monday - 05. Physics-5
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Lesson 25 DNA	Lesson 25 DNA

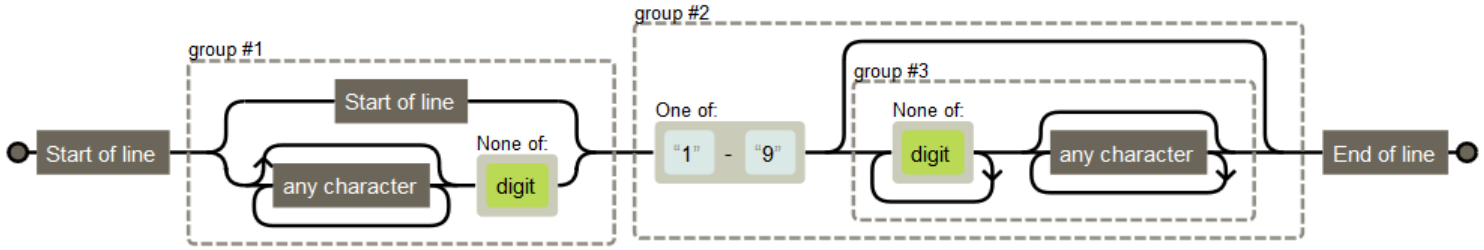
Sample:

```
·············Start·Length···············
Match 1: a1      0      2
Group 1: a       0      1
Group 2: 1       1      1
Group 3 did not participate in the match
```

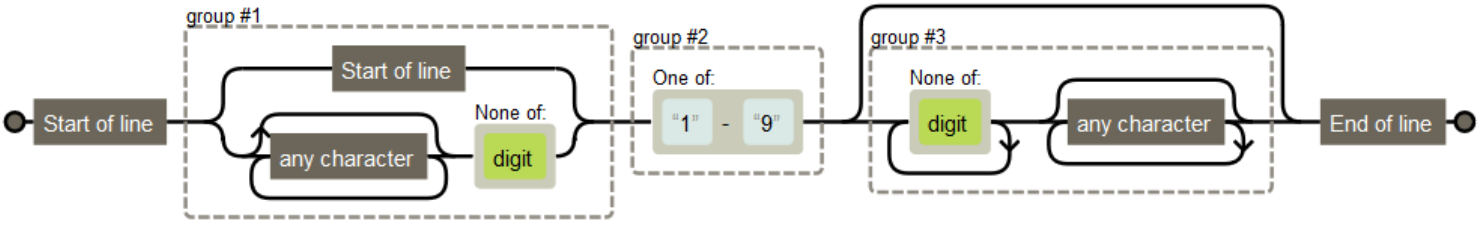
Tim Notes:

1. Although Group 3 still did not participate in the match, it does not display as Invalid, probably because it was nested in Group 2.

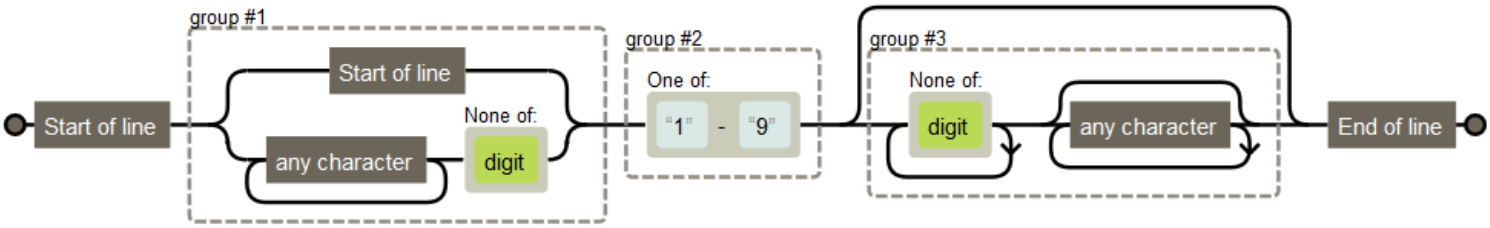
New - extending Capture Group 2 over the entire second half of the expression - worked with a1 without Replacement.



Modified - using .* - worked with a1 if 'Section #3: Replace' trick used - ^(^|.*?[^\d])([1-9])([^\d]+.*){0,1}$



Original - using .+ - didn't work with a1 - ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$



Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #6 cont.

Here's a RegEx that doesn't require the 'Replacement' trick. cont.

I am only concentrating on the second half of the RegEx which comprises Group 2 and Group 3 (nested). It is Group 2 that determines a match on 'a1'. Group 1 comprises the first half and has not changed from the 'modified' RegEx.

New Match: ^(^|.*?[^\d])([1-9]([^\d]+.*){0,1})$

as compared with:

Using .* Match: ^(^|.*?[^\d])([1-9])([^\d]+.*){0,1}$

Breaks down:

		1	2	3		
New	^	(^|.*?[^\d])	([1-9]	([^\d]+.*)	{0,1})	$

		1	2	3		
Using .*	^	(^|.*?[^\d])	([1-9])	([^\d]+.*)	{0,1}	$

Where :

a1:	\1 = a	\2 = 1	\3 = Invalid
2:	\1 = <entire string>	\2 = <entire string>	\3 = Invalid
Lesson 1:	\1 = Lesson	\2 = 1	\3 = Invalid
Lesson 1 DNA:	\1 = Lesson	\2 = 1 DNA	\3 = <space>DNA

2016.11.29 – Monday – 5. Physics-5: \1 = 2016.11.29 - Monday -<space> \2 = 5. Physics-5 \3 = .Physics-5

Observations:

1. Single numeric digits are interesting because both \1 and \2 = entire string, Replacment could have been either:

Replace: 0\1\2 or Replace: \10\2

But \10\2 had to be used to be universal with the other samples.

2. Capture Group 3 captures any following text past the numeric match
3. Capture Group 2 evalautes the entire expression. Will be explained next.

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Example #6 cont.

Here's a RegEx that doesn't require the 'Replacement' trick. cont.

Contributor and *Tim Analysis:*

1. Capture Group 2 groups around Capture Group 3, called 'nesting', so you don't need to account for Capture Group 3 in the Replacement String and therefore 'Section #3: Replace' is not required.

   Why? Because Capture Group 2 evaluates the entire second half of the expression now comprised of Capture Group 2 and Capture Group 3. When two or more capture Groups are captured, i.e., Capture Group 2, the evaluation of Capture Group 2 encompasses all Captured values of each Capture Group.

   If Capture Group 3 is invalid, the result of Capture Group 2 would not include Capture Group 3 because an Invalid expression captures nothing – it failed to evaluate. And because any value that Capture Group 3 held, if any, would be part of the value held in Capture Group 2, there is no requirement to use Capture Group 3 in the Replacement String, thus, avoiding the 'invalid' and eliminating the need for 'Section #3:Replace'.

   So if ([1-9]) = value 1 and ([^\d]+.*) = value 2 then Capture Group 2 = Value1Value2
   ([1-9]) = value 1 and ([^\d]+.*) = Invalid then Capture Group 2 = Value1

a1:	Capture Group 2 = 1 + invalid =	1
2:	Capture Group 2 = 2 + invalid =	2
Lesson 1:	Capture Group 2 = 1 + invalid =	1
Lesson 1 DNA:	Capture Group 2 = 1 + <space> DNA =	1 DNA

2016.11.29 – Monday – 5. Physics-5:

	Capture Group 2 = 5 + . Physics-5 =	5. Physics-5

The first value, i.e., '5' is evaluated from ([1-9]).
The second value, i.e., '.Physics-5' is evaluated from ([^\d]+.*)

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Using the original RegEx (Example #3) -

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$
Replace: \10\2\3

Use along with 'Section #3: Replace' as previously demonstrated.

This RegEx is meant to run multiple times. Each subsequent run will pad a zero unto a single numeric digit regardless if the numeric digit is delimited from the text portion or not.

Run #1

Name	New Name
1	01
2	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 1	Lesson 01
Lesson 1 DNA	Lesson 01 DNA
Chapter1 Lesson2 DNA	Chapter01 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 2 abc 3 456 7.txt	0 123 02 abc 3 456 7.txt

Sample:

```
........................
Match 1: Lesson 1 DNA
Group 1: Lesson
Group 2: 1
Group 3:  DNA
```

Run #2

Name	New Name
01	01
02	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 01	Lesson 01
Lesson 01 DNA	Lesson 01 DNA
Chapter01 Lesson2 DNA	Chapter01 Lesson02 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 02 abc 3 456 7.txt	0 123 02 abc 03 456 7.txt

Sample:

```
................................
Match 1: Chapter01 Lesson2 DNA
Group 1: Chapter01 Lesson
Group 2: 2
Group 3:  DNA
```

Run #3

Name	New Name
01	01
02	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 01	Lesson 01
Lesson 01 DNA	Lesson 01 DNA
Chapter01 Lesson02 DNA	Chapter01 Lesson02 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 02 abc 03 456 7.txt	0 123 02 abc 03 456 07.txt

Sample:

```
.............Start·Length.................
Match 1: 0 1      0       3
Group 1: 0        0       2
Group 2: 1        2       1
Group 3 did not participate in the match
```

Find Occurrences

Find any Single Digit Number anywhere in a Filename and add a preceding zero #2 cont.

Using the original RegEx (Example #3) – cont.

Match: ^(^|.+?[^\d])([1-9])([^\d]+.*){0,1}$
Replace: \10\2\3

along with 'Section #3: Replace' as previously demonstrated.

Remember, without using 'Section #3: Replace' – Capture Group 3 would be invalid and prevent you from renaming.

After Run #3

Name	New Name
01	01
02	02
05	05
10	10
Lesson 0	Lesson 0
Lesson 01	Lesson 01
Lesson 01 DNA	Lesson 01 DNA
Chapter01 Lesson02 DNA	Chapter01 Lesson02 DNA
Lesson 25 DNA	Lesson 25 DNA
0 123 02 abc 03 456 07.txt	0 123 02 abc 03 456 07.txt

All strings exhausted. No more matches possible. Success.

Tim Notes:

1. I have many more examples of Padding and Appending – also see the 'Add' section.

2. This example utilized two tricks – defeating the Invalid using an undefined reference after an expression before a quantifier and using the Replace function to remove the invalid group backreference if it appears in New Name, thus allowing previous matches to take place and permitting the Renaming process.

Additional material to my own, Contributor: RegexNinja

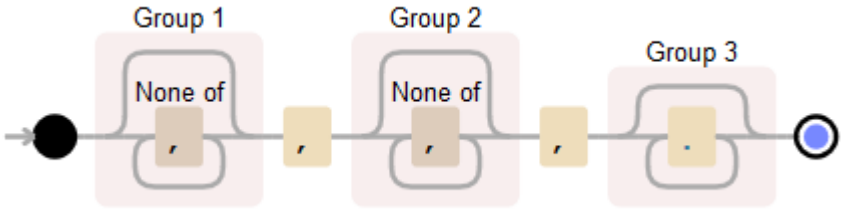
Find Occurrences

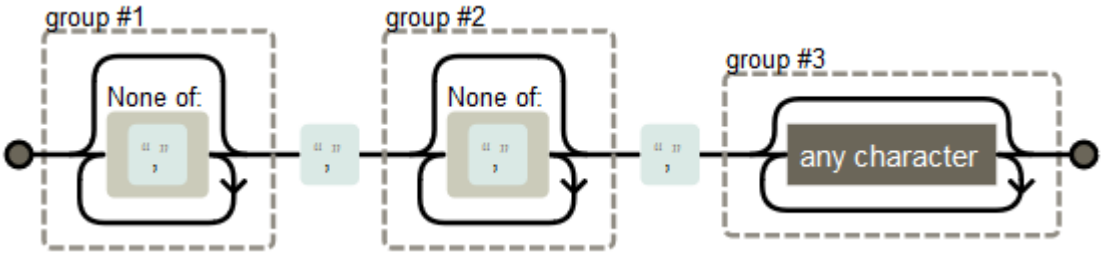
Insert text at second comma in string – i.e., 'brief'

Simons,Jaak,Brussels,13011910,Antwerp,25032009.jpg
to
Simons,Jaak (brief),Brussels,13011910,Antwerp,25032009.jpg

Match : ([^,]*),([^,]*),(.*)
Replace: \1,\2 (brief),\3

Name	New Name
Simons,Jaak,Brussels,13011910,Antwerp,25032009.jpg	Simons,Jaak (brief),Brussels,13011910,Antwerp,25032009.jpg





Sample:

```
.....................................................
Match 1: Simons,Jaak,Brussels,13011910,Antwerp,25032009
Group 1: Simons
Group 2: Jaak
Group 3: Brussels,13011910,Antwerp,25032009
```

Where:

\1 = Simons \2 = Jaak \3 = rest of string = Brussels,13011910,Antwerp,25032009

Contributor Notes:

1. change (brief) as needed.

Forum Contributors: Admin

Find Occurrences

Insert text at second comma in string – i.e., 'brief' cont.

Using: Simons,Jaak,Brussels,13011910,Antwerp,25032009.jpg

Match : ([^,]*),([^,]*),(.*)

Where:

\1 = Simons \2 = Jaak \3 = Brussels,13011910,Antwerp,25032009

Tim Analysis:

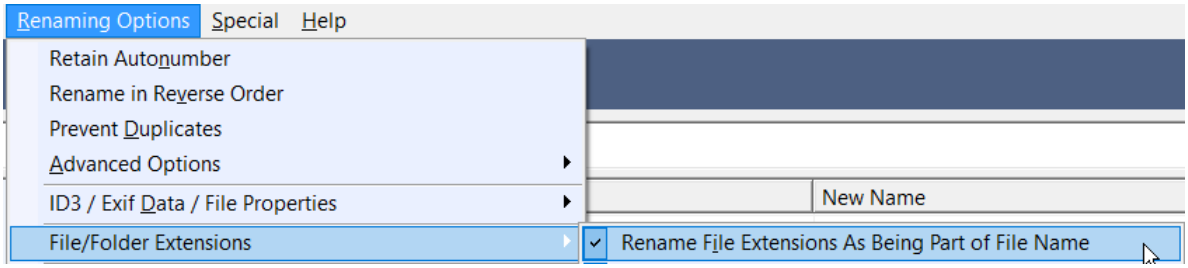
1. [^,]	Match against a Negated Class consisting of a <comma>	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = Simons
3. ,	Match against <comma> (Not Captured) = ' , ' This is the <comma> after 'Simons'	
4. [^,]	Match against a Negated Class consisting of a <comma>	Capture Group 2 = J
5. *	Make it Greedy	Capture Group 2 = Jaak
6. ,	Match against <comma> (Not Captured) = ' , ' This is the <comma> after 'Jaak'	
7. .*	Match against remaining string	Capture Group 3 = Brussels,13011910,Antwerp,25032009

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space>

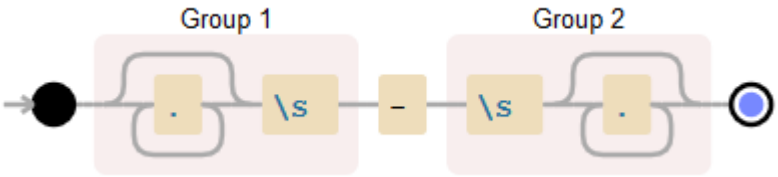
2016.11.29 - Monday - 5. Physics-5 to 2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs to 2016.10.06 - Thursday – 1 Environmental costs

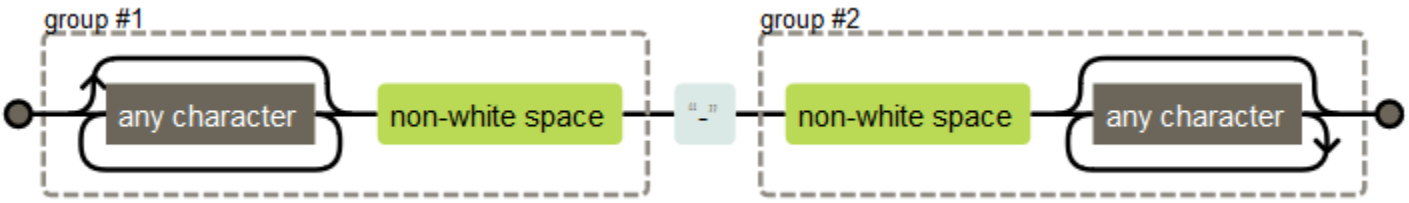
Because the samples use dots within the filenames, you have to enable 'Rename File extensions As Part of File Name' under the ID3 /Edif Data / File Properties submenu of the Renaming Options Menu.



Match: (.*?\S)-(\S.*)
Replace: \1 \2

Name	New Name
2016.11.29 - Monday - 5. Physics-5	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	2016.10.06 - Thursday – 1 Environmental costs





Sample:

```
.....................................................
Match 1: 2016.10.06 - Thursday – 1-Environmental costs
Group 1: 2016.10.06 - Thursday – 1
Group 2: Environmental costs
```

Forum Contributor: therube

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Using: 2016.10.06 - Thursday – 1-Environmental costs

Match: (.*?\S)-(\S.*)

Where:

\1 = 2016.10.06 - Thursday – 1 \2 = Environmental costs

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1= <entire string>
3. ?	But not too Greedy (Non-Greedy or Lazy) Makes the expression optional – Zero Occurrence Match results in null value.	Capture Group 1 = null
4. \S	Match against Non-Whitespace until…. Non-Whitespace is any character except <space>	Capture Group 1 = 2
5. -	… <hyphen> (Not Captured)	Capture Group 1= 2016.10.06 - Thursday – 1
6. \S	Match against Non-Whitespace until…. Non-Whitespace is any character except <space>	Capture Group 2 = E
7. .*	Match against remainder of string	Capture Group 2 = Environmental costs

Tim Notes:

1. For Newbies, what this does is partially parse the string and uses the Replacement String to insert the <space>.

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

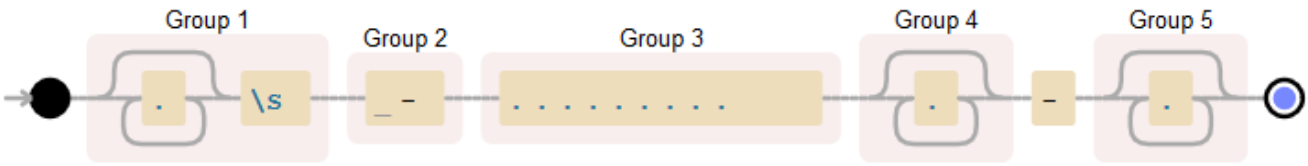
2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Tim Contribution –

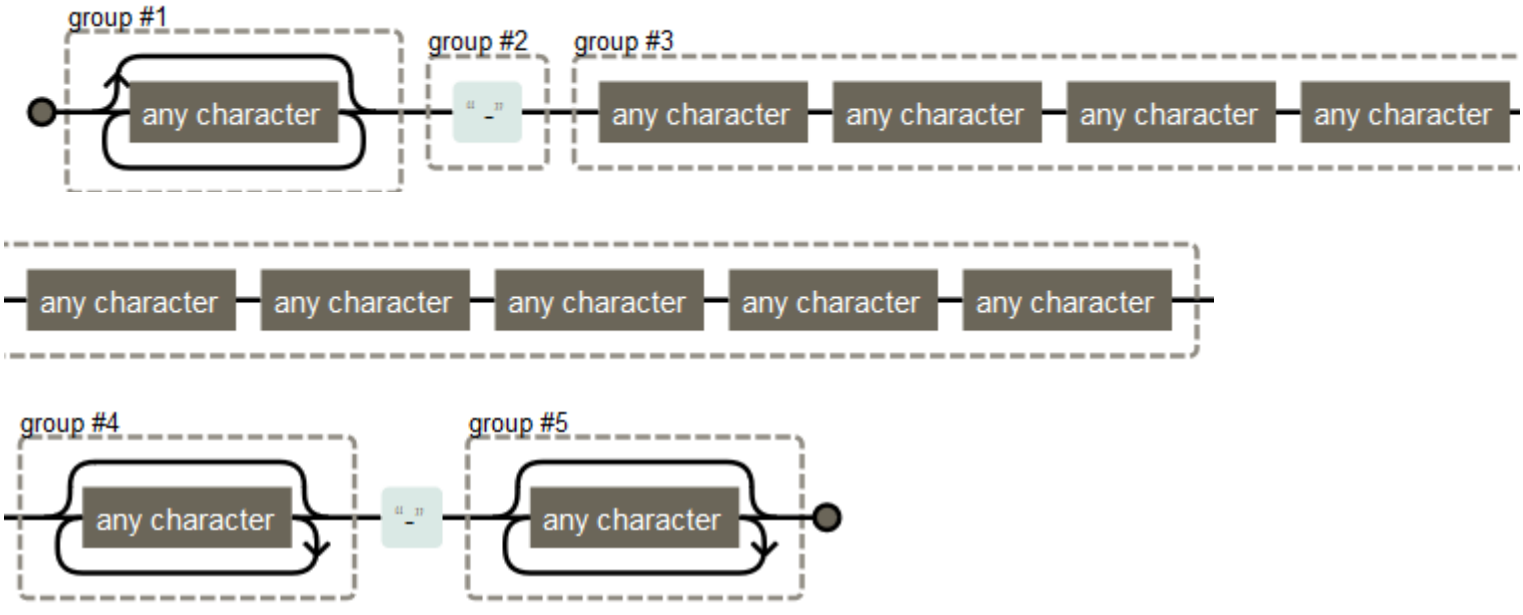
These are my own versions:

Match: (.*?)(-)(.....)(.*)-(.*)
Replace: \1\2\3\4 \5

Name	New Name
2016.10.06 - Thursday – 1-Environmental costs	2016.10.06 - Thursday – 1 Environmental costs
2016.11.29 - Monday - 5. Physics-5	2016.11.29 - Monday - 5. Physics 5



Expanded:



Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Match: (.*?)(-)(.....)(.*)-(.*)

Tim Contribution – cont.

Sample:

```
.....
Match 1: 2016.11.29 - Monday - 5. Physics-5
Group 1: 2016.11.29
Group 2:  -
Group 3:  Monday -
Group 4:  5. Physics
Group 5: 5
```

Sample:

```
.....
Match 1: 2016.10.06 - Thursday – 1-Environmental costs
Group 1: 2016.10.06
Group 2:  -
Group 3:  Thursday
Group 4:  – 1
Group 5: Environmental costs
```

Here I present my idea of a lazy RegEx (not to be confused with a Lazy Quantifier). It is unimaginative and very restrictive given the use of exact number of wildcards in Group 3. Shall I go on? Okay –so given all that – It works. It works as it is supposed to, on the pattern presented, and that's all that counts.

Sure, you may be a lousy Regular Expressions programmer, but that *isn't* the point. You may not be able to answer a poster's question by showing off your brilliance – but that *isn't* the point.

As you learn more about Regular Expressions, you may be able to write that RegEx that displays your talent – so far, from the look of the above, is defintitely well hidden. But newbie or not, the RegEx works to successfully rename the files in question and *that's* the point.

Tim Notes:

1. There is, however, another consideration – speed of processing. If the RegEx is going to be used on a small number of files. Fine. But if the processing will take place on a large number of files, then a badly coded RegEx could cost you in processing time. This is typically because of backtracking. I, myself is guilty of this, but my goal here is to teach you the tools to create your own RegEx. Proficiency will come with time.

2. No purpose in performing an analysis. This was just a simple illustration (although like most simple RegEx it will be a mess to analyze).

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

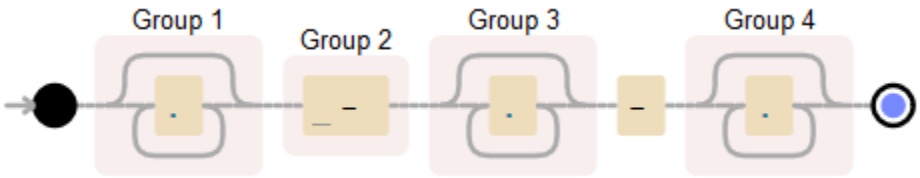
2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Tim Contribution – cont.

Here's another common mistake – and I, especially, am guilty of this- overwriting the code… You get so caught up in programming the RegEx – trying this, calculating that – that you just forget and stick something redundant in there. The above is a perfect example of this – it turns out the wildcards are not even needed. Here is the corrected RegEx:

Match: (.*?)(-)(.*)-(.*)
Replace: \1\2\3 \4

Name	New Name
2016.11.29 - Monday - 5. Physics-5	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	2016.10.06 - Thursday – 1 Environmental costs



Sample:

```
.....................................................
Match 1: 2016.10.06 - Thursday – 1-Environmental costs
Group 1: 2016.10.06
Group 2:  -
Group 3:  Thursday – 1
Group 4: Environmental costs
```

So, how do you find your mistakes? Well, for one – self analysis, much of what I am doing here. Take each bit of code and break it down. Find out what each expression does, or as I do, each part of the expression, and how it does it. Use Volume I, which has a very good RegEx manual in the Appendix, to define the various metacharacters and learn what they do. This book also contains a very extensive supplemental manual in the Appendix. There are also tools – both programs and online, that can help as well.

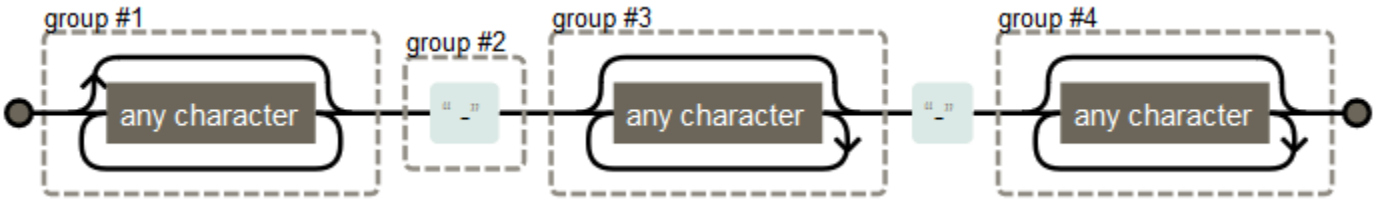
It is through the studying of the examples presented here and your own self analysis can you begin to understand what others are doing and learn at the same time what you can do. Yes, that is how I have learned RegEx coding and you can too. That is what this book is about – not a copy and paste method of doing what you want – because remember you are limited to the 'Pattern Matching' – and if the pattern of an example doesn't exactly fit what you want, then guess what? You need to learn a little to find out how to tweak it to your needs.

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Match: (.*?)(-)(.*)-(.*)



Tim Contribution – cont.

Where:

\1 = 2016.10.06 \2 = <space> <hyphen> <space> \3 = Thursday – 1 \4 = Environmental costs

Tim Analysis:

Using: 2016.10.06 - Thursday – 1-Environmental costs

1. .	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1= <entire string>
3. ?	But not too Greedy (Non-Greedy or Lazy) Makes the expression optional – Zero Occurrence Match results in null value.	Capture Group 1 = null
4. <space>	Match against the <space> followed by …	Capture Group 2 = <space> Changes Capture Group 1 = 2016.10.06
5. <hyphen>	.. <hyphen>	Capture Group 2 = <space> <hyphen>
6. .	Match against any character This is the <space> after '2016.10.06'	Capture Group 3 = <space>
7. *	Make it Greedy	Capture Group 3= <space> Thursday – 1- Environmental Environmental
8. -	Match against <hyphen> (Not Captured)	Changes Capture Group 1 = <space> Thursday – 1
9. .	Match against any character	Capture Group 4 = E
10. *	Make it Greedy	Capture Group 4 = Environmental costs

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

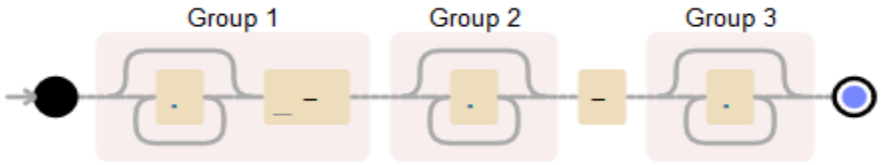
2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

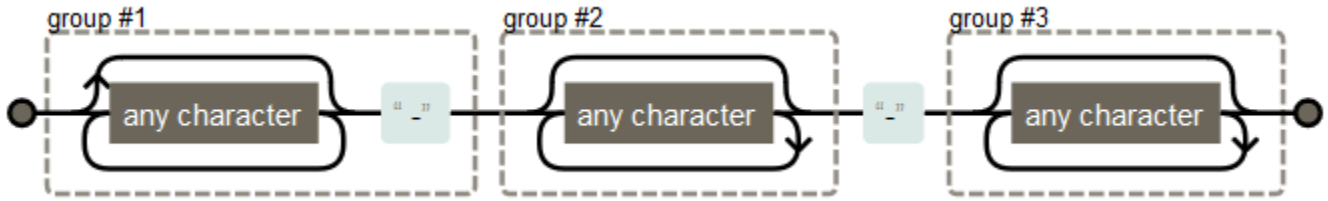
Tim Contribution – cont.

Another commonality you'll find I do in this book is overcapture, but this is deliberate– for example, there is no reason to capturing the <hyphen> in Capture Group 2 when it could have been done just as well in Capture Group 1 using:

Match: (.*? -)(.*)-(.*)
Replace: \1\2 \3

Name	New Name
2016.11.29 - Monday - 5. Physics-5	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	2016.10.06 - Thursday – 1 Environmental costs





Sample:

```
.....................................................
Match 1: 2016.10.06 - Thursday – 1-Environmental costs
Group 1: 2016.10.06 -
Group 2:  Thursday – 1
Group 3: Environmental costs
```

So why Did I? Answer - I find it best for illustrating parsing.

If I am short on Capture Groups, - Regular Expressions only allow a maximum of 9, then I can start cutting down by using Non-Marking Groups, or as I did here, combining and tweaking bringing the Capture Groups down from 5 to 3. I can also tweak to remove a capture from an expression or item that isn't going to be used in the Replacement String. And finally, instead of using a Capturing Group to capture an item such as a <space>, <hyphen>, <dash> etc., as was done here using Capture Group 2, instead, I could remove the capture and use the Replacement String to insert back the item.

Find Occurrences

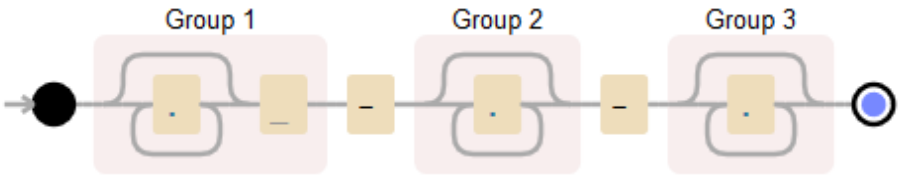
Replace last hyphen (in a 3 hyphen string) with <space> cont.

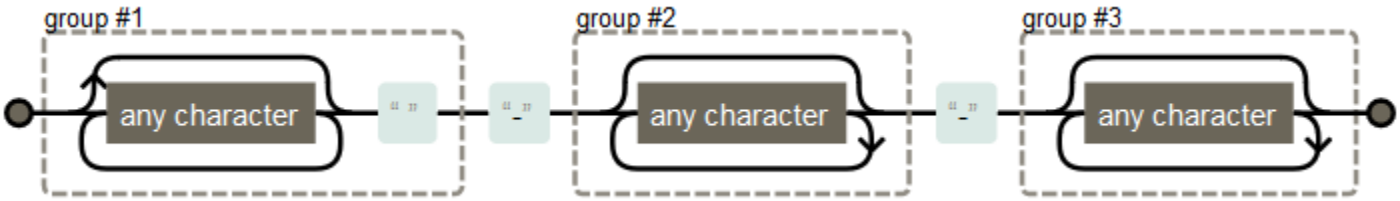
2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Tim Contribution – cont.

Instead of:		Use this:	
Match:	(.*?)(-)(.*)-(.*)	Match:	(.*?)-(.*)-(.*)
Replace:	\1\2\3 \4	Replace:	\1-\2 \3

Name	New Name
2016.11.29 - Monday - 5. Physics-5	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	2016.10.06 - Thursday – 1 Environmental costs





Sample:

```
.............................................
Match 1: 2016.11.29 - Monday - 5. Physics-5
Group 1: 2016.11.29
Group 2:  Monday - 5. Physics
Group 3: 5
```

Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

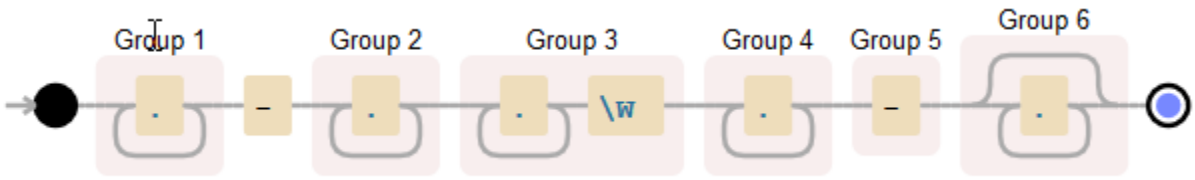
2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

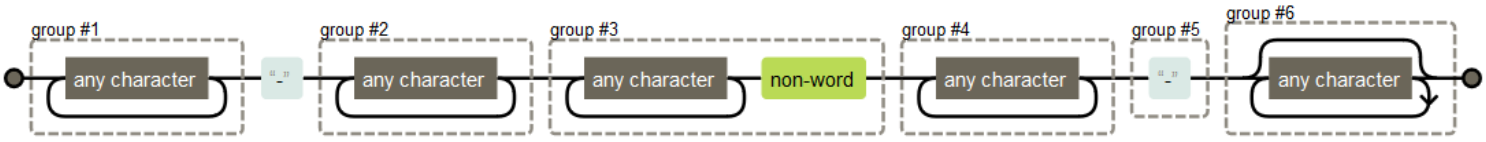
Tim Contribution – cont.

Here's another one that will work:

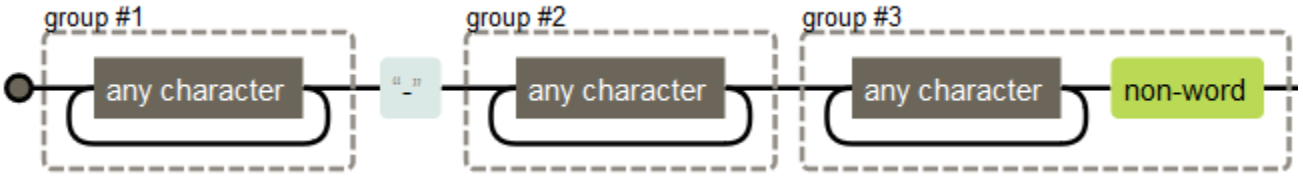
Match: (.+?)-(.+?)(.+?\W)(.+?)(-)(.*)
Replace: \1-\2\3\4 \6

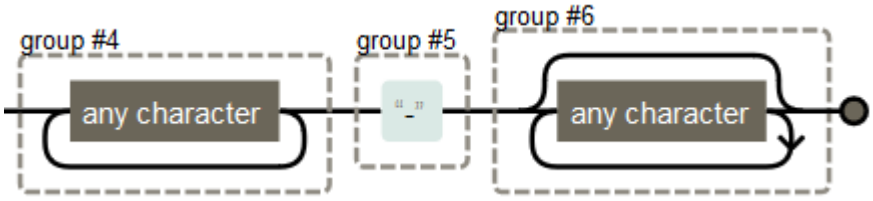
Name	New Name
2016.11.29 - Monday - 5. Physics-5	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday - 1-Environmental costs	2016.10.06 - Thursday - 1 Environmental costs





Expanded:





Find Occurrences

Replace last hyphen (in a 3 hyphen string) with <space> cont.

2016.11.29 - Monday - 5. Physics-5	to	2016.11.29 - Monday - 5. Physics 5
2016.10.06 - Thursday – 1-Environmental costs	to	2016.10.06 - Thursday – 1 Environmental costs

Tim Contribution – cont.

Match: (.+?)-(.+?)(.+?\W)(.+?)(-)(.*)

Sample:

```
Match 1: 2016.11.29 - Monday - 5. Physics-5
Group 1: 2016.11.29
Group 2:
Group 3: Monday
Group 4: - 5. Physics
Group 5: -
Group 6: 5
```

Sample:

```
Match 1: 2016.10.06 - Thursday – 1-Environmental costs
Group 1: 2016.10.06
Group 2:
Group 3: Thursday
Group 4: – 1
Group 5: -
Group 6: Environmental costs
```

Tim Notes:

1. \W is a Negated Class of a Word Character metacharacter (the opposite of \w). It will match against all Non-Word characters. This includes any characters *except* A-Z, a-z, 0-9 and the underscore character.

2. Word Characters are those characters used to form words and *include* A-Z, a-z, 0-9 and the underscore character. The Word Character Metacharacter is \w.

Where:

\1 = 2016.10.06 \2 = <space> \3 = Thursday <space> \4 = - 1
\5 = - \6 = Environmental costs

Tim Notes:

1. Do you see where a Capturing Group can be eliminated? Capture Group 5 captures a <hyphen>. Capture Group 5 is not included in the Replacement String and in fact, it is this <hyphen> we wish to be removed and replaced by the <space> in the Replacement String. So the RegEx can be tweaked to: (.+?)-(.+?)(.+?\W)(.+?)-(.*).

2. I'll leave the rest of the analysis to you. Hopefully I have provided enough information and tools to do your own.

Find Occurrences

Move everything prior to '-' to the end of file name

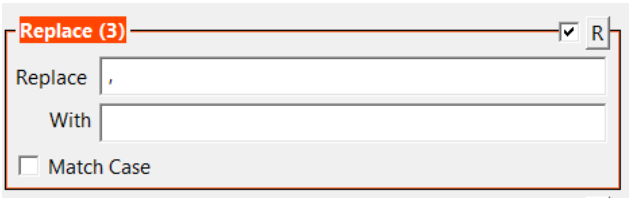
Files with comma –

Angela Hewitt - Beethoven, Piano Sonatas Vol. 1 (2006) [SACD] {Hyperion}
to
Beethoven - Piano Sonatas Vol. 1 (2006) [SACD] {Hyperion} (Angela Hewitt)

Files without comma –

Duo Lechner Tiempo - La Belle Epoque (2009) [SACD] {Pure Music}
to
La Belle Epoque (2009) [SACD] {Pure Music} (Duo Lechner Tiempo)

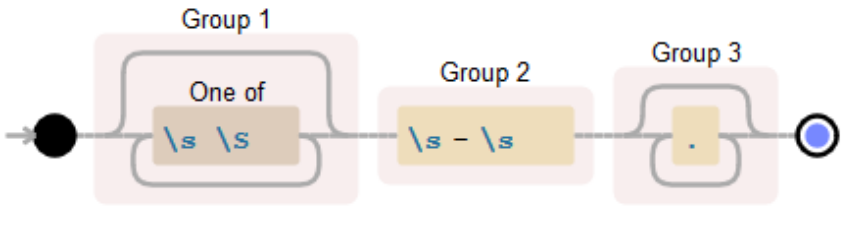
1. 'Section #3: Replace'

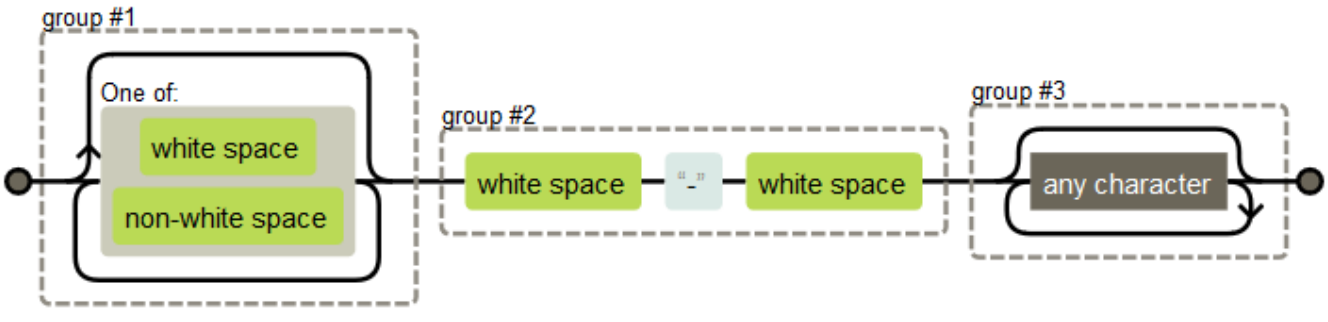


Replace: , (comma)
With: <space>

Match: ([\s\S]*?)(\s-\s)(.*)
Replace: \3 (\1)

Name	New Name
Angela Hewitt - Beethoven, Piano Sonatas Vol. 1 (2006) [SACD] {Hyperion}	Beethoven Piano Sonatas Vol. 1 (2006) [SACD] {Hyperion} (Angela Hewitt)
Duo Lechner Tiempo - La Belle Epoque (2009) [SACD] {Pure Music}	La Belle Epoque (2009) [SACD] {Pure Music} (Duo Lechner Tiempo)





Find Occurrences

Move everything prior to '-' to the end of file name cont.

Match: ([\s\S]*?)(\s-\s)(.*)

Sample:

```
.....................................................................
Match 1: Duo Lechner Tiempo - La Belle Epoque (2009) [SACD] {Pure Music}
Group 1: Duo Lechner Tiempo
Group 2:  -
Group 3: La Belle Epoque (2009) [SACD] {Pure Music}
```

Tim Notes:

1. The run in the sample will be the same for files with or without commas because the RegEx doesn't handle this, 'Section #2: Replace' does.

Forum Contributor: therube

Find Occurrences

Move everything prior to '-' to the end of file name cont.

Using: Duo Lechner Tiempo - La Belle Epoque (2009) [SACD] {Pure Music}

Match: ([\s\S]*?)(\s-\s)(.*)

Where:

\1 = Duo Lechner Tiempo \2 = <space><hyphen><space>
\3 = La Belle Epoque (2009) [SACD] {Pure Music}

Tim Analysis:

1. [\s\S]	Match against a class consisting of a Whitespace metacharacter and a Non-Whitespace metacharacter.	Capture Group 1 = D
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy Zero Occurrence Match = null	Capture Group 1 = null
4. \s	Match against a <space> character followed by ….	Capture Group 2 = <space> Changes Capture Group 1 = Duo
5. -	.. <hyphen. followed by ….	Capture Group 2 = <space> <hyphen> Changes Capture Group 1 = Duo Lechner Tiempo
6. \s	.. <space>	Capture Group 2 = <space><hyphen><space>
7. .*	Match against remainder of string	Capture Group 3 = La Belle Epoque (2009) [SACD] {Pure Music}

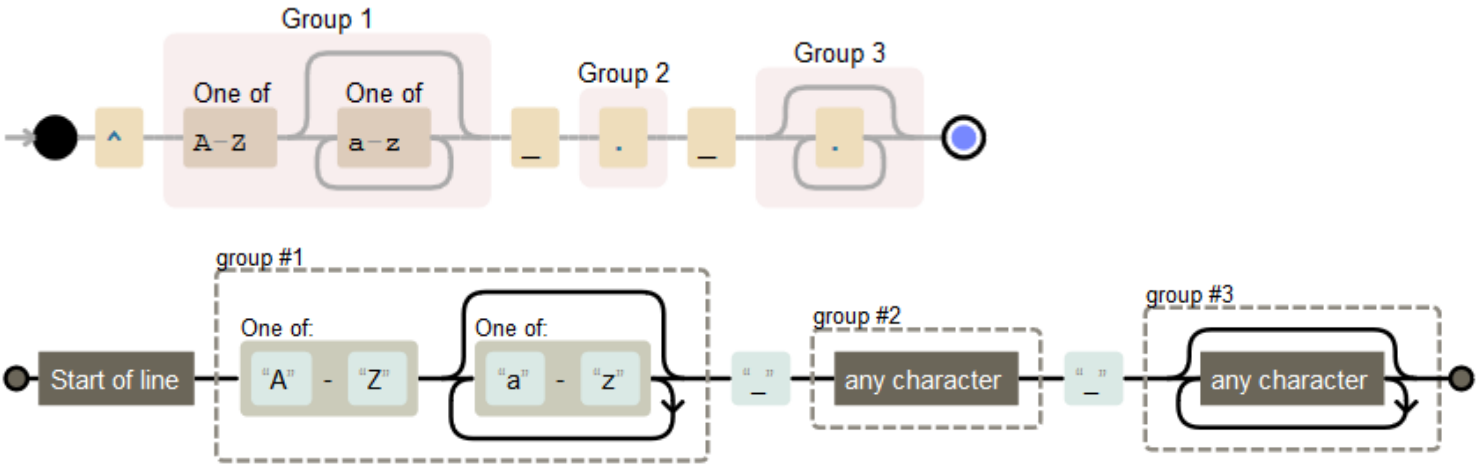
Find Occurrences

Move Person's Name to end, eliminate the underscore after initial and add underscore after

Kohn_L_I_12912185_1.jpg to I_12912185_1_Kohn_L.jpg

Match: ^([A-Z][a-z]*)_(.){1}_(.*)
Replace: \3_\1_\2

Name	New Name
Kohn_L_I_12912185_1.jpg	I_12912185_1_Kohn_L.jpg

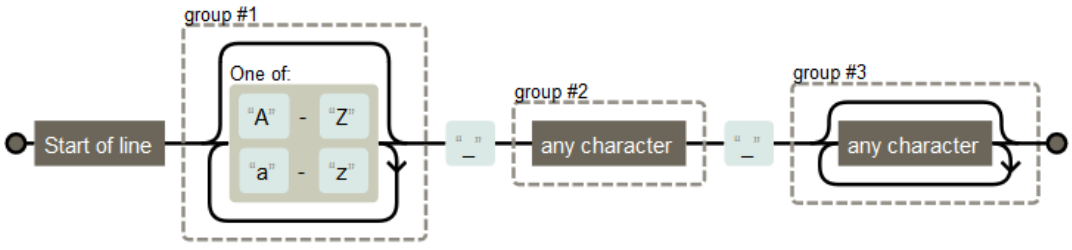


Sample:

```
Match 1: Kohn_L_I_12912185_1
Group 1: Kohn
Group 2: L
Group 3: I_12912185_1
```

Tim Notes:

1. By Using the (.){1}, it forces Capture Group 2 to limit itself only a to a single character, the 'L', but really not required because the single dot would only match against a single character. The {1} means the minimum value for match repetitions = 1.
2. Looks more complicated than it is, i.e., [A-Z] [a-z] simply means to match against an upper or lower case letter. It could also have been coded as, [A-Za-z]. Here is the visualization:



Forum Contributor: therube

Find Occurrences

Move Person's Name to end, eliminate the underscore after initial and add underscore after # cont.

Using: Kohn_L_I_12912185_1.jpg

Match: ^([A-Z][a-z]*)_(.){1}_(.*)

Where:

\1 = Kohn \2 = L \3 = I_12912185_1

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [A-Z][a-z]	[A-Za-z] Match against an upper or lowercase letter.	Capture Group 1 = K
3. *	Make it Greedy	Capture Group 1 = Kohn
4. _	Match against underscore (Not Captured)	Capture Group 1 = Unchanged
5. .	Match against any character	Capture Group 2 = L
6. {1}	Repeat last expression to match 1 iteration	
7. _	Match against underscore (Not Captured)	All Capture Groups = Unchanged
8. .*	Match against remaining string	Capture Group 3 = I_12912185_1

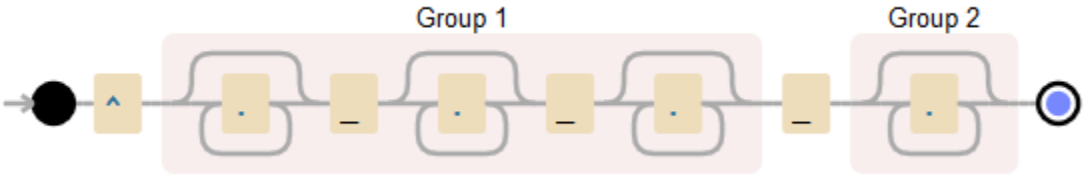
Find Occurrences

Replace third underscore with a space

john_doe_13456_20_m_v5 to john_doe_13456 20_m_v5

Match: ^(.*?_.*?_.*?)_(.*)
Replace: \1 \2

Name	New Name
john_doe_13456_20_m_v5	john_doe_13456 20_m_v5



Sample:

```
.....................................
Match 1: john_doe_13456_20_m_v5
Group 1: john_doe_13456
Group 2: 20_m_v5
```

Forum Contributor: therube

Find Occurrences

Replace third underscore with a space cont.

Using: john_doe_13456_20_m_v5

Match: ^(.*?_.*?_.*?)_(.*)

Where:

\1 = john_doe_13456 \2 = 20_m_v5 <space>

Tim Analysis:

1.	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = j
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. _	Match against the underscore	Capture Group 1 = john_
6..	Match against any character	Capture Group 1 = d
7. *	Make it Greedy	Capture Group 1 = <entire string>
8. ?	But not too Greedy	Capture Group 1 = john_
9. _	Match against the underscore	Capture Group 1 = john_doe_
10..	Match against any character	Capture Group 1 = 1
11. *	Make it Greedy	Capture Group 1 = <entire string>
12. ?	But not too Greedy	Capture Group 1 = john_doe_
13. _	Match against the underscore (Not Captured)	Changes Capture Group 1 = john_doe_13456
14. .*	Match against remaining string	Capture Group 2 = 20_m_v5<space>

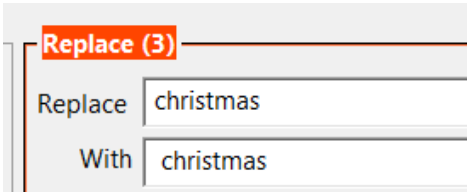
Find Occurrences

Add a space to certain files before/after a certain word

2016christmasdinner.mp4 to 2016 christmas dinner.mp4

Two methods:

(1) 'Section # 3: Replace' -



Replace: christmas
With: <space> christmas <space>

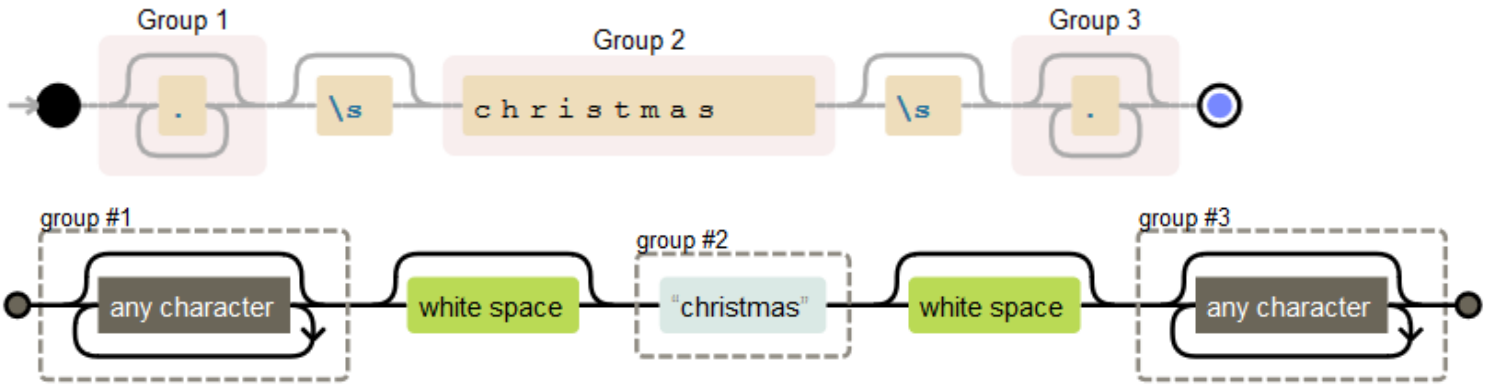
Name	New Name
2016christmasdinner.mp4	2016 christmas dinner.mp4

(2) RegEx

Match: (.*)\s?(christmas)\s?(.*)
Replace: \1 \2 \3

Name	New Name
2016christmasdinner.mp4	2016 christmas dinner.mp4

This also works if there isn't always a <space> both in front and behind the word 'christmas'.



Sample:

```
................................
Match 1: 2016christmasdinner
Group 1: 2016
Group 2: christmas
Group 3: dinner
```

Forum Contributor: KenP

Find Occurrences

Add a space to certain files before/after a certain word cont.

Using: 2016christmasdinner.mp4

Match: (.*)\s?(christmas)\s?(.*)

Where:

\1 = 2016 \2 = christmas \3 = dinner

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \s	Match against <space> character (Not Captured) = False	
4. ?	Make it optional (if no <space> character present) = True	
5. christmas	Match against string, 'christmas'	Capture Group 2 = christmas Changes Capture Group 1 = 2016
6. \s	Match against <space> character (Not Captured) = False	
7. ?	Make it optional (if no <space> character present) = True	
8. .*	Match against remaining string	Capture Group 3 = dinner

Find Occurrences

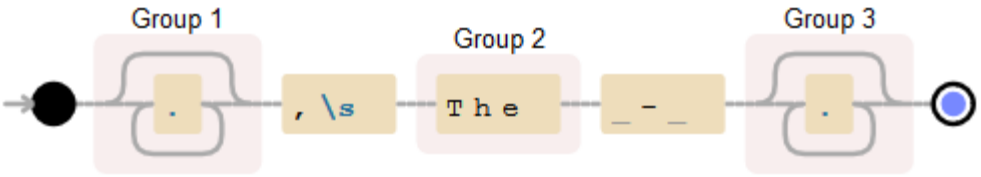
Move the word 'The' in first Instance Only to front of Title and Remove comma

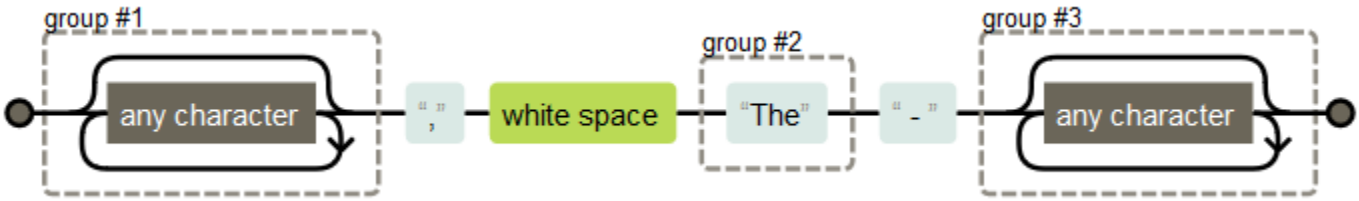
Wild Thornberrys, The - Animal Adventures
to
The Wild Thornberries - Animal Adventures

Secret of Googol 2b, The - Reshaping Googol Under the Ocean
to
The Secret of Googol 2b - Reshaping Googol Under the Ocean

Match: (.*),\s(The) - (.*)
Replace: \2 \1 - \3

Name	New Name
Secret of Googol 2b, The - Reshaping Googol Under the Ocean	The Secret of Googol 2b - Reshaping Googol Under the Ocean
Wild Thornberrys, The - Animal Adventures	The Wild Thornberrys - Animal Adventures





Sample:

```
.....................................................
Match 1: Wild Thornberrys, The - Animal Adventures
Group 1: Wild Thornberrys
Group 2: The
Group 3: Animal Adventures
```

Forum Contributor: KenP

Find Occurrences

Move the word 'The' in first Instance Only to front of Title and Remove comma cont.

Using: Wild Thornberrys, The - Animal Adventures

Match: (.*),\s(The) - (.*)

Where:

\1 = Wild Thornberrys \2 = The \3 = Animal Adventures

Tim Analysis:

1..	Match against any character	Capture Group 1 = W
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ,	Match against <comma> (Not Captured) followed by …	Changes Capture Group 1 = Wild Thornberrys
4. \s	.. Match against a <space> character (Not Captured) followed by …	
5. The	.. Match against a string 'The' followed by …	Capture Group 2 = The
6. <space>	Match against <space> character (Not Captured) followed by …	All Capture Groups = Unchanged
7. <hyphen>	.. Match against <hyphen> (Not Captured) followed by …	
8. <space>	.. Match against <space> character (Not Captured)	
9. .*	Match against remaining string	Capture Group 3 = Animal Adventures

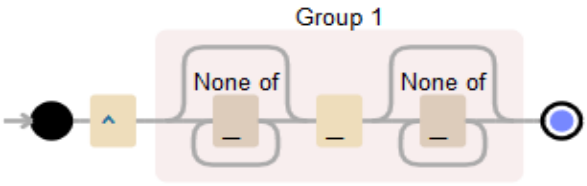
Find Occurrences

Remove all Characters after x occurrences of a character (Remove all after 2nd occurrence of underscore)

SEQ1(text of varying length)_SEQ2(more text)_SEQ3(some more text)_SEQ4(all sorts of text length).pdf
to
SEQ1(text of varying length)_SEQ2(more text).pdf

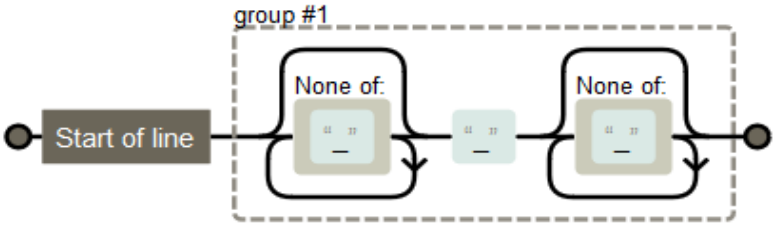
Match: ^([^_]*_[^_]*)
Replace: \1

Name	New Name
SEQ1(text of varying length)_SEQ2(more text)_SEQ3(some more...	SEQ1(text of varying length)_SEQ2(more text).pdf



Sample:

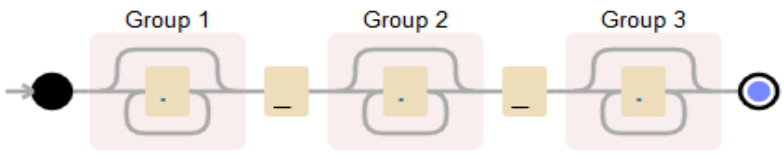
```
Match 1: SEQ1(text of varying length)_SEQ2(more text)
Group 1: SEQ1(text of varying length)_SEQ2(more text)
```

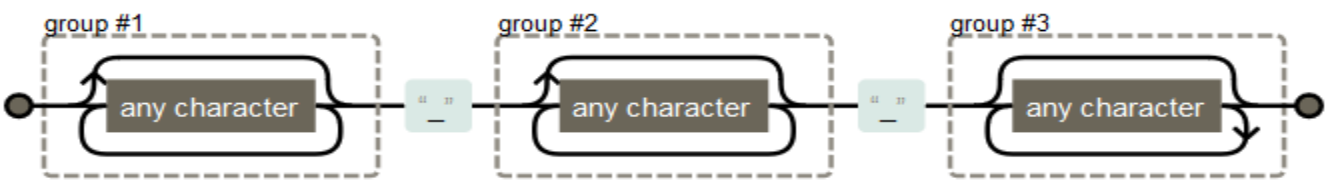


Modified - This will accomplish the same thing:

Match: (.*?)_(.*?)_(.*)
Replace: \1_\2

Name	New Name
SEQ1(text of varying length)_SEQ2(more text)_SEQ3(some more...	SEQ1(text of varying length)_SEQ2(more text).pdf





Find Occurrences

Remove all Characters after x occurrences of a character (Remove all after 2nd occurrence of underscore) cont.

Match: (.*?)_(.*?)_(.*)

Sample:

```
.....................................................................................................
Match 1: SEQ1(text of varying length)_SEQ2(more text)_SEQ3(some more text)_SEQ4(all sorts of text length)
Group 1: SEQ1(text of varying length)
Group 2: SEQ2(more text)
Group 3: SEQ3(some more text)_SEQ4(all sorts of text length)
```

Tim Notes:

1. Although the poster was concerned that each part had text of varying length, the RegEx doesn't take this in consideration and is only involved with the removal of extraneous information found after the specified underscore. Thus the varying lengths become a moot point.

Forum Contributors: KenP, therube

Find Occurrences

Remove all Characters after x occurrences of a character (Remove all after 2nd occurrence of underscore) cont.

Using: SEQ1(text of varying length)_SEQ2(more text)_SEQ3(some more text)_SEQ4(all sorts of text length).pdf

Original RegEx -

Match: ^([^_]*_[^_]*)

Where:

\1 = SEQ1(text of varying length)_SEQ2(more text)

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^_]	Match against a Negated Class consisting of an underscore character	Capture Group 1 = S
3. *	Make it Greedy	Capture Group 1 = SEQ1(text of varying length)
4. _	Match against underscore character	Capture Group 1 = SEQ1(text of varying length)_
5. [^_]	Match against a Negated Class consisting of an underscore character Match against anything but an underscore.	Capture Group 1 = SEQ1(text of varying length)_S
6. *	Make it Greedy	Capture Group 1 = SEQ1(text of varying length)_SEQ2 (more text)

Tim Notes:

1. Using the Negated class of the underscore in step 5 skips over the underscore after 'length)_' and matches at the 'S' of 'SEQ2'. Step 6 makes it Greedy and matches to the next underscore character preceding 'SEQ3'. The Remainder of the string is discarded (not captured).

Find Occurrences

Remove all Characters after x occurrences of a character (Remove all after 2nd occurrence of underscore) cont.

Using: SEQ1(text of varying length)_SEQ2(more text)_SEQ3(some more text)_SEQ4(all sorts of text length).pdf

Modified RegEx -

Match: (.*?)_(.*?)_(.*)

Where:

\1 = SEQ1(text of varying length) \2 = SEQ2(more text)
\3 = SEQ3(some more text)_SEQ4(all sorts of text length

Tim Analysis:

1. .	Match against any character This is the 'S' of 'SEQ1'	Capture Group 1 = S
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	Make the last expression optional	Capture Group 1 = BOL
5. _	Match against underscore character (Not Captured)	Changes Capture Group 1 = SEQ1(text of varying length)
6. .	Match against any character This is the 'S' of 'SEQ2'	Capture Group 2 = S
7. *	Make it Greedy Current position = EOL	Capture Group 2 = SEQ2(more text)_SEQ3(some more text)_ SEQ4(all sorts of text length)
8. ?	Make the last expression optional Backtracks to position before 'SEQ2'	Capture Group 2 =null Capture Group 1 = Unchanged
9. _	Match against underscore character (Not Captured) Moves forward to match against underscore preceding 'SEQ3'	Changes Capture Group 2 = SEQ2(more text)
10. *	Make it Greedy	Capture Group 3 = SEQ3(some more text)_ SEQ4(all sorts of text length)

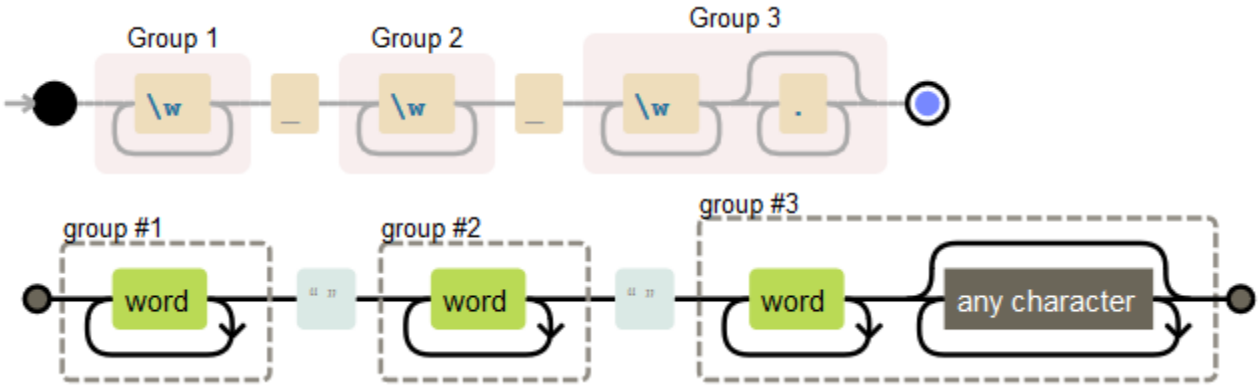
Find Occurrences

Insert 'hyphen' after second word in string

My Text Document has no Dash.txt to My Text - Document has no Dash.txt
My Document has no Dash.txt to My Document - has no Dash.txt

Match: (\w+) (\w+) (\w+.*)
Replace: \1 \2 - \3

Name	New Name
My Text Document has no Dash.txt	My Text - Document has no Dash.txt
My Document has no Dash.txt	My Document - has no Dash.txt



Sample:

```
.....................................
Match 1: My Text Document has no Dash
Group 1: My
Group 2: Text
Group 3: Document has no Dash
```

Tim Notes:

1. A good example of using Word Metacharacters \w to parse string
2. Dash is the wrong word – <hyphen> is the correct word , but sample used 'Dash' in title and that is what I used.
3. Not universal – Fails on: 10 C-Boy's Blues.mp3
4. Unfortunately, just using \w is not enough.

Forum Contributor: therube

Find Occurrences

Insert 'hyphen' after second word in string cont.

Using: My Text Document has no Dash.txt

Match: (\w+) (\w+) (\w+.*)

Where:

\1 = My \2 = Text \3 = Document has no Dash

Tim Analysis:

1. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = M
2. +	Make it Greedy	Capture Group 1 = My
3. <space>	Match against a <space> character (Not Captured)	
4. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 2 = T
5. +	Make it Greedy	Capture Group 2 = Text
6. <space>	Match against a <space> character (Not Captured)	
7. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = D
8. +	Make it Greedy	Capture Group 3 = Document
9. .*	Match against remaining string	Capture Group 3 = Document has no Dash

Find Occurrences

Insert 'hyphen' after second word in string cont.

Tim Contribution –

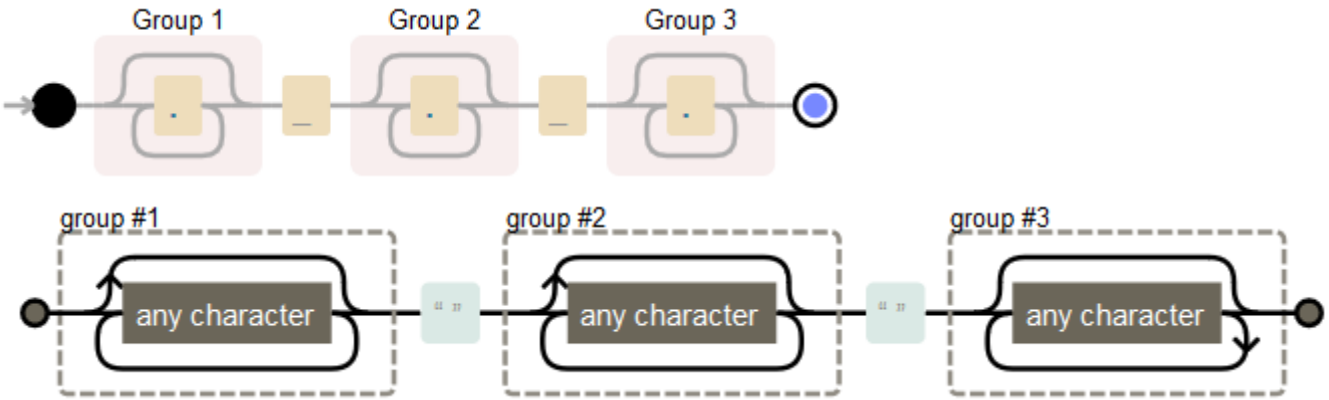
This one is –

Additional sample:

10 C-Boy's Blues

Match: (.*?) (.*?) (.*)
Replace: \1 \2-\3

Name	New Name
My Text Document has no Dash.txt	My Text-Document has no Dash.txt
My Document has no Dash.txt	My Document-has no Dash.txt
10 C-Boy's Blues.mp3	10 C-Boy's-Blues.mp3



Sample:

```
.............................
Match 1: 10 C-Boy's Blues
Group 1: 10
Group 2: C-Boy's
Group 3: Blues
```

Find Occurrences

Insert 'hyphen' after second word in string cont.

Tim Contribution – cont.

Using: 10 C-Boy's Blues

Match: (.*?) (.*?) (.*)

Where:

\1 = 10 \2 = C-Boy's \3 = Blues

Tim Analysis:

1. .	Match against any character This is the	Capture Group 1 = 1
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	Make the last expression optional	Capture Group 1 = BOL
5. <space>	Match against <space> character (Not Captured) This is the <space> after '10'	Changes Capture Group 1 =10
6. .	Match against any character	Capture Group 2 = C
7. *	Make it Greedy Current position = EOL	Capture Group 2 = C-Boy's Blues
8. ?	Make the last expression optional Backtracks to position before 'C'	Capture Group 2 = null
9. <space>	Match against <space> character (Not Captured) Moves forward to match against <space> preceding 'Blues'	Changes Capture Group 2 = C-Boy's
10. .*	Match against remaining string	Capture Group 3 = Blues

Find Occurrences

Copy Characters to Start of Name (Good explanation of \w)

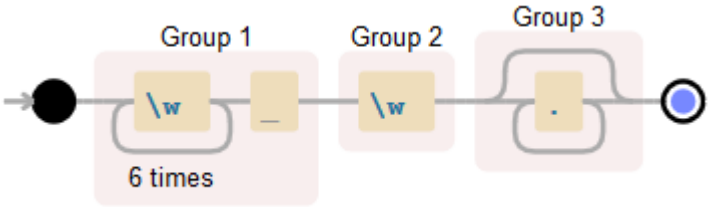
YYYYMM CustomerName Statement to CYYYYMM CustomerName Statement

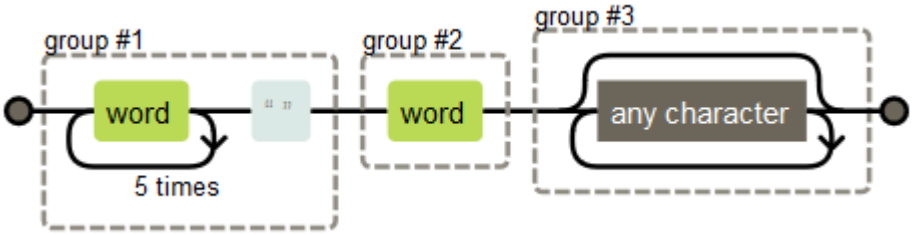
Observations:

1. Copy the 8th character from a file name and insert it at the 1st position.

Match: (\w{6})(\w)(.*)
Replace: \2\1\2\3

Name	New Name
YYYYMM CustomerName Statement	CYYYYMM CustomerName Statement





Sample:

```
.....................................
Match 1: YYYYMM CustomerName Statement
Group 1: YYYYMM
Group 2: C
Group 3: ustomerName Statement
```

Find Occurrences

Copy Characters to Start of Name (Good explanation of \w) cont.

Using: YYYYMM CustomerName Statement

Match: (\w{6})(\w)(.*)

Where:

\1 = YYYYMM \2 = C \3 = ustomerName Statement

Contributor and *Tim Analysis:*

1. \w{6}	Capture the first 6 characters before the one you want copied The \w means any letter or number (called A Word Metacharacter)	Capture Group 1 = YYYYMM
2. <space>	Match against the <space> Position = 7	Capture Group 1 = YYYYMMM<space>
3. \w	Match against the letter you want to copy. Position = 8	Capture Group 2 = C
4. .*	Capture the remainder of the string	Capture Group 3 = ustomerName Statement

To better understand, here's how it breaks down:

(I've used | to indicate the ends of text segments)

In the Match it gives the following groups

	|YYYYMM |	|C|	|ustomerName Statement|
	(\w{6})	(\w)	(.*)
Group	\1	\2	\3

In Replace string they are re-ordered:

	|C|	|YYYYMM |	|C|	|ustomerName Statement|
Group	\2	\1	\2	\3

Obviously, you can vary the number of characters in each block to suit your needs

Forum Contributor: Glenn

Find Occurrences

Remove all text after the 'last' occurrence of T# when followed by hyphen

14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
to
14692-2A-010-A-24-0-T2

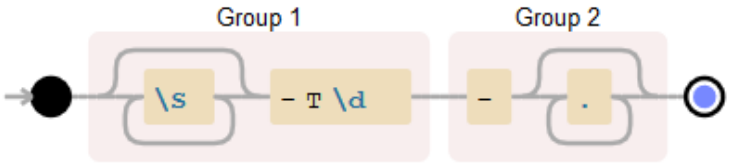
14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6
to
14692-2A-1100-A-32-5-T6

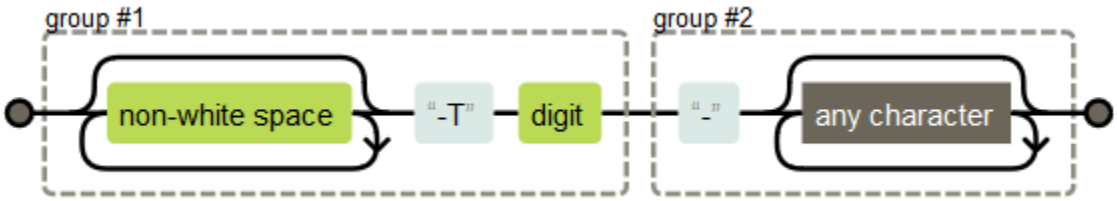
Observations:

1. Remove all text after the 'last' occurrence of T# when followed by <hyphen>

Match: (\S*-T\d)(-.*)
Replace: \1

Name	New Name
14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	14692-2A-010-A-24-0-T2
14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6	14692-2A-1100-A-32-5-T6





Sample:

```
.....................................................
Match 1: 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
Group 1: 14692-2A-010-A-24-0-T2
Group 2: -62-14692-2A-010-A-24-0 T2
```

Forum Contributor: KenP

Find Occurrences

Remove all text after the 'last' occurrence of T# when followed by hyphen cont.

Using: 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6

Match: (\S*-T\d)(-.*)

Where:

\1 = 14692-2A-1100-A-32-5-T6 \2 = 109-14692-2A-1100-A-32-5 T6

Tim Analysis:

1. \S	Match against first Non-Whitespace metacharacter.	Capture Group 1 = 1 (as in 14692)

Tim Notes:

1. Non-White space metacharacter refers to anything *but* a whitespace character. Whitespace characters in BRU PCRE (the RegEx engine used) are <space> characters – so anything but a <space>

2. *	Make it Greedy and match against .. until….	
3. -	.. <hyphen> character followed by…	
4. T	'T' character followed by…	Capture Group 1 = 14692-2A-1100-A-32-5 T 14692-2A-010-A-24-0 T
5. \d	.. a single numeric digit	Capture Group 1 = 14692-2A-1100-A-32-5 T6 14692-2A-010-A-24-0 T2
6. -	Match against next <hyphen> a. The <hyphen> immediately following the 'T6' and 'T2'	Capture Group 2 = -
7. .*	Match against remainder of string	Capture Group 2 = -109-14692-2A-1100-A-32-5 T6 -62-14692-2A-010-A-24-0 T2

Find Occurrences

Add Text after First or Last Space in String

Samples:

Single space filenames	Multiple Space filenames	Won't match (no <spaces>)
00- songname.mp3	0 123 2 abc 3 456 7.txt	out4.txt
test.pdf	Lesson 25 DNA	abc123.....txt
Lesson 0		a1
t00 -songname.mp3		

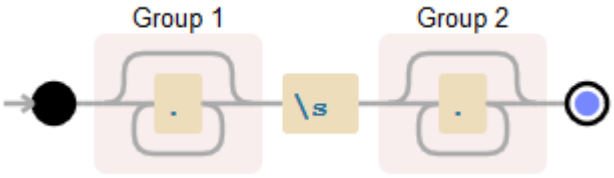
First Space:

Match: (.*?)\s(.*)
Replace: \1 {Inserted Text} \2

Name	New Name
00- songname.mp3	00- {Inserted Text} songname.mp3
out4.txt	out4.txt
test.pdf	{Inserted Text} test.pdf
Lesson 0	Lesson {Inserted Text} 0
abc123.....txt	abc123.....txt
0 123 2 abc 3 456 7.txt	0 {Inserted Text} 123 2 abc 3 456 7.txt
t00 -songname.mp3	t00 {Inserted Text} -songname.mp3
Lesson 25 DNA	Lesson {Inserted Text} 25 DNA
a1	a1

Sample:

```
Match 1: Lesson 25 DNA
Group 1: Lesson
Group 2: 25 DNA
```



Find Occurrences

Add Text after First or Last Space in String cont.

Samples:

Single space filenames	Multiple Space filenames	Won't match (<no spaces>)
00- songname.mp3	0 123 2 abc 3 456 7.txt	out4.txt
test.pdf	Lesson 25 DNA	abc123.....txt
Lesson 0		a1
t00 -songname.mp3		

Added Samples:

Chapter1 Lesson2 DNA
this is a test 01-Whip it Devo.wav

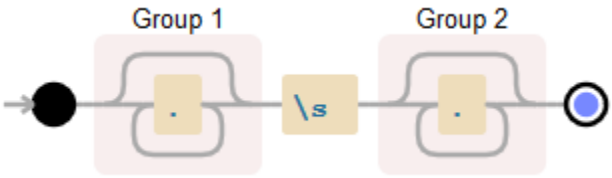
Last Space:

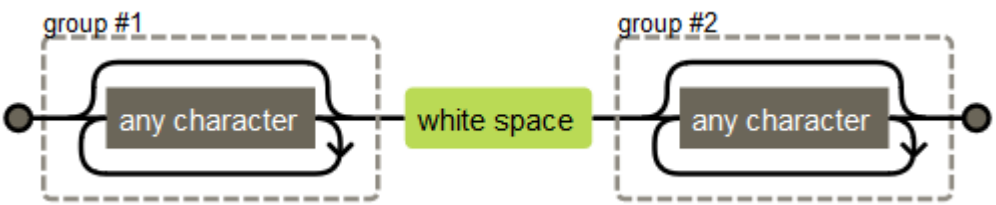
Match: (.*)\s(.*)
Replace: \1 {Inserted Text} \2

Name	New Name
00- songname.mp3	00- {Inserted Text} songname.mp3
out4.txt	out4.txt
test.pdf	{Inserted Text} test.pdf
Lesson 0	Lesson {Inserted Text} 0
abc123.....txt	abc123.....txt
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 {Inserted Text} 7.txt
Chapter1 Lesson2 DNA	Chapter1 Lesson2 {Inserted Text} DNA
t00 -songname.mp3	t00 {Inserted Text} -songname.mp3
Lesson 25 DNA	Lesson 25 {Inserted Text} DNA
a1	a1
this is a test 01-Whip it Devo.wav	this is a test 01-Whip it {Inserted Text} Devo.wav

Sample:

```
.............................
Match 1: Chapter1 Lesson2 DNA
Group 1: Chapter1 Lesson2
Group 2: DNA
```





Find Occurrences

Add Text after First or Last Space in String cont.

First Space:

Using: Lesson 25 DNA

Match: (.*?)\s(.*)

Where:

\1 = Lesson \2 = 25 DNA

Tim Analysis:

1. .	Match against any character	Capture Group 1 = L
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	Make the last expression optional	Capture Group 1 = BOL
5. \s	Match against <space> character (Not Captured) This is the <space> after 'Lesson'	Changes Capture Group 1 = Lesson
6. .*	Match against remaining string	Capture Group 2 = 25 DNA

Last Space:

Using: Chapter1 Lesson2 DNA

Match: (.*)\s(.*)

Where:

\1 = Chapter1 Lesson2 \2 = DNA

Tim Analysis:

1. .	Match against any character	Capture Group 1 = C
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \s	Match against <space> character (Not Captured) Backtracking. This is the <space> after 'Lesson2'	Changes Capture Group 1 = Chapter 1 Lesson2
4. .*	Match against remaining string	Capture Group 2 = DNA

Forum Contributor: information not available

Find Occurrences

Replace a Blank <space> with Text

8259438 some text <space>w f- 00086 jerry022 0 123 2 abc 3 456 7

044514_Freeware made by Emiel Wieldraaijer

Observations:

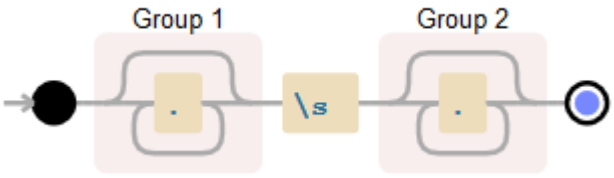
1. Parse string
2. Remove first <space>
3. Use Replacement String to add any characters or text in place of the missing <space>
4. Pattern Established: <text optional> <space> <text>

Match: (.*?)\s(.*)
Replace: \1 - \2

Name	New Name
8259438 some text	8259438 - some text
0 123 2 abc 3 456 7	0 - 123 2 abc 3 456 7
044514_Freeware made by Emiel Wieldraaijer	044514_Freeware - made by Emiel Wieldraaijer
w f- 00086 jerry022	- w f- 00086 jerry022

Sample:

```
.............................
Match 1: 0 123 2 abc 3 456 7
Group 1: 0
Group 2: 123 2 abc 3 456 7
```



Where:

\1 = 0 \2 = 123 2 abc 3 456 7

Tim Analysis:

1..	Match against any character	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = 0
4. \s	Match against <space> character (Not Captured) This is the <space> after '0'	Capture Group 1 = Unchanged
5. .*	Match against remaining string	Capture Group 2 = 123 2 abc 3 456 7

Forum Contributor: information not available

Find Occurrences

Replace the "." between Letters with a <space>, Ignore "." between Numbers

My.File.v.1.2.345	to	My File v.1.2.345
my.File2.v1.2.345	to	my File2 v1.2.345
this.Is.my.File.v.1.2.345	to	this Is my File v.1.2.345

Tim Contribution –

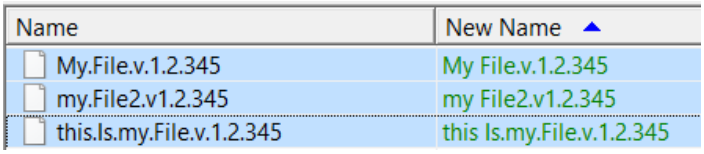
Match: (.*?)((?<=[A-Za-z])\.(?=[A-Za-z]))(.+)
Replace: \1 \3

Observations:

1. I used a RegEx to parse those 'words that had <dot> between them'. In order to be a valid match, the <dot> character had to come between two words and not a numeric digit previous or after.

2. Meant for multiple runs. Each run will isolate a word and a <dot> character. That word is then renamed using a <space> to replace the <dot> character.

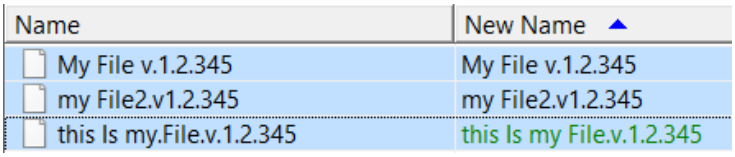
First Run:



Second Run:

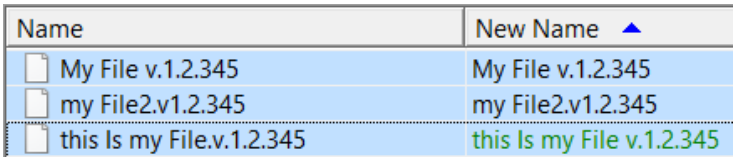
The second file only required one run.

Third Run:

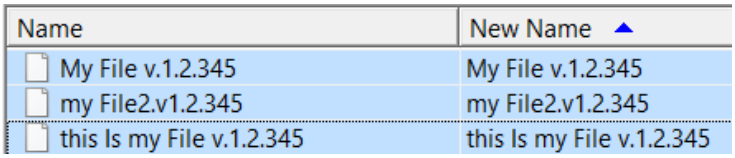


The first file was not renamed because the string was exhausted – meaning, no more Matches possible, by the second run.

Fourth Run:



After Fourth run, all strings exhausted:



Based on a RegEx originally by Forum Contributor: Glenn

Find Occurrences

Replace the "." between Letters with a <space>, Ignore "." between Numbers cont.

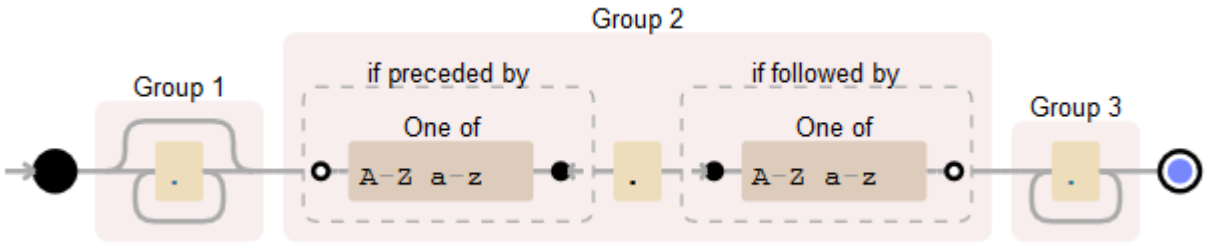
Using: my.File2.v1.2.345

Tim Contribution – cont.

Match: (.*?)((?<=[A-Za-z])\.(?=[A-Za-z]))(.+)

Sample:

```
Match 1: my.File2.v1.2.345
Group 1: my
Group 2: .
Group 3: File2.v1.2.345
```



Where:

\1 = my \2 = . \3 = File2.v1.2.345

Tim Analysis:

1..	Match against any character	Capture Group 1 = m
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. (?<=[A-Za-z])	Lookbehind. Look to the left. If character is a single uppercase or lowercase letter then expression = True and evaluation continues. = 'm' so TRUE	Capture Group 2 = position just after 'm' Changes Capture Group 1 = m
5. \.	Match against literal <dot> This is the <dot> after 'my'	Capture Group 2 = <dot> Changes Capture Group 1 = my
6. (?=[A-Za-z])	Lookahead. Look to the right. If next character is a single uppercase or lowercase letter then expression = True and evaluation continues. = 'F' so TRUE	Capture Group 2 = Unchanged
7. .+	Match against remaining string	Capture Group 3 = File2.v1.2.345

Find Occurrences

Replace the "." between Letters with a <space>, Ignore "." between Numbers cont.

Using: this.Is.my.File.v.1.2.345

Tim Contribution – cont.

Match: (.*?)((?<=[A-Za-z])\.(?=[A-Za-z]))(.+)

Sample:

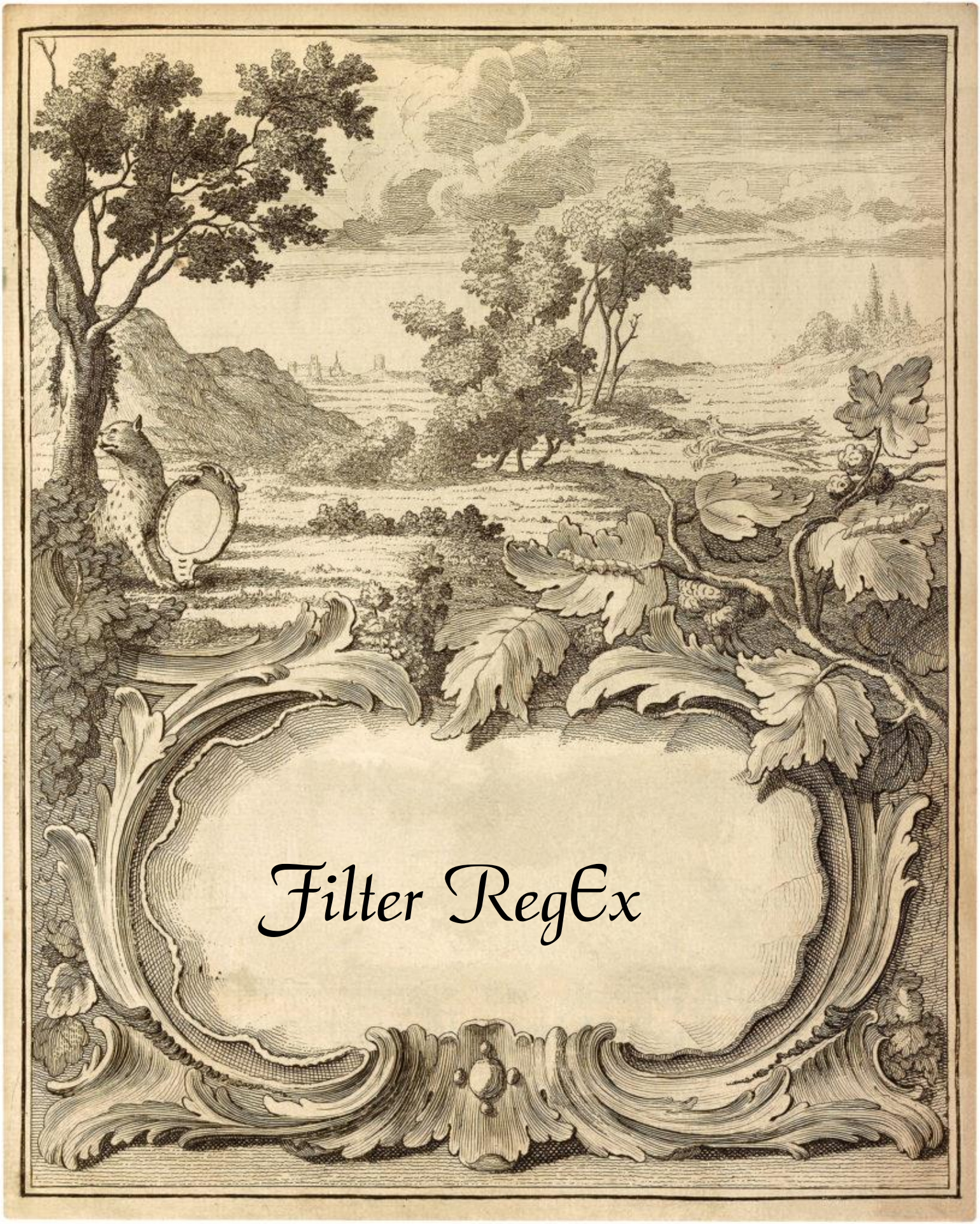
```
.....................................
Match 1: this.Is.my.File.v.1.2.345
Group 1: this
Group 2: .
Group 3: Is.my.File.v.1.2.345
```

Where:

\1 = this \2 = . \3 = Is.my.File.v.1.2.345

Tim Analysis:

1..	Match against any character	Capture Group 1 = t
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. (?<=[A-Za-z])	Lookbehind. Look to the left. If character is a single uppercase or lowercase letter then expression = True and evaluation continues. = 't' so TRUE	Capture Group 2 = position just after 't' Changes Capture Group 1 = t
5. \.	Match against literal <dot> This is the <dot> after 'this'	Capture Group 2 = <dot> Changes Capture Group 1 = this
6. (?=[A-Za-z])	Lookahead. Look to the right. If next character is a single uppercase or lowercase letter then expression = True and evaluation continues. = 'I' so TRUE	Capture Group 2 = Unchanged
7. .+	Match against remaining string	Capture Group 3 = Is.my.File.v.1.2.345


Filter RegEx

Section #12: Filter (RegEx)

It should be noted that Filter only affects what you see in the Content Panel. It does not perform manipulation. BRU supports using RegEx in the Filter data entry field.

Ending with 3 but not 4 digits

Limit File List only to filenames/folders that meet the filter specification of ending with an underscore followed by 3 numeric digits.

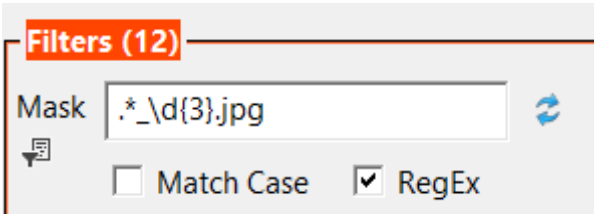
Serie_abc_123.jpg Display
Serie_def_1234.jpg Don't Display

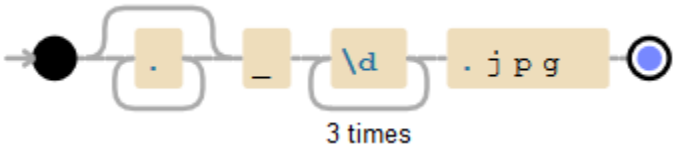
Before:

Name	New Name
1862_31_12_2019	1862_31_12_2019
1863_22_10_2019	1863_22_10_2019
Serie_abc_123.jpg	Serie_abc_123.jpg
Serie_def_1234.jpg	Serie_def_1234.jpg

'Section #12: Filters' –

Mask: .*_\d{3}.jpg
RegEx: Enabled





After:

Name	New Name
Serie_abc_123.jpg	Serie_abc_123.jpg

Forum Contributor: KenP

Section #12: Filter (RegEx)

Ending with 3 but not 4 digits cont.

Using:

Serie_abc_123.jpg
test_this_456.mp3
Another test_789.mp3

Mask: .*_\d{3}.*

Tim Analysis:

1. .	Match against any character	= s
2. *	Make it Greedy	= <entire string>
3. _	Match against underscore Backtracks to match underscore after 'abc'	= Serie_abc_
3. \d{3}	Match against 3 numeric digits	= Serie_abc_123

Tim Notes:

1. I changed the extension for this test to match all available samples: from '*.jpg' to '.*'
2. What it does is to filter any files that have the pattern:

Text_###

Where text can be any text – <spaces>, words, etc.

Name	New Name
Another test_789.mp3	Another test_789.mp3
serie_abc_123.jpg	serie_abc_123.jpg
test_this_456.mp3	test_this_456.mp3

3. Capture Groups are not required in the filter.

Section #12: Filter (RegEx)

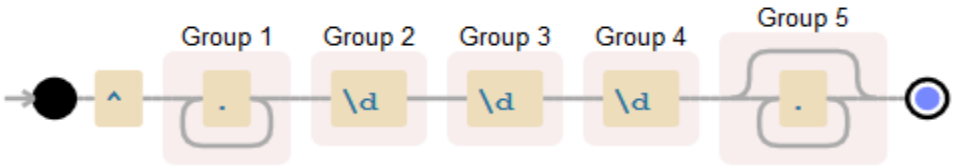
Displaying Filenames with one or more numeric digits anywhere in string

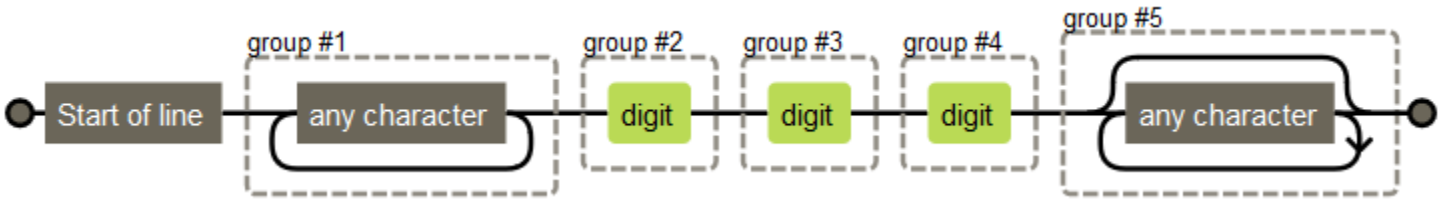
Tim Contribution –

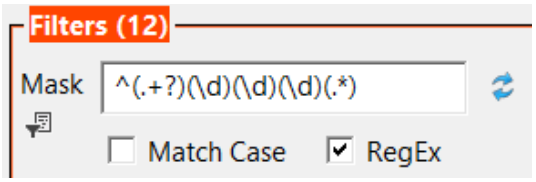
If you want to select only those files that have three or more numeric digits:

'Section #12: Filters' -

Mask: ^(.+?)(\d)(\d)(\d)(.*)
RegEx: Enabled







Click Filter Refresh Button (blue button to right of Mask data field to see matching files.

Name	New Name
file123.ext	file123.ext
file 1243.ext	file 1243.ext
file 021.ext	file 021.ext

Section #12: Filter (RegEx)

Displaying Filenames with one or more numeric digits anywhere in string cont.

Tim Contribution – cont.

If you want to select only those files that have three or more numeric digits: cont.

Using: file 021.ext

Mask: ^(.+?)(\d\d\d)(.*)

Tim Analysis:

1. ^	Start at beginning of line (BOL)	
2. .	Match against any character	Capture Group 1 = f
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. \d\d\d	Match against 3 numeric digits	Capture Group 2 = 021
	Backtracking.	Changes Capture Group 1 = file
6. .*	Match against remaining string	Capture Group 3 = EOL

Tim Notes:

1. I changed the individual Capture Groups to a singular Capture Group 2, (\d\d\d) for simplicity for this test.

   as you can see, this does not affect the filter outcome:

Name	New Name
file123.ext	file123.ext
file 021.ext	file 021.ext
file 1243.ext	file 1243.ext

2. Capture Group 3 would capture any text – or in this case, numeric digits past the three already allocated by Capture Group 2. Therefore the filter can find files that have three or more numeric digits at the end of the string. To ensure this, I could have added ' $ ' at the end.

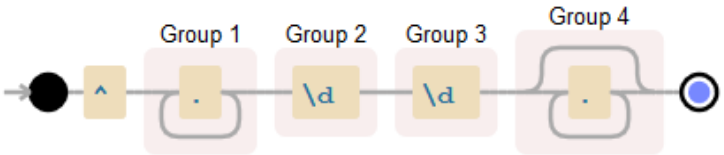
Section #12: Filter (RegEx)

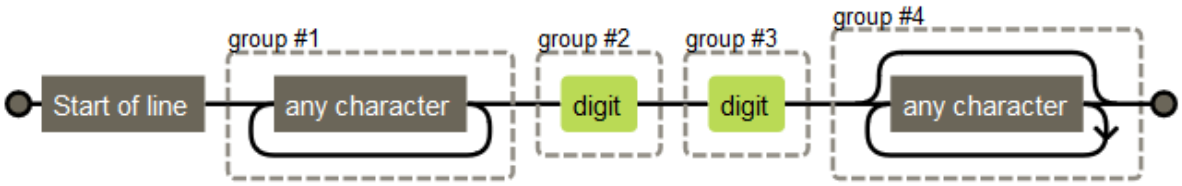
Displaying Filenames with one or more numeric digits anywhere in string cont.

Tim Contribution – cont.

For two or more, use this:

Mask: ^(.+?)(\d)(\d)(.*)

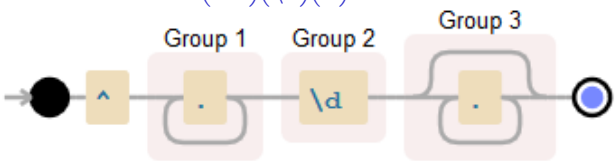


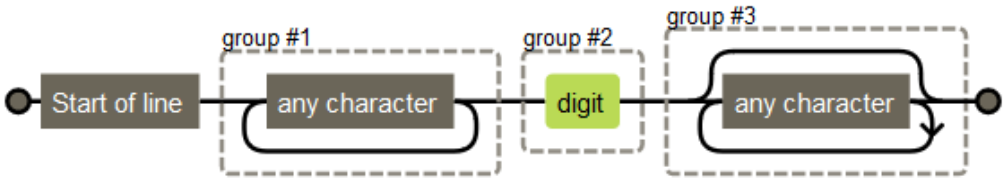


Name	New Name
file123.ext	file123.ext
file01.ext	file01.ext
file 1243.ext	file 1243.ext
file 021.ext	file 021.ext
file 01.ext	file 01.ext
1863_22_10_2019	1863_22_10_2019

For one or more, use this:

Mask: ^(.+?)(\d)(.*)



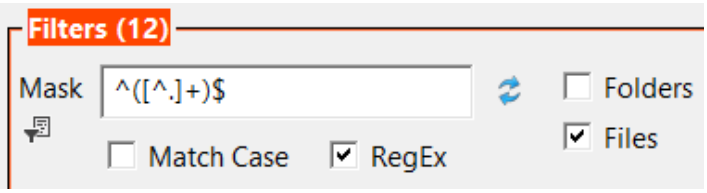


Name	New Name
file123.ext	file123.ext
FILE1.txt	FILE1.txt
file01.ext	file01.ext
file 1243.ext	file 1243.ext
file 021.ext	file 021.ext
file 01.ext	file 01.ext
1863_22_10_2019	1863_22_10_2019

Section #12: Filter (RegEx)

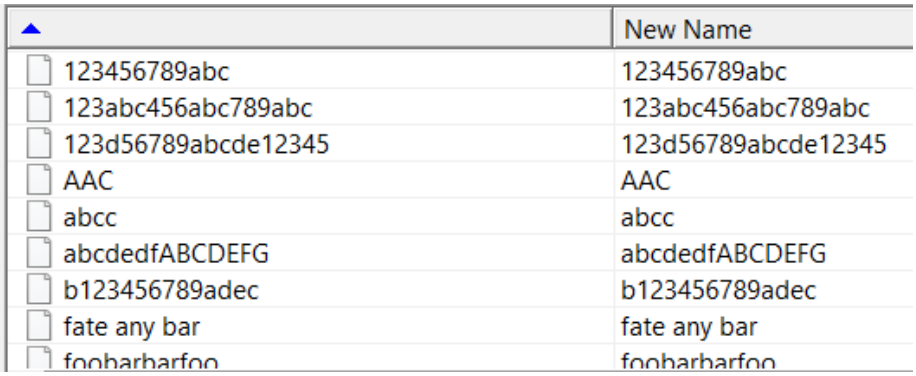
How to select files with no extension (to rename)

'Section #12: Filters' -

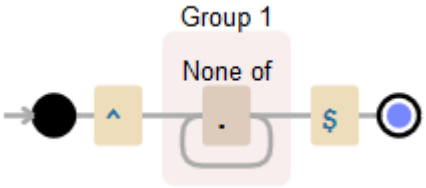


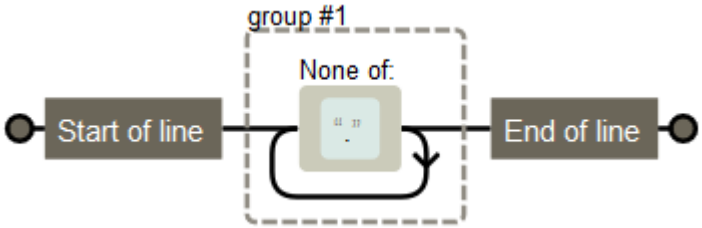
Mask: ^([^.]+)$
RegEx: Enabled

Click on the Filter Refresh button (the blue button to the right of the mask data field) to See Files or Press F5.



	New Name
123456789abc	123456789abc
123abc456abc789abc	123abc456abc789abc
123d56789abcde12345	123d56789abcde12345
AAC	AAC
abcc	abcc
abcdedfABCDEFG	abcdedfABCDEFG
b123456789adec	b123456789adec
fate any bar	fate any bar
foobarbarfoo	foobarbarfoo





Forum Contributor: Admin

Using: Lesson 1

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^.]	Match against a Negated Class consisting of a literal <dot> character.	Capture Group 1 = L
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. $	Must be at EOL (End of Line)	

Tim Notes:

1. Step 4 ensures that the string ends without an extension being present. This is because of the Negated Class which indicates the RegEx cannot match against a literal dot – therefore no extension after the EOL.

   Because this is a filter, it does not require 'Rename File Extensions As Being part of File Name' enabled from Renaming Options Menu.

Remove

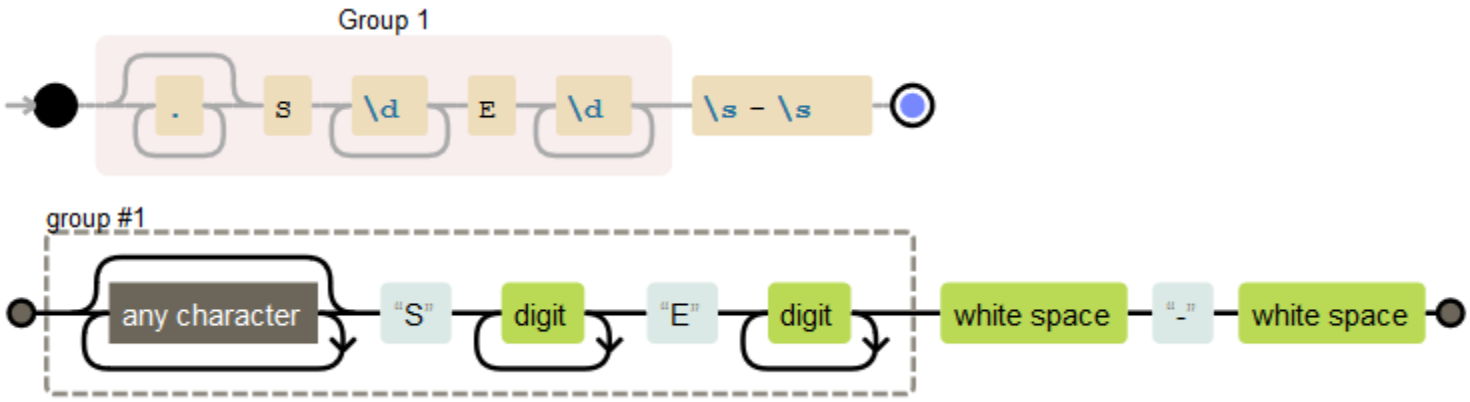
Remove

Delete Everything after SxxExx i.e., S07E05

American Horror Story S07E05 - Löcher.mkv to American Horror Story S07E05.mkv

Match: (.*S\d+E\d+)\s-\s
Replace: \1

Name	New Name
American Horror Story S07E05 - Löcher.mkv	American Horror Story S07E05.mkv



Sample:

```
                                          Start  Length
Match 1: American Horror Story S07E05 -       0      31
Group 1: American Horror Story S07E05         0      28
```

Contributor and *Tim Analysis:*

1..*S	Matches everything up to 'S' followed by …	Capture Group 1 = American Horror Story <space> S
2. \d+	..1 or more numeric digits, followed by …	Capture Group 1 = American Horror Story <space> S07
3. E	.. 'E' followed by …	Capture Group 1 = American Horror Story <space> S07E
4. \d+	.. 1 or more numeric digits, followed by …	Capture Group 1 = American Horror Story <space> S07E05
5. -	<space><hyphen><space> (Not Captured)	

Remove

Delete Everything after SxxExx i.e., S07E05 cont.

American Horror Story S07E05 - Löcher.mkv to American Horror Story S07E05.mkv

Match: (.*S\d+E\d+)\s-\s

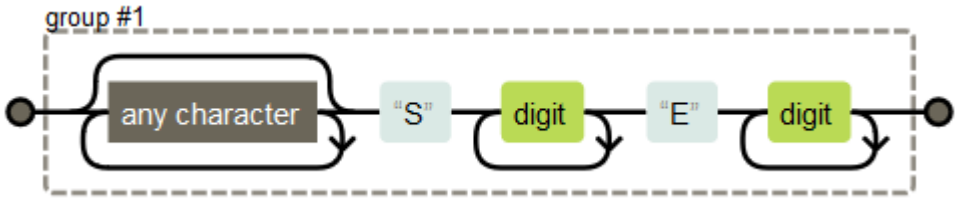
Contributor Notes:

1. If there are always two numeric digits, you could use \d\d or \d{2} instead of \d+.
2. If at the end of the string instead of <space><hyphen><space> there was –

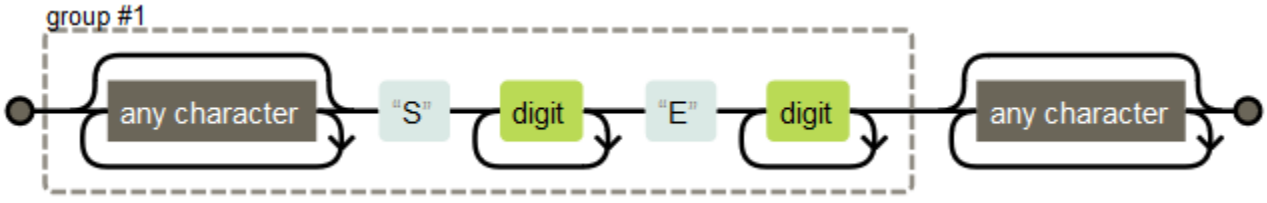
<space><hyphen>
text
<null>

You could use:

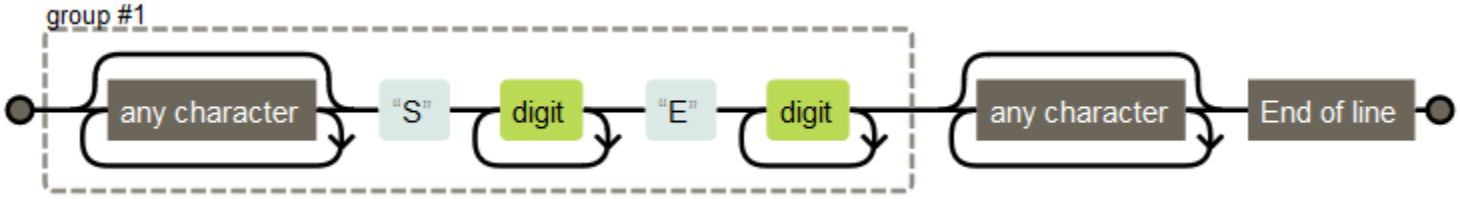
(.*S\d+E\d+)



(.*S\d+E\d+).*



(.*S\d+E\d+).*$



Forum Contributor: therube

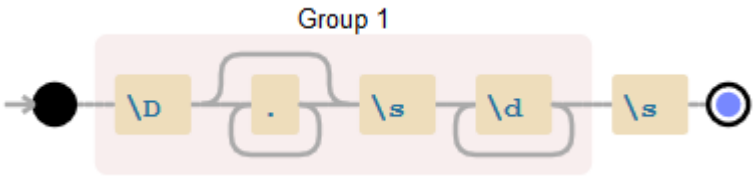
Remove

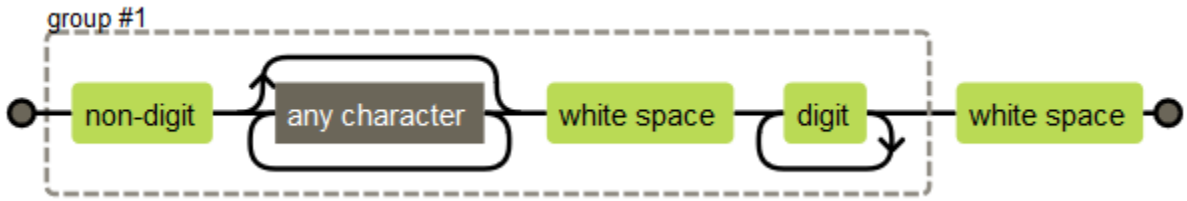
Remove all extraneous information after First group of Digits

Spider-Man 002 2009 Vol 4 Marvel to Spider-Man 002
BatMan 02 DC Vol 3 2014 to BatMan 02

Match: (\D.*?\s\d+)\s
Replace: \1

Name	New Name
Spider-Man 002 2009 Vol 4 Marvel.pdf	Spider-Man 002.pdf





Sample:

```
.........................
Match 1: Spider-Man 002
 Group 1: Spider-Man 002
Match 2:  Vol 4
 Group 1:  Vol 4
```

Contributor Analysis:

1. Non-Numeric Digit, followed by anything, non-greedy
2. Followed by a <space> & a set of numeric digit(s)
3. Remove the following <space> & everything else

Forum Contributor: therube

Tim Notes:

1. The sample shows a second match but BRU can only use the first match (Global switch that would allow more than one match is currently not supported).

Remove

Remove all extraneous information after First group of Digits cont.

Using: Spider-Man 002 2009 Vol 4 Marvel

Match: (\D.*?\s\d+)\s

Where:

\1 = Spider-Man 002

Tim Analysis:

1. \D	Match against any Non-Numeric Digit	Capture Group 1 = S
2..	Match against any character	Capture Group 1 = Sp
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = S
5. \s	Match against <space> character, followed by … This is the <space> after 'Man'	Capture Group 1 = Spider-Man<space>
6. \d	Match against Numeric Digit	Capture Group 1 = Spider-Man<space>0
7. +	Make it Greedy	Capture Group 1 = Spider-Man<space>002
8. \s	Match against <space> character (Not Captured)	

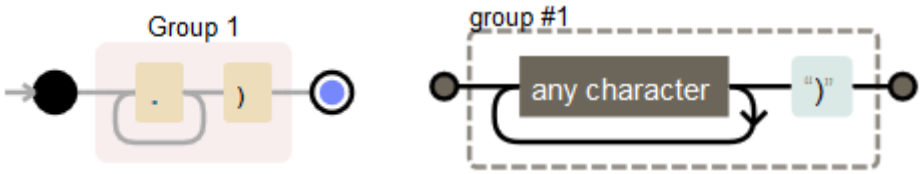
Remove

Remove Everything after first occurrence of ')'

Never Look Away (2018) - Drama, Thriller, History to Never Look Away (2018)

Match: (.+\))
Replace: \1

Name	New Name
Never Look Away (2018) - Drama, Thriller, History.pdf	Never Look Away (2018).pdf



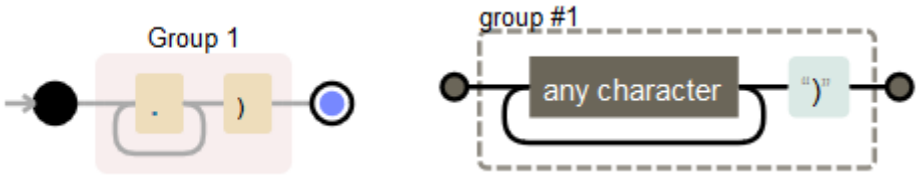
Sample:

```
.....................................
Match 1: Never Look Away (2018)
Group 1: Never Look Away (2018)
```

If you wanted to make it non-greedy:

Match: (.+?\))
Replace: \1

Name	New Name
Never Look Away (2018) - Drama, Thriller, History.pdf	Never Look Away (2018).pdf



Sample:

```
.....................................
Match 1: Never Look Away (2018)
Group 1: Never Look Away (2018)
```

Tim Notes:

1. The Visualizations Photos on left look the same for Greedy and Non-Greedy until you look at the path beneath in the Photos on the right).
2. The sample runs don't change the result, whether Greedy or Non-Greedy.

Forum Contributor: therube

Remove

Remove Everything after first occurrence of ')' cont.

Using: Never Look Away (2018) - Drama, Thriller, History

Where:

\1 = Never Look Away (2018)

Greedy:

Match: (.+\))

Tim Analysis:

1. .	Match against any character	Capture Group 1 = N
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \)	Match against a literal right parenthesis	Capture Group 1 =
	Backtracks to right parenthesis following '2018'	Never Look Away (2018)

Non-Greedy:

Match: (.+?\))

Tim Analysis:

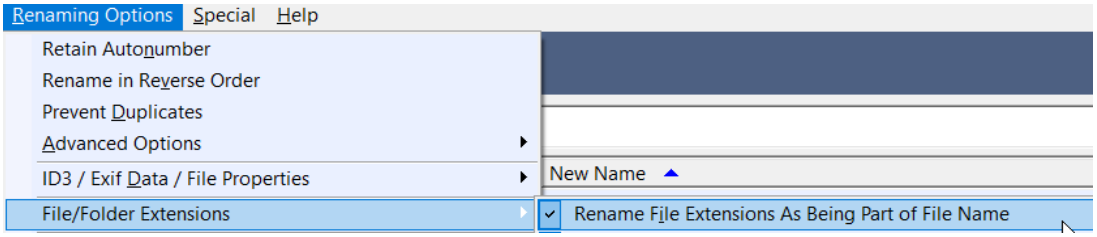
1. .	Match against any character	Capture Group 1 = N
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	
4. \)	Match against a literal right parenthesis	Capture Group 1 = N
	Moves forward to right parenthesis following '2018'	Never Look Away (2018)

Remove

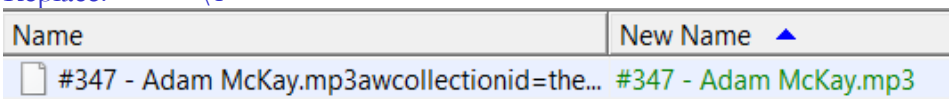
Remove all extraneous junk after extension

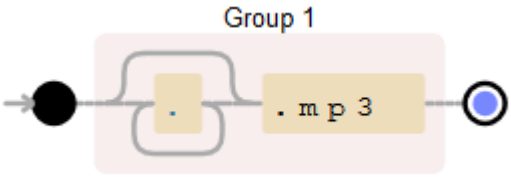
#347 - Adam McKay.mp3awcollectionid=the-empire-podcast&awepisodeid=14019
to
#347 - Adam McKay.mp3

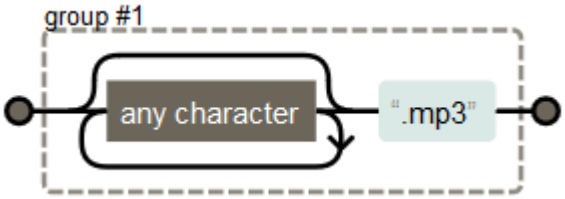
Enable : Rename File Extensions as Being part of File Name



Match: (.*\.mp3)
Replace: \1







Sample:

```
..............................
Match 1: #347 - Adam McKay.mp3
Group 1: #347 - Adam McKay.mp3
```

Tim Notes:

1. If the option wasn't enabled, BRU would treat the .mp3<and the rest of the text> as part of the extension and would not be considered in the renaming process (unless you enable 'Include Ext.' in 'Section #1: RegEx' as an alternative).
2. Because it is enabled, the RegEx matches everything until and inclusive of the encounter with '.mp3' at which time it stops, thus parsing the #347 – Adam McKay.mp3' from the rest of the string.

Forum Contributor: therube

Remove

Remove all extraneous junk after extension cont.

Using: #347 - Adam McKay.mp3awcollectionid=the-empire-podcast&awepisodeid=14019

Match: (.*\.mp3)

Where:

\1 = #347 - Adam McKay.mp3

Tim Analysis:

1..	Match against any character	Capture Group 1 = #
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \.	Match against literal <dot> This is the <dot> of the extension.	Capture Group 1 = #347 - Adam McKay.
4. mp3	Match against string 'mp3'	Capture Group 1 = #347 - Adam McKay.mp3

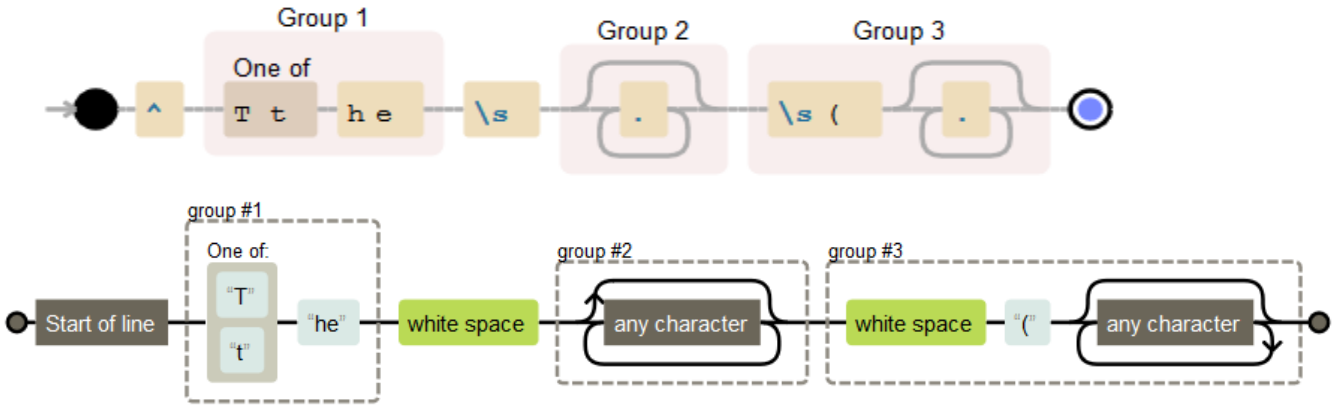
Remove

Remove 'The' from beginning to end of title (not end of filename)

The Straight Story (1999) (Disney) to Straight Story, The (1999) (Disney)

Match: ^([Tt]he)\s(.*?)(\s\(.*)
Replace: \2, \1\3

Name	New Name
The Straight Story (1999) (Disney)	Straight Story, The (1999) (Disney)



Sample:

```
.........................................
Match 1: The Straight Story (1999) (Disney)
Group 1: The
Group 2: Straight Story
Group 3:  (1999) (Disney)
```

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. ^([Tt]	Matches against either an Uppercase or Lowercase 'T' .. defined in a class followed by…	Capture Group 1 = T
3. he	the letters 'he'. This sets a Match = TRUE if the word is EITHER 'The' or 'the', followed by …	Capture Group 1 = The
4. \s	<space> (Not Captured)	
5. .	Match against next single character	Capture Group 2 = S
6. *	Make it Greedy	Capture Group 2 = Straight Story (1999) (Disney)
5. ?	but not too Greedy (Non-Greedy) until…	Capture Group 2 = entire string
6. \s	<space> followed by…	Capture Group 3 = <space> Changes Capture Group 2 = Straight
7. \(	.. left parenthesis	Capture Group 3 = <space> (
8. .*	Capture remainder of string	Capture Group 3 = <space>(1999) (Disney) Changes Capture Group 2 = Straight Story

Remove

Remove 'The' from beginning to end of title (not end of filename) cont.

Match: ^([Tt]he)\s(.*?)(\s\(.*)

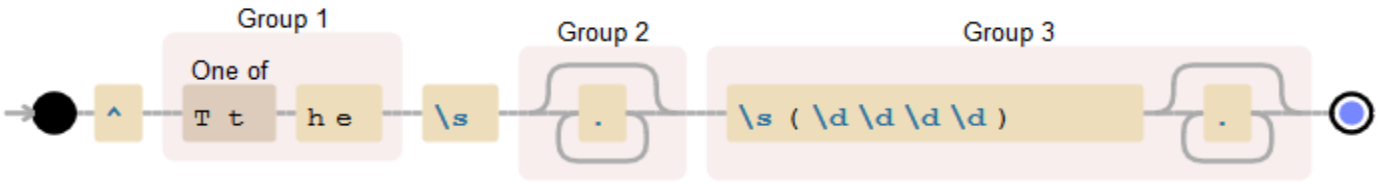
Problem: It also matches:

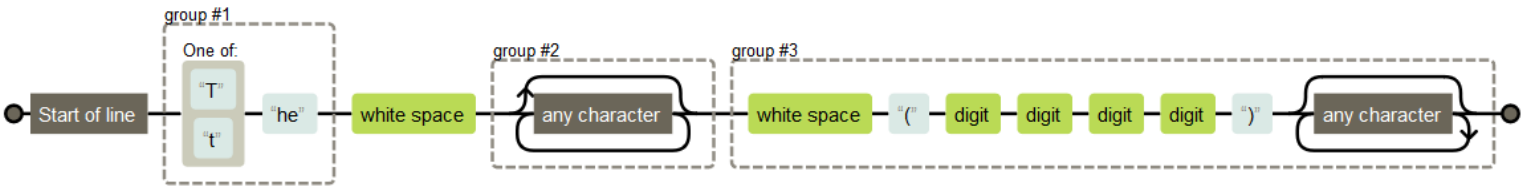
Name	
The Straight Story (1999) (Disney)	Straight Story, The (1999) (Disney)
The Straight Story (did you really intend on finding me)	Straight Story, The (did you really intend on finding me)

A variation Fixes this:

Match: ^([Tt]he)\s(.*?)(\s\(\d\d\d\d\).*)

Name	
The Straight Story (1999) (Disney)	Straight Story, The (1999) (Disney)
The Straight Story (did you really intend on finding me)	The Straight Story (did you really intend on finding me)



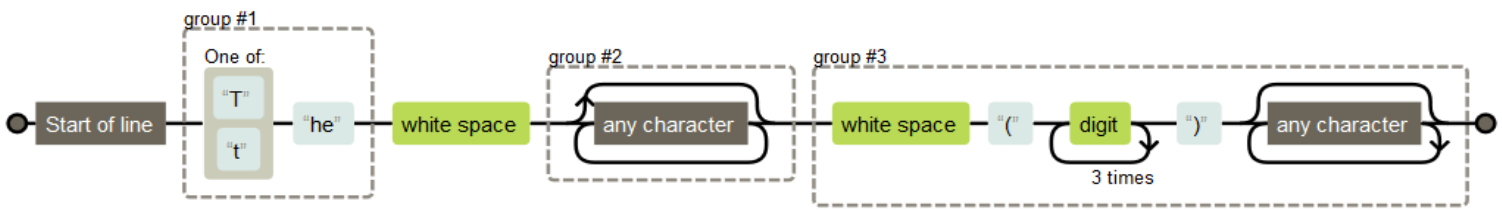


Tim Analysis:

1. The (d\d\d\d\).* looks specifically for 4 numerics (a year) and if True captures to the end of the string to EOL.
2. Could also have used \d{4}

Match: ^([Tt]he)\s(.*?)(\s\(\d{4}\).*)
Replace: \2, \1\3

Name	New Name
The Straight Story (1999) (Disney)	Straight Story, The (1999) (Disney)



Forum Contributor: therube

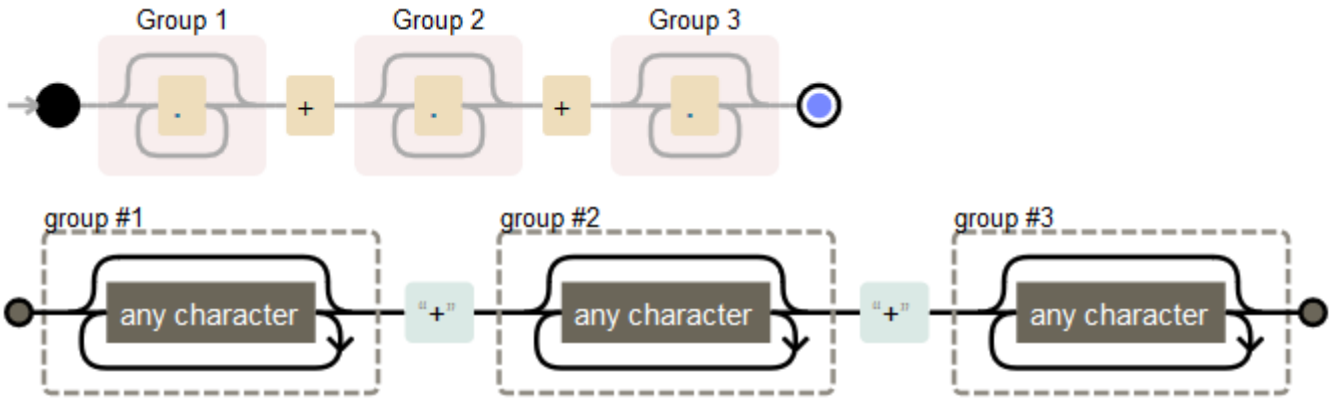
Remove

Remove second + and replace with Underscore (capture parts of filename and put back together)

someFile+1010101+1.jpg to someFile+1010101_1.jpg

Match: (.*)\+(.*)\+(.*)
Replace: \1+\2_\3

Name	New Name
someFile+1010101+1.jpg	someFile+1010101_1.jpg



Sample:

```
................................
Match 1: someFile+1010101+1
Group 1: someFile
Group 2: 1010101
Group 3: 1
```

Contributor Analysis:

1. Take each part of filename and place in capture groups but ignore the text character '+' each time
2. Put together string adding the '+' and '_' where needed.

Forum Contributor: Admin

Remove

Remove second + and replace with Underscore (capture parts of filename and put back together) cont.

Here's something I rarely do – I thought I would illustrate all of the steps using visuals so you can not only read what changed but see what changed and why.

Using: someFile+1010101+1.jpg

Match: (.*)\+(.*)\+(.*)

Where:

\1 = someFile \2 = 1010101 \3 = 1

Where:

Capture Group 1 is represented in this color
Capture Group 2 is represented in this color
Capture Group 3 is represented in this color
First literal '+' sign matched is represented in this color
Second literal '+' sign matched is represented in this color

Tim Analysis:

1.. Match against any character — Capture Group 1 = s

someFile+1010101+1

2. * Make it Greedy — Capture Group 1 = <entire string>

someFile+1010101+1 EOL

3. \+ Match against literal '+' character (Not Captured) — Changes Capture Group 1 = someFile+1010101
Backtracks to match '+' after '1010101'

someFile+1010101+1

4.. Match against any character — Capture Group 2 = 1
Moving forward matches '1' in the '+1'

someFile+1010101+1 EOL

Remove

Remove second + and replace with Underscore (capture parts of filename and put back together) cont.

Here's something I rarely do – I thought I would illustrate all of the steps using visuals so you can not only read what changed but see what changed and why. cont.

Using: someFile+1010101+1.jpg

Match: (.*)\+(.*)\+(.*)

Where:

\1 = someFile \2 = 1010101 \3 = 1

Tim Analysis: cont.

5. * Make it Greedy (already at EOL) Capture Group 2 = Unchanged

someFile+1010101+1 EOL

6. \+ Match against literal '+' character (Not Captured) Backtracks. This is the '+' after '1010101' Changes Capture Group 2 = 1010101

someFile+1010101+1

Changes the '+' Not Captured in step 3 to the '+' after 'someFile' Changes Capture Group 1 = someFile

7. .* Moving forward. Match against remaining string Capture Group 3 = 1

someFile+1010101+1 EOL

Tim Notes:

1. In step 6, Capture Group 1 is changed to 'someFile', because the matched '+' in step 3 changed from the position before the '1' at EOL to the position of the '+' after 'someFile.

   When this occurred, Capture Group 2 also changed, following the new position of the '+ sign to match against the string following 'someFile', to '1010101'. This left the current position at the '1' before EOL. Step 7 Captured this as Capture Group 3.

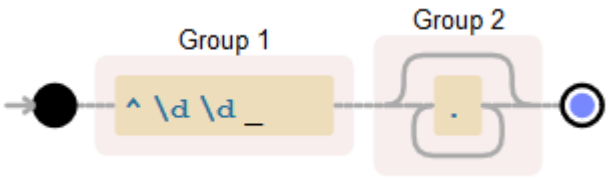
Remove

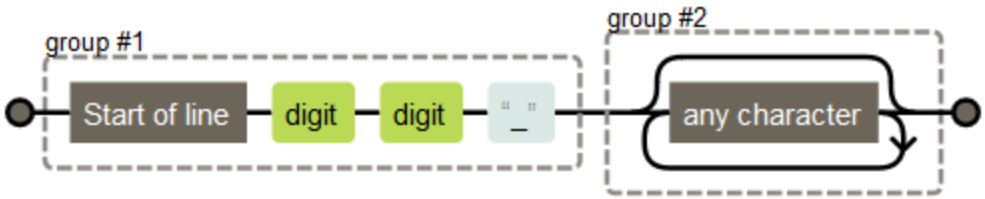
Remove first 3 characters if ##_

02_Starving Artist - The Hunger Song.mp3 to Starving Artist - The Hunger Song.mp3

Match: (^\d\d_)(.*)
Replace: \2

Name	New Name
02_Starving Artist - The Hunger Song.mp3	Starving Artist - The Hunger Song.mp3





Sample:

```
.................................................
Match 1: 02_Starving Artist - The Hunger Song
Group 1: 02_
Group 2: Starving Artist - The Hunger Song
```

Forum Contributor: therube

Where:

\1 = 02_ \2 = Starving Artist – The Hunger Song

Tim Analysis:

1. ^	Start at beginning	
2. \d\d	Match against 2 numeric digits	Capture Group 1 = 02
3. _	Match against underscore	Capture Group 1 = 02_
4. .*	Match against remaining string	Capture Group 2 = Starving Artist – The Hunger Song

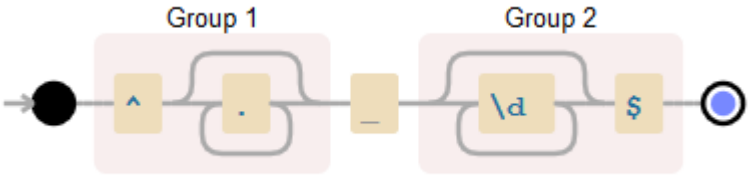
Remove

Remove the space before the number while ignoring the files that do not include a number

Alan H 1.pdf	to	Alan H1.pdf
Extremelylongname A.pdf	to	Ignore

Match: (^.*) (\d*$)
Replace: \1\2

Name	New Name
Alan H 1.pdf	Alan H1.pdf
Extremelylongname A.pdf	Extremelylongname A.pdf



Sample:

```
....................
Match 1: Alan H 1
Group 1: Alan H
Group 2: 1
```

Forum Contributor: therube

Remove

Remove the space before the number while ignoring the files that do not include a number cont.

Using: Alan H 1.pdf

Match: (^.*) (\d*$)

Where:

\1 = Alan H \2 = 1

Tim Analysis:

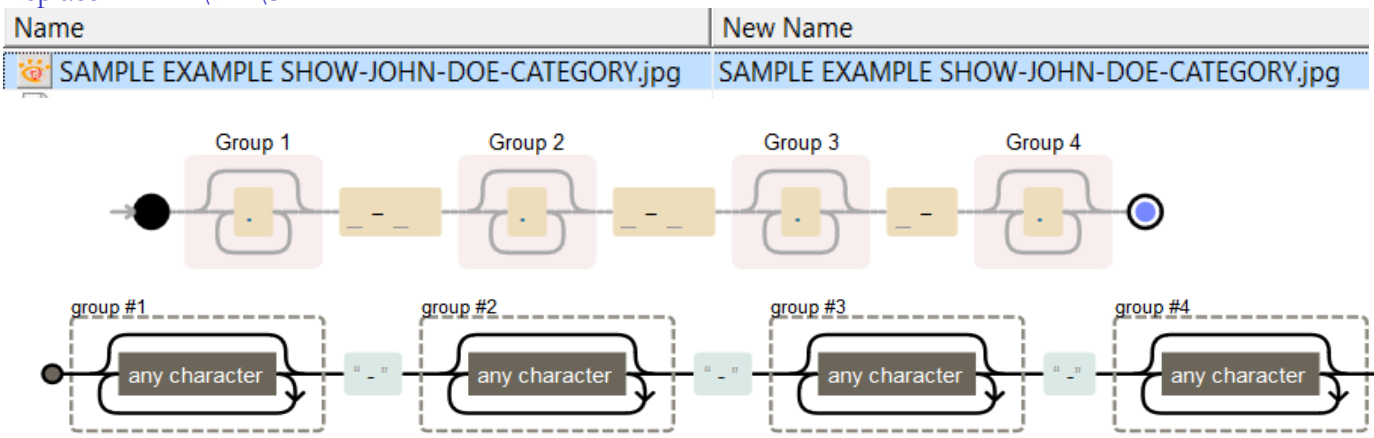
1. ^	Start at beginning	
2..	Match against any character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. <space>	Match against a <space> character (Not Captured)	Changes Capture Group 1 = Alan H
5. \d	Match against 1 numeric digit	Capture Group 2 = 1
6. .*	Make it Greedy	Capture Group 2 = Unchanged
7. $	Must be at EOL (End of Line)	

Remove

Remove all text before and after format -$-$-

SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg
to
JOHN-DOE.jpg

Match: (.*) - (.*) - (.*) -(.*)
Replace: \2 - \3



Didn't work.

Forum Contributor: therube

Tim Notes:

1. Obvious that the original RegEx is matched against a pattern of:

Text
<space> <hyphen> <space>
Text
<space> <hyphen> <space>
Text
<space><hyphen>
Text

and the sample is:

Text
<hyphen>
Text
<hyphen>
Text
<hyphen>
Text

Not even close.

Remove

Remove all text before and after format -$-$- cont.

SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg
to
JOHN-DOE.jpg

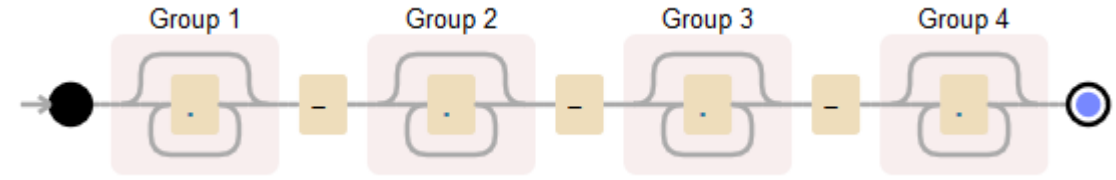
Tim Contribution –

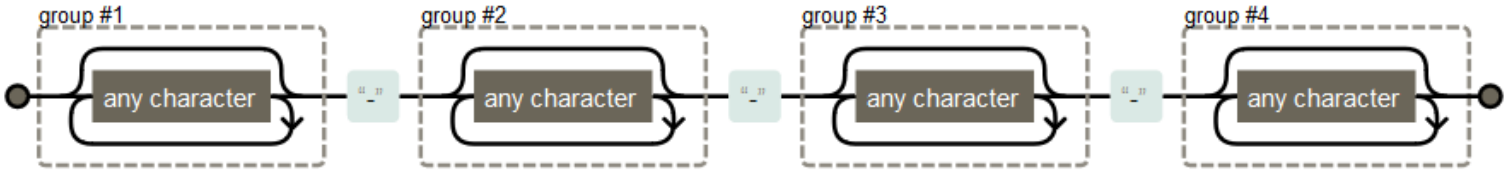
This will -

Corrected RegEx -

Match: (.*)-(.*)-(.*)-(.*)
Replace: \2-\3

Name	New Name
SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY	JOHN-DOE





Sample:

```
.....................................................
Match 1: SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY
Group 1: SAMPLE EXAMPLE SHOW
Group 2: JOHN
Group 3: DOE
Group 4: CATEGORY
```

Tim Notes:

1. Spacing in both the RegEx and the Replacement String was the problem

Remove

Remove all text before and after format -$-$- cont.

Using: SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg

Tim Contribution – cont.

Match: (.*)-(.*)-(.*)-(.*)

Where:

\1 = SAMPLE EXAMPLE SHOW \2 = JOHN \3 = DOE \4 = CATEGORY

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen> (Not Captured) Backtracks to <hyphen> after 'DOE'	Changes Capture Group 1 = SAMPLE EXAMPLE SHOW-JOHN-DOE
4..	Match against any character Moving forward.	Capture Group 2 = C
5. *	Make it Greedy Current position EOL	Capture Group 2 = CATEGORY
6. -	Match against <hyphen> (Not Captured) Backtracks to <hyphen> after 'DOE' Changes match of previous <hyphen> from step 3 to <hyphen> after 'JOHN'	Changes Capture Group 2 = DOE Changes Capture Group 1 = SAMPLE EXAMPLE SHOW-JOHN
7..	Match against any character	Capture Group 3 = C
8. *	Make it Greedy	Capture Group 3 = CATEGORY
9. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'DOE' Changes match of previous <hyphen> from step 6 to <hyphen> after 'JOHN' Changes match of previous <hyphen> from step 3 to <hyphen> after 'SHOW'	Changes Capture Group 3 = DOE Changes Capture Group 2 = JOHN Changes Capture Group 1 = SAMPLE EXAMPLE SHOW
10..	Match against any character	Capture Group 4 = C
11. *	Make it Greedy	Capture Group 4 = CATEGORY

Remove

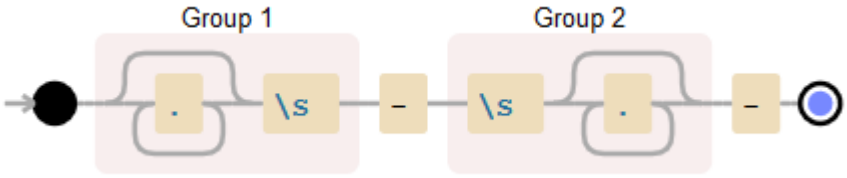
Remove all text before and after format -$-$- cont.

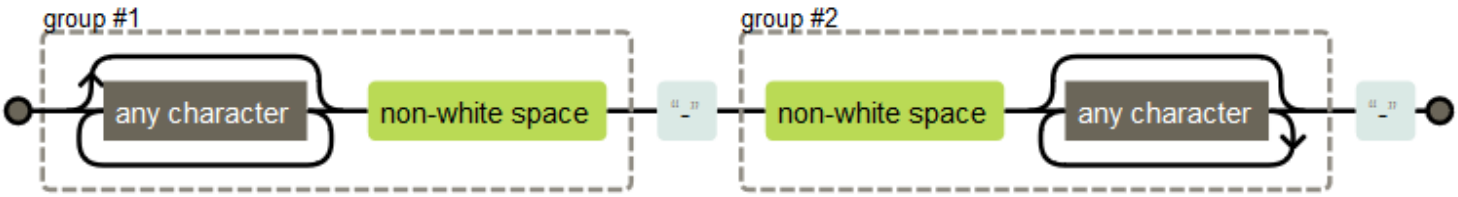
Tim Contribution – cont.

As will this -

Match: (.*?\S)-(\S.*)-
Replace: \2

Name	New Name
. EXAMPLE SHOW-JANE-WAYNE-CATEGORYB.jpg	JANE-WAYNE.jpg
SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg	JOHN-DOE.jpg





Sample:

```
........................................
Match 1: SAMPLE EXAMPLE SHOW-JOHN-DOE-
Group 1: SAMPLE EXAMPLE SHOW
Group 2: JOHN-DOE
```

Tim Notes:

1. Uses Non-Whitespace Metacharacters previously discussed.

Remove

Remove all text before and after format -$-$- cont.

Tim Contribution – cont.

Using: SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg

Match: (.*?\S)-(\S.*)-

Where:

\1 = SAMPLE EXAMPLE SHOW \2 = JOHN-DOE

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy Backtracks.	Capture Group 1 = BOL
4. \S	Matches any Non-Whitespace Character Moving forward.	Capture Group 1 = S
5. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'SHOW'	Changes Capture Group 1 = SAMPLE EXAMPLE SHOW
6. \S	Matches any Non-Whitespace Character	Capture Group 2 = J
7. .	Match against any character	Capture Group 2 = JO
8. *	Make it Greedy	Capture Group 2 = JOHN-DOE-CATEGORY
9. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'DOE' Does not affect <hyphen> from step 5	Changes Capture Group 2 = JOHN-DOE Capture Group 1 = Unchanged

A Word on Simple RegEx

We Interrupt this program to bring you an important message:

In Previous pages you may have noticed that I provide complex analysis to what appears to be simple RegEx.

Using: SAMPLE EXAMPLE SHOW-JOHN-DOE-CATEGORY.jpg

Match: (.*)-(.*)-(.*)-(.*)

Using: someFile+1010101+1.jpg

Match: (.*)\+(.*)\+(.*)

Both of these could have been summed up easily as each group of (.*), [or in some cases (.+)], holding the value of each word or phrase, separated by <hyphens> or the '+' character, to parse the string components.

The first example is:

SAMPLE EXAMPLE SHOW	-	JOHN	-	DOE	-	CATEGORY
(.*)	-	(.*)	-	(.*)	-	(.*)

The second example is:

someFile	+	1010101	+	1
(.*)	\+	(.*)	\+	(.*)

So why the needless complexity?

Because I feel it is important to illustrate that what the RegEx Engine is doing is not as easy as it appears. This teaches you that although the simplest RegEx may get the job done, it may not always be the best solution, especially when it involves an additional multitude of steps and backtracking. A simple analysis is not the best teacher anyway. When you begin programming RegEx on your own, you're better off understanding what the process *is* rather than just accepting that it works. For now, just realize this. Proficency will come with time.

A Word on Simple RegEx cont.

We Interrupt this program to bring you an important message: cont.

Even my own analysis, as thorough as I try to be, only touches upon what is happening behind the scenes.

For example, here is a debug output from the first example –

```
Beginning match attempt at character 0
 1 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGORY
 2 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGORY
 3 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGORYbacktrack
 4 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGOR
 5 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGOR
 6 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGORbacktrack
 7 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGO
 8 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGO
 9 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGObacktrack
10 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEG
11 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEG
12 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGbacktrack
13 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATE
14 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATE
15 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEbacktrack
16 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CAT
17 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CAT
18 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATbacktrack
```

This continues on for a total of 255 steps!

```
241 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATbacktrack
242 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CA
243 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CA
244 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CAbacktrack
245 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-C
246 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-C
247 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-Cbacktrack
248 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-
249 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-
250 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-backtrack
251 SAMPLE_EXAMPLE_SHOW-JOHN-DOE
252 SAMPLE_EXAMPLE_SHOW-JOHN-DOE
253 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-
254 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGORY
255 SAMPLE_EXAMPLE_SHOW-JOHN-DOE-CATEGORY
    Match found in 255 steps
```

But this book is not really about providing the best method. My own contributions are witness to that. No, I am simply an observer showing you what may be the consequences depending on how a RegEx is programmed.

Now back to our regularly scheduled program.

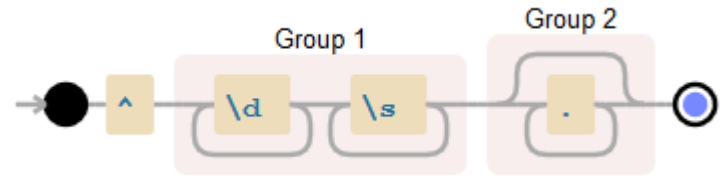
Remove

Strip away all prefix of numbers and space character

043924 style.css to style.css

Match: ^(\d+\s+)(.*)
Replace: \2

Name	New Name
043924 style.css	style.css



Sample:

```
.......................
Match 1: 043924 style
Group 1: 043924
Group 2: style
```

Forum Contributor: therube

Remove

Strip away all prefix of numbers and space character cont.

Using: 043924 style.css

Match: ^(\d+\s+)(.*)

Where:

\1 = 043924 \2 = style

Tim Analysis:

1. ^	Start at beginning	
2. \d	Match against Numeric Digit	Capture Group 1 = 0
3. +	Make it Greedy	Capture Group 1 = 043924
4. \s	Match against <space> character	Capture Group 1 = 043924<space>
5. +	Make it Greedy Accounts for additional <space>' characters if any.	Capture Group 1 = 043924<space>
6. .*	Match against remaining string	Capture Group 2 = style

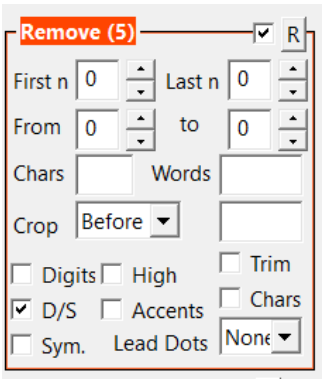
Remove

Remove everything after second hyphen. No change if only one hyphen

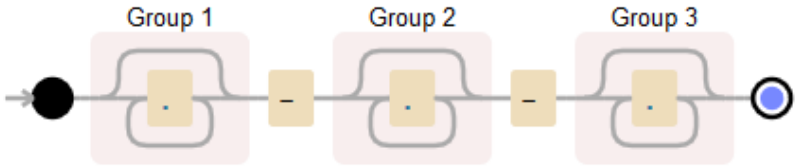
andy swimming - mandy watching - sandy dancing.jpg to andy swimming - mandy watching.jpg

Mary swimming - Andy watching.jpg to Ignore

1. 'Section #5: Remove' -

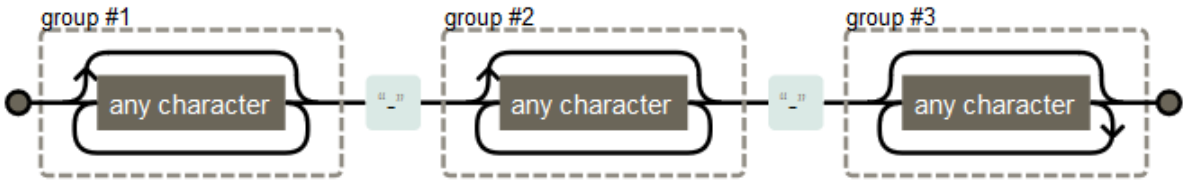


D/S – convert any occurrences of <double spaces> to <single spaces>



Match: (.*?)-(.*?)-(.*)
Replace: \1 - \2

Name	New Name
Mary swimming - Andy watching.jpg	Mary swimming - Andy watching.jpg
andy swimming - mandy watching - sandy dancing.jpg	andy swimming - mandy watching .jpg



Sample:

```
Match 1: andy swimming - mandy watching - sandy dancing
Group 1: andy swimming
Group 2:  mandy watching
Group 3:  sandy dancing
```

Contributor Analysis:

1. Match any character, non-greedy, up to " - "
2. Do the same again
3. Gather up rest of string

Forum Contributor: therube

Remove

Remove everything after second hyphen. No change if only one hyphen cont.

Using: andy swimming - mandy watching - sandy dancing

Match: (.*?)-(.*?)-(.*)

Where:

\1 = andy swimming \2 = <space> mandy watching \3 = <space> sandy dancing

Tim Analysis:

1..	Match against any character	Capture Group 1 = a
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. -	Match against <hyphen> (Not Captured)	Changes Capture Group 1 = andy swimming
5..	Match against any character This is the <space> after 'swimming'	Capture Group 2 = <space>
6. *	Make it Greedy	Capture Group 2 = <space> mandy watching - sandy dancing
7. ?	But not too Greedy Backtracks to position before <space> of 'mandy'	Capture Group 2 = null
8. -	Match against <hyphen> (Not Captured)	Changes Capture Group 2 = <space> mandy watching
9. .*	Match against remaining string	Capture Group 3 = <space> sandy dancing

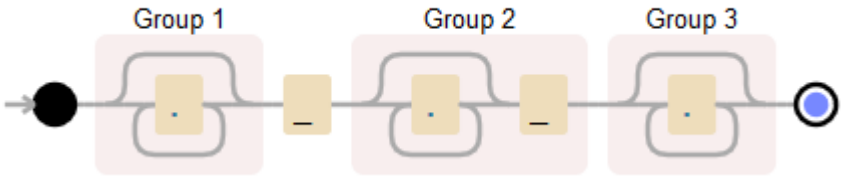
Remove

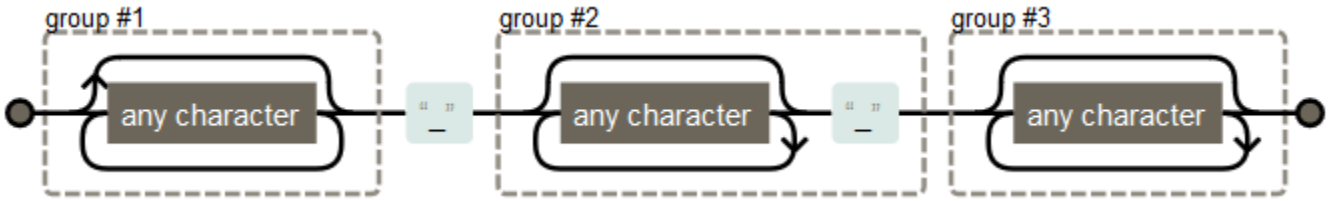
Remove everything between the first _ and last _

600128_rosa_choque_vestido to 600128_vestido

Match: (.*?)_(.*_)(.*)
Replace: \1_\3

Name	New Name
600128_rosa_choque_vestido	600128_vestido





Sample:

```
.....................................
Match 1: 600128_rosa_choque_vestido
Group 1: 600128
Group 2: rosa_choque_
Group 3: vestido
```

Forum Contributor: therube

Remove

Remove everything between the first _ and last _ cont.

Using: 600128_rosa_choque_vestido

Match: (.*?)_(.*_)(.*)

Where:

\1 = 600128 \2 = rosa_choque_ \3 = vestido

Tim Analysis:

1..	Match against any character	Capture Group 1 = 6
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. _	Match against underscore character (Not Captured)	Changes Capture Group 1 = 600128
5..	Match against any character	Capture Group 2 = r
6. *	Make it Greedy	Capture Group 2 = rosa_choque_vestido
7. _	Match against underscore character (Not Captured)	Changes Capture Group 2 = rosa_choque_
8. .*	Match against remaining string	Capture Group 3 = vestido

Tim Notes:

1. Anytime you begin a RegEx with (.*), you immediately move to the end of the string (EOL) and begin backtracking.

2. To move from there to the beginning of the string or to the position previous to the last expression, use the ' ? ' to make the Greedy Quantifier Lazy. In step 3, it backtracks to BOL.

Remove

Switch Around String and Remove Digits

songname_number9digits - artist.mp3 to artist - songname.mp3

Samples:

dabs#_256394142 - SUGHT.mp3	to	SUGHT - dabs#.mp3
5 AM_166267550 - (dreamchild).mp3	to	(dreamchild) - 5 AM.mp3
5_156628547 - redo.mp3	to	redo – 5.mp3
8;00_133775391 - Milk &Honey.mp3	to	Milk &Honey - 8;00.mp3
12nd - Grof_205805641 - 12nd (one second).mp3	to	12nd (one second) - 12nd - Grof.mp3
12nd - The Minute_299767163 - 12nd (one second).mp3	to	12nd (one second) - 12nd – The Minute.mp3
Antidote WAkachi_308796301 - SUGHT.mp3	to	SUGHT - Antidote WAkachi.mp3
Anymus_192638322 - Inkyz.mp3	to	Inkyz – Anymus.mp3
b&B_296170807 - psupac.mp3	to	psupac - b&B.mp3

Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Mac.mp3
to
K-Mac - Autoerotique - WTF (K-MAC Trap VIP).mp3

AWOLNATION - Run (B52 Remix)_217279053 - B52.mp3
to
B52 - AWOLNATION - Run (B52 Remix).mp3

Observations:

1. "songname" may include other symbols, <spaces>, etc.
2. "number9digits" the number is randomized and is usually 9 numeric digits
3. "songname" and "number9digits" are seperated by "_" with no <spaces> in between
4. "number9digits" and "artist" are seperated with " - " (<space> <hyphen> <space>)

Match: (.*)_\d{9} - (.*)
Replace: \2 - \1

Name	New Name
1862_31_12_2019	1862_31_12_2019
1863_22_10_2019	1863_22_10_2019
12nd - Grof_205805641 - 12nd (one second).mp3	12nd (one second) - 12nd - Grof.mp3
12nd - The Minute_299767163 - 12nd (one second).mp3	12nd (one second) - 12nd - The Minute.mp3
5 AM_166267550 - (dreamchild).mp3	(dreamchild) - 5 AM.mp3
5_156628547 - redo.mp3	redo - 5.mp3
8;00_133775391 - Milk &Honey.mp3	Milk &Honey - 8;00.mp3
Antidote WAkachi_308796301 - SUGHT.mp3	SUGHT - Antidote WAkachi.mp3
Anymus_192638322 - Inkyz.mp3	Inkyz - Anymus.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Ma...	Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Mac.mp3
AWOLNATION - Run (B52 Remix)_217279053 - B52.mp3	B52 - AWOLNATION - Run (B52 Remix).mp3
b&B_296170807 - psupac.mp3	psupac - b&B.mp3
dabs#_256394142 - SUGHT.mp3	SUGHT - dabs#.mp3

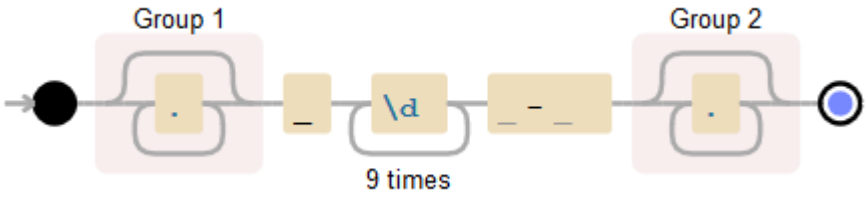
Didn't work:

The file Autoerotique, has 8 numeric digits. Not 9, so it doesn't match.

Remove

Switch Around String and Remove Digits cont.

Match: (.*)_\d{9} - (.*)



Modified - Try this instead -

Match: (.*)_(.*) - (.*)
Replace: \3 - \1

Name	New Name
1862_31_12_2019	1862_31_12_2019
1863_22_10_2019	1863_22_10_2019
12nd - Grof_205805641 - 12nd (one second).mp3	12nd (one second) - 12nd - Grof.mp3
12nd - The Minute_299767163 - 12nd (one second).mp3	12nd (one second) - 12nd - The Minute.mp3
5 AM_166267550 - (dreamchild).mp3	(dreamchild) - 5 AM.mp3
5_156628547 - redo.mp3	redo - 5.mp3
8;00_133775391 - Milk &Honey.mp3	Milk &Honey - 8;00.mp3
Antidote WAkachi_308796301 - SUGHT.mp3	SUGHT - Antidote WAkachi.mp3
Anymus_192638322 - Inkyz.mp3	Inkyz - Anymus.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Ma...	K-Mac - Autoerotique - WTF (K-MAC Trap VIP).mp3
AWOLNATION - Run (B52 Remix)_217279053 - B52.mp3	B52 - AWOLNATION - Run (B52 Remix).mp3
b&B_296170807 - psupac.mp3	psupac - b&B.mp3
dabs#_256394142 - SUGHT.mp3	SUGHT - dabs#.mp3

Remove

Switch Around String and Remove Digits cont.

Using: Anymus_192638322 - Inkyz.mp3

Original - Match: (.*)_\d{9} - (.*)

Where:

\1 = Anymus \2 = Inkyz

Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore (Not Captured)	Changes Capture Group 1 = Anymus
4. \d{9}	Match against 9 numeric digits (Not Captured) followed by … = 192638322	
5. <space>	Match against a <space> character This is the <space> following '192638322'	
6. -	Match against <hyphen> character (Not Captured)	
7. <space>	Match against a <space> character	
8. .*	Match against remaining string	Capture Group 2 = Inkyz

Modified - Match: (.*)_(.*) - (.*)

Where:

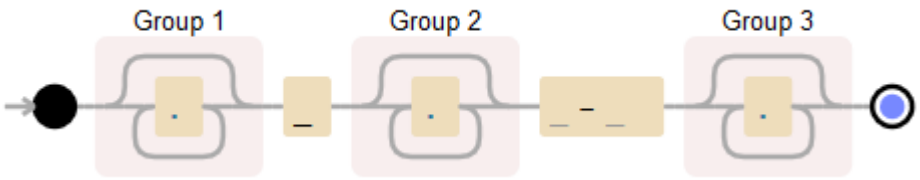
\1 = Anymus \2 = Inkyz

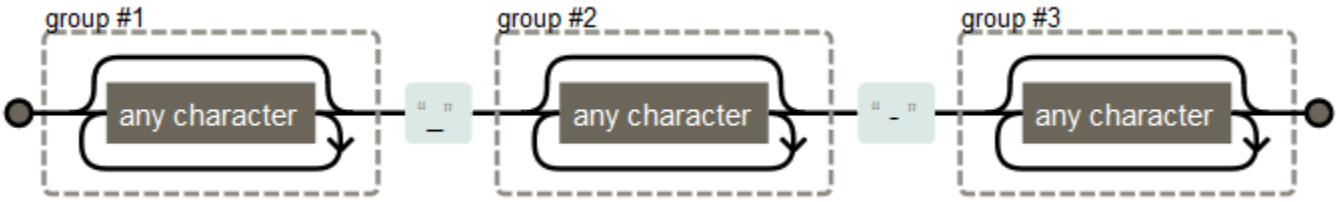
Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore (Not Captured)	Changes Capture Group 1 = Anymus
4..	Match against any character	Capture Group 2 = 1
5. *	Make it Greedy	Capture Group 2 = 192638322 - Inkyz
6. <space>	Match against a <space> character This is the <space> following '192638322'	Changes Capture Group 2 = 192638322
7. -	Match against <hyphen> character (Not Captured)	
8. <space>	Match against a <space> character	
9. .*	Match against remaining string	Capture Group 3 = Inkyz

Remove

Switch Around String and Remove Digits cont.





Sample:

```
.....................................................................
Match 1: Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Mac
Group 1: Autoerotique - WTF (K-MAC Trap VIP)
Group 2: 96732707
Group 3: K-Mac
```

Tim Notes:

1. The above did work

Forum Contributor: therube

Remove

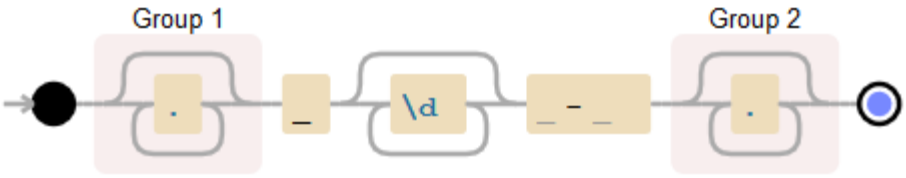
Switch Around String and Remove Digits cont.

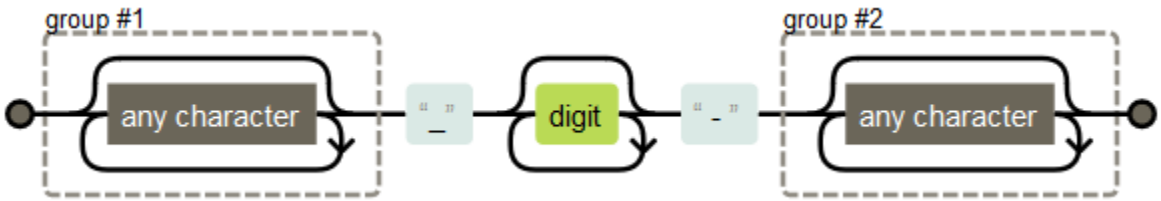
Tim Contribution –

The first RegEx example did not work for all files, while the second one did. Here's an additional RegEx that will as well:

Match: (.*)_\d* - (.*)
Replace: \2 - \1

Name	New Name
1862_31_12_2019	1862_31_12_2019
1863_22_10_2019	1863_22_10_2019
12nd - Grof_205805641 - 12nd (one second).mp3	12nd (one second) - 12nd - Grof.mp3
12nd - The Minute_299767163 - 12nd (one second).mp3	12nd (one second) - 12nd - The Minute.mp3
5 AM_166267550 - (dreamchild).mp3	(dreamchild) - 5 AM.mp3
5_156628547 - redo.mp3	redo - 5.mp3
8;00_133775391 - Milk &Honey.mp3	Milk &Honey - 8;00.mp3
Antidote WAkachi_308796301 - SUGHT.mp3	SUGHT - Antidote WAkachi.mp3
Anymus_192638322 - Inkyz.mp3	Inkyz - Anymus.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Ma...	K-Mac - Autoerotique - WTF (K-MAC Trap VIP).mp3
AWOLNATION - Run (B52 Remix)_217279053 - B52.mp3	B52 - AWOLNATION - Run (B52 Remix).mp3
b&B_296170807 - psupac.mp3	psupac - b&B.mp3
dabs#_256394142 - SUGHT.mp3	SUGHT - dabs#.mp3





Sample:

```
Match 1: Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Mac
Group 1: Autoerotique - WTF (K-MAC Trap VIP)
Group 2: K-Mac
```

Instead of relying on the pattern of 9 numeric digits, I made it greedy and matched all numeric digits so it would include any files that had less or more.

Remove

Switch Around String and Remove Digits cont.

Tim Contribution – cont.

Using: Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Mac.mp3

Match: (.*)_\d* - (.*)

Where:

\1 = Autoerotique - WTF (K-MAC Trap VIP) \2 = K-Mac

Tim Analysis:

1..	Match against any character	Capture Group 1 =
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore (Not Captured)	Changes Capture Group 1 = Autoerotique - WTF (K-MAC Trap VIP)
4. \d	Match against Numeric Digit (Not Captured) = '9'	
5. *	Make it Greedy (Not Captured) = '96732707'	
6. <space>	Match against a <space> character This is the <space> following '96732707'	
7. -	Match against <hyphen> character (Not Captured)	
8. <space>	Match against a <space> character	
9. .*	Match against remaining string	Capture Group 2 = K-Mac

Remove

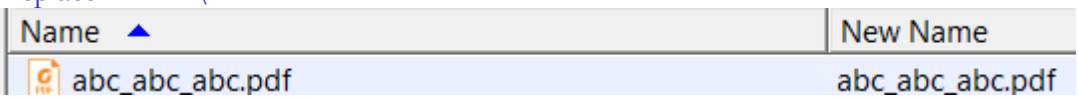
Keep last part of filename (preceded with underscore) with extension

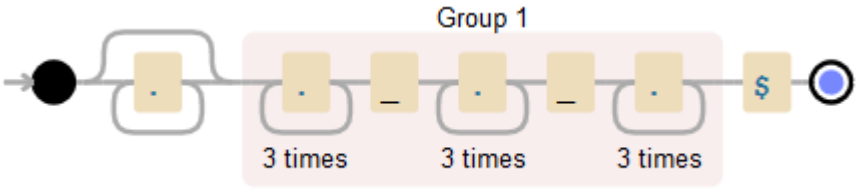
56t_cat_abc.pdf to abc.pdf
abc_abc_abc.pdf to abc.pdf

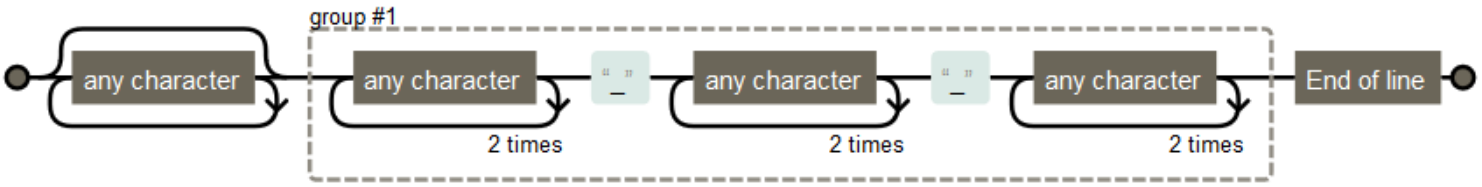
Observations:

All sections separated with underscore
Each section is made up of 3 characters

Match: .*(.{3}_.{3}_.{3})$
Replace: \1







Didn't work – although in forum it said it did.

Forum Contributor: KenP

Remove

Keep last part of filename (preceded with underscore) with extension cont.

56t_cat_abc.pdf to abc.pdf
abc_abc_abc.pdf to abc.pdf

Match: .*(.{3}_.{3}_.{3})$

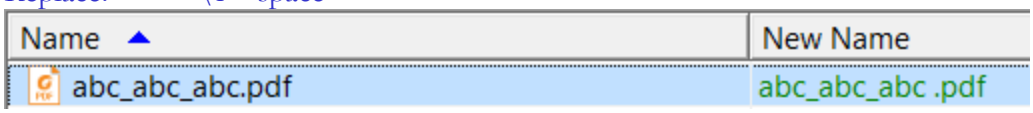
Tim Analysis:

The problem I can see is that all of the evaluation is done in Capture Group 1 gathering up the entire string.
This is further evidenced by the red indicator when you enter just '\' in the Replace.

Meaning that the RegEx is invalid – possible match but there is nothing new to display as far as New Name.

If you add a <space> to the Replacement String, you will see that the RegEx only gathered up the entire string by specifying it's exact pattern without parsing.

Match: .*(.{3}_.{3}_.{3})$
Replace: \1 <space>



This is further evidenced by the sample run which verifies the entire string is captured without parsing:

Sample:

```
.....................
Match 1: abc_abc_abc
Group 1: abc_abc_abc
```

Remove

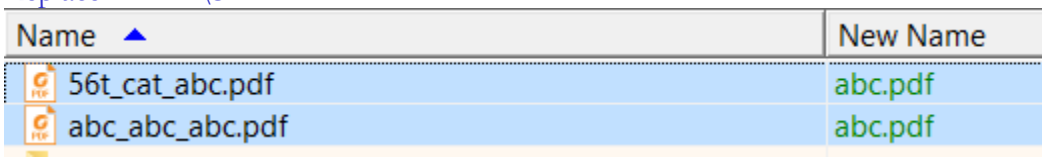
Keep last part of filename (preceded with underscore) with extension cont.

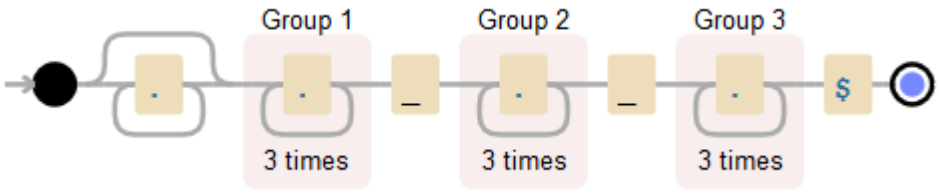
Tim Contribution –

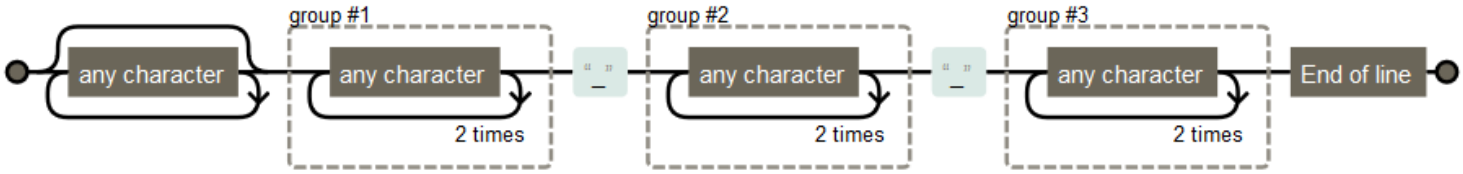
56t_cat_abc.pdf to abc.pdf
abc_abc_abc.pdf to abc.pdf

This works:

Match: .*(.{3})_(.{3})_(.{3})$
Replace: \3







Sample:

```
.......................
Match 1: abc_abc_abc
Group 1: abc
Group 2: abc
Group 3: abc
```

Sample:

```
.......................
Match 1: 56t_cat_abc
Group 1: 56t
Group 2: cat
Group 3: abc
```

Remove

Keep last part of filename (preceded with underscore) with extension cont.

Tim Contribution – cont.

Using: 56t_cat_abc

Match: .*(.{3})_(.{3})_(.{3})$

Where:

\1 = 56t \2 = cat \3 = abc

Tim Analysis:

1..	Match against any character (Not Captured) = '5'	
2. *	Make it Greedy (Not Captured) = <entire string>	
3. .	Match against any character	Capture Group 1 = c
4. {3}	Repeat last match for a total of three iterations	Capture Group 1 = abc
5. _	Match against underscore (Not Captured) This is the underscore after 'cat'	Changes Capture Group 1= cat
6. .	Match against any character	Capture Group 2 = a
7. {3}	Repeat last match for a total of three iterations	Capture Group 2 = abc
8. _	Match against underscore (Not Captured) This is the underscore after 'cat' Changes match of underscore in step 5 to underscore after '56t'. This in turn changes Capture Group 1 and 2.	Changes Capture Group 2 = cat Changes Capture Group 1 = 56t
9. {3}	Repeat last match for a total of three iterations	Capture Group 3 = abc
10. $	Must be at EOL	

Remove

Keep last part of filename (preceded with underscore) with extension cont.

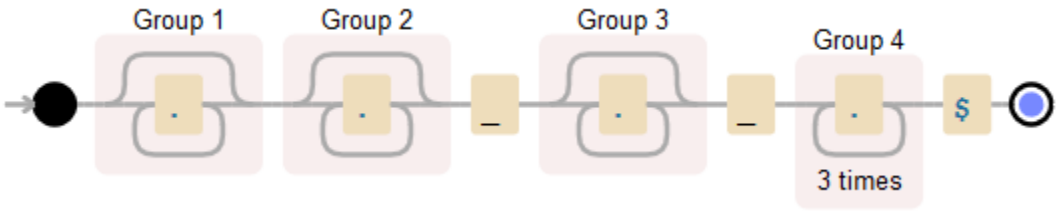
Tim Contribution –

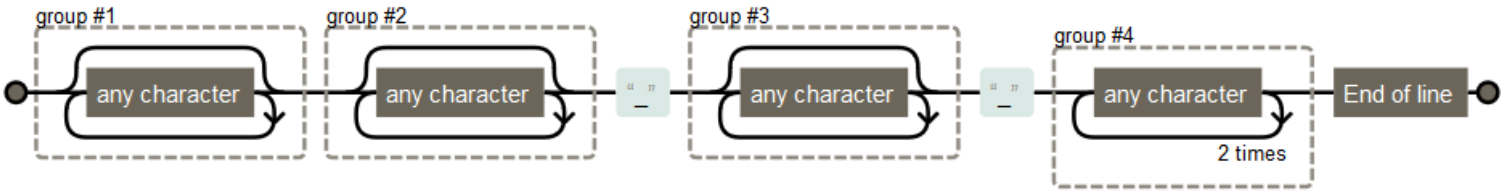
56t_cat_abc.pdf to abc.pdf
abc_abc_abc.pdf to abc.pdf

As does this:

Match: (.*)(.*)_(.*)_(.{3})$
Replace: \4

Name	New Name
56t_cat_abc.pdf	abc.pdf
abc_abc_abc.pdf	abc.pdf





Sample:

```
·····················Start·Length···
Match 1: 56t_cat_abc      0     11
Group 1: 56t              0      3
Group 2:                  3      0
Group 3: cat              4      3
Group 4: abc              8      3
```

Remove

Keep last part of filename (preceded with underscore) with extension cont.

Tim Contribution – cont.

Using: 56t_cat_abc

Match: (.*)(.*)_(.*)_(.{3})$

Where:

\1 = 56t \2 = null \3 = cat \4 = abc

Tim Analysis:

1..	Match against any character	Capture Group 1 = 5
2. *	Make it Greedy	Capture Group 1 = <entire string>
3..	Match against any character Backtracks from EOL to last character of string (of 'abc')	Capture Group 2 = c
4. *	Make it Greedy (Not Captured) = 'abc' This would be a null for New name because with step 1 and step 2 it matches against entire string again. This creates a Zero Occurrence Match resulting in a null value. Current position = EOL	Capture Group 2 = null
5. _	Match against underscore (Not Captured) This is the underscore after 'cat'	Changes Capture Group 1= 56t_cat Capture Group 2 = No Change
6..	Match against any character	Capture Group 3 = a
7. *	Make it Greedy Changes step 4 to 'cat' Changes step 2 to '56t'	Capture Group 3 = abc
8. _	Match against underscore (Not Captured) This is the underscore after 'cat' Changes match of underscore in step 5 to underscore after '56t'. This in turn changes Capture Group 1 and 3.	Changes Capture Group 3 = cat Capture Group 2 = No Change Changes Capture Group 1 = 56t
9..	Match against any character	Capture Group 4 = a
10. {3}	Repeat last match for a total of three iterations	Capture Group 4 = abc
11. $	Must be at EOL	

Remove

Keep last part of filename (preceded with underscore) with extension cont.

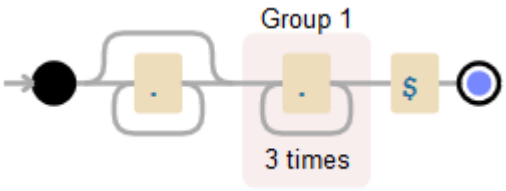
Tim Contribution – cont.

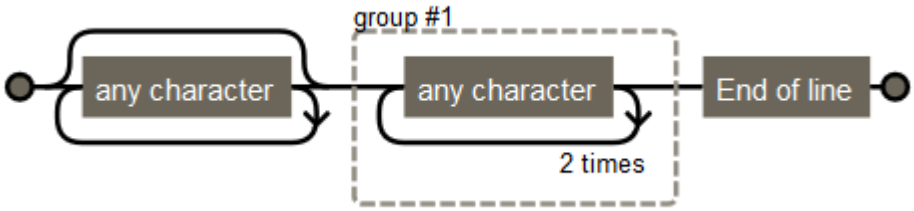
56t_cat_abc.pdf to abc.pdf
abc_abc_abc.pdf to abc.pdf

This will work regardless of the content before _abc:

Match: .*(.{3})$
Replace: \1

Name	New Name
56t_cat_abc.pdf	abc.pdf
abc_abc_abc.pdf	abc.pdf
_rty_tyff_testg_5776_abc.xml	abc.xml





Sample:

```
.......................
Match 1: 56t_cat_abc
Group 1: abc
```

Remove

Keep last part of filename (preceded with underscore) with extension cont.

Tim Contribution – cont.

Using: 56t_cat_abc.pdf

Match: .*(.{3})$

Where:

\1 = abc

Tim Analysis:

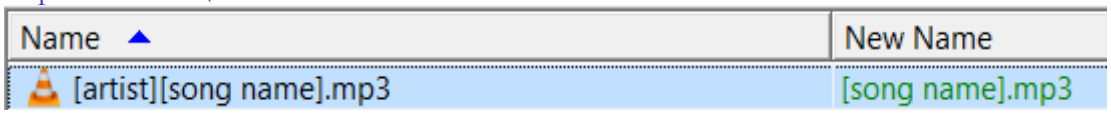
1..	Match against any character (Not Captured) = 5	
2. *	Make it Greedy (Not Captured) = <entire string>	
3. .	Match against any character Backtracking.	Capture Group 1 = c
4. {3}	Repeat last match for a total of 3 iterations	Capture Group 1 = abc

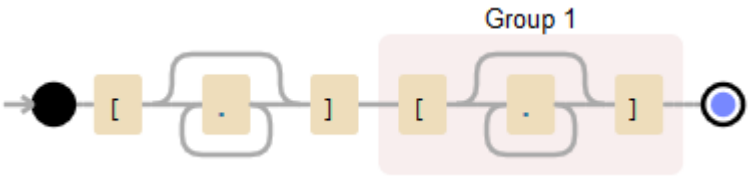
Remove

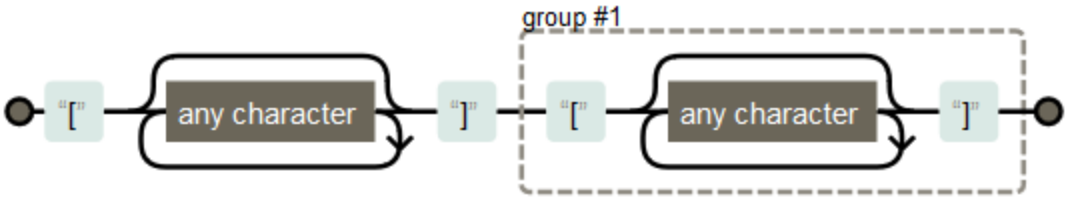
Remove everything within first set of square brackets

[artist][song name].mp3 to [songname].mp3

Match: \[.*\](\[.*\])
Replace: \1







Sample:

```
..............................
Match 1: [artist][song name]
Group 1: [song name]
```

Tim Notes:

1. This one is similar to 'Keep last part of filename (preceded with underscore) with extension'

Forum Contributor: KenP

Remove

Remove everything within first set of square brackets cont.

Use: [artist][song name].mp3

Match: \[.*\](\[.*\])

Where:

\1 = [song name]

Tim Analysis:

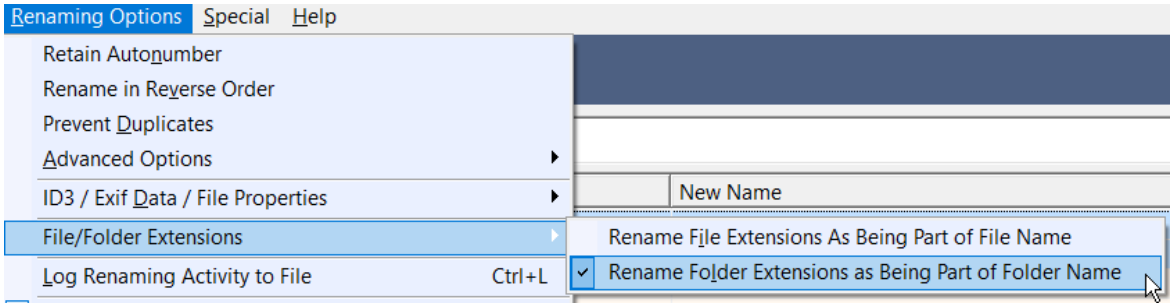
1. \[	Match against literal left bracket (Not Captured)	
2..	Match against any character (Not Captured) = 'a'	
3. *	Make it Greedy (Not Captured) = 'artist][song name]'	
4. \]	Match against literal right bracket (Not Captured)	
	Changes step 3 = '[artist]'	
5. \[	Match against literal left bracket	Capture Group 1 = [
6..	Match against any character	Capture Group 1 = [s
7. *	Make it Greedy	Capture Group 1 = [song name]
8. \]	Match against literal right bracket	Capture Group 1 = No Change

Remove

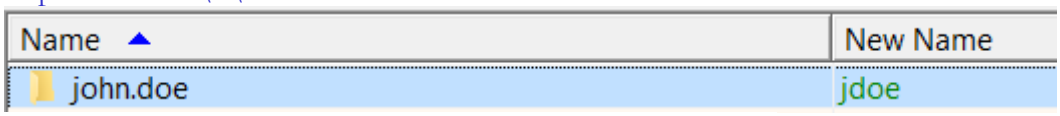
Change first name to Initials on folders

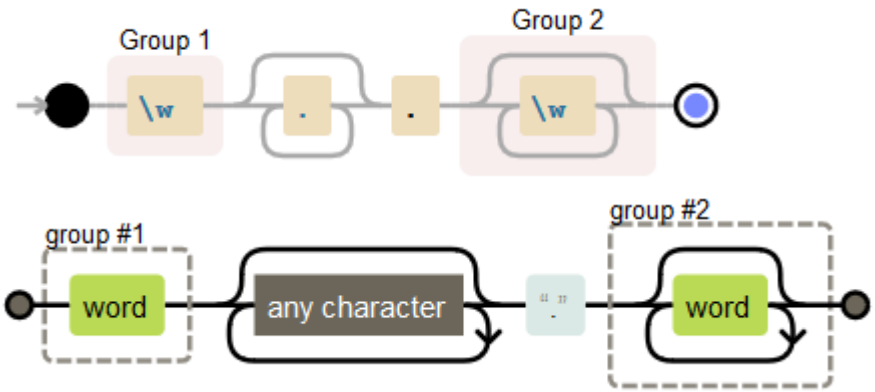
john.doe to jdoe

Because the folder names contain dot characters, set to Ignore file extensions.



Match: (\w).*\.(\w*)
Replace: \1\2





Sample:

```
.....................
Match 1: john.doe
Group 1: j
Group 2: doe
```

Tim Notes:

1. '\w' Word Metacharacters have previously been discussed.

Forum Contributor: KenP

Remove

Change first name to Initials on folders cont.

Using: john.doe

Match: (\w).*\.(\w*)

Where:

\1 = j \2 = doe

Tim Analysis:

1. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = j
1..	Match against any character (Not Captured) = 'o'	
2. *	Make it Greedy (Not Captured) = 'ohn.doe'	
3. .	Match against literal <dot> (Not Captured) Changes step 2 to 'ohn'	
4. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 2 = d
5. *	Make it Greedy	Capture Group 2 = doe

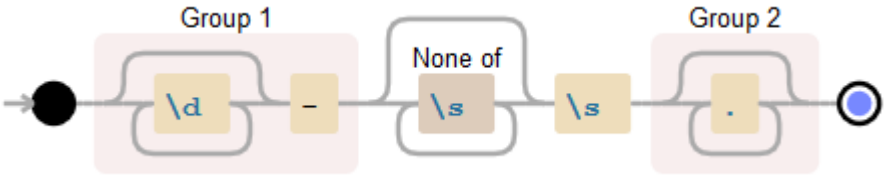
Remove

Delete the first word & <space> in string following the '-'

01-Devolv Whip it.wav	to	01-Whip it.wav
01-Devo Whip its.wav	to	01-Whip its.wav
01-Whip its Devo.wav	to	01-Devo.wav

Match: (\d*-)[^\s]*\s(.*)
Replace: \1\2

Name	New Name
01-Devolv Whip it.wav	01-Whip it.wav
01-Devo Whip its.wav	01-Whip its.wav
01-Whip its Devo.wav	01-its Devo.wav



Sample:

```
.............................
Match 1: 01-Devolv Whip it
Group 1: 01-
Group 2: Whip it
```

Forum Contributor: KenP

Remove

Delete the first word & <space> in string following the '-' cont.

Using: 01-Devolv Whip it.wav

Match: (\d*-)[^\s]*\s(.*)

Where:

\1 = 01- \2 = Whip it

Tim Analysis:

1. \d	Match against Numeric Digit	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = 01
3. -	Match against <hyphen>	Capture Group 1 = 01-
4. [^\s]	Match against a Negated Class consisting of a <space> (Not Captured) = 'D'	
5. *	Make it Greedy (Not Captured) = 'Devolv'	Capture Group 1 = Unchanged
6. \s	Match against <space> character (Not Captured) This is the <space> after 'Devolv'	
7..*	Match against remaining string	Capture Group 2 = Whip it

Remove

Delete the first word & <space> in string following the '-' cont.

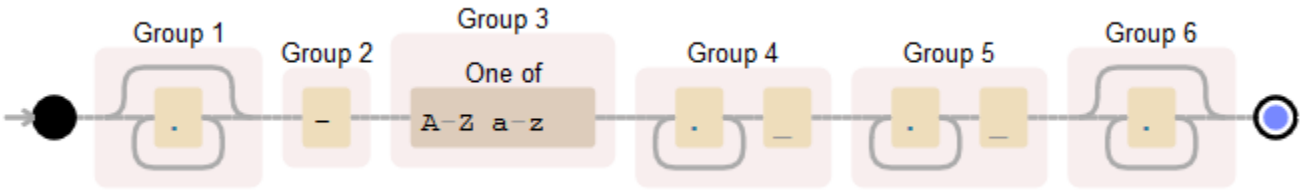
Tim Contribution – less elegant but it shows the parsing of the string more clearly, better for learning

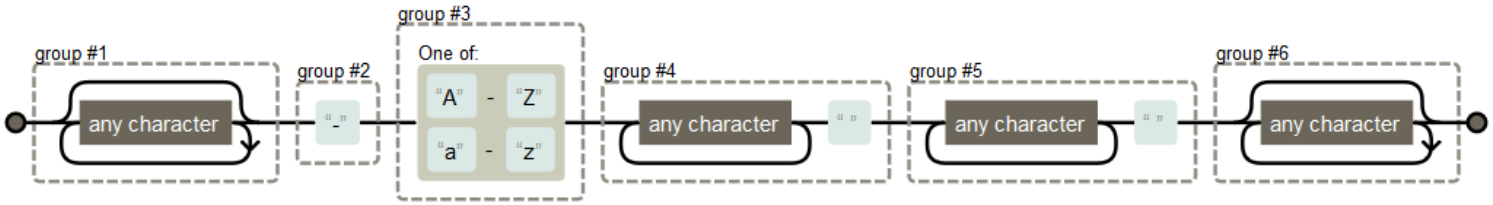
You may have noticed by now that some of my contributions utilize more Capture Groups than required. I do this to illustrate the parsing involved. You can see this for yourself by using separate back references \1 \2 \3 etc. and see the values assigned. This way you can learn to parse on your own. Many of the requests in the forum have to do with parsing. 'Give a man a fish and you feed him for a day; Teach a man to fish and you feed him for a lifetime.' – The Bible.

01-Devolv Whip it.wav	to	01-Whip it.wav
01-Devo Whip its.wav	to	01-Whip its.wav
01-Whip its Devo.wav	to	01-Devo .wav

Match: (.*)(-)([A-Za-z])(.+?)(.+?)(.*)
Replace: \1\2\5\6

Name	New Name
01-Devo Whip its.wav	01-Whip its.wav
01-Devolv Whip it.wav	01-Whip it.wav
01-Whip its Devo.wav	01-its Devo.wav





Sample:

```
Match 1: 01-Devolv Whip it
Group 1: 01
Group 2: -
Group 3: D
Group 4: evolv
Group 5: Whip
Group 6: it
```

This example got me thinking. I thought I would expound on this more:

Remove

Delete the first word & <space> in string following the '-' cont.

Tim Contribution – less elegant but it shows the parsing of the string more clearly, better for learning cont.

Using: 01-Devolv Whip it.wav

Match: (.*)(-)([A-Za-z])(.+?)(.+?)(.*)

Where:

\1 = 01 \2 = - \3 = D \4 = evolve \5 = Whip \6 = it

Tim Analysis:

1..	Match against any character	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen>	Capture Group 2 = -
	This is the <hyphen> after '01'	Changes Capture Group 1 = 01
4. [A-Za-z]	Match against an upper or lowercase letter	Capture Group 3 = D
5. .	Match against any character	Capture Group 4 = e
6. +	Make it Greedy	Capture Group 4 = evolv Whip it
7. ?	But not too Greedy	Capture Group 4 = e
8. <space>	Match against <space>	Capture Group 4 = evolv
	This is the <space> after 'Devolv'	
9. .	Match against any character	Capture Group 5 = W
10. +	Make it Greedy	Capture Group 5 = Whip it
11. ?	But not too Greedy	Capture Group 5 = W
12. <space>	Match against <space>	Capture Group 5 = Whip
	This is the <space> after 'Whip'	
11. .*	Match against remaining string	Capture Group 6 = it

Remove

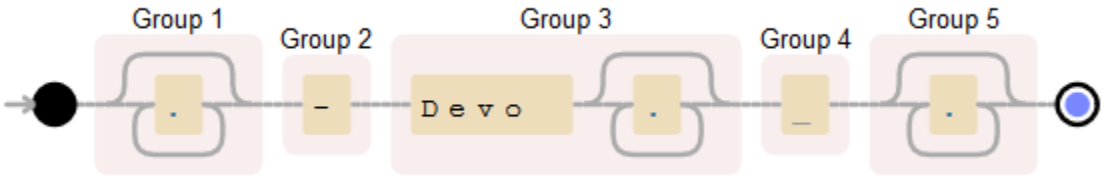
Remove 'Devo* <space>' from title

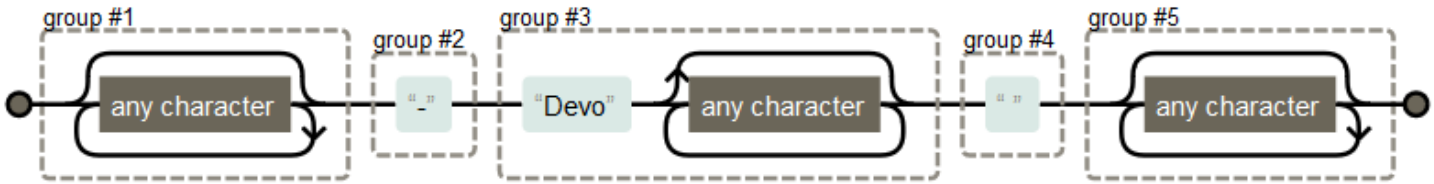
Tim Contribution –

01-Devolv Whip it.wav	to	01-Whip it.wav
01-Devo Whip its.wav	to	01-Whip its.wav
01-Whip its Devo.wav	to	01-Devo .wav

Match: (.*)(-)(Devo.*?)( )(.*)
Replace: \1\2\5

Name	New Name
01-Devolv Whip it.wav	01-Whip it.wav
01-Devo Whip its.wav	01-Whip its.wav
01-Whip its Devo.wav	01-Whip its Devo.wav





Sample:

```
·····························Start·Length··
Match 1: 01-Devolv Whip it       0      17
Group 1: 01                      0       2
Group 2: -                       2       1
Group 3: Devolv                  3       6
Group 4:                         9       1
Group 5: Whip it                10       7
```

Tim Notes:

1. A <space> must follow Devo, thus it would not work with '01-Whip its Devo.wav'

Remove

Remove 'Devo* <space>' from title cont.

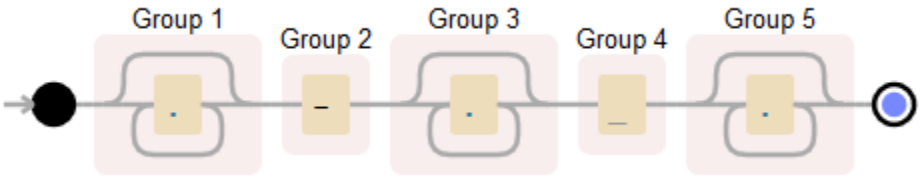
Tim Contribution – cont.

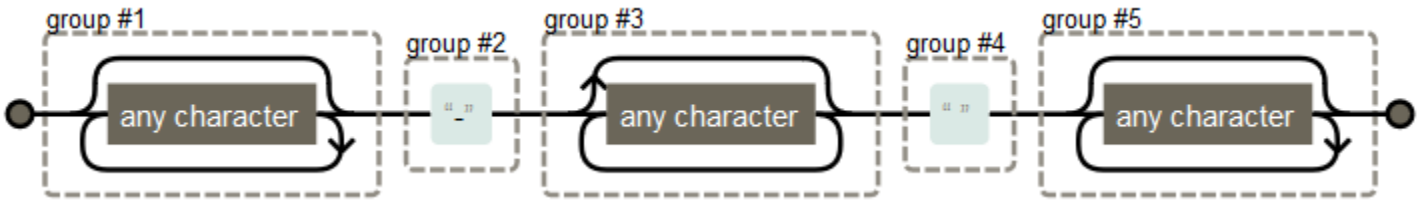
01-Devolv Whip it.wav	to	01-Whip it.wav
01-Devo Whip its.wav	to	01-Whip its.wav
01-Whip its Devo.wav	to	01-Devo .wav

To make it universal (to these samples) so it just retains last two words – change (Devo.*?) to:

Match: (.*)(-)(.*?)( )(.*)
Replace: \1\2\5

Name	New Name
01-Devo Whip its.wav	01-Whip its.wav
01-Devolv Whip it.wav	01-Whip it.wav
01-Whip its Devo.wav	01-its Devo.wav





Sample:

```
·····························Start·Length··
Match 1: 01-Devolv Whip it      0      17
Group 1: 01                     0       2
Group 2: -                      2       1
Group 3: Devolv                 3       6
Group 4:                        9       1
Group 5: Whip it               10       7
```

Simple.

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Using: 01-Devolv Whip it.wav

Match: (.*)(-)(.*?)( )(.*)

Where:

\1 = 01 \2 = - \3 = Devolv \4 = <space> \5 = Whip it

Tim Analysis:

1..	Match against any character	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen>	Capture Group 2 = -
		Changes Capture Group 1 = 01
4..	Match against any character	Capture Group 3 = D
5. *	Make it Greedy	Capture Group 3 = Devolv Whip it
6. ?	But not too Greedy	Capture Group 3 = BOL
7. <space>	Match against a <space> character	Capture Group 4 = <space>
	This is the <space> after 'Devolv'	Changes Capture Group 3 = Devolv
8. .*	Match against remaining string	Capture Group 5 = Whip it

Remove

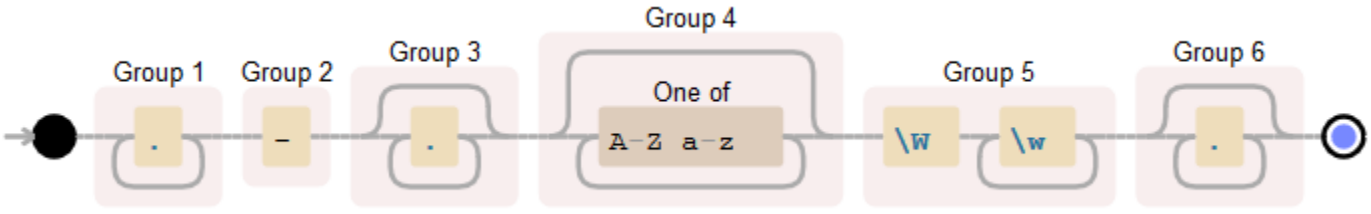
Remove 'Devo* <space>' from title cont.

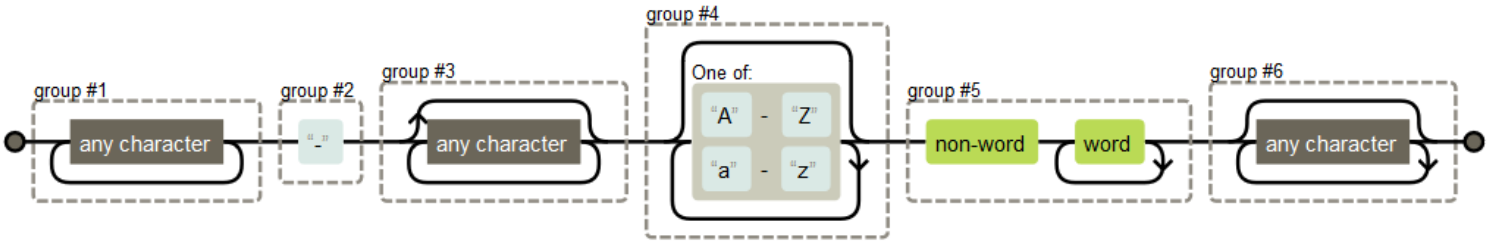
Tim Contribution – cont.

Remove all words before last word in string retaining characters at beginning up to <hyphen>

01-Devolv Whip it.wav	to	01-Whip it.wav
01-Devo Whip its.wav	to	01-Whip its.wav
01-Whip its Devo.wav	to	01-Devo.wav

Match: (.+)(-)(.*?)([A-Za-z]*)(\W\w+)(.*)





Sample:

	Start	Length
Match 1: 01-Devolv Whip it	0	17
Group 1: 01	0	2
Group 2: -	2	1
Group 3:	3	0
Group 4: Devolv	3	6
Group 5: Whip	9	5
Group 6: it	14	3

Where:

\1 = 01 \2 = - \3 = "" \4 = Devolv \5 = <space> Whip \6 = <space> it

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Remove all words before last word in string retaining characters at beginning up to <hyphen> cont.

01-Devolv Whip it.wav to 01-Whip it.wav
01-Devo Whip its.wav to 01-Whip its.wav
01-Whip its Devo.wav to 01-Devo.wav

Match: (.+)(-)(.*?)([A-Za-z]*)(\W\w+)(.*)

Replace: \1-\4 = first word

Name	New Name
01-Devolv Whip it.wav	01-Devolv.wav
01-Devo Whip its.wav	01-Devo.wav
01-Whip it Devo.wav	01-Whip.wav

Replace: \1-\5 = Second word

Name	New Name
01-Devolv Whip it.wav	01- Whip.wav
01-Devo Whip its.wav	01- Whip.wav
01-Whip it Devo.wav	01- it.wav

Replace: \1-\6 = Third Word

Name	New Name
01-Devolv Whip it.wav	01- it.wav
01-Devo Whip its.wav	01- its.wav
01-Whip it Devo.wav	01- Devo.wav

Tim Notes:

1. Since the <hyphen> was also captured, you could replace or do away with this as well in the Replace String.

Replace: \1 \4 \5 \6

Name	New Name
01-Devolv Whip it.wav	01 Devolv Whip it.wav
01-Devo Whip its.wav	01 Devo Whip its.wav
01-Whip it Devo.wav	01 Whip it Devo.wav

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Remove all words before last word in string retaining characters at beginning up to <hyphen> cont.

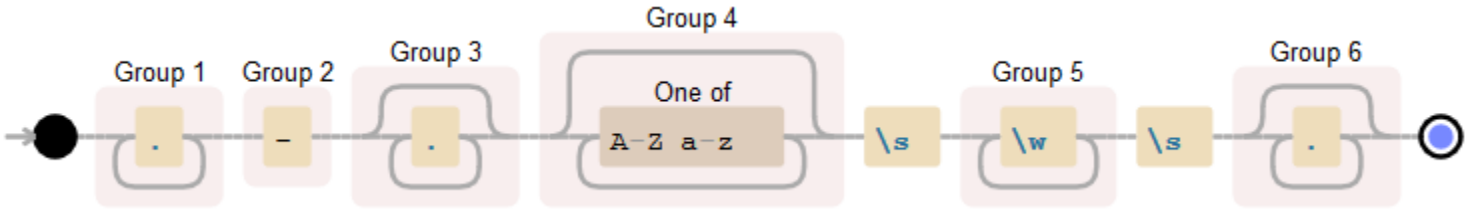
01-Devolv Whip it.wav	to	01-Whip it.wav
01-Devo Whip its.wav	to	01-Whip its.wav
01-Whip its Devo.wav	to	01-Devo.wav

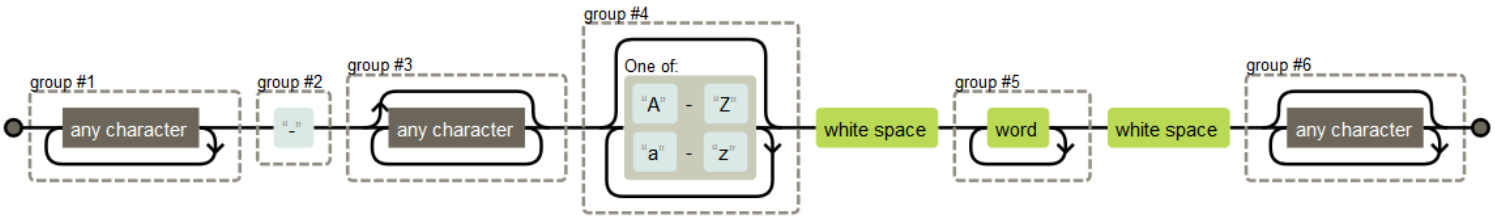
If you want to get rid of the <space> before the third word and second word in RegEx, use this:

Match: (.+)(-)(.*?)([A-Za-z]*)\s(\w+)\s(.*)

Replace: same as on previous page.. i.e., for Third Word – 1-\6

Name	New Name
01-Devolv Whip it.wav	01-it.wav
01-Devo Whip its.wav	01-its.wav
01-Whip it Devo.wav	01-Devo.wav





Sample:

```
                            Start  Length
Match 1: 01-Devolv Whip it      0      17
Group 1: 01                     0       2
Group 2: -                      2       1
Group 3:                        3       0
Group 4: Devolv                 3       6
Group 5: Whip                  10       4
Group 6: it                    15       2
```

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Remove all words before last word in string retaining characters at beginning up to <hyphen> cont.

Where … using the sample, 01-Whip it Devo.wav:

\1 = 01 \2 = - \3 = null \4 = Whip \5 = <space>it \6 =<space> Devo

… and the Original RegEx:

Match: (.+)(-)(.*?)([A-Za-z]*)(\W\w+)(.*) (the RegEx that doesn't remove the <spaces>)

Tim Analysis:

1. .	Match against first character	Capture Group 1 = 0
2. +	Make it Greedy until….	Capture Group 1 = entire string
3. -	<hyphen>	Capture Group 2 = <hyphen> Changes Capture Group 1 = 01
4. .	Match against next character	Capture Group 3 = W
5. *	Make it Greedy	Capture Group 3 = Whip It Devo
6. ?	But not too Greedy (Non-Greedy) – Zero Occurrence Match	Capture Group 3 = null
7. [A-Za-z]	Match against an upper or lowercase letter	Capture Group 4 = W
8. *	Make it Greedy	Capture Group 4 = Whip
9. \W	Match against a Non-Word Metacharacter followed by … This is the <space> following 'Whip'	Capture Group 5 = <space>
10. \w	.. a Word Metacharacter This is the '<space>' before 'it' and the 'i' in 'it'	Capture Group 5 = <space> i
11. +	Make it Greedy	Capture Group 5 = <space> it
12. .	Match against next character This is the <space> before 'Devo' RegEx Engine is backtracking	Capture Group 6 = <space>
13. *	Make it Greedy RegEx moves forward to end of string	Capture Group 6 = <space>Devo

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Remove all words before last word in string retaining characters at beginning up to <hyphen> cont.

Where … using the sample, 01-Whip it Devo.wav:

\1 = 01 \2 = - \3 = null \4 = Whip \5 = it \6 = Devo

… and the Modified RegEx:

Match: (.+)(-)(.*?)([A-Za-z]*)\s(\w+)\s(.*) (the RegEx that does remove the <spaces>)

Tim Analysis:

1. .	Match against first character	Capture Group 1 = 0
2. +	Make it Greedy until….	Capture Group 1 = entire string
3. -	<hyphen>	Capture Group 2 = <hyphen> Changes Capture Group 1 = 01
4. .	Match against next character	Capture Group 3 = W
5. *	Make it Greedy	Capture Group 3 = Whip It Devo
6. ?	But not too Greedy (Non-Greedy) - Zero Occurrence Match	Capture Group 3 = null
7. [A-Za-z]	Match against an upper or lowercase letter	Capture Group 4 = W
8. *	Make it Greedy	Capture Group 4 = Whip
9. \s	Match against Whitespace Metacharacter (Not Captured) = <space> (this is the <space> before 'it'	
10. \w	Match against a Word Metacharacter	Capture Group 5 = i
11. +	Make it Greedy	Capture Group 5 = it
12. \s	Match against Whitespace Metacharacter (Not Captured) = <space> (this is the <space> before 'Devo'	
13. .	Match against next character	Capture Group 6 = D
14. *	Make it Greedy	Capture Group 6 = Devo

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

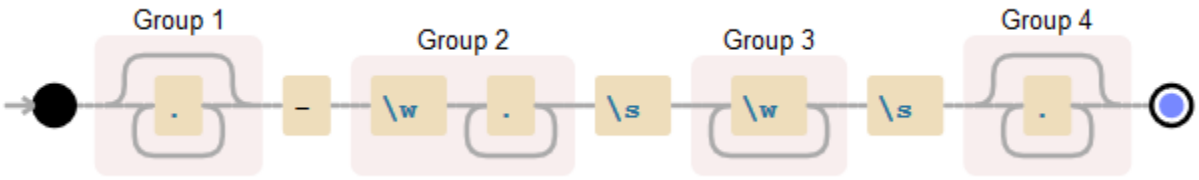
Tim Notes:

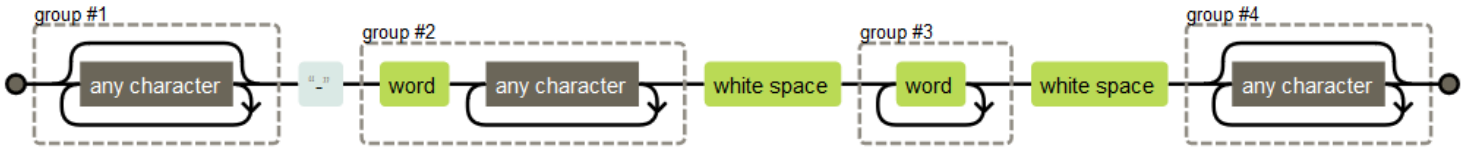
1. Are there better ways to do this – absolutely- In RegEx there are always (usually?) more than one way to accomplish your goal. I used a method that would parse each word so I could put it togther in a number of ways. Nothing new there but I also wanted to demonstrate how this could be accomplished by more than just the standard way of using '(.+)' or '(\d\d)-' etc.

Another method of parsing the String

01-Devolv Whip it.wav	to	01-Whip.wav
01-Devo Whip its.wav	to	01-Whip.wav
01-Whip its Devo.wav	to	01-its.wav

Match: (.*)-(\w.+)\s(\w+)\s(.*)





Sample:

```
..............................
Match 1: 01-Devolv Whip it
Group 1: 01
Group 2: Devolv
Group 3: Whip
Group 4: it
```

Tim Notes:

1. Removes that Zero Occurrence Match that created a null value for Capture Group 3.
2. Reduces Capture Groups down from 6 to 4 because of Capture Group 3 redefined and <hyphen> not captured.

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Another method of parsing the String cont.

01-Devolv Whip it.wav	to	01-Whip.wav
01-Devo Whip its.wav	to	01-Whip.wav
01-Whip its Devo.wav	to	01-its.wav

Match: (.*)-(\w.+)\s(\w+)\s(.*)

Replace: \2 = First Word

Name	New Name
01-Devolv Whip it.wav	Devolv.wav
01-Devo Whip its.wav	Devo.wav
01-Whip its Devo.wav	Whip.wav

Replace: \3 = Second word

Name	New Name
01-Devo Whip its.wav	Whip.wav
01-Devolv Whip it.wav	Whip.wav
01-Whip its Devo.wav	its.wav

Replace: \4 = Third Word

Name	New Name
01-Devolv Whip it.wav	it.wav
01-Devo Whip its.wav	its.wav
01-Whip its Devo.wav	Devo.wav

Replace: \3 \4 = Last two Words

Name	New Name
01-Devolv Whip it.wav	Whip it.wav
01-Devo Whip its.wav	Whip its.wav
01-Whip its Devo.wav	its Devo.wav

Remove

Remove 'Devo* <space>' from title cont.

Tim Contribution – cont.

Another method of parsing the String cont.

Using … 01-Whip its Devo.wav as the sample:

Match: (.*)-(\w.+)\s(\w+)\s(.*)

Where:

\1 = 01 \2 = Whip \3 = its \4 = Devo

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 0
2. *	Make it Greedy until…	Capture Group 1 = entire string
3. -	.. <hyphen> (Not Captured)	Changes Capture Group 1 =01
4. \w	Match against a Word Metacharacter	Capture Group 2 = W
5. +	Make it Greedy	Capture Group 2 = Whip
6. \s	Match against Whitespace Metacharacter (Not Captured) = <space> (this is the <space> after 'Whip'	
7. \w	Match against a Word Metacharacter	Capture Group 3 = i
8. +	Make it Greedy	Capture Group 3 = its
9. \s	Match against Whitespace Metacharacter (Not Captured) = <space> (this is the <space> after 'its'	
10. .	Match against next character	Capture Group 4 = D
11. *	Make it Greedy	Capture Group 4 = Devo

Remove

Remove Underscore from Title, Replace with <space> and Remove Brackets

1999_06_00_Darkangel_[Promo]	to	1996_06_00 Darkangel (Promo)
2000_04_03_Standing_-_Burning_Empires	to	2000_04_03 Standing - Burning Empires
2002_03_00_Beloved	to	2002_03_00 Beloved

1. Remove all underscores in title
2. Remove any brackets in title

Solution: Two separate runs:

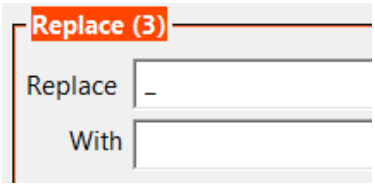
Run #1

1. Use 'Section #3: Replace' to remove ALL underscores and replace with a <space>
2. Use 'Section #14: Character Translation' to replace the brackets with parentheses

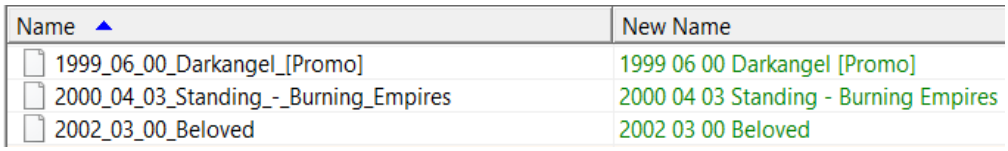
Run #2
Use RegEx to restore the underscores in the dates.

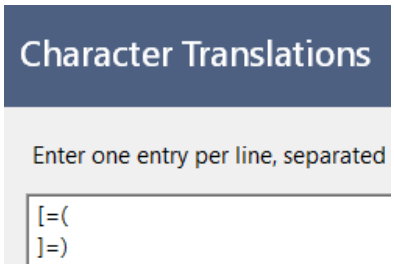
First run -

Replace underscore with <space>



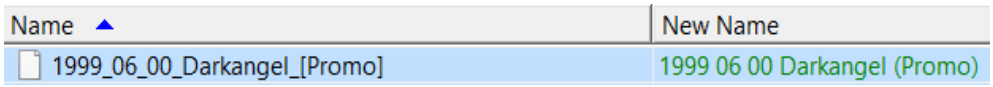
So far –



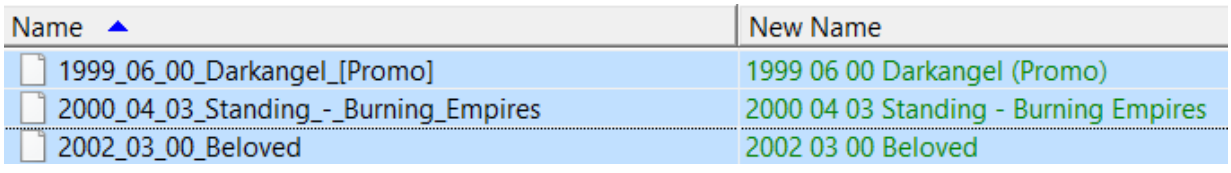


Character Translation for converting brackets to parentheses

This is the result after Character Translation –



After First run:



Remove

Remove Underscore from Title, Replace with <space> and Remove Brackets cont.

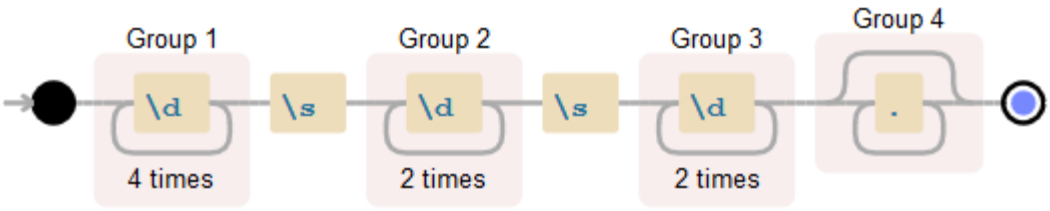
Second run -

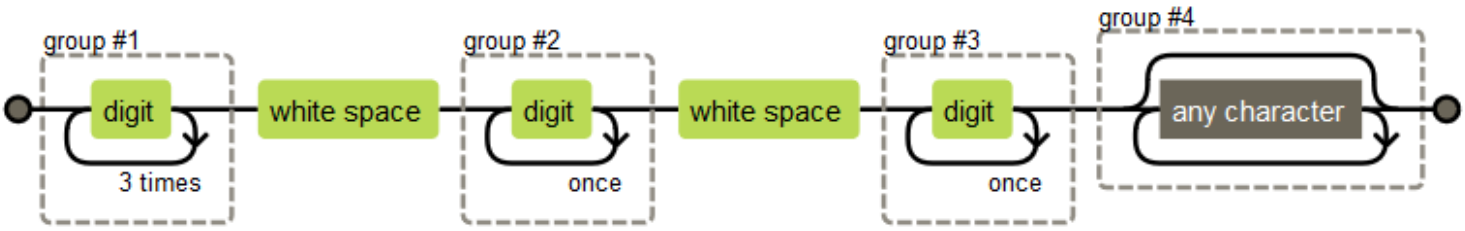
Restore underscore in dates using RegEx

Match: (\d{4})\s(\d{2})\s(\d{2})(.*)
Replace: \1_\2_\3\4

After Second run:

Name	New Name
1999 06 00 Darkangel (Promo)	1999_06_00 Darkangel (Promo)
2000 04 03 Standing - Burning Empires	2000_04_03 Standing - Burning Empires
2002 03 00 Beloved	2002_03_00 Beloved





Using: 1999 06 00 Darkangel_(Promo)

Sample:

```
                                        Start  Length
Match 1: 1999 06 00 Darkangel_(Promo)       0      28
Group 1: 1999                               0       4
Group 2: 06                                 5       2
Group 3: 00                                 8       2
Group 4:  Darkangel_(Promo)                10      18
```

Forum Contributor: KenP

Remove

Remove Underscore from Title, Replace with <space> and Remove Brackets cont.

Using: 1999 06 00 Darkangel_(Promo)

Match: (\d{4})\s(\d{2})\s(\d{2})(.*)

Where:

\1 = 1999 \2 = 06 \3 = 00 \4 = <space> Darkangel_(Promo)

Tim Analysis:

1. \d {4}	Match against 4 numeric digits	Capture Group 1 = 1999
2. \s	Match against a <space> character (Not Captured) This is the <space> following '1999'	
3. \d {2}	Match against 2 numeric digits	Capture Group 2 = 06
4. \s	Match against a <space> character (Not Captured) This is the <space> following '06'	
5. \d {2}	Match against 2 numeric digits	Capture Group 3 = 00
6. .*	Match against remaining string	Capture Group 4 = <space>Darkangel_(Promo)

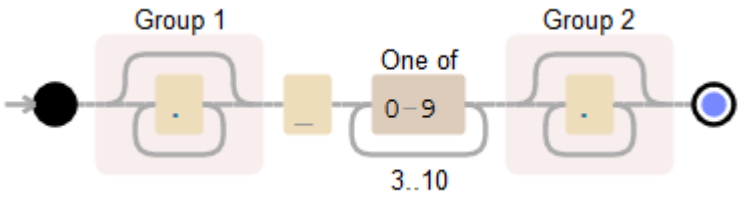
Remove

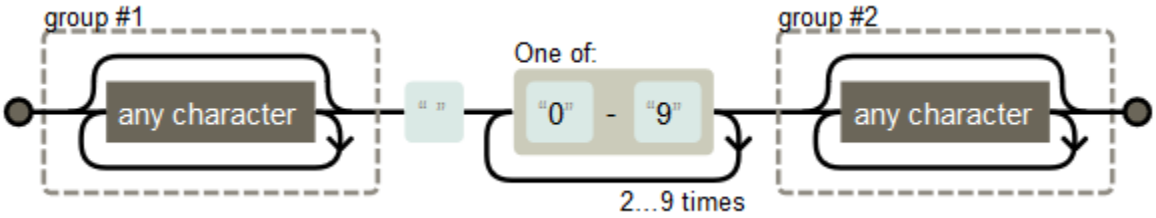
Remove Numeric Digits for any files that have Three or more

mushy 1234567 fruit.cat to mushy fruit.cat
mushy 12 fruit.cat ignore

Match: (.*) [0-9]{3,10}(.*)
Replace: \1\2

Name	New Name
mushy 12 fruit.cat	mushy 12 fruit.cat
mushy 1234567 fruit.cat	mushy fruit.cat





Sample:

```
................................
Match 1: mushy 1234567 fruit
Group 1: mushy
Group 2:  fruit
```

Tim Notes:

1. Don't forget the <space> after (.*)
2. A more descriptive title would be –
   'Remove 1st 3-10 Numeric Digits after <space> and everything up-to/including a final <space>'

Forum Contributor: deselby

Remove

Remove Numeric Digits for any files that have Three or more cont.

Using: mushy 1234567 fruit.cat

Match: (.*) [0-9]{3,10}(.*)

Where:

\1 = mushy \2 = <space>fruit

Tim Analysis:

1..	Match against any character	Capture Group 1 = m
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> following '7'	Changes Capture Group 1 = mushy 1234567
3. [0-9]	Match against a class consisting of a numeric digit '0' - '9' (Not Captured) = 1	Changes Capture Group 1 = mushy

Tim Notes:

1. The '7' is the last numeric digit in the string, so it moves forward until it hits the EOL, at which point it begins to backtrack attempting to match against a <space> followed by a numeric digit. Matches against the <space> after 'mushy' and captures the '1'.

4. {3,10}	Repeat last expression for a minimum of 3 iterations up to a maximum of 10. = '1234567'	
5. .*	Match against remaining string	Capture Group 2 = <space> fruit

Remove

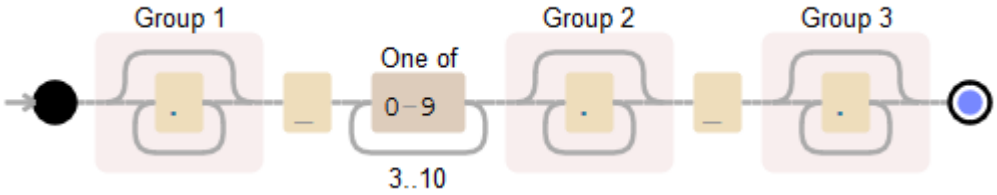
Remove Numeric Digits for any files that have Three or more cont.

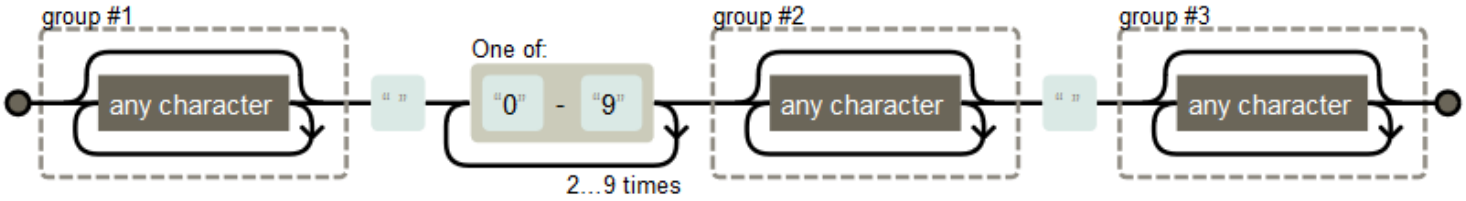
Tim Contribution –

Here is some alternate code:

Match: (.*) [0-9]{3,10}(.*) (.*) Don't forget the <space> after (.*)
Replace: \1 \3

Name	New Name
mushy 12 fruit.cat	mushy 12 fruit.cat
mushy 1234567 fruit.cat	mushy fruit.cat





Sample:

```
                              Start  Length
Match 1: mushy 1234567 fruit      0      19
Group 1: mushy                    0       5
Group 2:                         13       0
Group 3: fruit                   14       5
```

Where:
\1 = mushy \2 = null \3 = fruit

Remove

Remove Numeric Digits for any files that have Three or more cont.

Tim Contribution – cont.

Using: mushy 1234567 fruit.cat

Match: (.*) [0-9]{3,10}(.*) (.*)

Where:
\1 = mushy \2 = null \3 = fruit

Tim Analysis:

1. .	Match against next character	Capture Group 1 = m
2. *	Make it Greedy until…	Capture Group 1 = entire string
3. <space>	… <space> after 'mushy'	Capture Group 1 – mushy 1234567
4. [0-9]	Match against a numeric digit in class 0-9 (Not Captured) = '1'	Changes Capture Group 1 = mushy
5. {3,10}	Specifies a minimum of 3 and maximum of 10 repetitions until …. = '1234567'	Capture Group 1 = Unchanged
6. .	Matches against next character This is <space> after ' 7 '	Capture Group 2 = <space>
7. *	Make it Greedy until …	Capture Group 2 = <space>fruit
8. <space>	… <space>	Changes Capture Group 2 = null
9. .	Matches against next character	Capture Group 3 = f
		Capture Group 3 = fruit

Tim Notes:

1. By setting {3,10}, the RegEx can remove from the 3rd to the 10th of numeric digits after the <space>. In the above example there are 7, so it falls within this range. It would not affect filenames that have only 2 numeric digits and that is what the original poster wanted - not to affect 'mushy 12 fruit.cat'.

2. A more descriptive title would be –

'Remove 1st 3-10 Numeric Digits after <space> and everything up-to/including a final <space>'

Remove

Remove all except the 'pattern in bold' part of filename

Samples:

F_Personal_UP_Job_102_Balustrade_Balustrade Model_100617_Library_Balustrade_PlotFiles_Prototypes_Single Part_**TP401-V22-BP2** A3 Advance Steel

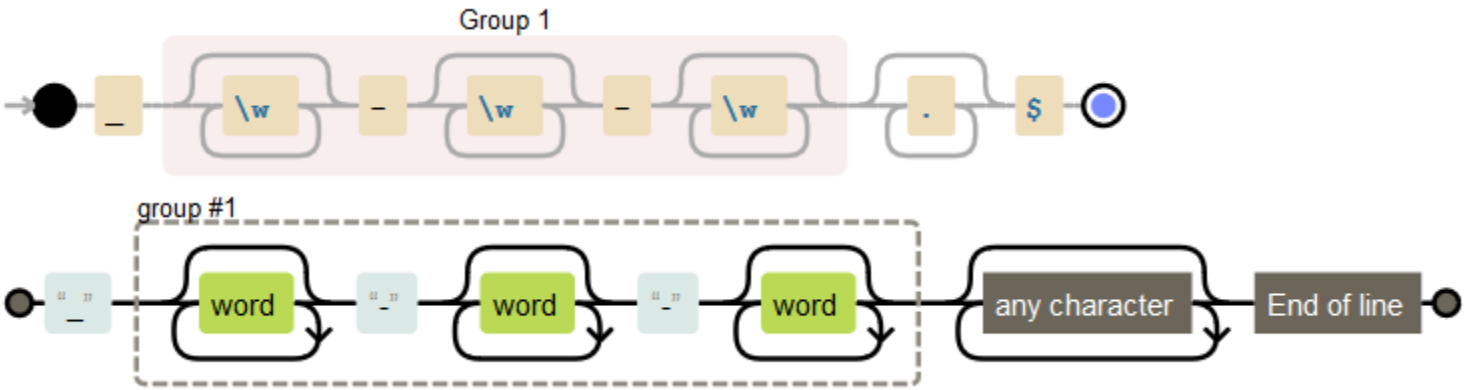
F_Personal_UP_Job_102_Balustrade_Balustrade Model_100617_Library_Balustrade_PlotFiles_Prototypes_Single Part_**TP401-V22-p2** A3 Advance Steel

Observations:

1. The bolded part is the only section that uses <hyphens>. The rest of the string uses underscores.
2. Pattern of bolded part is <alpha/numeric> '-' <alpha/numeric> '-' <alpha/numeric>
3. The RegEx is going to find the occurrence of the pattern and isolate it.

Match: _(\w*-\w*-\w*).*$
Replace: \1

Name	New Name
F_Personal_UP_Job_102_Balustrade_Balustrade Model_100617_Li...	TP401-V22-p2
F_Personal_UP_Job_102_Balustrade_Balustrade Model_100617_Li...	TP401-V22-BP2



Sample:

```
........................................
Match 1: _TP401-V22-BP2 A3 Advance Steel
Group 1: TP401-V22-BP2
```

Tim Notes:

1. NAME too long to see in photo. They are the sample files, trust me.

Forum Contributor: KenP

Remove

Remove all except the 'pattern in bold' part of filename cont.

Using: F_Personal_UP_Job_102_Balustrade_Balustrade Model_100617_Library_Balustrade_PlotFiles_Prototypes_Single Part_**TP401-V22-BP2** A3 Advance Steel

Match: _(\w*-\w*-\w*).*$

Where:

\1 = **TP401-V22-BP2**

Tim Analysis:

1. _	Match against underscore character (Not Captured) This is the underscore preceding 'Personal'	
2. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = P
3. *	Make it Greedy Includes all words and underscores up to the <space> preceding 'Model'	Capture Group 1 = Personal_UP_Job_102_ Balustrade_Balustrade
4. -	Match against <hyphen>	Capture Group 1 = TP401-

This is not so clear.

Here is what is happening at this point.

When the <hyphen> is added, the RegEx Engine begins testing *each* character for:

(a) underscore (not captured)

(b) if matched, it looks for word characters (this is the \w*) followed by a <hyphen>
1. Each time it will move forward through the string until the <space> after 'Single'
2. It has not matched against the <hyphen>

(c) If no match, starts RegEx over [from (a)] and tests the *next* character

This won't match until it reaches the underscore after 'Part'

(d) Matched, looks for word characters followed immediately by a <hyphen>
1. Matches against TP401-

Capture Group 1 = TP401-

Remember the Underscore of (a) is not captured and that is why Capture Group 1 <> ' _TP401-'

Remove

Remove all except the 'pattern in bold' part of filename cont.

Using: F_Personal_UP_Job_102_Balustrade_Balustrade Model_100617_Library_Balustrade_PlotFiles_Prototypes_Single Part_**TP401-V22-BP2** A3 Advance Steel

Match: _(\w*-\w*-\w*).*$

Where:

\1 = **TP401-V22-BP2**

Tim Analysis: cont.

5. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = TP401-V
6. *	Make it Greedy The <hyphen> is not a word character so it stops matching at the <hyphen> following 'V22'	Capture Group 1 = TP401-V22
7. -	Match against <hyphen>	Capture Group 1 = TP401-V22-
8. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = TP401-V22-B
9. *	Make it Greedy Matches until the <space> after 'BP2'	Capture Group 1 = TP401-V22-BP2
10. .*	Match against remaining string (Not Captured) = '<space>A3 Advanced Steel'	
11. $	Must be at end of string (EOL)	

Remove

Get rid of the URL part of the name

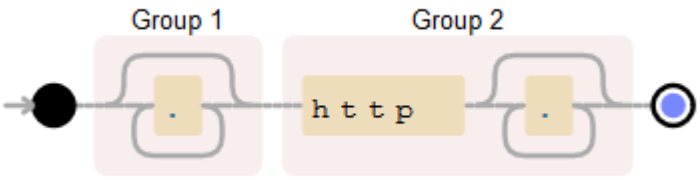
dd(20160625) hh(163602'') -Nobel Prize -httpwww.iflscience.combrainnobel-prize -.url
to
dd(20160625) hh(163602'') -Nobel Prize - .url

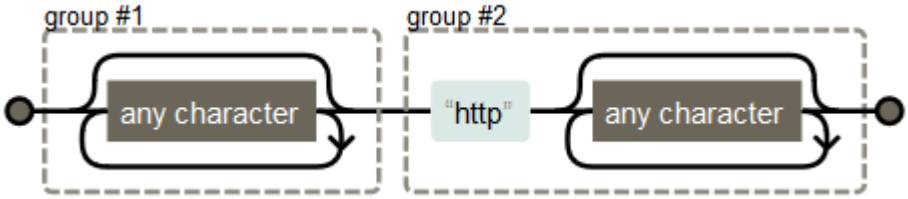
Observations:

1. Portion to exclude always start with http and all the files end with .url

Match: (.*)(http.*)
Replace: \1

Name	New Name
dd(20160625) hh(163602'') -Nobel Prize -httpwww.iflscience.co...	dd(20160625) hh(163602'') -Nobel Prize -.url





Sample:

```
Match 1: dd(20160625) hh(163602'') -Nobel Prize -httpwww.iflscience.combrainnobel-prize -
Group 1: dd(20160625) hh(163602'') -Nobel Prize -
Group 2: httpwww.iflscience.combrainnobel-prize -
```

Forum Contributor: KenP

Remove

Get rid of the URL part of the name cont.

Using: dd(20160625) hh(163602") -Nobel Prize -httpwww.iflscience.combrainnobel-prize -.url

Match: (.*)(http.*)

Where:

\1 = dd(20160625) hh(163602") -Nobel Prize - \2 = httpwww.iflscience.combrainnobel-prize -

Tim Analysis:

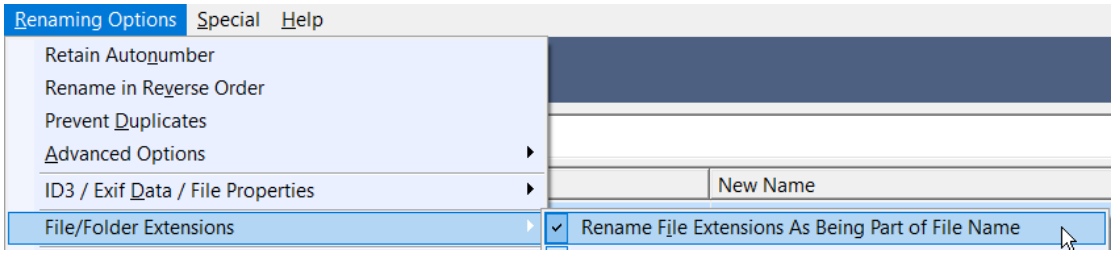
1..	Match against any character	Capture Group 1 =
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. http	Match against string 'http'	Capture Group 2= http Changes Capture Group 1 = dd(20160625) hh(163602") -Nobel Prize -
4. .*	Match against remaining string	Capture Group 2 = httpwww.iflscience.combrainnobel-prize –

Remove

Remove Extraneous Junk at end of Filename including Extension and all Spaces in Filename

A-001 - General Notes Symbols Abbreviations.pdf-M-0-0-A-001 - General Notes Symbols Abbreviations
to
A-001-GeneralNotesSymbolsAbbreviations

Set to Ignore File names:

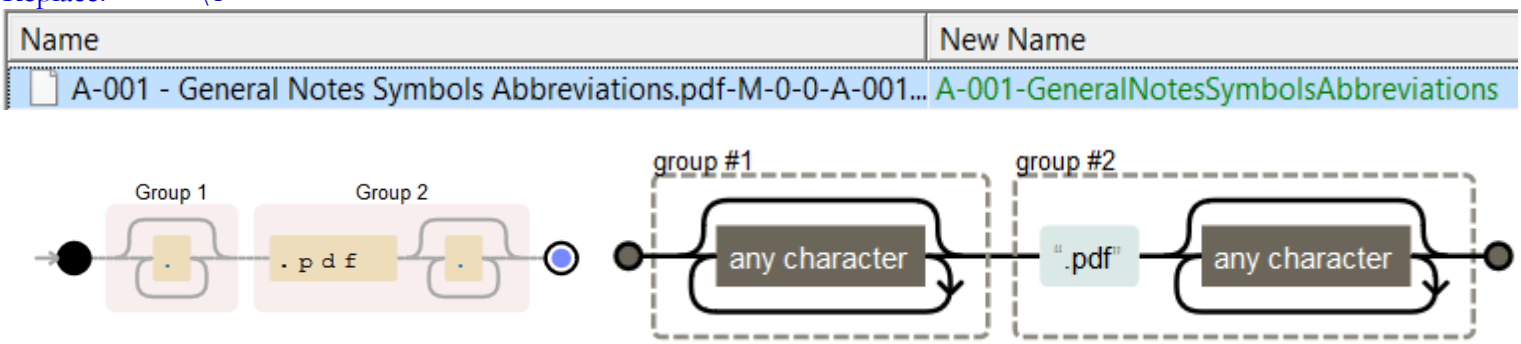


1. Set 'Section #3: Replace' to remove <spaces>:

Replace: <space>
With: "" null (nothing)

2. RegEx:

Match: (.*)(\.pdf.*)
Replace: \1



Sample:

```
Match 1: A-001 - General Notes Symbols Abbreviations.pdf-M-0-0-A-001 - General Notes Symbols Abbreviations
Group 1: A-001 - General Notes Symbols Abbreviations
Group 2: .pdf-M-0-0-A-001 - General Notes Symbols Abbreviations
```

Forum Contributor: therube

Remove

Remove Extraneous Junk at end of Filename including Extension and all Spaces in Filename cont.

Using: A-001 - General Notes Symbols Abbreviations.pdf-M-0-0-A-001 - General Notes Symbols Abbreviations

Match: (.*)(\.pdf.*)

Where:

\1 = A-001 - General Notes Symbols Abbreviations
\2 = pdf-M-0-0-A-001 - General Notes Symbols Abbreviations

Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \.	Match against literal <dot> character	Capture Group 2 = . Changes Capture 1 = A-001 - General Notes Symbols Abbreviations
4. pdf	Match against string 'pdf'	Capture Group 2= .pdf
5. .*	Match against remaining string	Capture Group 2 = pdf-M-0-0-A-001 - General Notes Symbols <space> Abbreviations

Remove

Remove single numeric digit (and preceding underscore) from files that end in single digit

filename_1.jpg to filename.jpg
filename_11.jpg Ignore

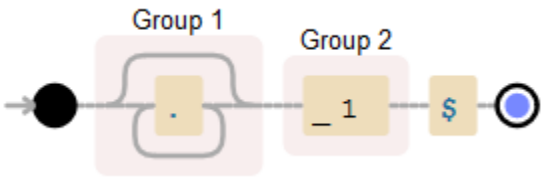
For limiting only to a value of '1'

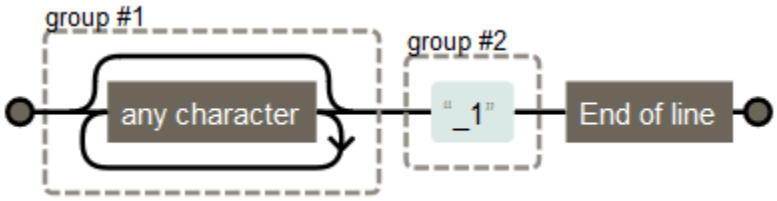
Match: (.*)(_1)$
Replace: \1

Name	New Name
filename_11.jpg	filename_11.jpg
filename_1.jpg	filename.jpg

Sample:

```
Match 1: filename_1
Group 1: filename
Group 2: _1
```





Forum Contributor: KenP

Tim Contribution –

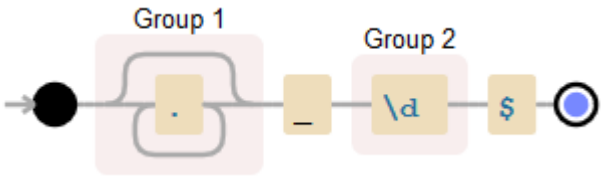
To specify any single numeric digit –

Match: (.*)_(\d)$
Replace: \1

Name	New New Name
filename_a.jpg	filename_a.jpg
filename_8.jpg	filename.jpg
filename_345.jpg	filename_345.jpg
filename_2.jpg	filename.jpg
filename_11.jpg	filename_11.jpg
filename_1.jpg	filename.jpg

Sample:

```
Match 1: filename_1
Group 1: filename
Group 2: 1
```



Tim Notes:

1. To specify multiple numeric digits, change the \d to \d* (make it Greedy).

Remove

Remove single numeric digit (and preceding underscore) from files that end in single digit cont.

Using: filename_1.jpg

Original - Match: (.*)(_1)$

Where:

\1 = filename \2 = _1

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore	Capture Group 2 = _ Changes Capture Group 1 = filename
4. 1	Match against string '1'	Capture Group 2 = _1
5. $	Must be at EOL	

Tim Contribution –

Match: (.*)_(\d)$

Where:

\1 = filename \2 = 1

Tim Analysis:

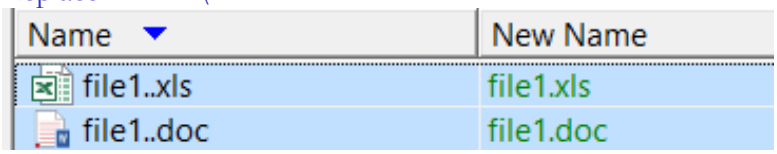
1..	Match against any character	Capture Group 1 = f
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore (Not Captured) = '_'	Changes Capture Group 1 = filename
4. \d	Match against Numeric Digit	Capture Group 2 = 1
5. $	Must be at EOL	

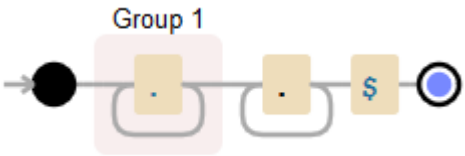
Remove

Remove Trailing Dots

file1..doc to file1.doc
file1..xls to file1.xls

Match: (.+?)\.+$
Replace: \1





Sample:

```
.................
Match 1: file1..
Group 1: file1
```

Forum Contributor: truth

Remove

Remove Trailing Dots cont.

Using: file1..doc

Match: (.+?)\.+$

Where:

\1 = file1

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \.	Match against literal <dot> (Not Captured) This is the <dot> after 'file1'	Changes Capture Group 1 = file1
4. +	make it Greedy (will handle multiple <dot> characters – this sample doesn't have any)	Capture Group 1 = No Change
5. $	Must be at EOL	

Remove

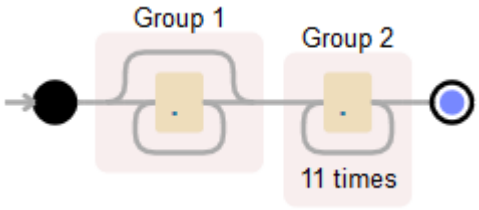
Get last x characters

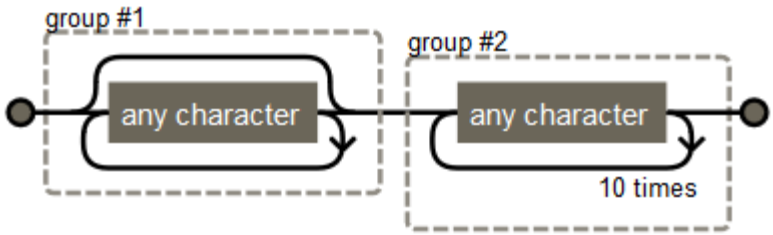
My Folder - Updated - AxYs34ep1ko to AxYs34ep1ko

In this sample, the last x characters refers to the last 11 characters

Match: (.*)(.{11})
Replace: \2

Name	New Name
My Folder - Updated - AxYs34ep1ko	AxYs34ep1ko





Sample:

```
Match 1: My Folder - Updated - AxYs34ep1ko
Group 1: My Folder - Updated -
Group 2: AxYs34ep1ko
```

Contributor Notes:

1. This only works where each file name contains the exact number of characters you need to capture.

Forum Contributor: KenP

Remove

Get last x characters cont.

Using: My Folder - Updated - AxYs34ep1ko

Match: (.*)(.{11})

Where:

\1 = My Folder - Updated - <space> \2 = AxYs34ep1ko

Tim Analysis:

1..	Match against any character	Capture Group 1 = M
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. .	Match against any character	Capture Group 2 = o
	Backtracking.	Changes Capture Group 1 =
	This is the 'o' of 'AxYs34ep1ko'	My Folder - Updated - AxYs34ep1k
	After match of 'o', current position = EOL	
4. {11}	Match last expression for 11 iterations.	Capture Group 2= AxYs34ep1ko
	at EOL so it backtracks.	Changes Capture Group 1 =
		My Folder - Updated - <space>

Remove

Get last x characters cont.

My Folder - Updated - AxYs34ep1ko to AxYs34ep1ko

Tim Contribution –

You can also easily adapt this to keep the front end and remove the back:

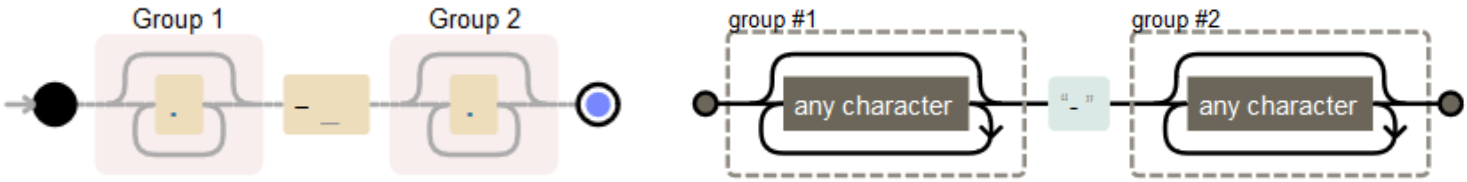
Replace: \1

Name	New Name
My Folder - Updated - AxYs34ep1ko	My Folder - Updated

As well as modify it to capture all the last characters as long as the pattern is a <hyphen> <space> preceding:

Match: (.*)- (.*)
Replace: \2

Name	New Name
My Folder - Updated - AxYs34ep1ko	AxYs34ep1ko



Sample:

```
.............................................
Match 1: My Folder - Updated - AxYs34ep1ko
Group 1: My Folder - Updated
Group 2: AxYs34ep1ko
```

Tim Notes:

1. There has to be a pattern. If the pattern at the end of the string, for example, was <space> underscore, then the RegEx would be: (.*) _(.*)

Remove

Get last x characters cont.

Using: My Folder - Updated - AxYs34ep1ko

Tim Contribution – cont.

Match: (.*)- (.*)

Where:

\1 = My Folder – Updated <space> \2 = AxYs34ep1ko

Tim Analysis:

1..	Match against any character	Capture Group 1 = M
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'Updated <space>'	Changes Capture Group 1 = My Folder – Updated <space>
4. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Updated <space><hyphen>	
5. .*	Match against remaining string	Capture Group 2 = AxYs34ep1ko

Tim Notes:

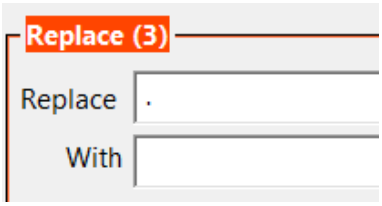
1. If you didn't want the trailing <space> after 'Updated' change to: (.*) - (.*)

Remove

Remove all after year and convert dots to spaces and add parentheses where noted

Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4 to Title - Second Title (2016).mp4

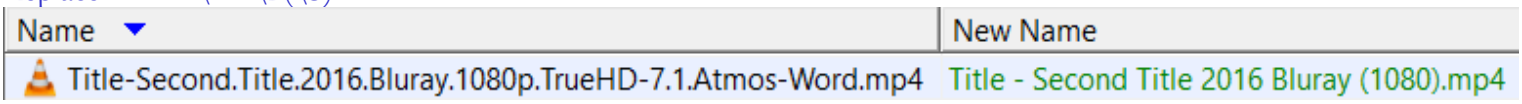
1. 'Section #3: Replace'

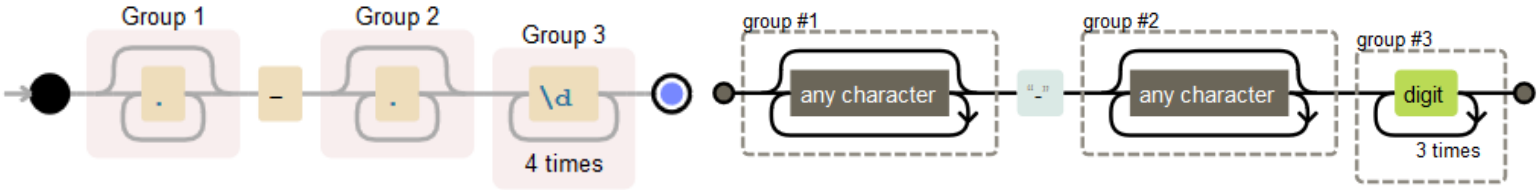


Replace: dot
With: <space>

2. RegEx

Match: (.*)-(.*)(\d{4})
Replace: \1 - \2(\3)





Sample:

```
.....................................................
Match 1: Title-Second.Title.2016.Bluray.1080
Group 1: Title
Group 2: Second.Title.2016.Bluray.
Group 3: 1080
```

Where:

\1 = Title \2 = Second.Title.2016.Bluray. \3 = 1080

Forum Contributor: KenP

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Using: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4

Match: (.*)-(.*)(\d{4})

Where:

\1 = Title \2 = Second.Title.2016.Bluray. \3 = 1080

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen> (Not Captured) This is the <hyphen> after 'Atmos'	Changes Capture Group 1 = Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos
4..	Match against any character	Capture Group 2 = W
5. *	Make it Greedy Current position = EOL	Capture Group 2 = Word
6. \d	Match against Numeric Digit Backtracks. This is the '1' of '7.1'	Capture Group 3 = 1 Changes Capture Group 2 = 7
7. {4}	Repeat last match for 4 iterations total Backtracks. There are not 4 numeric digits after the '1' of '7.1', therefore has to continue to backtrack until it matches against the '1080'.	Capture Group 3 = 1080 Changes Capture Group 2 = Second.Title.2016.Bluray. Changes Capture Group 1 = Title

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4
to
Title - Second Title (2016).mp4

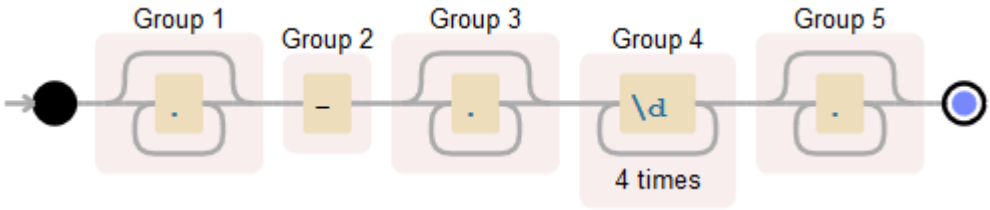
Tim Contribution –

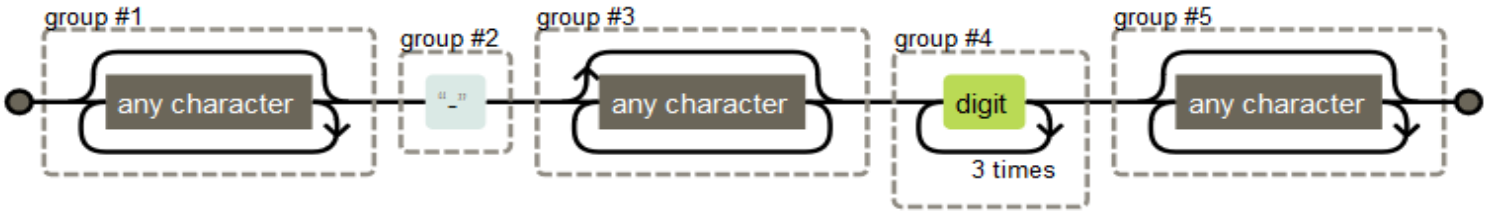
Still requires 'Section #3: Replace' using same settings.

This will work as well (demonstrate additional parsing of string)-

Match: (.*)(-)(.*?)(\d{4})(.*)
Replace: \1 \2 \3(\4)

Name	New Name
Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4	Title - Second Title (2016).mp4





Sample:

```
.....................................................................
Match 1: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
Group 1: Title
Group 2: -
Group 3: Second.Title.
Group 4: 2016
Group 5: .Bluray.1080p.TrueHD-7.1.Atmos-Word
```

Where:

\1 = Title \2 = - \3 = Second Title. \4 = 2016 \5 = .Bluray.1080p.TrueHD-7.1.Atmos-Word

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Using: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4

Tim Contribution – cont.

Match: (.*)(-)(.*?)(\d{4})(.*)

Where:

\1 = Title \2 = - \3 = Second Title. \4 = 2016 \5 = .Bluray.1080p.TrueHD-7.1.Atmos-Word

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen> This is the <hyphen> after 'Atmos'	Capture Group 2 = - Changes Capture Group 1 = Title-Second.Title.2016.Bluray.1080p.
4..	Match against any character	Capture Group 3 = W
5. *	Make it Greedy Current position = EOL	Capture Group 3 = Word
6. ?	But not too Greedy Current position = <hyphen> after 'Atmos'	Capture Group 3 = null Changes Capture Group 1 = Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos
7. \d	Match against Numeric Digit Backtracks. This is the '7' of '7.1' <hyphen> in Capture Group 2 changes to the <hyphen> before '7.1'	Capture Group 4 = 7 Changes Capture Group 2 = - (Not Same)

Tim Notes:

1. As it backtracks it tests each character (.*) looking for a dash (-) followed by a numeric digit (\d). It matches at the '-7' of '-7.1'

2. The Numeric digit does not have to be immediately following [because of the (.*?)] - it can be <hyphen> text <numeric digit>. It will match against the first numeric digit found after the <hyphen> so in this case it was the '-7'.

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Using: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4

Tim Contribution – cont.

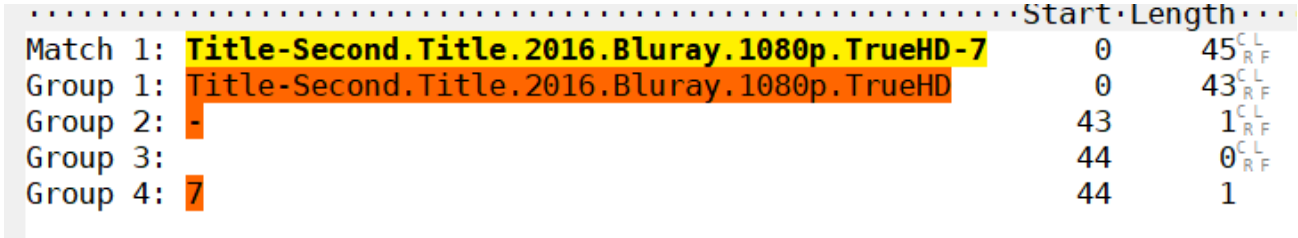
Match: (.*)(-)(.*?)(\d{4})(.*)

Where:

\1 = Title \2 = - \3 = Second Title. \4 = 2016 \5 = rest of string = .Bluray.1080p.TrueHD-7.1.Atmos-Word

Tim Analysis: cont.

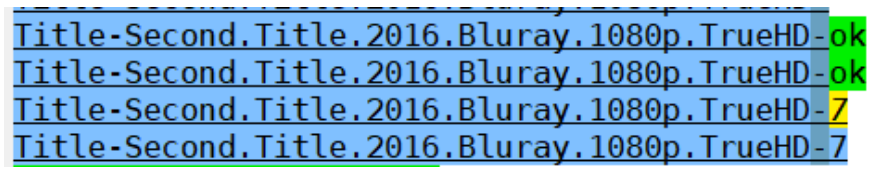
This is what it looks like up to now:



Once the <hyphen> is matched it moves forward looking for a numeric digit.

Finding none, it backtracks looking for a preceding <hyphen>

Finds one and moves forward to match against the '7'



Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Using: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4

Tim Contribution – cont.

Match: (.*)(-)(.*?)(\d{4})(.*)

Where:

\1 = Title \2 = - \3 = Second Title. \4 = 2016 \5 = rest of string = .Bluray.1080p.TrueHD-7.1.Atmos-Word

Tim Analysis: cont.

8. {4} Repeat last match for 4 iterations total Backtracks.

Capture Group 4 = 2016
Changes Capture Group 3 = Second Title.
Changes Capture Group 1 = Title

Tim Notes:

1. Same explanation as before. Although the current position is the '7' of '7.1' in order to satisfy the match it has to be in the pattern of a<hyphen> followed by a <numeric digit> but does not have to be immediately following.

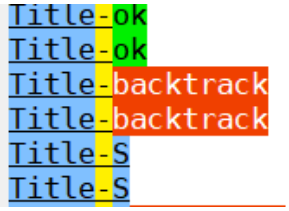
2. There is not a 4 numeric digit sequence after the '7' so it backtracks to the next previous <hyphen>. This will be the <hyphen> following 'Title'.

3. It moves forward to match against the '2016' rather than the '1080' because the '2016' is encountered by the RegEx Engine first after the <hyphen> is matched.

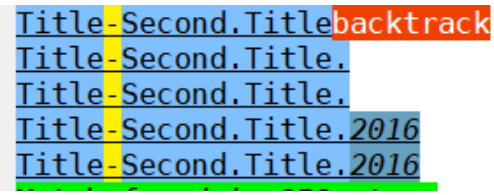
It now looks like this:

```
.....................................
Match 1: Title-Second.Title.2016
Group 1: Title
Group 2: -
Group 3: Second.Title.
Group 4: 2016
```

Once the <hyphen> is matched, it moves forward looking for 4 numeric digits:



Matches against the '2016'



9. .* Match against remaining string

Capture Group 5 = .Bluray.1080p.TrueHD-7.1.Atmos-Word

Remove

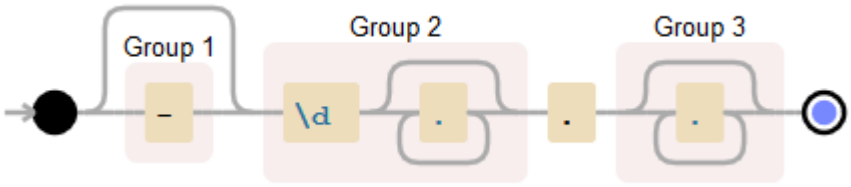
Remove all after year and convert dots to spaces and add parentheses where noted cont.

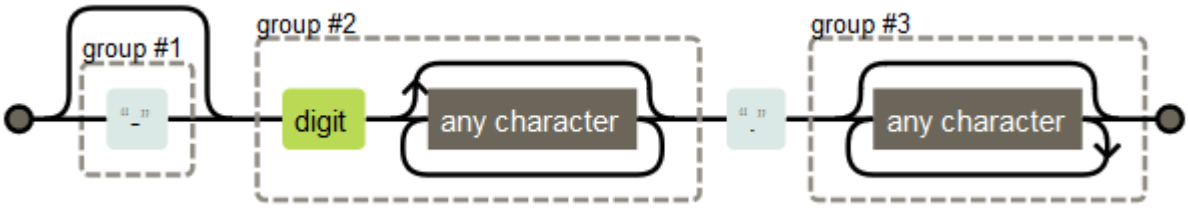
Tim Contribution – cont.

To capture from the year to the end of the string and remove the first part -

Match: (-)?(\d.*?)\.(.*)
Replace: \2 - \3

Name	New Name
Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4	2016 - Bluray 1080p TrueHD-7 1 Atmos-Word.mp4





Sample:

```
                                                  Start  Length
Match 1: 2016.Bluray.1080p.TrueHD-7.1.Atmos-Word     19      39
Group 1 did not participate in the match
Group 2: 2016                                        19       4
Group 3: Bluray.1080p.TrueHD-7.1.Atmos-Word          24      34
```

Where:

\1 = no value — Didn't capture <hyphen> because (-)? Made expression optional
\2 = 2016
\3 = Bluray.1080p.TrueHD-7.1.Atmos-Word

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Tim Contribution – cont.

To capture from the year to the end of the string and remove the first part – cont.

Using: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4

Match: (-)?(\d.*?)\.(.*)

Where:

\1 = no value \2 = 2016 \3 = Bluray.1080p.TrueHD-7.1.Atmos-Word

Tim Analysis:

1. -	Match against <hyphen> followed by … This is the <hyphen>after 'Title'	Capture Group 1 = -
2. ?	Entire expression made optional. Capture Group 1 never evaluated.	Changes Capture Group 1 = no value
3. \d	Match against Numeric Digit This is the '2' of '2016'	Capture Group 2 = 2
4..	Match against any character	Capture Group 2 = 0
5. *	Make it Greedy	Capture Group 2 = 2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
6. ?	But not too Greedy	Capture Group 2 = 2
7. \.	Match against literal <dot> (Not Captured) This is the <dot>following '2016'	Changes Capture Group 2 =2016
8. .*	Match against remaining string	Capture Group 3 = Bluray.1080p.TrueHD-7.1.Atmos-Word

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Tim Contribution – cont.

To capture from the year to the end of the string and remove the first part – cont.

Tim Notes:

1. Step 2 has to do with placing the '?' Quantifier outside of the Capture Group rather than inside with the expression. Inside the expression (-?) would make the match of the <hyphen> 'optional'. It might match but 'skip' over capturing the value of the <hyphen> resulting in Zero Occurrence Match. Capture Group 1's value would be null and display as:

```
                                                  Start  Length
Match 1: 2016.Bluray.1080p.TrueHD-7.1.Atmos-Word     19      39
Group 1:                                             19       0
Group 2: 2016                                        19       4
Group 3: Bluray.1080p.TrueHD-7.1.Atmos-Word          24      34
```

Instead, by placing the Quantifier outside of the Capture Group, it made the entire Capture Group evaluation optional resulting in no evaluation taking place and displays as:

```
                                                  Start  Length
Match 1: 2016.Bluray.1080p.TrueHD-7.1.Atmos-Word     19      39
Group 1 did not participate in the match
Group 2: 2016                                        19       4
Group 3: Bluray.1080p.TrueHD-7.1.Atmos-Word          24      34
```

Capture Group 1 has no value or undefined value which is not the same as a Zero Occurrence Match. In a Zero Occurrence Match a match has taken place but the Capture Group holds no value or a 'null' value as evidenced by the length of Group 1 with zero in the first photo, whereas in the second photo it displays Group 1 as not having participated in the match at all. No evaluation – no value or undefined value. This is not the same as an Invalid Capture Group either. An Invalid Capture may be because of a syntax error although the match may still be viable.

```
 1  ok
 2  ok
 3  2
 4  2ok
 5  2
 6  2backtrack
 7  20
 8  20
 9  20backtrack
10  201
11  201
12  201backtrack
13  2016
14  2016
15  2016.
16  2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
17  2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
    Match found in 17 steps
```

```
 1  backtrack
 2  ok
 3  2
 4  2ok
 5  2
 6  2backtrack
 7  20
 8  20
 9  20backtrack
10  201
11  201
12  201backtrack
13  2016
14  2016
15  2016.
16  2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
17  2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
    Match found in 17 steps
```

Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Tim Contribution – cont.

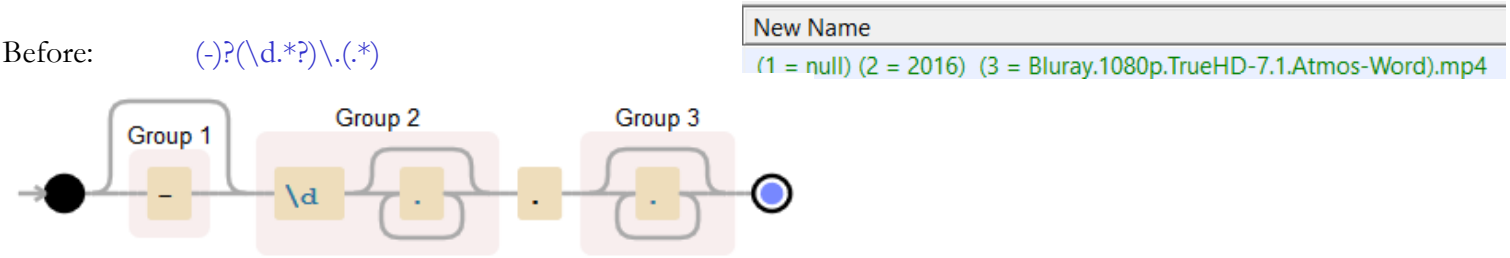
Tim Notes:

1. By placing the Lazy Quantifier outside of Group 1, it made the entire expression optional and so it never evaluated. This pushed the RegEx Engine forward to the Numeric Digits, '1080'
2. Placing the Lazy Quantifier inside Group 1 has the same result, but a Zero Occurrence Match will result in a null value for Capture Group 1 instead of a no value or undefined value.

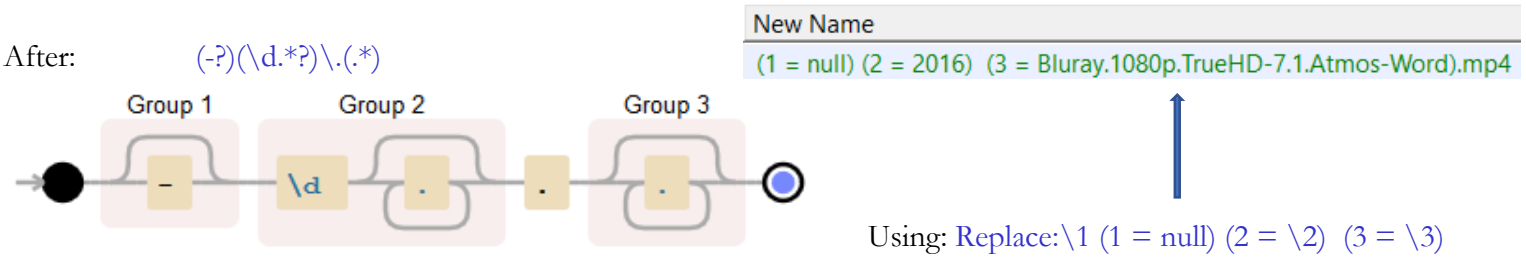
Sample:

```
                                                  Start  Length
Match 1: 2016.Bluray.1080p.TrueHD-7.1.Atmos-Word     19      39
Group 1:                                             19       0
Group 2: 2016                                        19       4
Group 3: Bluray.1080p.TrueHD-7.1.Atmos-Word          24      34
```

Before: (-)?(\d.*?)\.(.*)

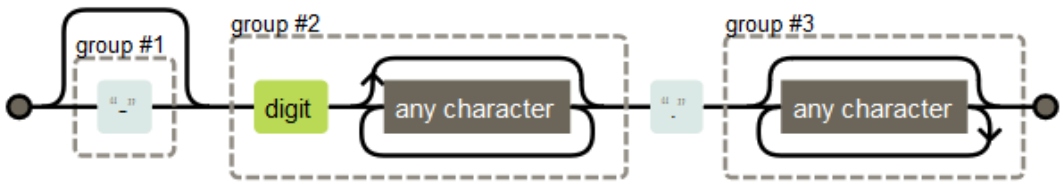


After: (-?)(\d.*?)\.(.*)



Using: Replace:\1 (1 = null) (2 = \2) (3 = \3)

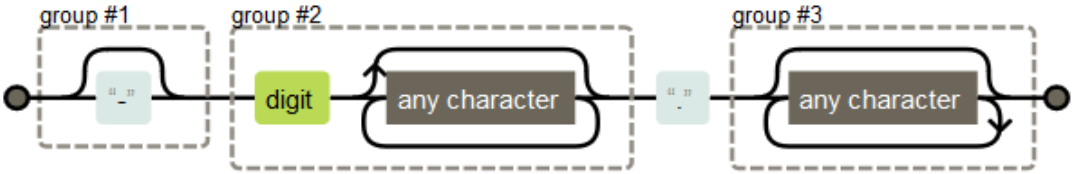
Before: (-)?(\d.*?)\.(.*)



Tim Notes:

1. Same two RegEx as above, just displaying the different Visualizations.

After: (-?)(\d.*?)\.(.*)



Remove

Remove all after year and convert dots to spaces and add parentheses where noted cont.

Tim Contribution – cont.

Using: Title-Second.Title.2016.Bluray.1080p.TrueHD-7.1.Atmos-Word.mp4

After: (-?)(\d.*?)\.(.*) cont.

Where:

\1 – null \2 = 2016 \3 = Bluray.1080p.TrueHD-7.1.Atmos-Word

Tim Notes:

1. This is an analysis of the 'After' example on the previous page to illustrate the '?' Quantifier placed inside the Capture Group with the expression, as opposed to it originally placed outside the Capture Group. Again, this is to illustrate the difference between a Zero Occurrence Match and an unevaluated Capture. The result doesn't change, just in how we get there.

Tim Analysis:

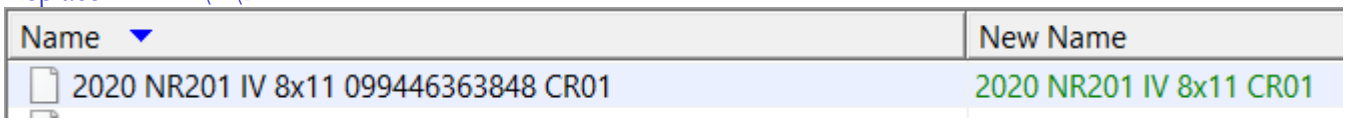
1. -	Match against <hyphen> followed by … This is the <hyphen>after 'Title'	Capture Group 1 = -
2. ?	The expression is made optional. This results in a Zero Occurrence Match of a null value	Changes Capture Group 1 = null
3. \d	Match against Numeric Digit This is the '2' of '2016'	Capture Group 2 = 2
4..	Match against any character	Capture Group 2 = 0
5. *	Make it Greedy	Capture Group 2 = 2016.Bluray.1080p.TrueHD-7.1.Atmos-Word
6. ?	But not too Greedy	Capture Group 2 = 2
7. \.	Match against literal <dot> (Not Captured) This is the <dot>following '2016'	Changes Capture Group 2 =2016
8. .*	Match against remaining string	Capture Group 3 = Bluray.1080p.TrueHD-7.1.Atmos-Word

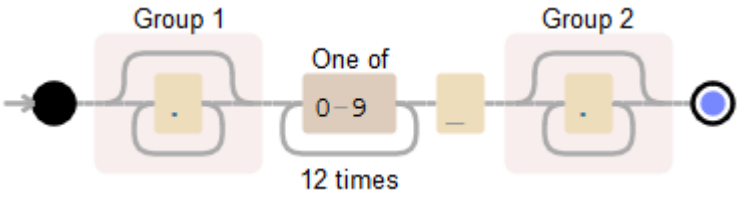
Remove

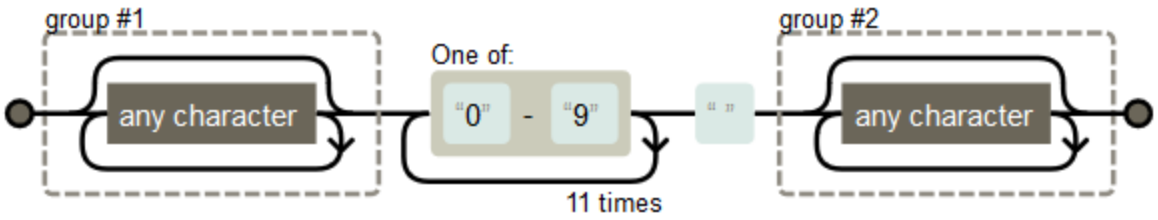
How to find 12 digit number and just remove it

2020 NR201 IV 8x11 099446363848 CR01 to 2020 NR201 IV 8x11 CR01

Match: (.*)[0-9]{12} (.*)
Replace: \1\2







Sample:

```
.............................................
Match 1: 2020 NR201 IV 8x11 099446363848 CR01
Group 1: 2020 NR201 IV 8x11
Group 2: CR01
```

Tim Notes:

1. This is similar to 'Remove Numeric Digits for any files that have Three or more'.
2. By specifying a class of [0-9] with a repetition of {12}, the RegEx matches against all 12 numeric digits

Forum Contributor: therube

Remove

How to find 12 digit number and just remove it cont.

Using: 2020 NR201 IV 8x11 099446363848 CR01

Match: (.*)[0-9]{12} (.*)

Where:

\1 = 2020 NR201 IV 8x11<space> \2 = CR01

Tim Analysis:

1..	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [0-9]	Match against a class consisting of a numeric digit 0-9 (Not Captured) = 1 Backtracking. This is the '1' of 'CR01'	Changes Capture Group 1 = 2020 NR201 IV 8x11 099446363848 CR0
4. {12}	Repeat last expression for a total of 12 iterations (Not Captured). Changes to – '099446363848'	Changes Capture Group 1 = 2020 NR201 IV 8x11<space>
5. <space>	Match against a <space> character (Not Captured) This is the <space> preceding 'CR01'	
6. .*	Match against remaining string	Capture Group 2 = CR01

Remove

How to find 12 digit number and just remove it cont.

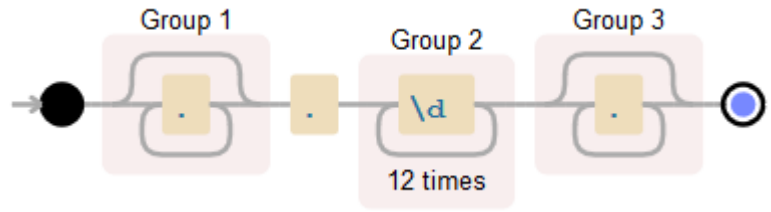
2020 NR201 IV 8x11 099446363848 CR01 to 2020 NR201 IV 8x11 CR01

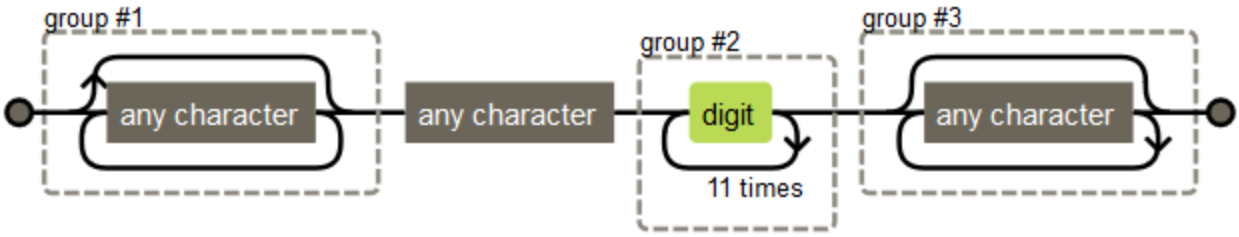
Tim Contribution –

It can be re-written as:

Match: (.*?).(\d{12})(.*)
Replace: \1\3

Name	New Name
2020 NR201 IV 8x11 099446363848 CR01	2020 NR201 IV 8x11 CR01





Sample:

```
Match 1: 2020 NR201 IV 8x11 099446363848 CR01
Group 1: 2020 NR201 IV 8x11
Group 2: 099446363848
Group 3:  CR01
```

Where:
\1 = 2020 NR201 IV 8x11 \2 = 099446363848 \3 = <space>CR01

Tim Notes:

1. \d{12} simply indicates to match against 12 numeric digits

Remove

How to find 12 digit number and just remove it cont.

2020 NR201 IV 8x11 099446363848 CR01 to 2020 NR201 IV 8x11 CR01

Tim Contribution – cont.

Match: (.*?).(\d{12})(.*)

Where:
\1 = 2020 NR201 IV 8x11 \2 = 099446363848 \3 = <space>CR01

Tim Analysis:

1..	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. .	Match against any character (Not Captured) = '1' This is the '1' of 'CR01' Current position = EOL	Changes Capture Group 1 = 2020 NR201 IV 8x11 099446363848 CR0
4. \d	Match against Numeric Digit Changes Not Captured = '0'. This is the '0' of 'CR01' Backtracking. Captures the '1' of 'CR01'	Capture Group 2 = 1 Changes Capture Group 1 = 2020 NR201 IV 8x11 099446363848 CR
5. {12}	Repeat last expression for a total of 12 iterations (Not Captured). Changes to – <space>. This is the <space> after '8x11'	Changes Capture Group 2 = 099446363848 Changes Capture Group 1 = 2020 NR201 IV 8x11
6. .*	Match against remaining string	Capture Group 3 = <space>CR01

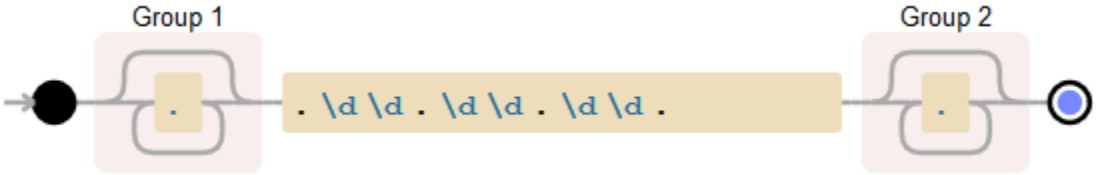
Remove

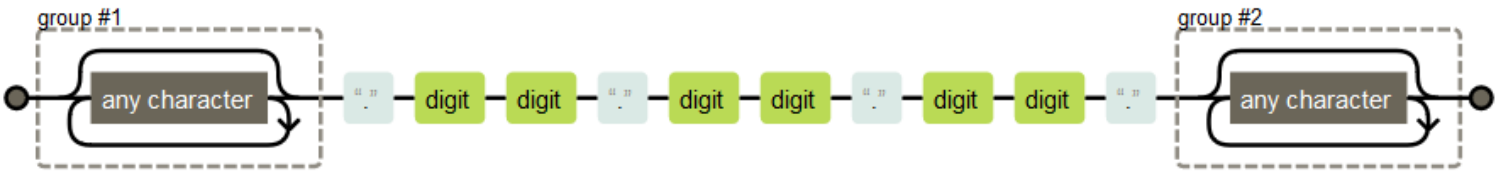
Remove the timestamp and change dot to hyphen

vikings.15.02.16.episode.2 to vikings - episode.2

Match: (.*)\.\d\d\.\d\d\.\d\d\.(.*)
Replace: \1 - \2

Name	New Name
vikings.15.02.16.episode.2	vikings - episode.2





Sample:

```
.....................................
Match 1: vikings.15.02.16.episode
Group 1: vikings
Group 2: episode
```

Forum Contributor: the rube

Remove

Remove the timestamp and change dot to hyphen cont.

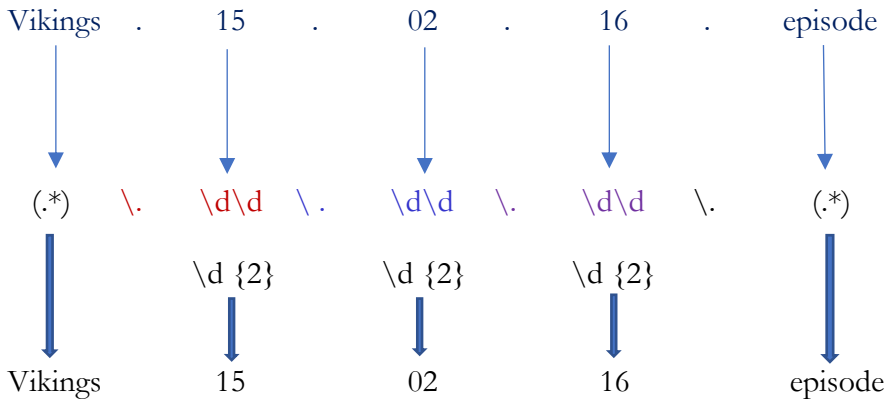
Using: vikings.15.02.16.episode.2

Match: (.*)\.\d\d\.\d\d\.\d\d\.(.*)

Where:

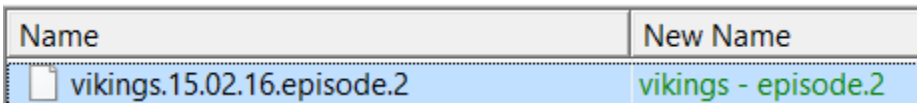
\1 = Vikings \2 = episode

Whenever you see an array of \d's or other Metacharacters in a RegEx, you know it can be simplified much of the time. The analysis, however, doesn't change.



Tim Notes:

1. The '2' in the string is treated as the file extension – '.2' It doesn't matter if 'Rename File Extensions As Being part of File Name' is enabled from the ID3 / Exif Data / File Properties sub-menu of the Renaming Options Menu because the outcome will be the same.



2. In addition, to assist with the analysis, I added additional Capture Groups. The final outcome is unaffected.

Tested RegEx will be:

Match: (.*)\.(\d{2})\.(\d{2})\.(\d{2})\.(.*)

Remove

Remove the timestamp and change dot to hyphen cont.

Using: vikings.15.02.16.episode.2

Match: (.*)\.(\d{2})\.(\d{2})\.(\d{2})\.(.*)

Where:

\1 = vikings \2 = 15 \3 = 02 \4 = 16 \5 = episode

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \.	Match against literal <dot> (Not Captured) Backtracks. This is the <dot> after '16'	Changes Capture Group 1 = vikings.15.02.16
4. \d{2}	Match against 2 numeric digits Moves forward to find two numeric digits. Not found Backtracks to match against '16'.	Capture Group 2 = 16 Changes Capture Group 1 = Vikings.15.02
5. \.	Match against literal <dot> (Not Captured) Moves forward. This is the <dot> after '16'	
6. \d{2}	Match against 2 numeric digits Backtracks. Matches against '16'	Capture Group 3 = 16 Changes Capture Group 2 = 02 Changes Capture Group 1 = vikings.15
7. \.	Match against literal <dot> (Not Captured) Moves forward. This is the <dot> after '16'	
8. \d{2}	Match against 2 numeric digits Backtracks. Matches against '16'	Capture Group 4 = 16 Changes Capture Group 3 = 02 Changes Capture Group 2 = 15 Changes Capture Group 1 = vikings
9. \.	Match against literal <dot> (Not Captured) Moves forward. This is the <dot> after '16'	
10. .*	Match against remaining string	Capture Group 5 = episode

Remove

Remove the timestamp and change dot to hyphen cont.

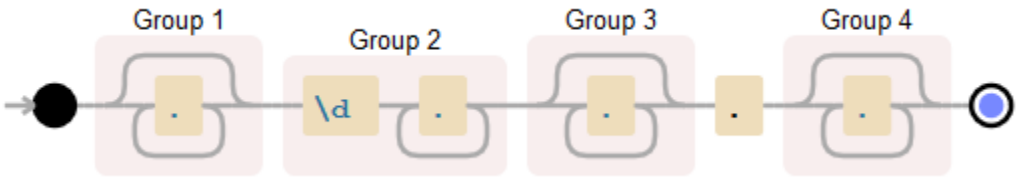
Tim Contribution –

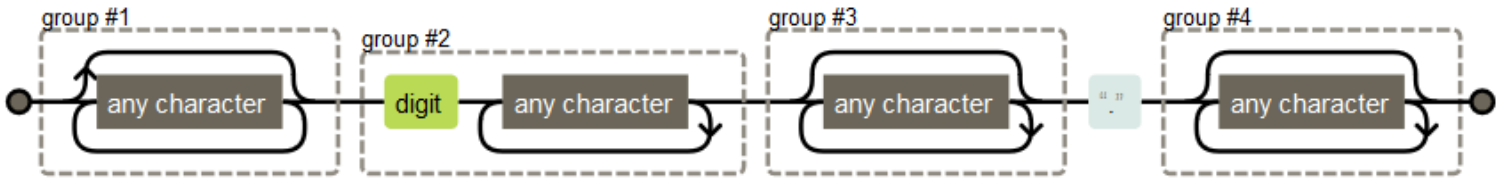
This will also work but has the advantage of working with any number of numeric digits in the title:

For example, 'test.15.02.16.56.78.episode.2'

Match: (.*?)\.(\d.+)(.*)\.(.*)
Replace: \1.\4

Name	New Name
vikings.15.02.16.episode.2	vikings.episode.2
test.15.02.16.56.78.episode.2	test.episode.2





Sample:

	Start	Length
Match 1:	vikings.15.02.16.episode	0	24
Group 1:	vikings	0	7
Group 2:	15.02.16	8	8
Group 3:		16	0
Group 4:	episode	17	7

Sample:

	Start	Length
Match 1:	test.15.02.16.56.78.episode	0	27
Group 1:	test	0	4
Group 2:	15.02.16.56.78	5	14
Group 3:		19	0
Group 4:	episode	20	7

Remove

Remove the timestamp and change dot to hyphen cont.

Tim Contribution – cont.

This will also work but has the advantage of working with any number of numeric digits in the title:

Using: test.15.02.16.56.78.episode.2

Match: (.*?)\.(\d.+)(.*)\.(.*)

Where:

\1 = test \2 = 15.02.16.56.78 \3 = null \4 = episode

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \.	Match against literal <dot> (Not Captured) This is the <dot> after 'test'	Changes Capture Group 1 = test
5. \d	Match against a numeric digit Matches against '1' of '15'	Capture Group 2 = 1
6..	Match against any character	Capture Group 2 =15
7. +	Make it Greedy Current position = EOL	Capture Group 2 = 15.02.16.56.78.episode
8..	Match against any character Backtracks	Capture Group 3 = e Changes Capture Group 2= 15.02.16.56.78.episod
9. *	Make it Greedy Capture Group 3 gave up all of its value = null	Capture Group 3 = null Changes Capture Group 2= 15.02.16.56.78.episode
10. \.	Match against literal <dot> (Not Captured) Backtracks. This is the <dot> after '78'	Changes Capture Group 2 = 15.02.16.56.78
11. .*	Match against remaining string	Capture Group 4 = episode

Remove

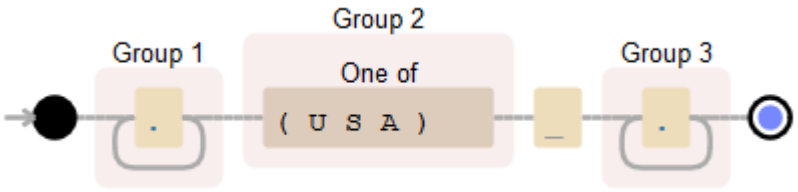
Remove Everything after (USA)

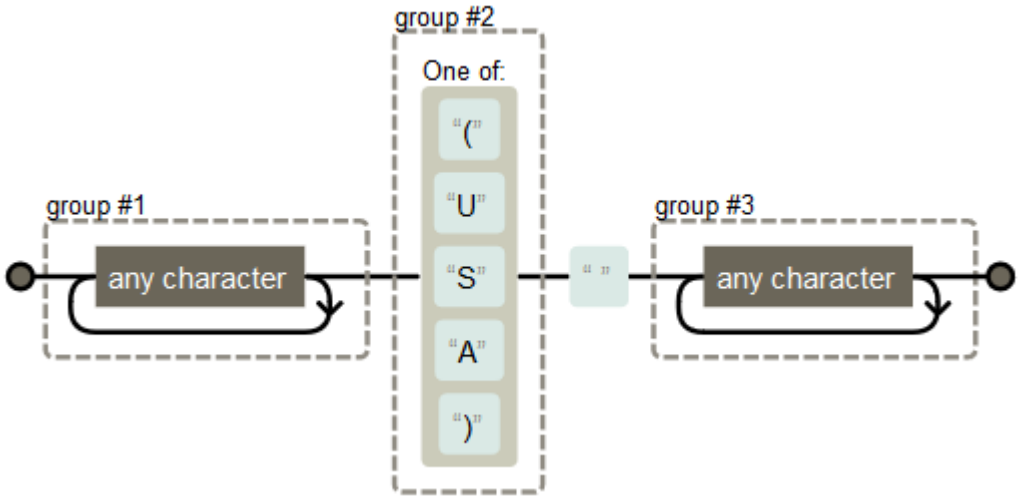
Filename1 (USA) (en, Fr, Jp).zip	to	Filename1 (USA).zip
Filename2 (USA) (en).zip	to	Filename2 (USA).zip
Filename3 (USA) (en, Fr, Jp, Aus, Ger).zip	to	Filename3 (USA).zip
Filename4 (USA) (en, Fr).zip	to	Filename4 (USA).zip

If the files were always 'USA', use this:

Match: (.+)([(USA)]) (.+)
Replace: \1\2

Name	New Name
Filename1 (USA) (en, Fr, Jp).zip	Filename1 (USA).zip
Filename2 (USA) (en).zip	Filename2 (USA).zip
Filename3 (USA) (en, Fr, Jp, Aus, Ger).zip	Filename3 (USA).zip
Filename4 (USA) (en, Fr).zip	Filename4 (USA).zip





Sample:

```
.....................................
Match 1: Filename1 (USA) (en, Fr, Jp)
Group 1: Filename1 (USA
Group 2: )
Group 3: (en, Fr, Jp)
```

Remove

Remove Everything after (USA) cont.

Using: Filename1 (USA) (en, Fr, Jp).zip

If the files were always 'USA', use this: cont.

Match: (.+)([(USA)]) (.+)

Where:

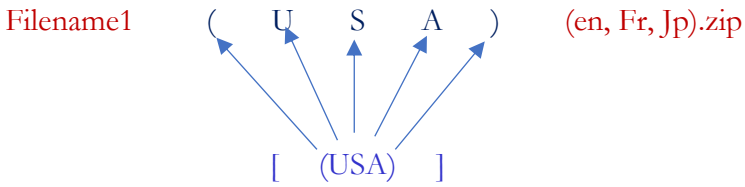
/1 = Filename1 (USA \2 =) \3 = (en, Fr, Jp)

Tim Analysis:

1..	Match against any character	Capture Group 1 = F
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. [(USA)]	Match against a class consisting of string characters, '(' 'U' 'S' 'A' ')'	Capture Group 2 =) Changes Capture Group 1 = Filename1 (USA

Tim Notes:

1. Unlike a regular match of a string, '(USA)', this only matches individual characters of whichever comes first. Because the string '(USA)' can be found together in the sample, the matches take place one character after the other:



The last match matches the right parenthesis and that is the final value captured. Remember that a class only matches a single value. Capture Group 2 = ')'. Capture Group 1 is changed to 'Filename1 (USA'.

4. <space>	Match against a <space> character (Not Captured) This is the <space> after '(USA)'	
5..	Match against any character	Capture Group 3 = (
6. +	Make it Greedy	Capture Group 3 = (en, Fr, Jp)

Remove

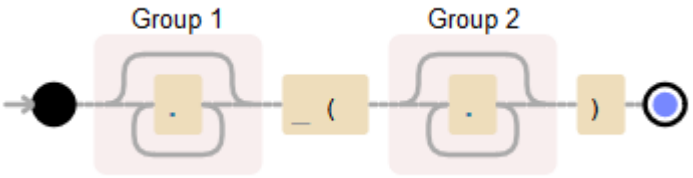
Remove Everything after (USA) cont.

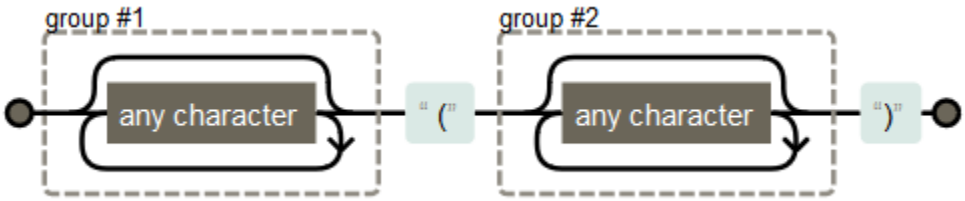
Filename1 (USA) (en, Fr, Jp).zip	to	Filename1 (USA).zip
Filename2 (USA) (en).zip	to	Filename2 (USA).zip
Filename3 (USA) (en, Fr, Jp, Aus, Ger).zip	to	Filename3 (USA).zip
Filename4 (USA) (en, Fr).zip	to	Filename4 (USA).zip

If not, use this:

Match: (.*) \((.*)\)
Replace: \1

Name	New Name
Filename1 (USA) (en, Fr, Jp).zip	Filename1 (USA).zip
Filename2 (Hello) (en).zip	Filename2 (Hello).zip
Filename3 (USA) (en, Fr, Jp, Aus, Ger).zip	Filename3 (USA).zip
Filename4 (This is a Test) (en, Fr).zip	Filename4 (This is a Test).zip





Sample: Using - Filename4 (This is a Test) (en, Fr, Jp).zip

```
.....................................................
Match 1: Filename4 (This is a Test) (en, Fr, Jp)
Group 1: Filename4 (This is a Test)
Group 2: en, Fr, Jp
```

Tim Notes:

1. A more descriptive title would be –
Remove Everything after (<space> OR U<space> OR S<space> OR)<space>

Forum Contributors: RobsterUK, hns

Remove

Remove Everything after (USA) cont.

Using: Filename4 (This is a Test) (en, Fr, Jp).zip

If not, use this: cont.

Match: (.*) \((.*)\)

Where:

\1 = Filename4 (This is a Test) \2 = en, Fr, Jp

Tim Analysis:

1..	Match against any character	Capture Group 1 = F
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Fr,'	Changes Capture Group 1 = Filename4 (This is a Test) <space> (en, Fr,
4. \(	Match against literal left parenthesis (Not Captured) This is the left parenthesis of '(en, Fr, Jp)'	Changes Capture Group 1= Filename4 (This is a test)
5..	Match against any character	Capture Group 2 = e
6. *	Make it Greedy	Capture Group 2 = en, Fr, Jp)
7. \(	Match against literal right parenthesis (Not Captured) This is the right parenthesis of '(en, Fr, Jp)'	Changes Capture Group 2= en, Fr, Jp

Remove

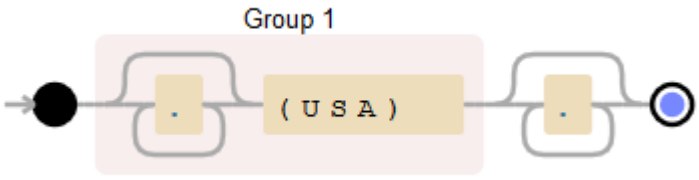
Remove Everything after (USA) cont.

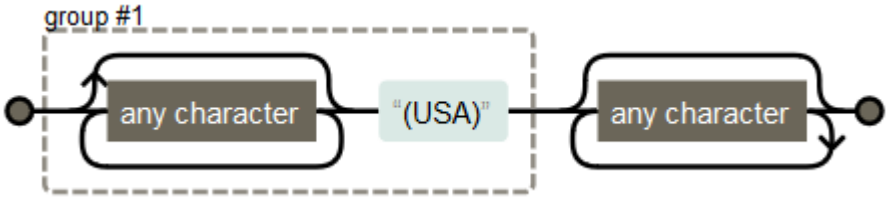
Filename1 (USA) (en, Fr, Jp).zip	to	Filename1 (USA).zip
Filename2 (USA) (en).zip	to	Filename2 (USA).zip
Filename3 (USA) (en, Fr, Jp, Aus, Ger).zip	to	Filename3 (USA).zip
Filename4 (USA) (en, Fr).zip	to	Filename4 (USA).zip

Alternate: This one is more closely related to the title 'Remove Everything after (USA)'

Match: (.*?\(USA\)).*
Replace: \1

Name	New Name
Filename1 (USA) (en, Fr, Jp).zip	Filename1 (USA).zip
Filename2 (USA) (en).zip	Filename2 (USA).zip
Filename3 (USA) (en, Fr, Jp, Aus, Ger).zip	Filename3 (USA).zip
Filename4 (USA) (en, Fr).zip	Filename4 (USA).zip





Sample:

```
Match 1: Filename1 (USA) (en, Fr, Jp).zip
Group 1: Filename1 (USA)
```

Forum Contributor: RegexNinja

Remove

Remove Everything after (USA) cont.

Using: Filename1 (USA) (en, Fr, Jp).zip

Alternate: This one is more closely related to the title 'Remove Everything after (USA)' cont.

Match: (.*?\(USA\)).*

Where:

\1 = Filename1 (USA)

Tim Analysis:

1..	Match against any character	Capture Group 1 = F
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \(	Match against literal left parenthesis This is the left parenthesis of '(USA)'	Changes Capture Group 1= Filename1 (
5. USA	Match against the string 'USA'	Capture Group 1 = Filename1 (USA
6. \(	Match against literal right parenthesis This is the right parenthesis of '(USA)'	Capture Group 1 = Filename1 (USA)
7..	Match against any character = <space> (Not Captured)	
8. *	Make it Greedy = '<space> (en, Fr, Jp)'	

Tim Notes:

1. You may have noticed a pattern with RegEx behavior. When you want to set up Capture Group 1, use (.*) to match against the entire string. The position is at EOL so whatever it does next is going to backtrack and place a value in Capture Group 1 based on where the next evaluation places the current position in the sample string.

2. Another behavior, is once at EOL initially, you can start back at the beginning of the string to match from there using the (.*?) to make the last expression optional. When evaluated, it will match to the EOL then backtrack to the BOL.

   Capture Group 1 is still matched (the initial <dot>) but holds no value – not until the next expression is evaluated. BOL is not a value but a string position. Position zero to be precise. Once the next expression is evaluated, then the (.*) portion of the sub-expression, (.*?) is reevaluated and the value captured, again, based on where the next evaluation places the current position in the sample string.

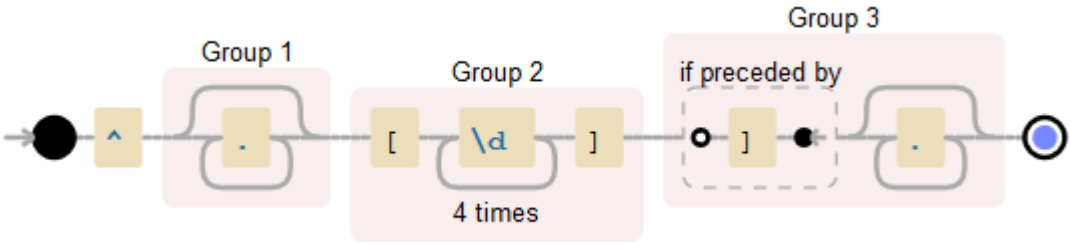
Remove

Remove all before last Bracket

Blood Ceremony [2008] Blood Ceremony	to	[2008] Blood Ceremony
Blood Ceremony [2011] Living With The Ancients	to	[2011] Living with the Ancients
Blood Ceremony [2013] The Eldritch Dark	to	[2013] The Eldritch Dark

Match: ^(.*)(\[\d{4}\])((?<=\]).*)
Replace: \2\3

Name	New Name
Blood Ceremony [2013] The Eldritch Dark	[2013] The Eldritch Dark
Blood Ceremony [2011] Living With The Ancients	[2011] Living With The Ancients
Blood Ceremony [2008] Blood Ceremony	[2008] Blood Ceremony



Sample:

```
........................................................
Match 1: Blood Ceremony [2008] Blood Ceremony
Group 1: Blood Ceremony
Group 2: [2008]
Group 3:  Blood Ceremony
```

Forum Contributor: DavidLago

Remove

Remove all before last Bracket cont.

Blood Ceremony [2008] Blood Ceremony	to	[2008] Blood Ceremony
Blood Ceremony [2011] Living With The Ancients	to	[2011] Living with the Ancients
Blood Ceremony [2013] The Eldritch Dark	to	[2013] The Eldritch Dark

Using: Blood Ceremony [2008] Blood Ceremony

Match: ^(.*)(\[\d{4}\])((?<=\]).*)

Where:

/1 = Blood Ceremony<space> \2 = [2008] \3 = <space> Blood Ceremony

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .	Match against next character	Capture Group 1 = B
3. *	Make it Greedy until…	Capture Group 1 = entire string
4. \[	.. Left Bracket followed by …	Capture Group 2 = [Changes Capture Group 1 = Blood Ceremony<space>
5.\d{4}	.. 4 numeric digits, followed by …	Capture Group 2 = [2008
6. \]	.. Right bracket	Capture Group 2 = [2008]
7. (?<=\])	Lookbehind and if last character is a single right bracket literal then expression = True and evaluation continues.= '] ' so TRUE	All Capture Groups = No Change
8. .	Match against next character This is the <space> following the right bracket	Capture Group 3 = <space>
9. *	Make it Greedy	Capture Group 3 = <space> Blood Ceremony

If there was an extension, the <space> at the end of Blood Ceremony would need to be removed either through enabling the 'Trim' option in 'Section #5: Remove' or within the RegEx itself.

Remove

Remove all before last Bracket cont.

Tim Analysis: cont.

Tim Notes:

1. Change the (.*) for Group 1 to (.*?) to remove first bracket.

2. (?<=) This is a Positive Lookbehind. While moving through the string, this indicates to the RegEx to look left) and see if there is a certain character or expression – and if there is, the expression = True and the evaluation continues. The Lookbehind does not change the current position.

   If evaluated TRUE, as in this example, the current position would be just before the <space> immediately following the right bracket.

3. How did I determine that the Lookbehind was True or not? I captured the value – but it doesn't display a value, so I added the (..) within the capture to see what followed. This is also a good way to determine if the RegEx engine is backtracking or not.

   ((?<=\])….) = <space> Blo

   Here the first character after the Lookbehind is the <space>. From this we can tell that the current position is just before the <space> immediately following the right bracket as was explained in the first notation. I can also tell from this that the RegEx engine is moving forward because the next few characters are B, l, o. If the RegEx engine was backtracking the next few characters would have been], 8, 0.

4. For further information, refer to 'Lookarounds' in the RegEx Manual found in the Appendix of Volume I and also in the Appendix found in this book.

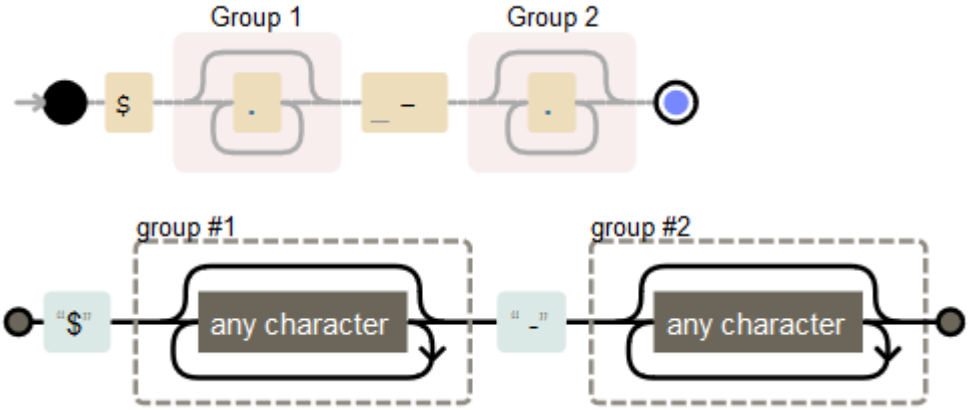
Remove

Remove numeric dollar amount from string

$1125.80 - Business Internet-Static IPs	to	$ - Business Internet-Static IPs
$713.65 - Shop Tools-Supplies	to	$ - Shop Tools-Supplies

Match: \$(.*) -(.*)
Replace: $\2

Name	New Name
$713.65 - Shop Tools-Supplies	$ Shop Tools-Supplies
$1125.80 - Business Internet-Static IPs	$ Business Internet-Static IPs



Sample:

```
.....................................................
Match 1: $1125.80 - Business Internet-Static IPs
Group 1: 1125.80
Group 2:  Business Internet-Static IPs
```

Tim Notes:

1. Rather than just try to capture the numeric digits, a simpler approach of parsing the string was used.
2. Originally this was for folders. Since my testing involves only using filenames, I had to change the Renaming Options to include the File Extension as part of the Renaming in order to make the examples work.

Remove

Remove numeric dollar amount from string cont.

$1125.80 - Business Internet-Static IPs

Match: \$(.*) -(.*)

Where:

1 = 1125.80 \2 = <space> Business Internet-Static IPs

Tim Analysis:

1. \$	Match against literal ' $ ' (Not Captured)	
2..	Match against any character	Capture Group 1 = 1
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. <space>	Match against a <space> character (Not Captured) This is the <space> character after 'Static'	Changes Capture Group 1 = 1125.80 - Business Internet-Static
5. -	Match against <hyphen> (Not Captured) This is the <hyphen> following '1125.80 <space>' This changes step 3 to the <space> after '1125.80'.	Changes Capture Group 1= 1125.80
6. .*	Match against remaining string	Capture Group 2 = <space> Business Internet-Static IPs

Remove

Convert TV Series Season Information from s(##)e(##) to just numeric

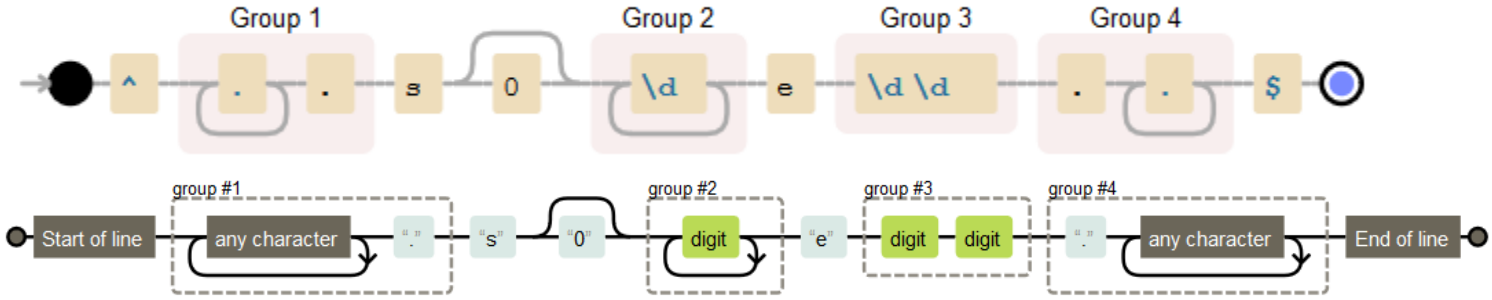
midnightspecial.s01e01.480p.avi	to	midnightspecial.101.480p.avi
midnightspecial.s10e01.480p.avi	to	midnightspecial.1001.480p.avi
midnightspecial.s14e01.480p.avi	to	midnightspecial.1401.480p.avi

Observations:

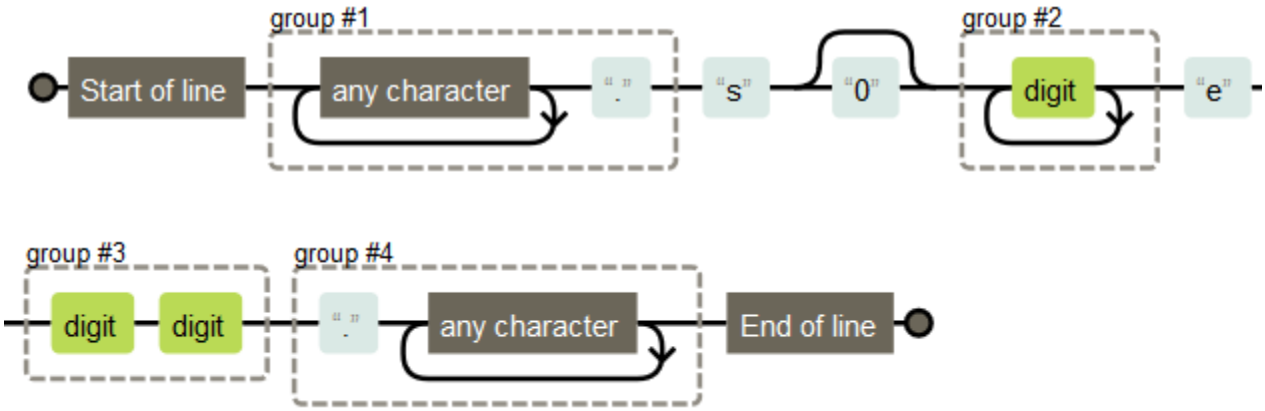
1. For example, Season 1 Episode 2 would be changed to 102.
2. Remove the series 's', the episode 'e', and the leading zero from series numbering.

Match: ^(.+\.)s0?(\d+)e(\d\d)(\..+)$
Replace: \1\2\3\4

Name	New Name
midnightspecial.s14e01.480p.avi	midnightspecial.1401.480p.avi
midnightspecial.s10e01.480p.avi	midnightspecial.1001.480p.avi
midnightspecial.s01e01.480p.avi	midnightspecial.101.480p.avi



Expanded:



Remove

Convert TV Series Season Information from s(##)e(##) to just numeric cont.

Match: ^(.+\.)s0?(\d+)e(\d\d)(\..+)$

Sample:

```
.....................................
Match 1: midnightspecial.s01e01.480p
Group 1: midnightspecial.
Group 2: 1
Group 3: 01
Group 4: .480p
```

Contributor Analysis:

1. Match everything till a literal dot, followed by a literal 's'.
2. followed by a zero or not, followed by one-or-more numeric digits (1 or 10 or 14).
3. followed by a literal 'e', followed by two numeric digits (01).
4. followed by a literal dot and the rest till the end.
5. Replace with what is captured into the (..) parentheses, while dropping what was Not Captured.

Tim Notes:

1. I decided to leave the Contributor's original analysis because it does a pretty good job of explaining things.

Forum Contributor: Stefan

Remove

Convert TV Series Season Information from s(##)e(##) to just numeric cont.

Using: midnightspecial.s01e01.480p.avi

Match: ^(.+\.)s0?(\d+)e(\d\d)(\..+)$

Where:

\1 = midnightspecial. \2 = 1 \3 = 01 \4 = .480p

Tim Notes:

1. Remember what I said earlier? When you have what looks like a complex RegEx, break it down.

anchor	Capture Group 1	literal	Greedy	Capture Group 2	literal	Capture Group 3	Capture Group 4	anchor
^	(.+ \.)	s0	?	(\d+)	e	(\d\d)	(\. .+)	$
BOL	Greedy	literal		Numeric Digit Greedy		Numeric Digit	Literal Greedy	EOL

Not so bad now is it?

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = m
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. \.	Match against literal <dot> character	Capture Group 1 = midnightspecial.
5. s0	Match against string 's0' (Not Captured)	Capture Group 1 = No Change
6. ?	Repeat last match 'Once or None' (optional) 's0' value still viable. Evaluated.	Capture Group 1 = No Change
7. \d	Match against Numeric Digit This is the '1' of 's01'	Capture Group 2 = 1
8. +	Make it Greedy There are no further numeric digits immediately following 's01'. Next character is an 'e'	Capture Group 2 = Unchanged
9. e	Match against literal 'e' (Not Captured) This is the 'e' of 'e01'	
10. \d\d	Match against 2 numeric digits This is the '01' of 'e01'	Capture Group 3 = 01
11. \.	Match against literal <dot> character This is the <dot> following 'e01'	Capture Group 4 = .
12. .+	Match against remaining string	Capture Group 4 = .480p
13. $	Must be at EOL	

Remove

Delete everything in Brackets without numbers

Tim Notes:

Here is an interesting learning session. The contributor decided on presenting two different versions, each breaking down the request with a single purpose RegEx before presenting the 'one' RegEx that would satisfy the request. I also modified the wording for clarification.

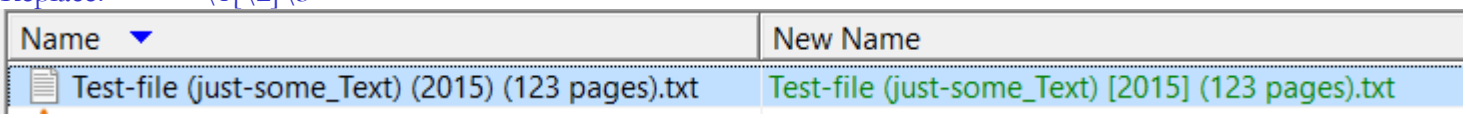
> Test-file (just-some_Text) (2015) (123 pages).txt
> to
> Test-file (2015).txt

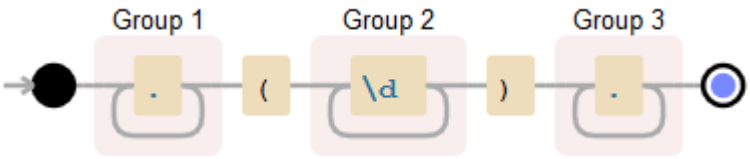
Observations:

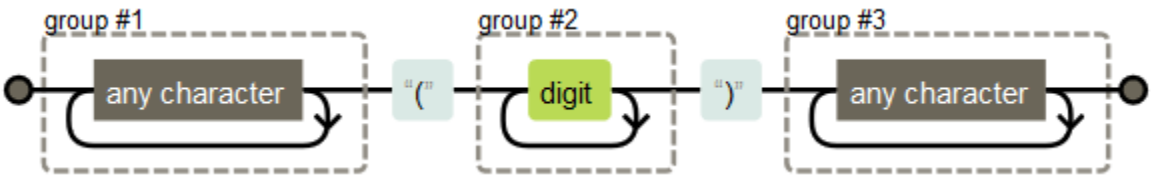
1. Pattern established - xxx (garbage) (numeric digits) (garbage)

Protect Brackets with numbers only:

Match: (.+)\((\d+)\)(.+)
Replace: \1[\2]\3







Sample:

```
.....................................................
Match 1: Test-file (just-some_Text) (2015) (123 pages)
Group 1: Test-file (just-some_Text)
Group 2: 2015
Group 3:  (123 pages)
```

Remove

Delete everything in Brackets without numbers cont.

Using: Test-file (just-some_Text) (2015) (123 pages).txt

Protect Brackets with numbers only: cont.

Match: (.+)\((\d+)\)(.+)

Where:

\1 = Test-file (just-some_Text)<space> \2 = 2015 \3 = <space>(123 pages)

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \(	Match against literal left parenthesis (Not Captured) This is the left parenthesis of '(123'	Changes Capture Group 1 = Test-file (just-some_Text) <space> (2015) <space>
4. \d	Match against Numeric Digit This is the '1' of '123'	Capture Group 2 = 1
5. +	Make it Greedy	Capture Group 2 = 123
6. \)	Match against literal right parenthesis (Not Captured) This is the right parenthesis of '(2015)'	Changes Capture Group 2 = 2015 Changes Capture Group 1 = Test-file (just-some_Text) <space>
7. .+	Match against remaining string	Capture Group 3 = <space>(123 pages)

Tim Notes:

1. In step 6 you will notice that matching a character without having to capture it is all that is required to have other Capture Groups reevaluate themselves. When the match against the right parenthesis is made, Capture Group 2 reevaluates to change its value to '2015' and this in turn forces Capture Group 1 to drop the '2015' from it's value.

   It has to backtrack in order to match against the right parenthesis of '(2015)'. Why didn't it just move forward and match against the right parenthesis of '(123 pages)'?

   Because it is still testing using the, (\d+)\) – in other words the match is not True unless a numeric digit group is immediately followed by a right parenthesis. There is text separating the '123' from the ') ', and therefore doesn't match. The RegEx engine backtracks to match against the ') ' of '2015'. This in turn changes the values of the preceding Capture Groups (as required).

Remove

Delete everything in Brackets without numbers cont.

Tim Notes:

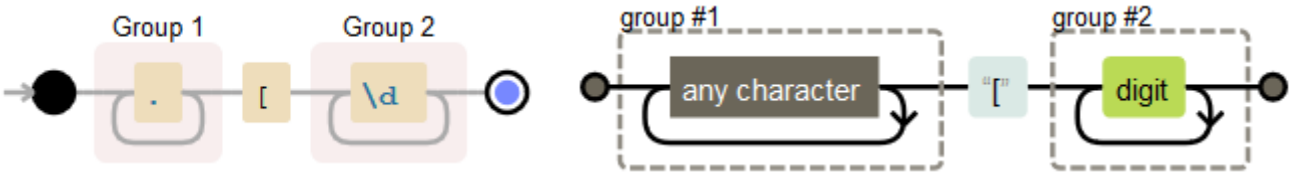
Took me a little bit of time to comprehend, but I guess in some countries, parentheses are referred to as brackets whereas in other countries, they are distinguished separately. These countries tend to refer to what we call brackets [ ] as square brackets. The poster is referring to parentheses as brackets, while the contributor is understandably misinterpreting the request and adding brackets [ ], at least in this example.

Replace [brackets] back to (parentheses):

i.e., Test-file (just-some_Text) [2015] (123 pages).txt

Match: (.+)\[(\d+)
Replace: \1(\2)

Name	New Name
Test-file (just-some_Text) [2015] (123 pages).txt	Test-file (just-some_Text) (2015).txt



Sample:

```
----------------------------------------
Match 1: Test-file (just-some_Text) [2015
Group 1: Test-file (just-some_Text)
Group 2: 2015
```

Remove

Delete everything in Brackets without numbers cont.

Replace [brackets] back to (parentheses): cont.

Using: Test-file (just-some_Text) [2015] (123 pages).txt

Match: (.+)\[(\d+)

Where:

\1 = Test-file (just-some_Text)<space> \2 = 2015

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \[	Match against literal left bracket (Not Captured) This is the left bracket of '[2015'	Changes Capture Group 1 = Test-file (just-some_Text) <space>
4. \d	Match against Numeric Digit This is the '2' of '2015'	Capture Group 2 = 2
5. +	Make it Greedy	Capture Group 2 = 2015

Remove

Delete everything in Brackets without numbers cont.

Tim Notes:

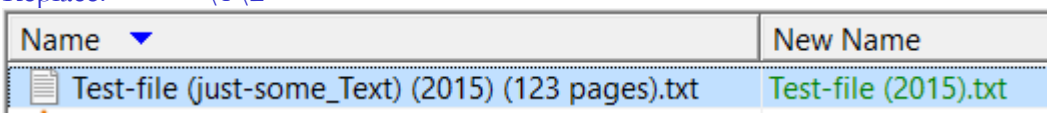
1. See Previous note concerning Brackets.

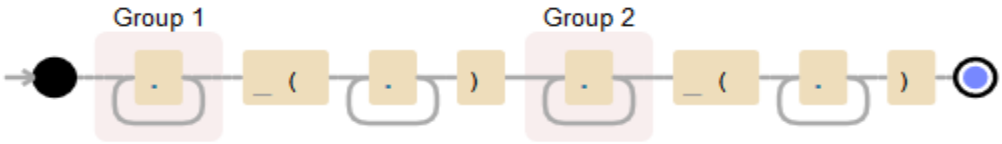
Replace [brackets] back to (parentheses):

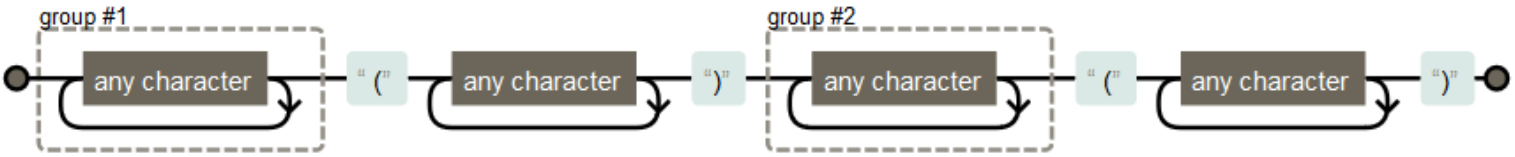
i.e., Test-file (just-some_Text) [2015] (123 pages).txt

Final RegEx - Parse string and remove garbage:

Match: (.+) \(.+\)(.+) \(.+\)
Replace: \1\2







Sample:

```
.....................................................
Match 1: Test-file (just-some_Text) [2015] (123 pages)
Group 1: Test-file
Group 2:  [2015]
```

Forum Contributor: Stefan

Remove

Delete everything in Brackets without numbers cont.

Replace [brackets] back to (parentheses): cont.

Using: Test-file (just-some_Text) [2015] (123 pages).txt

Final RegEx - Parse string and remove garbage: cont.

Match: (.+) \(.+\)(.+) \(.+\)

Where:

\1 = Test-file \2 = <space> [2015]

Tim Analysis:

1..	Match against any character	Capture Group 1 =
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after '[2015]'	Changes Capture Group 1 = Test-file (just-some_Text) [2015]
4. \(	Match against literal left parenthesis (Not Captured) This is the left parenthesis of '(123'	
5..	Match against any character (Not Captured) = 1	
6. +	Make it Greedy (Not Captured) = 123 pages)	
7. \)	Match against literal right parenthesis (Not Captured) This is the right parenthesis of 'Text)'	
8..	Match against any character This is the <space> after 'Text)'	Capture Group 2 = <space>
9. +	Make it Greedy Changes Not Captured to 'Just-some_Text) Current position = EOL	Capture Group 2 = <space>[2015] (123 pages) Changes Capture Group 1 = Test-file

Tim Notes:

1. The Not Captured in steps 4 through 6 was a value of '(123 pages)'. When Capture Group 2 captured '(123 pages) as part of it's value, this reevaluates steps 4 through 6 to match against another left parenthesis followed by text. It backtracked to match against the left parenthesis of '(just', followed by the text leading up to '<space> [2015]', a value now held by Capture Group 2. Thus, the Steps 4 through 6 now were equal to '(just-some_Text).

   Step 3's value of the <space> now held by Capture Group 2, is changed to the next <space> encountered, (still backtracking), the <space> after 'Test-file'.

   This in turn reevaluates Capture Group 1 which was forced to give up the characters '(just-some_Text)' from it's captured value for the uncaptured value along with the '<space> [2015]' value now held by Capture Group 2, leaving Capture Group 1 = 'Test-file'

Remove

Delete everything in Brackets without numbers cont.

Replace [brackets] back to (parentheses): cont.

Using: Test-file (just-some_Text) [2015] (123 pages).txt

Final RegEx - Parse string and remove garbage: cont.

Match: (.+) \(.+\)(.+) \(.+\)

Where:

\1 = Test-file \2 = <space> [2015]

Tim Analysis: cont.

10. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after '(123'	Changes Capture Group 2 = <space>[2015] (123
11. \(	Match against literal left parenthesis (Not Captured) This is the left parenthesis of '(123' This also changes the <space> in step 10 to the <space> after '[2015]' since the value of '(123<space> is now discarded. This changes Capture Group 2 = <space>[2015]. Current position = ' (' of '(123'.	Changes Capture Group 2 = <space>[2015]
12..	Match against any character (Not Captured) = 1 This is the '1' of '123	
13. +	Make it Greedy (Not Captured) = 123 pages)	
14. \)	Match against literal right parenthesis (Not Captured) This is the right parenthesis of '123 pages). A bit redundant since this was already matched in step 13.	

Tim Notes:

1. Not sure why the Contributor added .+\) (steps 12-14) because the final match required was in step 11.

Remove

Remove hyphen as first character in string

-3.text	to	3.text
-file_01-09-05.pdf	to	file_01-09-05.pdf
-[Word1-Word2] - 2009.ext	to	[Word1-Word2] - 2009.ext
-Calendars packing slip 09-29-11.pdf	to	Calendars packing slip 09-29-11.pdf
-2015-01-01 - Market Center Alpha San Jose	to	2015-01-01 - Market Center Alpha San Jose

Method #1:

1. 'Section #12: Filters' –

Mask: -* <hyphen> followed by *

This will cause BRU to display only filenames starting with a single <hyphen>

Name	New Name
-file_01-09-05.pdf	-file_01-09-05.pdf
-Calendars packing slip 09-29-11.pdf	-Calendars packing slip 09-29-11.pdf
-[Word1-Word2] - 2009.ext	-[Word1-Word2] - 2009.ext
-3.text	-3.text
-2015-01-01 - Market Center Alpha San Jose	-2015-01-01 - Market Center Alpha San Jose

2. 'Section #5: Remove' -

First n: 1
This removes that <hyphen>

Name	New Name
-file_01-09-05.pdf	file_01-09-05.pdf
-Calendars packing slip 09-29-11.pdf	Calendars packing slip 09-29-11.pdf
-[Word1-Word2] - 2009.ext	[Word1-Word2] - 2009.ext
-3.text	3.text
-2015-01-01 - Market Center Alpha San Jose	2015-01-01 - Market Center Alpha San Jose

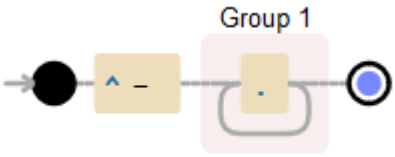
Remove

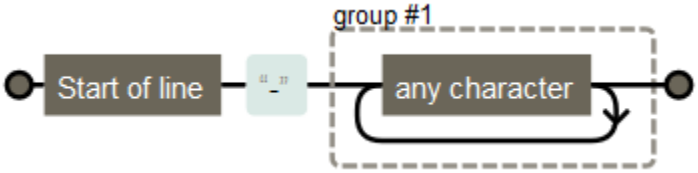
Remove hyphen as first character in string cont.

Method #2: RegEx

Match: ^-(.+)
Replace: \1

Name	New Name
-file_01-09-05.pdf	file_01-09-05.pdf
-Calendars packing slip 09-29-11.pdf	Calendars packing slip 09-29-11.pdf
-[Word1-Word2] - 2009.ext	[Word1-Word2] - 2009.ext
-3.text	3.text
-2015-01-01 - Market Center Alpha San Jose	2015-01-01 - Market Center Alpha San Jose





Sample:

```
.........................
Match 1: -file_01-09-05
Group 1: file_01-09-05
```

Contributor Analysis:

1. Matches all filenames starting with a <hyphen> (the ^ means the start of the filename - Beginning of Line or BOL), saving everything after the <hyphen> in Capture Group 1

Forum Contributor: bitmonger

Where:

\1 = file_01-09-05

Using: -file_01-09-05.pdf

Tim Analysis:

1. ^ Start at beginning of string (BOL)
2. - Match against <hyphen> (Not Captured)
3. .+ Match against remaining string — Capture Group 1 = file_01-09-05

Remove

How to keep just a 4 digit or 5 digit number

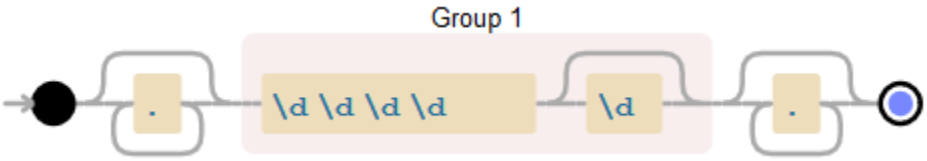
ZL02.5largedish3344.pdf to 3344.pdf
zl02.44567largeplate.pdf to 44567.pdf

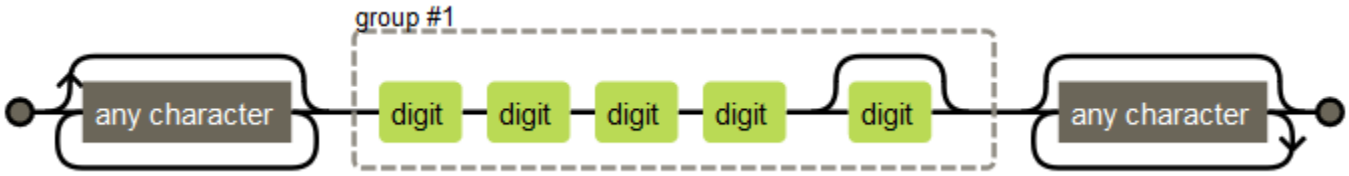
Observations:

1. There is only one 4 or 5 numeric digit contiguous number buried in the filename.

Match: .*?(\d\d\d\d\d?).*
Replace: \1

Name	New Name
ZL02.5largedish3344.pdf	3344.pdf
zl02.44567largeplate.pdf	44567.pdf





Sample:

```
................................
Match 1: ZL02.5largedish3344.pdf
Group 1: 3344
```

Forum Contributor: bitmonger

Remove

How to keep just a 4 digit or 5 digit number cont.

Using: ZL02.5largedish3344.pdf

Match: .*?(\d\d\d\d\d?).*

Where:

\1 = 3344

For the sake of simplicity I will change the RegEx to:

Match: .*?(\d{5}?).*

This doesn't affect the final outcome. – or does it?

Tim Analysis:

1..	Match against any character (Not Captured) = 'Z'
2. *	Make it Greedy (Not Captured) = <entire string>
3. ?	But not too Greedy (Not Captured) Changes string position to BOL
4. \d{5}	Match against 5 numeric digits = False

didn't work.

Tim Notes:

Why? Do you know?

My change is what caused the problem. Yes, sometimes you can simplify a RegEx that has multiple Metacharacters using a Quantifier, but not always.

The purpose of the RegEx is to be able to match against filenames that contain *either* 4 or 5 numeric digits. By using the, \d{5}, I eliminated the possibility of matching anything but a string containing 5 numeric digits.

So, let's try again – this time using the original RegEx and see what happens.

Remove

How to keep just a 4 digit or 5 digit number cont.

Using: ZL02.5largedish3344.pdf

Match: .*?(\d\d\d\d\d?).*

Where:

\1 = 3344

Tim Analysis:

1..	Match against any character (Not Captured) = 'Z'	
2. *	Make it Greedy (Not Captured) = <entire string>	
3. ?	But not too Greedy (Not Captured) Changes string position to BOL	
4. \d	Match against a numeric digit	Capture Group 1 = 3
5. \d	Match against a numeric digit	Capture Group 1 = 33
6. \d	Match against a numeric digit	Capture Group 1 = 334
7. \d	Match against a numeric digit	Capture Group 1 = 3344
8. \d	Match against a numeric digit = False There are only 4 numeric digits.	
9. ?	Make the last match optional. The last match of the fifth \d is made optional, therefore the \d\d\d\d is matched.	
10. .*	Match against remaining string (Not Captured) This would handle filenames that have text after the numeric digits.	Capture Group 1 = No Change

Tim Notes:

1. There is some redundancy that could be modified. Normally, using .*? at the beginning of the RegEx is a good technique to set up Capture Group 1; by moving to the end of the string to EOL and backtracking to the beginning of the string to BOL.

   However, Capture Group 1 appears later in the RegEx. If I were to remove the sub expression, there would be no change to the outcome since the RegEx engine will move forward through the string to match up against the numeric digits.

Match: (\d\d\d\d\d?).*
Replace: \1

Name	New Name
zl02.44567largeplate.pdf	44567.pdf
ZL02.5largedish3344.pdf	3344.pdf

Remove

Remove Front end of filename using pattern <space> <hyphen> <space>

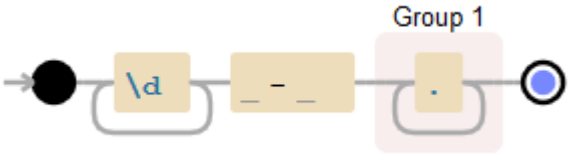
12 - Duran Duran – Album Title	to	Duran Duran – Album Title
02 - Fallout Boys – Album Title	to	Fallout Boys – Album Title
231 - Bruce Springsteen – Album Title	to	Bruce Springsteen – Album Title

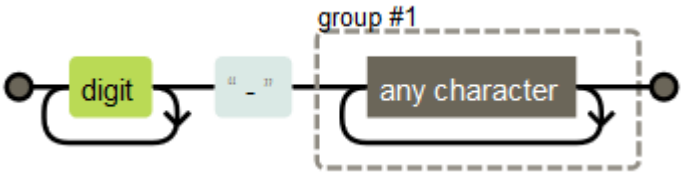
Observations:

1. Remove all before the group name.
2. Pattern established – first occurrence of <space><hyphen><space>

Match: \d+ - (.+)
Replace: \1

Name	New Name
231 - Bruce Springsteen – Album ...	Bruce Springsteen – Album Title
12 - Duran Duran – Album Title	Duran Duran – Album Title
02 - Fallout Boys – Album Title	Fallout Boys – Album Title





Sample:

```
.....................................
Match 1: 12 - Duran Duran – Album Title
Group 1: Duran Duran – Album Title
```

Contributor Analysis:

1. \d match one numeric digits
2. + match one-or-more of the last expression, here the \d
3. ' - ' match <space> <hyphen> <space>
4. (.+) match one-or-more characters, and store it in group one, which match can be accessed by '\1' in the replacement.

Remove

Remove Front end of filename using pattern <space> <hyphen> <space> cont.

This also works:

Match: \d - (.+)
Replace: \1

Name	New Name
231 - Bruce Springsteen – Album ...	Bruce Springsteen – Album Title
12 - Duran Duran – Album Title	Duran Duran – Album Title
02 - Fallout Boys – Album Title	Fallout Boys – Album Title

Sample:

```
.....................................
Match 1: 2 - Duran Duran – Album Title
Group 1: Duran Duran – Album Title
```

Contributor Analysis:

1. This works but should it? Pattern changed to <numeric digit> <space> <hyphen><space>

   a. \d still works because the RegEx still matches part of the line.
   for example, '12 - Duran Duran - Wild Boys.mp3', the \d - will match the '2 - '. The (.+) matches the rest.

   b. It's true that it doesn't match the whole line but we aren't capturing anything before the \d so it is ignored and dropped when the replacement takes place.

2. As long as the complete RegEx matches within the line it will work, even if that match doesn't include the whole line.

   a. It does make a difference (won't match) if we change the RegEx to:

   ^\d - (.+)

   In this case it is trying to force a match with a single leading numeric digit beginning at the start of the line (Beginning of Line or BOL).

Forum Contributors: Stefan, Glenn

Remove

Similar but Not the same - Remove Front end of filename using pattern

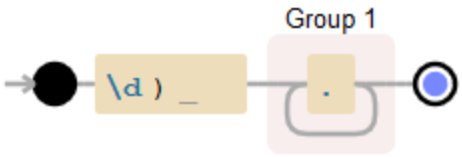
1) Origin 01.cbz to Origin 01.cbz

Observations:

1. Pattern is <numeric digit><close parenthesis><space>
2. Pattern established

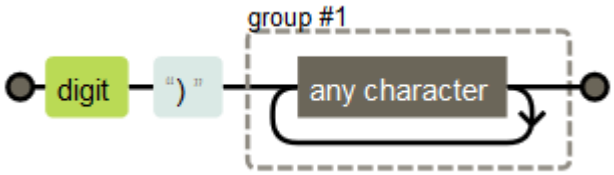
Match: \d\) (.+)
Replace: \1

Name	New Name
1) Origin 01.cbz	Origin 01.cbz



Sample:

```
.......................
Match 1: 1) Origin 01
Group 1: Origin 01
```



Tim Notes:

1. I included this to show that the same RegEx, '\d+ - (.+)', in 'Remove Front end of filename using pattern' cannot be used here– the RegEx must reflect the pattern of the samples.

Forum Contributor: bitmonger

Where:

\1 = Origin (01)

Tim Analysis:

1. \d Match against Numeric Digit (Not Captured) = 1
2. \) Match against right parenthesis (Not Captured) =) Not Captured = 1)
3. <space> Match against a <space> character (Not Captured) Not Captured = 1) <space>
4. .+ Match against remaining string Capture Group 1 = Origin (01)

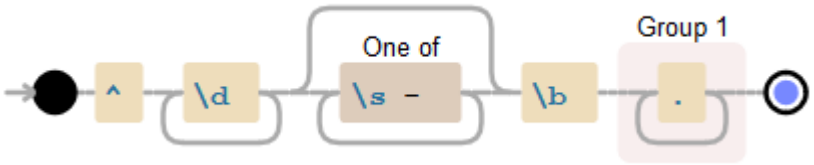
Remove

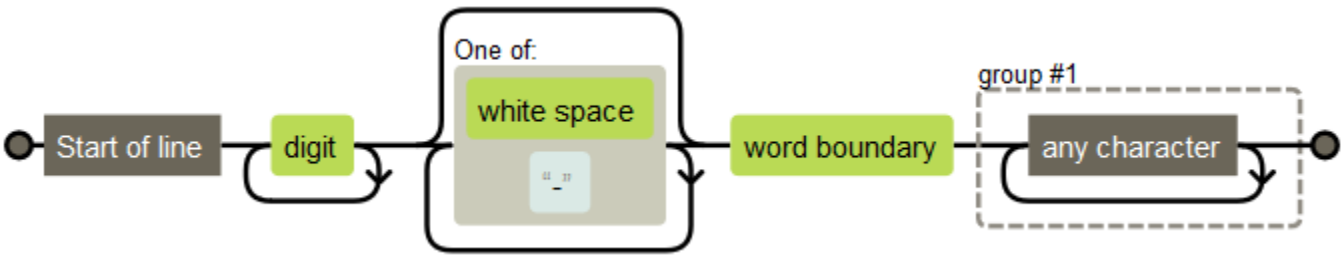
Remove numeric digits, spaces and hyphen at the beginning of filename

01 hello1.mp3	to	hello1.mp3
01 - hello2.mp3	to	hello2.mp3
01- hello3.mp3	to	hello3.mp3"
01 -hello4.mp3	to	hello4.mp3
01-hello5.mp3	to	hello5.mp3
01.mp3	to	Ignore

Match: ^\d+[\s-]*\b(.+)
Replace: \1

Name	New Name
01 hello1.mp3	hello1.mp3
01 - hello2.mp3	hello2.mp3
01- hello3.mp3	hello3.mp3
01 -hello4.mp3	hello4.mp3
01-hello5.mp3	hello5.mp3
01.mp3	01.mp3





Sample:

```
.....................
Match 1: 01 hello1
Group 1: hello1
```

Forum Contributor: TheRegExpMaster

Remove

Remove numeric digits, spaces and hyphen at the beginning of filename cont.

Using 01 - Hello2.mp3

Where:

\1 = hello2

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. \d	Match against first numeric digit (Not Captured) = '0'	
3. +	Make it Greedy (Not Captured) = '01'	
4. [\s-]	Match against a class that is either a <space> or a <hyphen> (Not Captured) = <space>	
5. *	Make it Greedy (Not Captured) = <space> <hyphen> <space>	
6. \b	Word Boundary. Matches against a letter (Upper or lowercase, a numeric digit, or an underscore character. = 'h' = TRUE	
7. .	Match against next character	Capture Group 1 = h
8.+	Make it Greedy	Capture Group 1 = hello2

Tim Notes:

1. /b Normally this establishes the start of a Word Boundary - it can be used to isolate whole words using the syntax is \b …. \b where the expressions with the boundary as defined by the beginning of the boundary and the end of the boundary represented by the \b metacharacter, are characters with no whitespace. This includes letters both upper and lowercase, numeric digits and the underscore. It can also be used separately as is done in this example.

2. In this example, it is used to exclude the 01.mp3 file, because there are no letter, numeric digits or an underscore that follows after the 01. In step 4, this would have excluded the file, because there is no <space> or <hyphen> following but by making it Greedy in step 5, it included to the EOL (End of Line) and so it matched again. By using the /b, it Fails.

   The Word Boundary does not capture. To determine if TRUE, I enclose it in a Capture Group then use my little trick of (…) to ascertain the current position.

   (\b...) = hel

   The current position = ' h ' = TRUE. The evaluation continues.

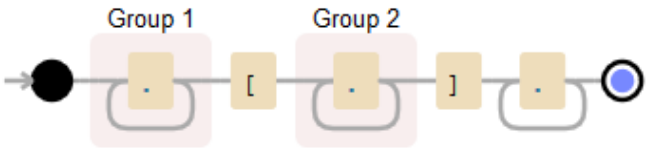
Remove

Remove everything after brackets at end of filename and replace Brackets with Parentheses

Hard To Say I'm Sorry 2K9 [Radio Edit]Radio Edit	to	Hard To Say I'm Sorry 2K9 (Radio Edit)
It Just Won't Do [Bigroom Radio Edit]Bigroom Radio Edit	to	It Just Won't Do (Bigroom Radio Edit)
Music Matters Feat. Cass Fox [Axwell Edit]Axwell Edit	to	Music Matters Feat. Cass Fox (Axwell Edit)
Memory [Klaas Mix]Klaas Mix	to	Memory (Klaas Mix)

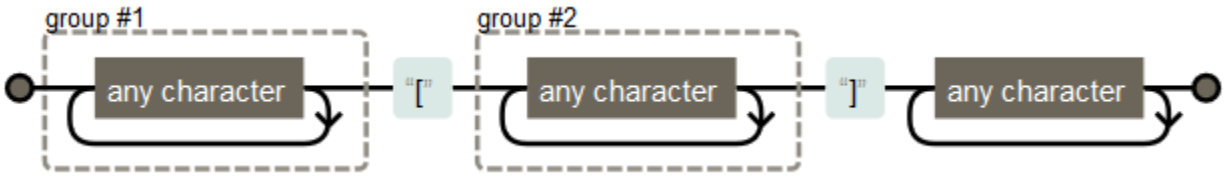
Match: (.+)\[(.+)\].+
Replace: \1(\2)

Name	New Name
Music Matters Feat. Cass Fox [Axwell Edit]Axwell Edit	Music Matters Feat. Cass Fox (Axwell Edit)
Memory [Klaas Mix]Klaas Mix	Memory (Klaas Mix)
It Just Won't Do [Bigroom Radio Edit]Bigroom Radio E...	It Just Won't Do (Bigroom Radio Edit)
Hard To Say I'm Sorry 2K9 [Radio Edit]Radio Edit	Hard To Say I'm Sorry 2K9 (Radio Edit)



Sample:

```
Match 1: Hard To Say I'm Sorry 2K9 [Radio Edit]Radio Edit
Group 1: Hard To Say I'm Sorry 2K9
Group 2: Radio Edit
```



Tim Notes:

1. Because of the dot character in Music Matters filename, I needed to enable, 'Rename File Extensions as Part of the Filename' in Renaming Options Menu for this sample to work.

Contributor Notes:

1. Be aware the RegEx cannot check if the text inside the brackets is the same as the one afterwards.
   i.e., Memory [Klaas Mix]Hellooooooooooworld.mp3 will be renamed Memory (Klaas Mix).mp3

2. The file name must have a string both before the brackets, inside the brackets and after the brackets to match the pattern.
   a. If you want to allow empty strings, use ".*" instead of ".+").

   i.e. [Klaas Mix]Klaas Mix.mp3 will be unchanged
   Memory [Klaas Mix].mp3 will be unchanged
   Memory []Klaas Mix.mp3 will be unchanged

Forum Contributor: TheRegExpMaster

Remove

Remove everything after brackets at end of filename and replace Brackets with Parentheses cont.

Using: Memory [Klaas Mix]Klaas Mix

Match: (.+)\[(.+)\].+

Where:

\1 = Memory <space> \2 = Klaas Mix

Tim Analysis:

1..	Match against any character	Capture Group 1 = M
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \[	Match against literal left bracket (Not Captured)	Changes Capture Group 1 = Memory<space>
4..	Match against any character	Capture Group 2 = K
5. *	Make it Greedy	Capture Group 2 = Klaas Mix]Klaas Mix
6. \]	Match against literal right bracket (Not Captured)	Changes Capture Group 2 = Klaas Mix
7..	Match against any character (Not Captured) = EOL	
8. *	Make it Greedy (Not Captured) = No Change	

Tim Notes:

1. As far as I can see, the final, ' .+ ' is redundant since all text after brackets is discarded in step 6.

Remove

Remove everything after brackets at end of filename and replace Brackets with Parentheses cont.

Tim Contribution –

Back to the original RegEx –

Match: (.+)\[(.+)\].+
Replace: \1(\2)

Where:

\1 = <all of the text leading up to the bracketed section> \2 = <text (only) in bracketed section>

Looking back …

Contributor Notes:

1. Be aware the RegEx cannot check if the text inside the brackets is the same as the one afterwards.
   i.e., Memory [Klaas Mix]Hellooooooooooworld.mp3 will be renamed Memory (Klaas Mix).mp3

Name	New Name
Hard To Say I'm Sorry 2K9 [Radio Edit]Radio Edit	Hard To Say I'm Sorry 2K9 (Radio Edit)
It Just Won't Do [Bigroom Radio Edit]Bigroom Radio Edit	It Just Won't Do (Bigroom Radio Edit)
Music Matters Feat. Cass Fox [Axwell Edit]Axwell Edit	Music Matters Feat. Cass Fox (Axwell Edit)
Memory [Klaas Mix]Hellooooooooooworld	Memory (Klaas Mix)
Memory [Klaas Mix]Klaas Mix	Memory (Klaas Mix)

Okay so what is the original Contributor's intent?

It isn't just to remove extraneous information after the brackets. As you can see from the samples, all of them have a pattern of redundant data appearing in both the bracketed section and the text following after. Therefore the goal is to match only those files with redundant data, and that is why the Contributor specifically points out that a file such as:

Memory [Klaas Mix]Hellooooooooooworld.mp3

..that does NOT contain redundant data would still be inadvertently included in the Renaming process as evidenced in the above photo.

This is what the Contributor meant when stated that the RegEx does not look inside the bracketed text to verify that it will match the text that follows outside of the bracketed section.

Read on for a solution.

Remove

Remove everything after brackets at end of filename and replace Brackets with Parentheses cont.

Tim Contribution – cont.

However, this can:

Using a Backreference in a RegEx –

Memory [Klaas Mix]Hellooooooooooworld to ignore

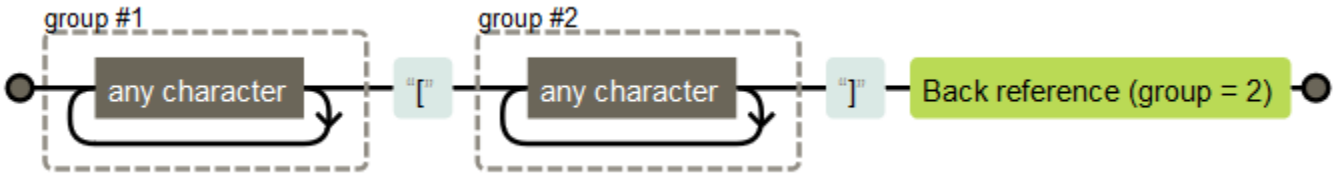
Match: (.+)\[(.+)\]\2
Replace: \1(\2)

Name	New Name
Hard To Say I'm Sorry 2K9 [Radio Edit]Radio Edit	Hard To Say I'm Sorry 2K9 (Radio Edit)
It Just Won't Do [Bigroom Radio Edit]Bigroom Radio Edit	It Just Won't Do (Bigroom Radio Edit)
Music Matters Feat. Cass Fox [Axwell Edit]Axwell Edit	Music Matters Feat. Cass Fox (Axwell Edit)
Memory [Klaas Mix]Hellooooooooooworld	Memory [Klaas Mix]Hellooooooooooworld
Memory [Klaas Mix]Klaas Mix	Memory (Klaas Mix)

Ignores file in rename because it doesn't fit the redundant pattern –

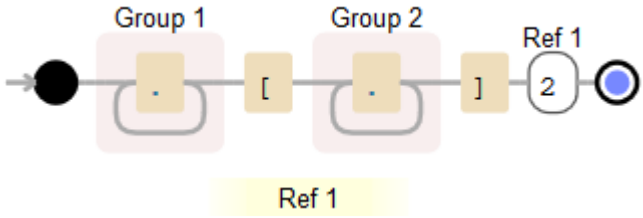
Where:

\2 = Klaas Mix



Sample: The modified RegEx will still match existing samples.

```
.....................................
Match 1: Memory [Klaas Mix]Klaas Mix
Group 1: Memory
Group 2: Klaas Mix
```



Tim Notes:

1. I have used Backreferences in a RegEx in another example in the book. I refer you to 'Remove Duplicate Title From Multiple Music Filenames' which can be found in this section.

2. The reference to backreference \2 in the RegEx is how the expression determines NOT to rename files that do not contain redundant data because if the duplicate text is NOT matched, the evaluation Fails and the filename will not be renamed.

Remove

Remove everything after brackets at end of filename and replace Brackets with Parentheses cont.

Tim Contribution – cont.

Using a Backreference in a RegEx – cont.

Using: Memory [Klaas Mix]Hellooooooooooworld

Match: (.+)\[(.+)\]\2

Goal is to fail the file so it won't be renamed.

Tim Analysis:

1..	Match against any character	Capture Group 1 = M
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \[	Match against literal left bracket (Not Captured)	Changes Capture Group 1 = Memory<space>
4..	Match against any character	Capture Group 2 = K
5. *	Make it Greedy	Capture Group 2 = Klaas Mix]Klaas Mix
6. \]	Match against literal right bracket (Not Captured)	Changes Capture Group 2 = Klaas Mix
7. \2	Backreference to Capture Group 2. Capture Group 2 contains the value, 'Klaas Mix'. Therefore we are trying to match up against the string, 'Klaas Mix' = 'Hellooooooooooworld' = False. RegEx Fails.	

Tim Notes:

1. This is the simplified analysis considering that the RegEx itself completes the task in 1,909 steps! (Regex Buddy can name that tune in 2,664 steps!!) What it comes down to at the end, is that when it can't match against the 'Klaas Mix' value referenced by backreference \2, it will backtrack (continually) but it can't match the bracketed string '[Klass Mix]' either, because that value is held by Capture Group 2, which is the literal value it is trying to match (can't match against itself).

   --- as Kirk says to the Android, 'Everything that Mudd says is a lie'. Mudd then tells the Android that he is lying. Android shuts down not able to process conflicting data. Anyone hungry for 'Pi' ??

With thanks to RegexNinja

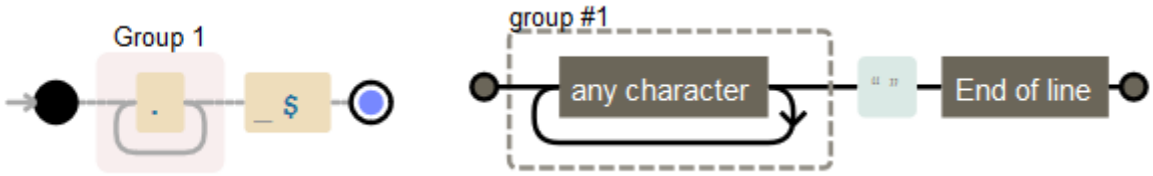
Remove

Remove Space from end of filename (to demonstrate the $ anchor)

this is a test .jpg to this is a test.jpg

Match: (.+) $
Replace: \1

Name	New Name
this is a test .jpg	this is a test.jpg



Sample:

```
.........................
Match 1: this is a test
Group 1: this is a test
```

Tim Notes:

1. There is a <space> between the closing parenthesis and the dollar sign (anchor).
2. The $ indicates the end of the filename (EOL), so this only matches the names that end in a <space>.

Forum Contributor: bitmonger

Where:

\1 = this is a test

Tim Analysis:

1..	Match against any character	Capture Group 1 = t
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) Backtracks. This is the <space> after 'test'	Changes Capture Group 1 = this is a test
4. $	Must be at EOL (End of Line)	

Remove

Remove text between two Dots

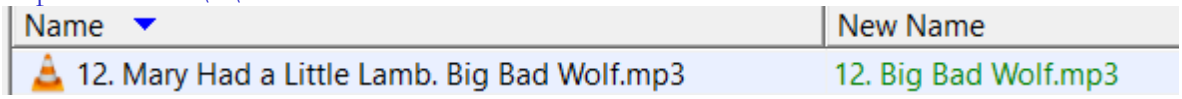
12. Mary Had a Little Lamb. Big Bad Wolf.mp3
to
12. Big Bad Wolf.mp3

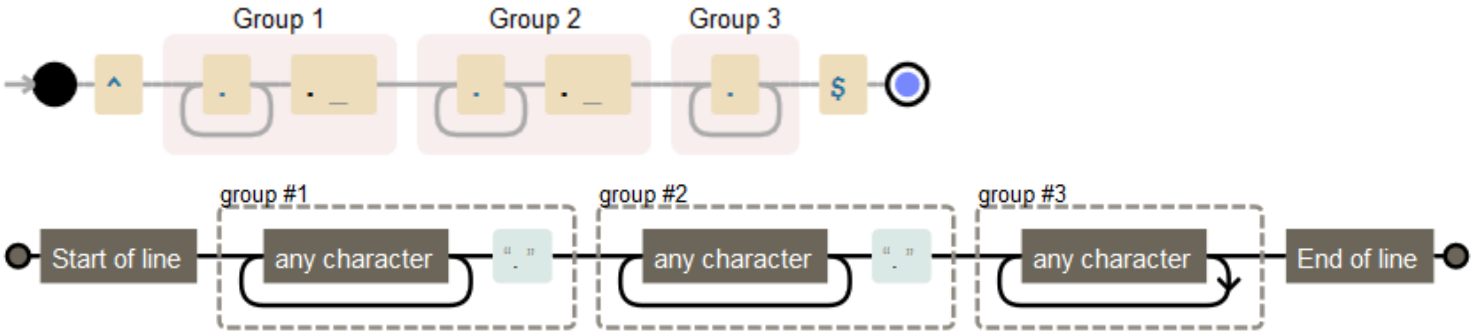
Observations:

1. Remove the text between the first and second dot (including the second dot).

Not restrictive pattern <text><dot><text><dot>

Match: ^(.+?\.)(.+?\.)(.+)$
Replace: \1\3





Sample:

```
.....................................................
Match 1: 12. Mary Had a Little Lamb. Big Bad Wolf
Group 1: 12.
Group 2: Mary Had a Little Lamb.
Group 3: Big Bad Wolf
```

Contributor Analysis:

1. Match everything up till first dot.
2. Match everything up till second dot.
3. Capture rest of string.

Remove

Remove text between two Dots cont.

Using: 12. Mary Had a Little Lamb. Big Bad Wolf.mp3

Not restrictive pattern <text><dot><text><dot> cont.

Match: ^(.+?\.)(.+?\.)(.+)$

Where:

\1 = 12.<space> \2 = Mary Had a Little Lamb. <space> \3 = Big Bad Wolf

Breaks down to:

^ (.+? \.) (.+? \.) (.+) $

Tim Analysis:

1.	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = 1
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy (Lazy)	Capture Group 1 = 1

Tim Notes:

1. If you have been following my text up to now, I have told you how using the sub-expression, ' (.*?) ', at the beginning of the RegEx, is an interesting way to set up Capture Group 1. The <dot> Metacharacter captures the first character in the string. The Greedy Quantifier then matches to the end of the string (the EOL), capturing the entire string. The Lazy Quantifier then moves back to the beginning of the string (The BOL), forcing Capture Group 1 to give up it's initial value, but still matching (a null value). The next sub-expression in the RegEx will change Capture Groups' value depending on the position in the string at the end of this evaluation.

   Using the sub-expression, ' (.+?) ', at the beginning of the RegEx is similar. The first character is captured and the RegEx Engines matches to the end of the string as before using the Greedy Quantifier ' + '. But when the Lazy Quantifier is added, instead of moving to the BOL, it moves to the previous capture position, retaining it's original value.

   This is because of the difference between the Greedy Quantifiers, ' * ', and ' + '. The asterisk is used to match **ZERO** or more times. When the Lazy Quantifier is applied, it makes the previous match optional – meaning, give it an out, and it will take it – so **zero** it is – back to BOL. (This will happen regardless of the 'start of line anchor', ' ^ ' used in this RegEx – it just happened to have one). The plus sign, on the other hand, is used to match at least **ONCE**. When the Lazy Quantifier is applied, it also makes the previous match optional, but it will match at least the **one** time, so Capture Group 1 gives up the match of the entire string but retains its original matched value, the '1'.

Remove

Remove text between two Dots cont.

Using: 12. Mary Had a Little Lamb. Big Bad Wolf.mp3

Not restrictive pattern <text><dot><text><dot> cont.

Match: ^(.+?\.)(.+?\.)(.+)$

Where:

\1 = 12. \2 = Mary Had a Little Lamb. <space> \3 = Big Bad Wolf

Tim Analysis: cont.

5. \. Match against literal <dot> character Capture Group 1 = 12.

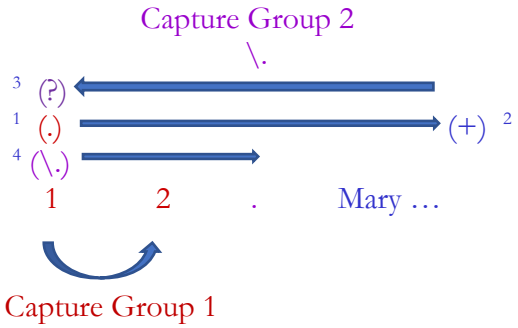
Tim Notes:

1. Although Capture Group 1 retained it's initial value of the '1', just as using the, ' (.*?) ' would force Capture Group 1 to reevaluate its value based on the evaluation of the next sub-expression, so too, will this have an effect using ' (.+?) '. It's not as clear here because the next sub-expression is part of Capture Group 1, so all you are seeing is the additional capture of the '2' with an added <dot> character. But, if you had:

(.+?)(\.)

1. Match against any character — Capture Group 1 = 1
2. + Make it Greedy — Capture Group 1= <entire string>
3. ? But not too Greedy — Capture Group 1 = 1
4. \. Match against literal <dot> character — Capture Group 2 = .

Changes Capture Group 1= 12



Capture Group 1 = 12
Capture Group 2 = .

Now you can see that the evaluation of the literal <dot> character moved the RegEx Engine to match against the <dot> following '12', changing Capture Group 1's initial value of '1' by adding the '2' value to the '1', while still capturing the <dot> character's value for Capture Group 2. In other words, the affect on Capture Group's 1' value, based on the next sub-expression's evaluation, is the same for using either, ' (.*?) ' or ' (.+?) ', it is only the retention of the initial value that is different.

Remove

Remove text between two Dots cont.

Using: 12. Mary Had a Little Lamb. Big Bad Wolf.mp3

Not restrictive pattern <text><dot><text><dot> cont.

Match: ^(.+?\.)(.+?\.)(.+)$

Where:

\1 = 12.<space> \2 = Mary Had a Little Lamb. <space> \3 = Big Bad Wolf

Tim Analysis: cont.

6. <space>	Match against a <space> character	Capture Group 2 = 12.<space>
7..	Match against any character	Capture Group 2 = M
8. +	Make it Greedy	Capture Group 2 = Mary Had a Little Lamb. Big Bad Wolf
9. ?	But not too Greedy (Lazy)	Capture Group 2 = M
10. <space>	Match against a <space> character	Capture Group 2 = Mary Had a Little Lamb. <space>
11. \.	Match against literal <dot> character	Capture Group 3 = B
12. .+	Match against remaining string	Capture Group 3 = Big Bad Wolf
13. $	Must be at EOL (End of Line)	

Remove

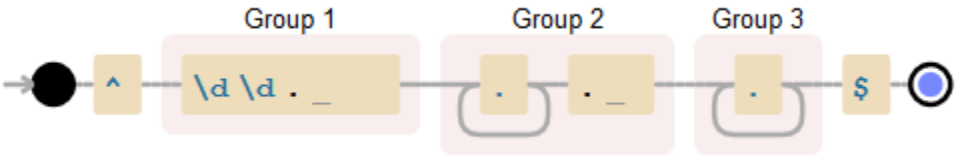
Remove text between two Dots cont.

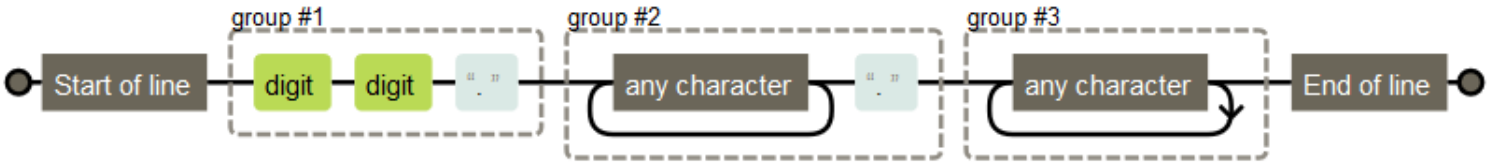
12. Mary Had a Little Lamb. Big Bad Wolf.mp3 to 12. Big Bad Wolf.mp3

With two leading numeric digits - more restrictive pattern <##><dot><text><dot>

Match: ^(\d\d\.)(.+?\.)(.+)$
Replace: \1\3

Name	New Name
12. Mary Had a Little Lamb. Big Bad Wolf.mp3	12. Big Bad Wolf.mp3





Sample:

```
.....................................................
Match 1: 12. Mary Had a Little Lamb. Big Bad Wolf
Group 1: 12.
Group 2: Mary Had a Little Lamb.
Group 3: Big Bad Wolf
```

Contributor Analysis:

1. Match first two numeric digits followed by a dot character.
2. Match all text till second dot.
3. Capture rest of string to EOL (End of Line).

Remove

Remove text between two Dots cont.

Using: 12. Mary Had a Little Lamb. Big Bad Wolf.mp3

With two leading numeric digits - more restrictive pattern <##><dot><text><dot> cont.

Match: ^(\d\d\.)(.+?\.)(.+)$

Where:

\1 = 12.<space> \2 = Mary Had a Little Lamb. <space> \3 = Big Bad Wolf

Tim Analysis:

1.	Start at beginning of string (BOL)	
2. \d\d	Match against 2 numeric digits	Capture Group 1 = 12
3. \.	Match against literal <dot> character	Capture Group 1 = 12.
4. <space>	Match against a <space> character	Capture Group 2 = 12.<space>
5..	Match against any character	Capture Group 2 = M
6. +	Make it Greedy	Capture Group 2 = Mary Had a Little Lamb. Big Bad Wolf
7. ?	But not too Greedy (Lazy)	Capture Group 2 = M
8 <space>	Match against a <space> character	Capture Group 2 = Mary Had a Little Lamb. <space>
9. \.	Match against literal <dot> character	Capture Group 3 = B
10. .+	Match against remaining string	Capture Group 3 = Big Bad Wolf
11. $	Must be at EOL (End of Line)	

Tim Notes:

1. Except for step 2, the analysis is the same. Step 2 restricts the match to only two numeric digits.

Remove

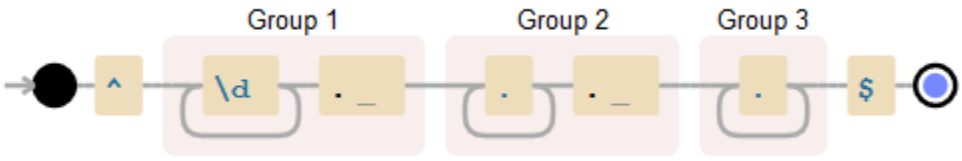
Remove text between two Dots cont.

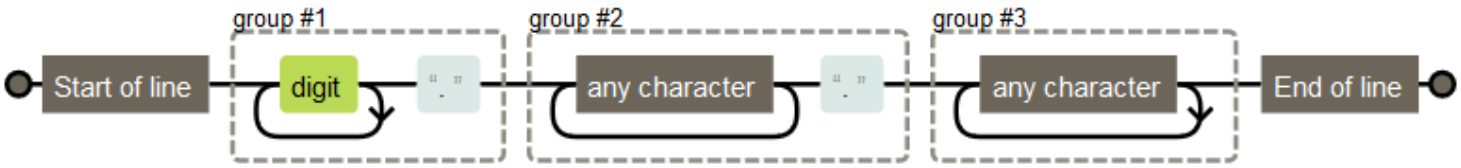
12. Mary Had a Little Lamb. Big Bad Wolf.mp3 to 12. Big Bad Wolf.mp3

Different amount of leading numeric digits - less restrictive pattern <any #><dot><text><dot>

Match: ^(\d+\.)(.+?\.)(.+)$
Replace: \1\3

Name	New Name
12. Mary Had a Little Lamb. Big Bad Wolf.mp3	12. Big Bad Wolf.mp3





Sample:

```
.....................................................
Match 1: 12. Mary Had a Little Lamb. Big Bad Wolf
Group 1: 12.
Group 2: Mary Had a Little Lamb.
Group 3: Big Bad Wolf
```

Forum Contributor: Stefan

Remove

Remove text between two Dots cont.

Using: 12. Mary Had a Little Lamb. Big Bad Wolf.mp3

Different amount of leading numeric digits - less restrictive pattern <any #><dot><text><dot>

Match: ^(\d+\.)(.+?\.)(.+)$

Where:

\1 = 12.<space> \2 = Mary Had a Little Lamb. <space> \3 = Big Bad Wolf

Tim Analysis:

1.	Start at beginning of string (BOL)	
2. \d	Match against a numeric digit	Capture Group 1 = 1
3. +	Make it Greedy	Capture Group 1 = 12
4. \.	Match against literal <dot> character	Capture Group 1 = 12.
5. <space>	Match against a <space> character	Capture Group 2 = 12.<space>
6..	Match against any character	Capture Group 2 = M
7. +	Make it Greedy	Capture Group 2 = Mary Had a Little Lamb. Big Bad Wolf
8. ?	But not too Greedy (Lazy)	Capture Group 2 = M
9 <space>	Match against a <space> character	Capture Group 2 = Mary Had a Little Lamb. <space>
10. \.	Match against literal <dot> character	Capture Group 3 = B
11. .+	Match against remaining string	Capture Group 3 = Big Bad Wolf
12. $	Must be at EOL (End of Line)	

Tim Notes:

1. Except for step 2 and step 3, the analysis is the same. The ' \d+' allows the RegEx to match against any number of numeric digits

Remove

Remove Duplicate Title From Multiple Music Filenames

Black Box - Everybody, Everybody - 01 - Everybody, Everybody (Love Unlimited edit)
to
Black Box - Everybody, Everybody (Love Unlimited edit)

Black Box - Everybody, Everybody - 02 - Everybody, Everybody (Le Freak mix)
to
Black Box - Everybody, Everybody (Le Freak mix)

Robin S - Show Me Love - 01 - Show Me Love (Montego Mix)	to	Robin S - Show Me Love (Montego Mix)
Robin S - Show Me Love - 02 - Show Me Love (Dub Mix)	to	Robin S - Show Me Love (Dub Mix)
Robin S - Show Me Love - 03 - Show Me Love (New York Mix)	to	Robin S - Show Me Love (New York Mix)
Jack B - Test to show Unchanged (Required for Pattern)	to	Ignore

Observations:

1. Match Repeating Duplicate Text
2. Ignore files already in correct format
3. Pattern established:
< -> <text> < -> <space> <numeric> <space> < -> <space> <open parenthesis> <text> <close parenthesis>

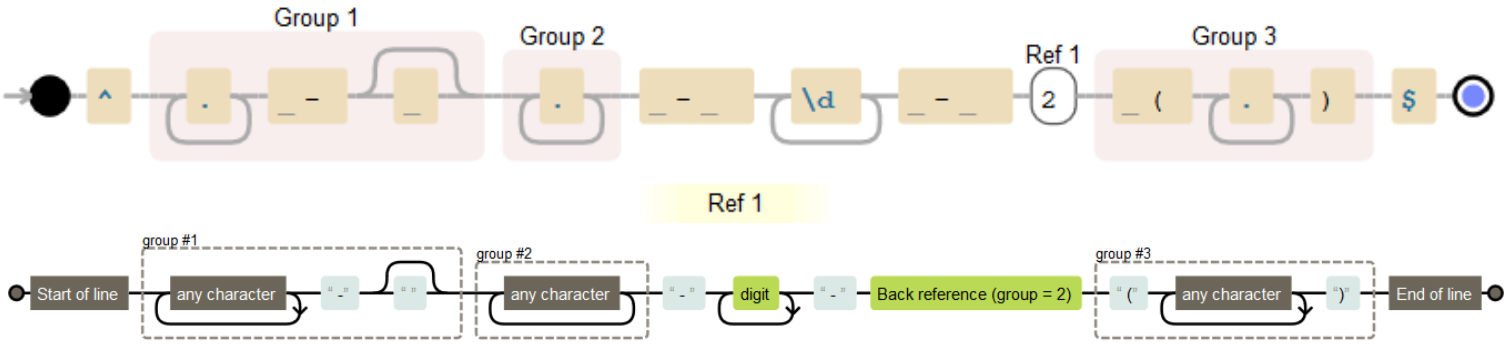
Match: ^(.+ - ?)(.+?) - \d+ - \2(\(.+\))$
Replace: \1\2\3

Name	New Name
Robin S - Show Me Love - 03 - Show Me Love (New York Mix)	Show Me Love
Robin S - Show Me Love - 02 - Show Me Love (Dub Mix)	Show Me Love
Robin S - Show Me Love - 01 - Show Me Love (Montego Mix)	Show Me Love
Black Box - Everybody, Everybody - 02 - Everybody, Everybody (Le Freak mix)	Everybody, Everybody
Black Box - Everybody, Everybody - 01 - Everybody, Everybody (Love Unlimited edit)	Everybody, Everybody
Jack B - Test to show Unchanged (Required for Pattern)	Jack B - Test to show Unchanged (Required for Pattern)

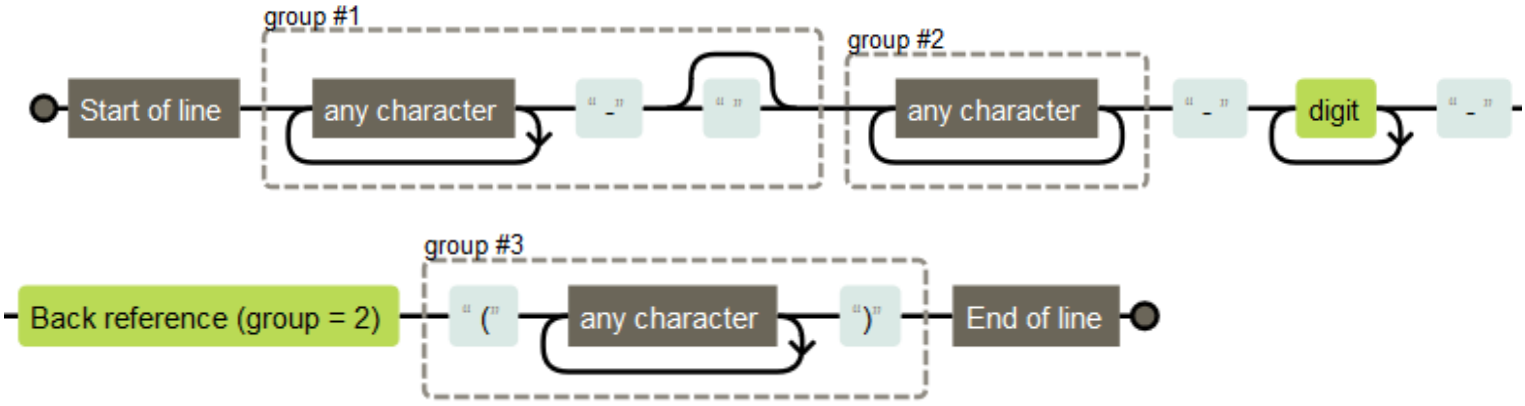
Remove

Remove Duplicate Title From Multiple Music Filenames cont.

Match: ^(.+ - ?)(.+?) - \d+ - \2(\(.+\))$



Expanded:



Sample:

```
.....................................................................
Match 1: Robin S - Show Me Love - 01 - Show Me Love (Montego Mix)
Group 1: Robin S -
Group 2: Show Me Love
Group 3:  (Montego Mix)
```

Contributor Notes:

1. It mandates - # - in between the repeating-text so it won't touch names already in the correct format.
   a. No need to worry about which files to select.

Forum Contributor: truth

Remove

Remove Duplicate Title From Multiple Music Filenames cont.

Using: Robin S - Show Me Love - 03 - Show Me Love (New York Mix)

Match: ^(.+ - ?)(.+?) - \d+ - \2(\(.+\))$

```
.....................................................................
Match 1: Robin S - Show Me Love - 03 - Show Me Love (New York Mix)
Group 1: Robin S -
Group 2: Show Me Love
Group 3:  (New York Mix)
```

Where:

\1 = Robin S - <space> \2 = Show Me Love \3 = <space> (New York Mix)

Match: ^(.+ - ?)(.+?) - \d+ - \2(\(.+\))$

Tim Analysis:

This is interesting because it demonstrates how a value captured in a *previous* Group can be utilized in the SAME expression (another example of using a back reference in the RegEx).

1. ^	Start at beginning (Beginning of Line or BOL)	
2. .	Match against any character	Capture Group 1 = R
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. <space>	Match against <space> Backtracking. Locates first <space> before 'York'	Capture Group 1 = Robin S - Show Me Love - 03 - Show Me Love<space>New York
5. <hyphen>	Match against <hyphen>	Capture Group 1 = Robin S - Show Me Love - 03 –
6. <space>	Match against <space>	Capture Group 1 = Robin S - Show Me Love - 03 –<space>
7. ?	Make Previous match of <space> optional	Capture Group 1 = Unchanged
8. .	Match against any character This is the 'S' of the second 'Show'.	Capture Group 2 = S
9. +	Make it Greedy	Capture Group 2 = Show Me Love (New York Mix)
10. ?	But not too Greedy	Capture Group 2 = S
11. <space>	Match against <space> (Not Captured) This is the <space> after the second 'Show. Capture Group 2 is the second 'Show' in string.	Changes Capture Group 2 = Show <space>
12. <hyphen>	Match against <hyphen> (Not Captured) Backtracking.	Changes Capture Group 2 = 03
13. <space>	Match against <space> (Not Captured)	

Remove

Remove Duplicate Title From Multiple Music Filenames cont.

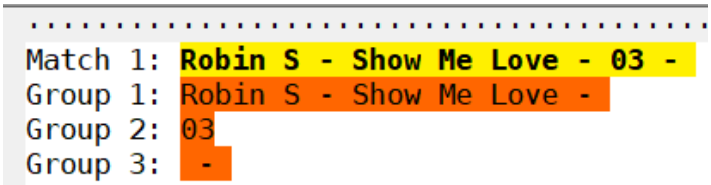
Using: Robin S - Show Me Love - 03 - Show Me Love (New York Mix)

Match: ^(.+ - ?)(.+?) - \d+ - \2(\(.+\))$

Where:

\1 = Robin S - <space> \2 = Show Me Love \3 = <space> (New York Mix)

This is what we have at this point:



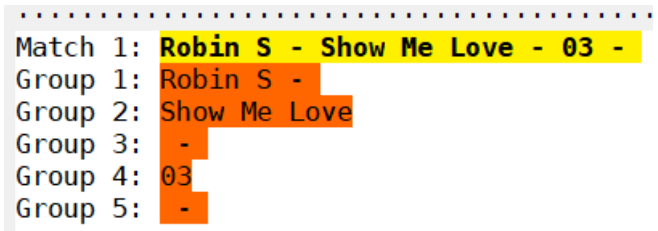
For clarification, I added a Capture Group to the <space> <hyphen> <space> following the (.+?) – see Group 3 in the photo to the left. It illustrates that the Group 3 capture are the literals following the '03'.
^(.+ - ?)(.+?)(-)

Tim Analysis: cont.

14. \d Match against numeric digit (Not Captured) = '0'. Backtracks to match the '0' of '03'.
15. + Make it Greedy = '03' (Not Captured) Capture Group 2 is now the first 'Show me Love'

Changes Capture Group 2 = Show Me Love
Changes Capture Group 1 = Robin S - <space>

Added extra Capture Groups for Illustration only.



^(.+ - ?)(.+?)(-)(\d+)(-)

The following is (Not Captured) -
16. <space> Match against <space> This is the <space> after '03'
17. <hyphen> Match against <hyphen>
18. <space> Match against <space>

Again for clarification only, I added a Capture Group around the <space> <hyphen> <space> following (\d+). This is portrayed as Group 5 in the photo to the right.

19. \2 Reference Backreference \2 Value. This is Capture Group 2' value of the first 'Show Me Love' in the string. (Not Captured)
20. <space> Match against <space> This is the <space> after 'Love'

Capture Group 3 = <space>

21. \(Match against right parenthesis
22. . Match against any character
23. + Make it Greedy
24. $ Must be at EOL

Capture Group 3 = <space> (
Capture Group 3 = <space>(N
Capture Group 3 = <space>(New York Mix)

Tim Notes:

1. The reference to backreference \2 in the RegEx is how the expression determines NOT to rename files that already have this format because if the duplicate text is NOT matched, the evaluation Fails and the filename won't be renamed.

Remove

Removing the Numeric string '_#####' from the middle of file names

123-ABD_98765(Type-A).pdf to 123-ABD (Type A).pdf

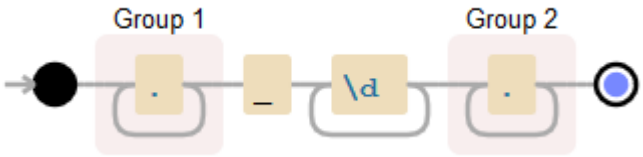
Observations:

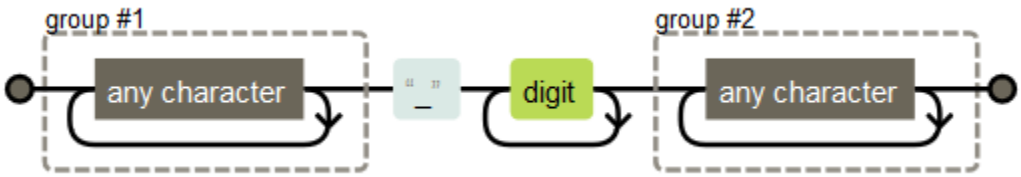
1. Files are currently in the form: <Number I want to keep>_<Useless ID Ref>(Real ID I need to keep).ext
2. File names are of random length, and the number of numeric digits in the _<numbers here> string is arbitrary, but right now, between 1 and 7 digits.

To remove 'all' consecutive-numeric digits following '_'

Match: (.+)_\d+(.+)
Replace: \1 \2

Name	New Name
123-ABC_98765(Type-A).p...	123-ABC (Type-A).pdf





Sample:

```
.....................................
Match 1: 123-ABC_98765(Type-A)
Group 1: 123-ABC
Group 2: (Type-A)
```

Remove

Removing the Numeric string '_#####' from the middle of file names cont.

Using: 123-ABC_98765(Type-A).pdf

To remove 'all' consecutive-numeric digits following '_' cont.

Match: (.+)_\d+(.+)

Where:

\1 = 123-ABC \2 = (Type-A)

Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. _	Match against underscore (Not Captured) Backtracks. This is the underscore after 'ABC'	Changes Capture Group 1 = 123-ABC
4. \d	Match against Numeric Digit (Not Captured) = '9'	Capture Group 1 = Unchanged
5. +	Make it Greedy (Not Captured) = '98765'	Capture Group 1 = Unchanged
6. .+	Match against remaining string	Capture Group 2 = (Type-A)

Remove

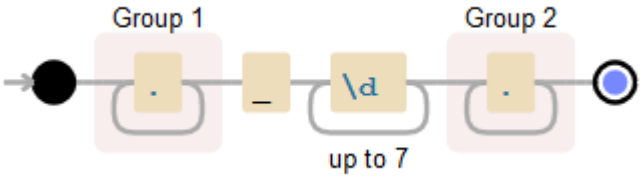
Removing the Numeric string '_#####' from the middle of file names cont.

123-ABC_98765(Type-A).pdf to 123-ABC (Type A).pdf

To remove a max of 7-numeric digits following '_'

Match: (.+)_\d{1,7}(.+)
Replace: \1 \2

Name	New Name
123-ABC_98765(Type-A).p...	123-ABC (Type-A).pdf



Sample:

```
................................
Match 1: 123-ABC_98765(Type-A)
Group 1: 123-ABC
Group 2: (Type-A)
```

Contributor Notes:

1. Both expressions remove the last occurrence of _## within OrigNames
2. Neither will remove _## from the very beginning/end of OrigName
3. If that functionality is needed, change .+ into .*

Tim Notes:

1. Uses 'Minimum, Maximum' Range Quantifier {##,##} previously discussed.

Forum Contributor: truth

Remove

Removing the Numeric string '_#####' from the middle of file names cont.

Using: 123-ABC_98765(Type-A).pdf

To remove a max of 7-numeric digits following '_' cont.

Match: (.+)_\d{1,7}(.+)

Where:

\1 = 123-ABC \2 = (Type-A)

Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. _	Match against underscore (Not Captured) Backtracks. This is the underscore after 'ABC'	Changes Capture Group 1 = 123-ABC
4. \d	Match against Numeric Digit (Not Captured) = '9'	Capture Group 1 = Unchanged
5. {1,7}	Repeat last expression for a minimum match of one iteration to a maximum match of 7 iterations (Not Captured) = '98765'	
6. .+	Match against remaining string	Capture Group 2 = (Type-A)

Remove

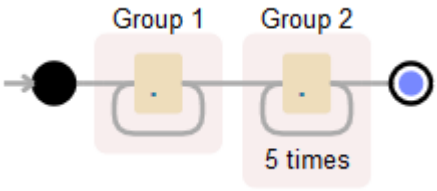
Simple Removes – Keep last 5 Characters of string, Keep first 6 Characters of string

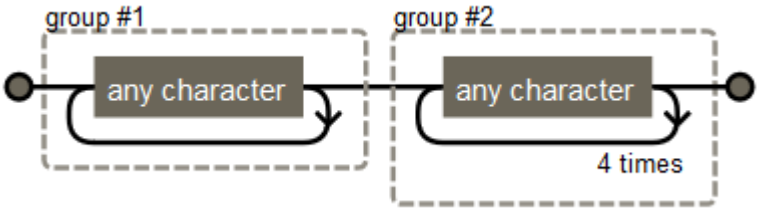
Keep Last -

asdfasd&asdf.a893-638 aaaaa_11111.ext	to	11111.ext
bbbbdfd sdf . asdf _ df 34 $ # bbbbbb_22222.ext	to	22222.ext
cc_3asd ed3 345455 $&_33333.ext	to	33333.ext

Match: (.+)(.{5})
Replace: \2

Name	New Name
bbbbdfd sdf . asdf _ df 34 $ # bbbbbb_22222.ext	22222.ext
asdfasd&asdf.a893-638 aaaaa_11111.ext	11111.ext
cc_3asd ed3 345455 $&_33333.ext	33333.ext





Sample:

```
.............................................
Match 1: asdfasd&asdf.a893-638 aaaaa_11111
Group 1: asdfasd&asdf.a893-638 aaaaa_
Group 2: 11111
```

Remove

Simple Removes – Keep last 5 Characters of string, Keep first 6 Characters of string cont.

Keep Last – cont.

Using: asdfasd&asdf.a893-638 aaaaa_11111.ext

Match: (.+)(.{5})

Where:

\1 = asdfasd&asdf.a893-638 aaaaa_ \2 = 11111

Tim Analysis:

1..	Match against any character	Capture Group 1 = a
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. .	Match against any character	Capture Group 2 = 1 Changes Capture Group 1 = asdfasd&asdf.a893-638 aaaaa_1111
4. {5}	Repeat last expression to match 5 iterations.	Capture Group 2 = 11111 Changes Capture Group 1 = asdfasd&asdf.a893-638 aaaaa_

Remove

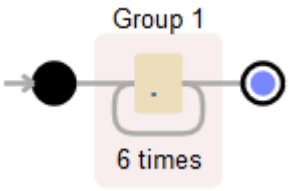
Simple Removes – Keep last 5 Characters of string, Keep first 6 Characters of string cont.

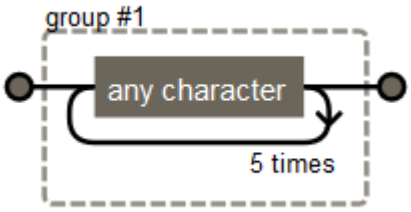
Keep First –

BBD302a 24x36.plt to BBD302.plt

Match: (.{6})
Replace: \1

Name	New Name
BBD302a 24x36.plt	BBD302.plt





Sample:

```
.................
Match 1: BBD302
 Group 1: BBD302
Match 2: a 24x3
 Group 1: a 24x3
```

Forum Contributor: Stefan

Where:

\1 = BBD302

Tim Analysis:

1. .	Match against any character	Capture Group 1 = B
2. {6}	Repeat last expression to match 6 iterations.	Capture Group 2 = BBD302

Remove

Remove all Except last Numeric Digits greater than zero

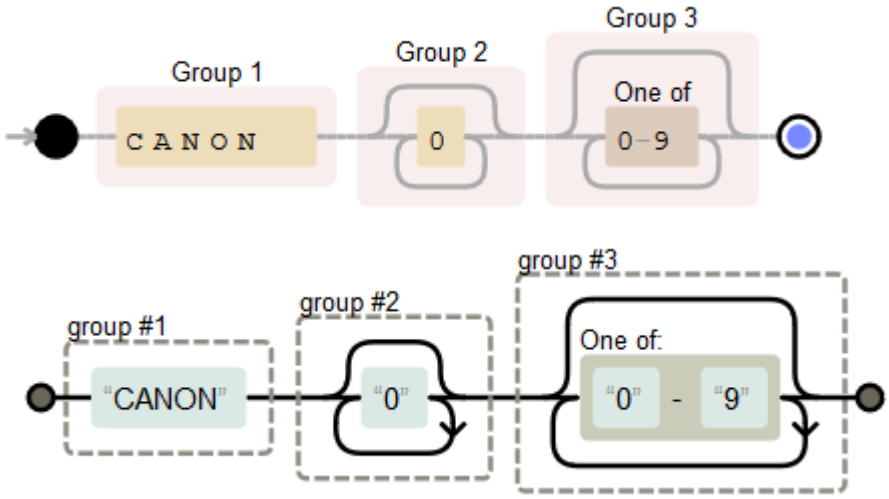
CANON0002	to	2
CANON0009	to	9
CANON0031	to	31

Observations:

1. Remove all text and zero numeric digits
2. Pattern is text immediately followed by Zeros ending in non-zero numeric

Match: (CANON)(0*)([0-9]*)
Replace: \3

Name	New Name
CANON0031	31
CANON0009	9
CANON0002	2



Sample:

```
....................
Match 1: CANON0002
Group 1: CANON
Group 2: 000
Group 3: 2
```

Sample:

```
....................
Match 1: CANON0031
Group 1: CANON
Group 2: 00
Group 3: 31
```

Forum Contributor: jimwillsher (Original Publisher BRU)

Remove

Remove all Except last Numeric Digits greater than zero cont.

Using: CANON0002

Match: (CANON)(0*)([0-9]*)

Where:

\1 = CANON \2 = 000 \3 = 2

Tim Analysis:

1. CANON	Match against string, 'CANON'	Capture Group 1 = CANON
2. 0	Match against literal '0'	Capture Group 2 = 0
3. *	Make it Greedy	Capture Group 2 = 000
4. [0-9]	Match against a class consisting of a numeric digit 0-9.	Capture Group 3 = 2
5. *	Make it Greedy This will capture any additional numeric digits at the end of the string, i.e., CANON0031 = 31	Capture Groups = Unchanged

Final Observations:

1. Limited and restrictive. Only matches 'CANON'
2. Specifically matches Zeros.
3. Does parse completely
   a. Isolates 'CANON'
   b. Isolates Zeros
   c. Isolates numbers greater than zero

Remove

Remove all Except last Numeric Digits greater than zero cont.

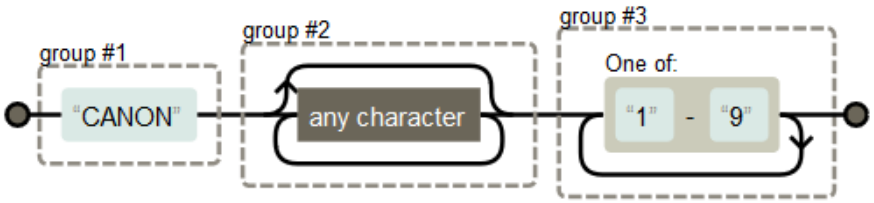
Tim Contribution –

Using: CANON0031

Example #1 - Remove zero dependency:

Example #1 - Match: (CANON)(.*?)([1-9]+)
Replace: \3

Name	New Name
CANON0031	31
CANON0009	9
CANON0002	2

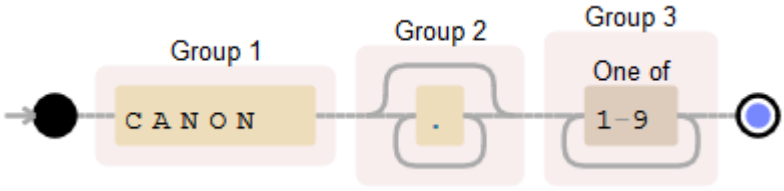


Sample:

```
Match 1: CANON0031
Group 1: CANON
Group 2: 00
Group 3: 31
```

Where:

\1 = CANON \2 = 00 \3 = 31



Tim Analysis:

1. CANON	Match against string, 'CANON'	Capture Group 1 = CANON
2. .	Match against any character	Capture Group 2 = 0
3. *	Make it Greedy Current position = EOL	Capture Group 2 = 0031
4. ?	But not too Greedy	Capture Group 2 = BOL
5. [1-9]	Match against a class consisting of a numeric digit 1-9.	Capture Group 3 = 3 Changes Capture Group 2 = 00
5. +	Make it Greedy This will capture any additional numeric digits at the end of the string, i.e., CANON0031 = 31	Capture Group 3 = 31

Tim Notes:

1. This removes the dependence on the Zeros, using, ' (.*?) ' instead of ' (0*) '
2. Same Final Observations as before otherwise.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

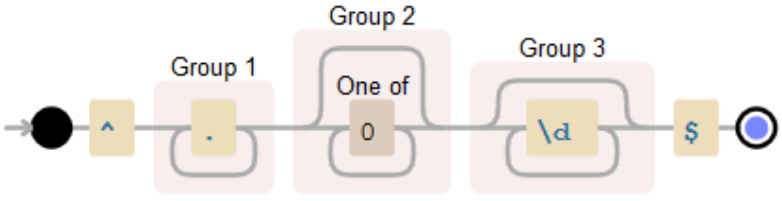
Larger sample group -

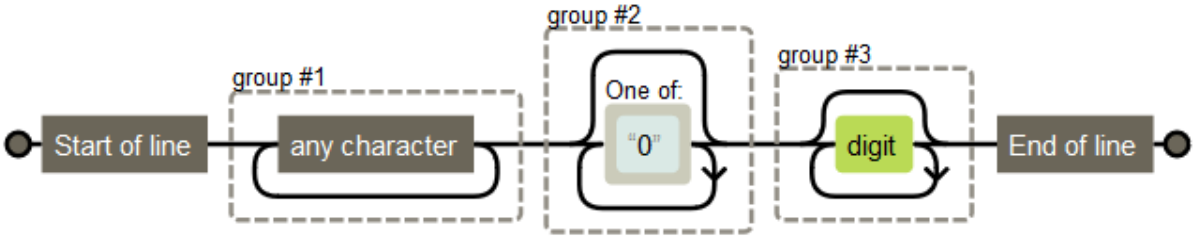
CANON0009	Lesson 1	BRC32.exe	Test056430
CANON0031	The Rain in Spain00010	TEST0001456	a1
CANON002500	LookingGood 126355	Test0456067800	

Example #2 - Match numeric digits other than zero using \d:

Match: ^(.+?)([0]*)(\d*)$
Replace: \3

Name	New Name
CANON0009	9
CANON0031	31
CANON002500	2500
Lesson 0	Lesson 0
Lesson 1	1
Lesson 1 DNA	Lesson 1 DNA
Lesson 25 DNA	Lesson 25 DNA
The Rain in Spain00010	10
LookingGood 126355	126355
BRC32.exe	32.exe
TEST0001456	1456
Test0456067800	456067800
Test056430	56430
a1	1
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA





Tim Notes:

1. Won't match against a numeric that only has a zero – 'Lesson 0' won't work (see Example #3)
2. Will not work with strings that have text following after numeric digits (see Example #3).

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Example #2 - Match numeric digits other than zero using \d: cont.

Match: ^(.+?)([0]*)(\d*)$

Sample:

```
·····················Start·L
Match 1: CANON0009       0
Group 1: CANON           0
Group 2: 000             5
Group 3: 9               8
```

Sample:

```
·····························
Match 1: The Rain in Spain00010
Group 1: The Rain in Spain
Group 2: 000
Group 3: 10
```

Sample:

```
···················Start·Length···
Match 1: Lesson 1      0       8
Group 1: Lesson        0       7
Group 2:               7       0
Group 3: 1             7       1
```

Sample:

```
·············Start·Length···
Match 1: a1      0       2
Group 1: a       0       1
Group 2:         1       0
Group 3: 1       1       1
```

Tim Notes:

1. If there are no leading Zeros, then a Zero Occurrence Match will capture a null value for Capture Group 2.

Using: The Rain in Spain00010

Where:

\1 = The Rain in Spain \2 = 000 \3 = 10

See next page for full analysis.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Using: The Rain in Spain00010

Example #2 - Match numeric digits other than zero using \d: cont.

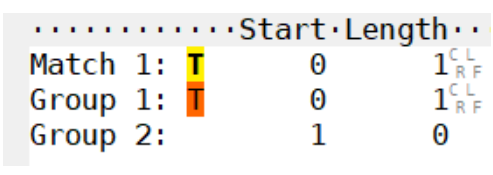
Match: ^(.+?)([0]*)(\d*)$

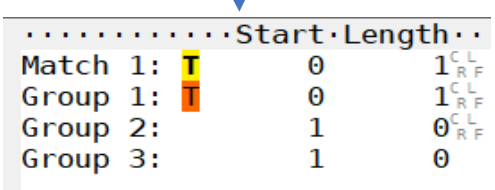
Tim Analysis:

1. ^	Start at beginning (Beginning of Line or BOL)	
2. .	Match against any character	Capture Group 1 = T
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = T
5. [0]	Match against a class consisting of a zero This is the first zero following 'Spain'	Capture Group 2 = 0 Changes Capture Group 1 = The Rain in Spain
6. *	Make it Greedy	Capture Group 2 = null Changes Capture Group 1 = T
7. \d	Match against numeric digit	Capture Group 3 = 1 Changes Capture Group 2 = 000 Changes Capture Group 1 = The Rain in Spain
8. *	Make it Greedy	Changes Capture Group 3 = null Changes Capture Group 2 = null Changes Capture Group 1 = T
9. $	Last Match has to be from End of String Backtracks (End of Line or EOL)	Changes Capture Group 3 = 10 Changes Capture Group 2 = 000 Changes Capture Group 1 = The Rain in Spain

Unfortunately, this is one of those where my analysis doesn't quite fit. Why is Capture Group 2 made null in step 6? The obvious explanation is when Group 1's value changed to 'T' from 'The Rain in Spain' in step 6 because it backtracked, it wiped out the value in Capture Group 2.

Also other questions are why did Capture Groups 3 and 2 null out in step 8? Same reason, it backtracked wiping out the values for Capture Groups 3 and 2 when it changed Capture Group 1 back to 'T'. Generally, a null value - not an Undefined value, is the result of a Zero Occurrence Match because of an action – typically backtracking, that causes the Capture Group to 'give up' (reevaluate as I call it) all of its characters.





This is an example where taking the RegEx apart piece by piece doesn't quite tell the entire story. Sometimes an analysis has to be done by looking at the sections of an expression rather than each individual piece.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Using: The Rain in Spain00010

Example #2 - Match numeric digits other than zero using \d: cont.

Match: ^(.+?)([0]*)(\d*)$

Where:

\1 = The Rain in Spain \2 = 000 \3 = 10

Tim Summary Analysis:

1. Each character is tested against the expression—

testing is performed by:

a. character (.+?) → tested against [0] = false → $

b. test next character (.+?)
c. Continue until …
d. [0] = '0'
e. [0] * = '000'
f. \d = '1'
g. \d* = '10'

Capture Group 1 = The Rain in Spain

Capture Group 2 = 000

Capture Group 3 = 10

Final Observations:

1. Matches any preceding text before Zeros.
2. Specifically matches Zeros.
3. Incomplete parsing of larger sample groups
   a. Isolates preceding text
   b. Isolates Zeros
   c. Isolates numbers greater than zero
   d. Does NOT handle strings that have following text after numeric digits (See Example #3)
   e. Does NOT handle strings that start with zero (See Example #3)

Tim Notes:

1. Teaser - All of these issues will be addressed in the Final RegEx – Example #5, but we are not there yet.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Larger Sampling

BRC32.exe	CANON002500	05	Lesson 1 DNA
a1	CANON0031	Lesson 1	Chapter1 Lesson2 DNA
CANON0009	Test0456067800	Lesson 25 DNA	
TEST0001456	Test056430	Lesson 0	

Tim Contribution: cont.

Example #3 - Match any text preceding or following numeric digits:

Observations:

1. Pattern is now: text followed by numeric digits.
2. Leading Zeros can be optionally included
3. Will match files that have text after the numeric digits.

Separate the Numeric from text: Universal – not including leading Zeros – 'Lesson 0' still won't work

Match: ^(.*?)([A-Za-z\s]*)([0]*)(.*)$
Replace: \4

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	9
TEST0001456	1456
CANON002500	2500
CANON0031	31
Test0456067800	456067800
Test056430	56430
05	5
Lesson 1	1
Lesson 25 DNA	25 DNA
Lesson 0	Lesson 0
Lesson 1 DNA	1 DNA
Chapter1 Lesson2 DNA	1 Lesson2 DNA

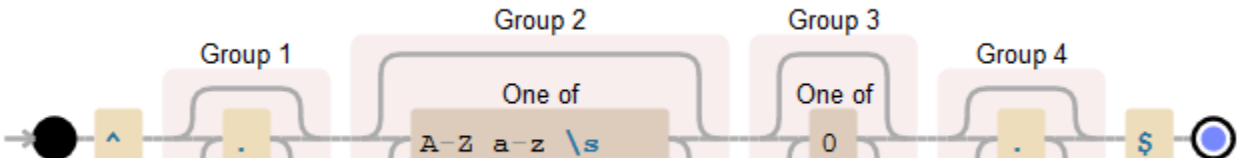
Remove

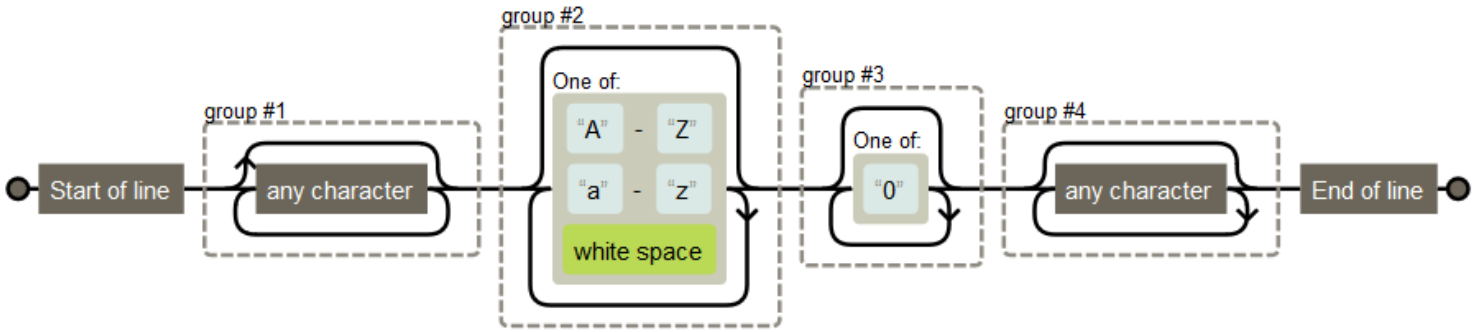
Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Example #3 - Match any text preceding or following numeric digits: cont.

Match: ^(.*?)([A-Za-z\s]*)([0]*)(.*)$





Sample:

Match 1:	CANON0031	0	9
Group 1:		0	0
Group 2:	CANON	0	5
Group 3:	00	5	2
Group 4:	31	7	2

Sample:

```
Match 1: Test0456067800
Group 1:
Group 2: Test
Group 3: 0
Group 4: 456067800
```

Sample:

Match 1:	Lesson 1	0	8
Group 1:		0	0
Group 2:	Lesson	0	7
Group 3:		7	0
Group 4:	1	7	1

Sample:

Match 1:	Lesson 25 DNA	0	13
Group 1:		0	0
Group 2:	Lesson	0	7
Group 3:		7	0
Group 4:	25 DNA	7	6

Tim Notes:

1. In all cases, Capture Group 1 holds a null value.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Example #3 - Match any text preceding or following numeric digits: cont.

Match: ^(.*?)([A-Za-z\s]*)([0]*)(.*)$

Tim Notes:

1. Will not work with strings that begin with a numeric because, after all, it is meant to separate numeric from text presuming that numeric falls after text, not before it.

To clarify, It will match, but no change in the name, accordingly, no New Name:

i.e.,

01-Devolv Whip it.wav
using Replace String with <space> changes the file name to allow New Name -

\3\4 <space>

Name	New Name
01-Devolv Whip it.wav	01-Devolv Whip it .wav

To include leading Zeros, use Replacement String (but see Tim Note #1 above still applies):

Replace: \3\4

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	0009
TEST0001456	0001456
CANON002500	002500
CANON0031	0031
Test0456067800	0456067800
Test056430	056430
05	05
Lesson 1	1
Lesson 25 DNA	25 DNA
Lesson 0	0
Lesson 1 DNA	1 DNA
Chapter1 Lesson2 DNA	1 Lesson2 DNA

Tim Notes:

1. Sample '05' is not included. It requires no change since it contains no preceding text.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Using: The Rain in Spain00010

Example #3 - Match any text preceding or following numeric digits: cont.

Match: ^(.*?)([A-Za-z\s]*)([0]*)(.*)$

Where:

\1 = null \2 = The Rain in Spain \3 = 000 \4 = 10

Tim Analysis:

1. ^	Start at beginning (Beginning of Line or BOL)	
2. .	Match against any character	Capture Group 1 = T
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = T
5. [A-Za-z\s]	Match against a class consisting of an uppercase or lowercase letter or a <space> The <space> takes care of those strings that have a <space> between the end of the text and the beginning of the numeric digits.	Capture Group 2 = T Changes Group 1 = null
6. *	Make it Greedy	Capture Group 2 = The Rain in Spain
7. [0]	Match against a class consisting of a zero This is the first zero following 'Spain'	Capture Group 3 = 0
8. *	Make it Greedy	Capture Group 3 = 000 Capture Group 2 = Unchanged
9. .	Match against any character	Capture Group 4 = 1 Capture Group 3 = Unchanged Capture Group 2 = Unchanged Capture Group 1 = Unchanged
10. *	Make it Greedy	Changes Capture Group 4 = 10 All other Groups remain Unchanged
11. $	Last match has to be from end of string (End of Line or EOL)	

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution: cont.

Example #3 - Match any text preceding or following numeric digits: cont.

Larger sample group -

CANON0009	Lesson 1	BRC32.exe	
CANON0031	The Rain in Spain00010	TEST0001456	a1
CANON002500	LookingGood 126355	Test0456067800	

Match: ^(.*?)([A-Za-z\s]*)([0]*)(.*)$

Where:

\1 = null \2 = <preceding text> \3 = <Zeros>
\4 = <numbers greater the zero with any following text>

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	9
TEST0001456	1456
CANON002500	2500
CANON0031	31
Test0456067800	456067800
Test056430	56430
05	5
Lesson 1	1
Lesson 25 DNA	25 DNA
Lesson 0	Lesson 0
Lesson 1 DNA	1 DNA
Chapter1 Lesson2 DNA	1 Lesson2 DNA

To display only the Numeric Digits Greater than zero
Replace: \4

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	0009
TEST0001456	0001456
CANON002500	002500
CANON0031	0031
Test0456067800	0456067800
Test056430	056430
05	05
Lesson 1	1
Lesson 25 DNA	25 DNA
Lesson 0	0
Lesson 1 DNA	1 DNA
Chapter1 Lesson2 DNA	1 Lesson2 DNA

To display those files that include Zeros
Replace: \3\4

Final Observations:

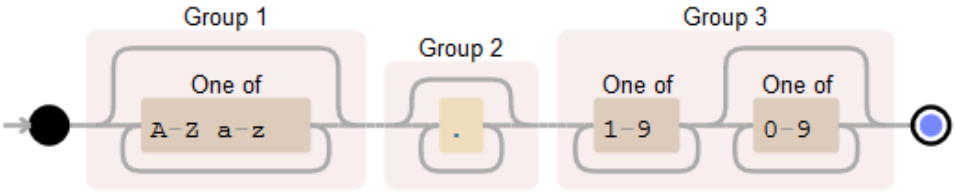
1. Matches any preceding text before Zeros.
2. Specifically matches Zeros.
3. Incomplete parsing of larger sample groups
   a. Isolates preceding text (Capture Group 2)
   b. Isolates Zeros (Capture Group 3)
   c. Does NOT completely parse numbers greater than zero (Included in Capture Group 4)
      1. Includes any following text
   d. Handles strings that have following text after numeric digits (included in Capture Group 4)
   e. Does handle strings that start with zero
   f. Intentionally does NOT handle strings that contain only a numeric (goes against its purpose)

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

CANON0031	to	31
CANON0009	to	9
CANON0002	to	2



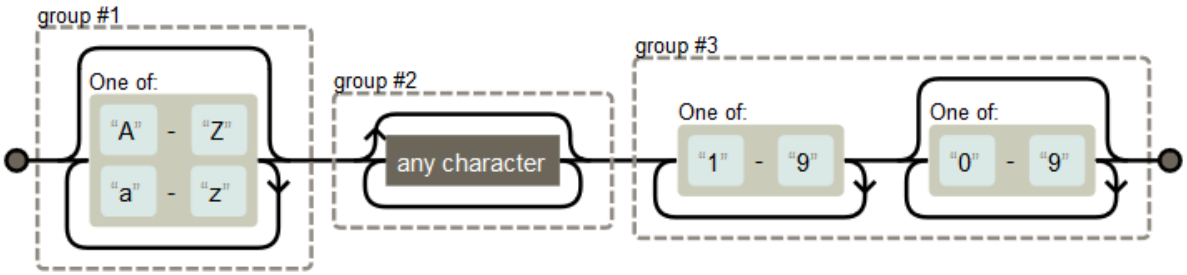
Example #4 - No dependency on Zeros:

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)
Replace: \3

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	9
TEST0001456	1456
CANON002500	2500
CANON0031	31
Test0456067800	456067800
Test056430	56430
05	5
Lesson 1	1
Lesson 25 DNA	25
Lesson 0	Lesson 0
Lesson 1 DNA	1
CANON0002	2
Chapter1 Lesson2 DNA	1
LookingGood 126355	126355
The Rain in Spain00010	10

Sample:

```
....................
Match 1: CANON0031
Group 1: CANON
Group 2: 00
Group 3: 31
```



Tim Notes:

1. To include following text after the numeric digits add this at the end - (.*)$ - see Example #4a

   Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)(.*)$

2. Modeled after original Contributor's RegEx using 'CANON' as text. Restricted as a result to capturing the first word of string rather than all text leading up to numeric digits.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

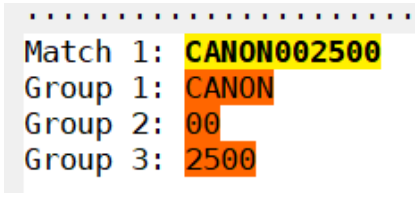
Expanded Sample Group:

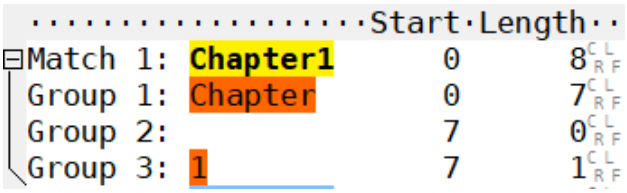
CANON0031	a1	Lesson 1	TEST0001456
CANON0009	CANON002500	Lesson 1 DNA	Test0456067800
CANON0002	Chapter1 Lesson2 DNA	Lesson 25 DNA	Test056430
05	Lesson 0	LookingGood 126355	The Rain in Spain00010

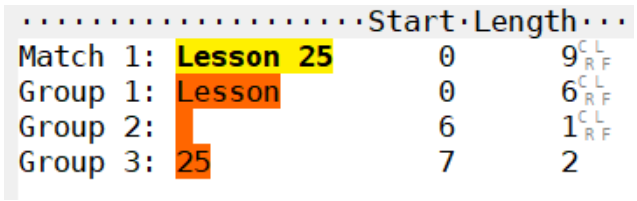
Example #4 - No dependency on Zeros: cont.

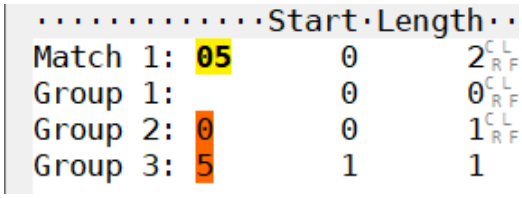
Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)

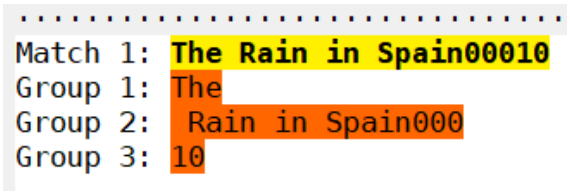
Samples:

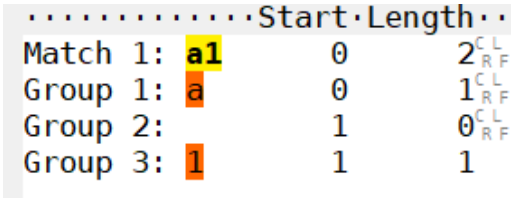


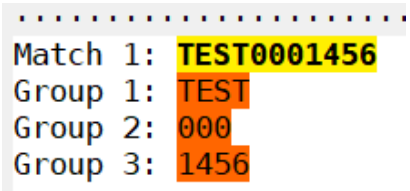


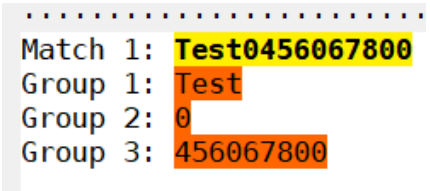












Tim Notes:

1. Lesson 0 - ignored – no numeric digits greater than zero
2. Zeros captured in Capture Group 2. If No Zeros, a Zero Occurrence Match takes place resulting in a null value. Unfortunately, if the preceding text contains more than a single word, this too is captured in Capture Group 2.
3. Preceding word is captured in Capture Group 1. If none, a Zero Occurrence Match takes place resulting in a null value.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: CANON0031

Example #4 - No dependency on Zeros: cont.

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)

Where:

\1 = CANON \2 = 00 \3 = 31

Tim Analysis:

1. [A-Za-z]	Match against a class consisting of an uppercase or lowercase letter	Capture Group 1 = C
2. *	Make it Greedy	Capture Group 1 = CANON
3. .	Match against any character	Capture Group 2 = 0
4. *	Make it Greedy Current position = EOL	Capture Group 2 = 0031
5. ?	But not too Greedy	Capture Group 2 = BOL
6. [1-9]	Match against a class consisting of a numeric digit 1-9.	Capture Group 3 = 3 Changes Capture Group 2 = 00

Tim Notes:

1. If there were no preceding Zeros, a Zero Match Occurrence would take place resulting in a null value for Capture Group 2. This does not apply if there is more text that follows after the initial word is captured in Capture Group 1 in which case, the additional text is captured as Capture Group 2. See Samples.

7. +	Make it Greedy This will capture any additional numeric digits at the end of the string, i.e., CANON0031 = 31	Capture Group 3 = 31

Tim Notes:

1. This removes the dependence on the string, 'CANON', but only captures the first word ([A-Za-z]*)

2. This is fine for samples like CANON0031, but to expand to a larger sample group, the ([0-9]*) was added. The problem was that it could not handle any further Zeros that appear. So numbers like 4500, or 12307805 were evaluated as 45 and 123 respectively. This addition takes care of that. See Example #4a.

Remove

Remove all Except last Numeric Digits greater than zero cont.

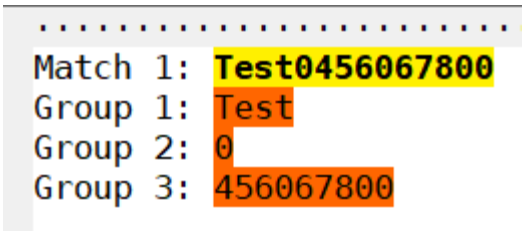
Tim Contribution – cont.

Using: Test0456067800

Example #4 - No dependency on Zeros: cont.

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)

To illustrate this, I will change the sample.



Using: Test0456067800

Where (thus far):

\1 = Test \2 = 0 \3 = 45

Tim Analysis: cont.

8. [0-9]	Match against a class consisting of a numeric digit 0-9.	Capture Group 3 = 456
9. *	Make it Greedy. This captures any additional numeric digits containing Zeros or not.	Capture Group 3 = 456067800

Final Observations:

1. Limited - Will match any first word of text.
2. No dependence on Zeros.
3. Incomplete parsing of larger sample base (Only handles pattern <text><Zeros><numeric digits>)
   a. Isolates first word of any preceding text only (Capture Group 1)
      1. Additional text is captured in Capture Group 2.
   b. Isolates Zeros (Capture Group 2)
   c. Isolates numbers greater than zero (Capture Group 3)
   d. Does NOT capture any text following

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

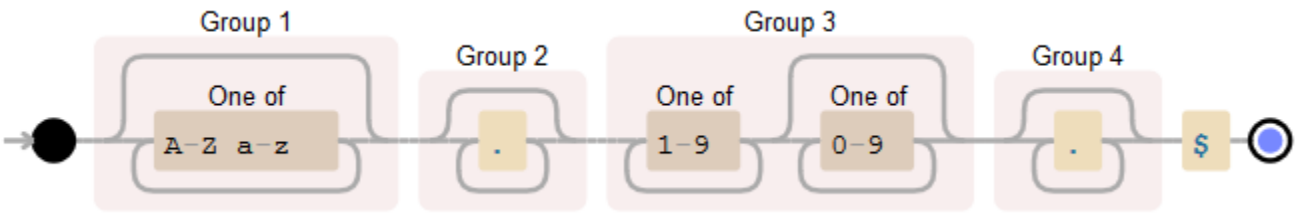
CANON0031	a1	Lesson 1	TEST0001456
CANON0009	CANON002500	Lesson 1 DNA	Test0456067800
CANON0002	Chapter1 Lesson2 DNA	Lesson 25 DNA	Test056430
05	Lesson 0	LookingGood 126355	The Rain in Spain00010

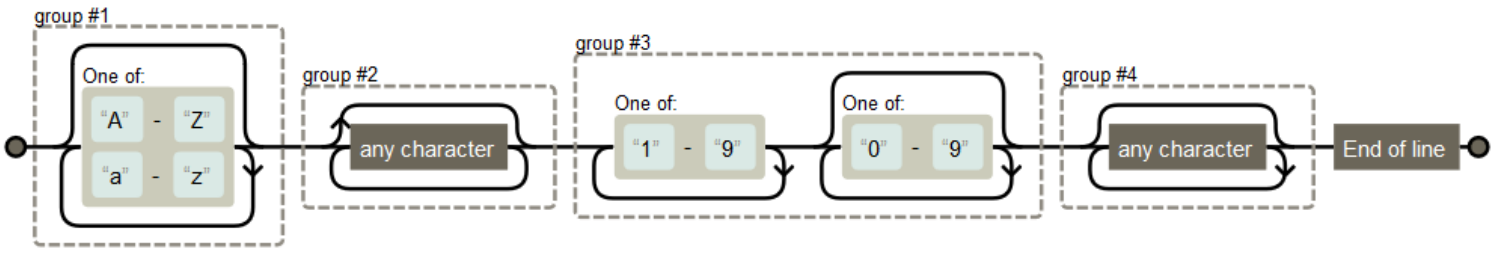
Example #4a- No dependency on Zeros and include following text:

To include following text after the numeric digits add this at the end - (.*)$

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)(.*)$
Replace: \3

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	9
TEST0001456	1456
CANON002500	2500
CANON0031	31
Test0456067800	456067800
Test056430	56430
05	5
Lesson 1	1
Lesson 25 DNA	25
Lesson 0	Lesson 0
Lesson 1 DNA	1
CANON0002	2
Chapter1 Lesson2 DNA	1
LookingGood 126355	126355
The Rain in Spain00010	10





Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

CANON0031	a1	Lesson 1	TEST0001456
CANON0009	CANON002500	Lesson 1 DNA	Test0456067800
CANON0002	Chapter1 Lesson2 DNA	Lesson 25 DNA	Test056430
05	Lesson 0	LookingGood 126355	The Rain in Spain00010

Example #4a- No dependency on Zeros and include following text: cont.

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)(.*)$

Samples:

	Start	Length
Match 1: **Lesson 25 DNA**	0	13
Group 1: Lesson	0	6
Group 2:	6	1
Group 3: 25	7	2
Group 4: DNA	9	4

	Start	Length
Match 1: **CANON002500**	0	11
Group 1: CANON	0	5
Group 2: 00	5	2
Group 3: 2500	7	4
Group 4:	11	0

	Start	Length
Match 1: **Chapter1 Lesson2 DNA**	0	20
Group 1: Chapter	0	7
Group 2:	7	0
Group 3: 1	7	1
Group 4: Lesson2 DNA	8	12

	Start	Length
Match 1: **a1**	0	2
Group 1: a	0	1
Group 2:	1	0
Group 3: 1	1	1
Group 4:	2	0

	Start	Length
Match 1: **LookingGood 126355**	0	18
Group 1: LookingGood	0	11
Group 2:	11	1
Group 3: 126355	12	6
Group 4:	18	0

	Start	Length
Match 1: **The Rain in Spain00010**	0	22
Group 1: The	0	3
Group 2: Rain in Spain000	3	17
Group 3: 10	20	2
Group 4:	22	0

Observations:

1. If there is only 1 preceding word, then a Zero Occurrence Match occurs with Capture Group 2
2. If there are additional preceding words, this is also captured in Capture Group 2.
3. Capture Group 4 is used to hold the value of any following text. If none, a Zero Occurrence Match takes place.

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: Lesson 25 DNA

Example #4a- No dependency on Zeros and include following text: cont.

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)(.*)$

Where:

\1 = Lesson \2 = <space> \3 = 25 \4 = <space> DNA

Tim Analysis:

1. [A-Za-z]	Match against a class consisting of an uppercase or Lowercase letter	Capture Group 1 = L
2. *	Make it Greedy	Capture Group 1 = Lesson
3. .	Match against any character This is the <space> after 'Lesson'	Capture Group 2 = <space>
4. *	Make it Greedy Current position = EOL	Capture Group 2 = <space>25 DNA
5. ?	But not too Greedy Current position = just after 'Lesson'	Capture Group 2 =null
6. [1-9]	Match against a class consisting of a numeric digit 1-9.	Capture Group 3 = 2 Changes Capture Group 2 = <space>
7. +	Make it Greedy This will capture any additional numeric digits at the end of the string except any Zeros.	Capture Group 3 = 25

Tim Notes:

1. Up to this point, the RegEx could not handle any further Zeros that appear. So numbers like 4500, or 12307805 were evaluated as 45 and 123 respectively. The following addition takes care of this

8. [0-9]	Match against a class consisting of numeric digit 0-9.	Capture Group 3 = Unchanged
9. *	Make it Greedy. This captures any additional numeric digits containing Zeros or not.	Capture Group 3 = Unchanged
10. .*	Match against remaining string This matches against any following text.	Capture Group 4 = <space>DNA
11. $	Must match at EOL	

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: Lesson 25 DNA

Example #4a- No dependency on Zeros and include following text: cont.

Match: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)(.*)$

Where:

\1 = Lesson \2 = <space> \3 = 25 \4 = <space> DNA

Final Observations:

1. Limited - Will match any first word of text.
2. No dependence on Zeros.
3. Incomplete parsing of larger sample base
   a. Separates first word of any preceding text only (Capture Group 1)
      1. Additional text and leading Zeros are captured in Capture Group 2.
   b. Does NOT isolate Zeros if there is additional text following the first word (Capture Group 2)
   c. Isolates numbers greater than zero (Capture Group 3)
   d. Does capture any text following (Capture Group 4)

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

CANON0031	a1	Lesson 1	TEST0001456
CANON0009	CANON002500	Lesson 1 DNA	Test0456067800
CANON0002	Chapter1 Lesson2 DNA	Lesson 25 DNA	Test056430
05	Lesson 0	LookingGood 126355	The Rain in Spain00010

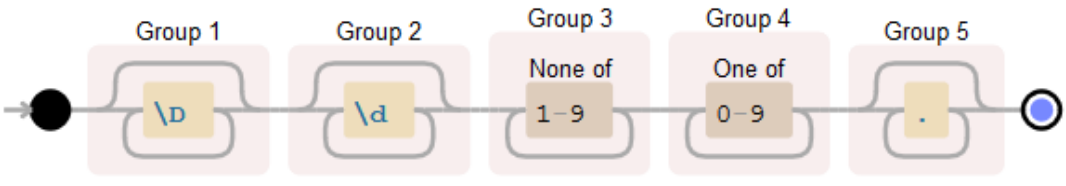
Example #5 – Using a Negated Class to isolate Zeros:

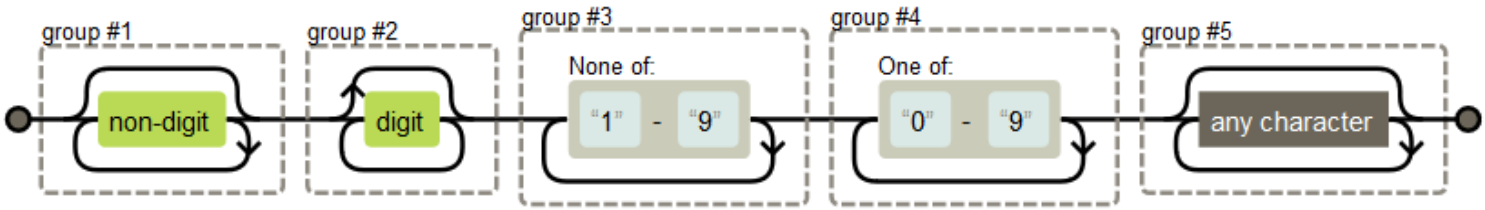
Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)
Replace: \4

Name	New Name
BRC32.exe	32.exe
a1	1
CANON0009	9
TEST0001456	1456
CANON002500	2500
CANON0031	31
Test0456067800	456067800
Test056430	56430
05	5
Lesson 1	1
Lesson 25 DNA	25
Lesson 0	0
Lesson 1 DNA	1
CANON0002	2
Chapter1 Lesson2 DNA	2
LookingGood 126355	126355
The Rain in Spain00010	10

Observations:

1. Matches all samples.





Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

CANON0031	a1	Lesson 1	TEST0001456
CANON0009	CANON002500	Lesson 1 DNA	Test0456067800
CANON0002	Chapter1 Lesson2 DNA	Lesson 25 DNA	Test056430
05	Lesson 0	LookingGood 126355	The Rain in Spain00010

Example #5 – Using a Negated Class to isolate Zeros: cont.

Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)

Samples:

	Start	Length
Match 1: **The Rain in Spain00010**	0	22
Group 1: The Rain in Spain	0	17
Group 2:	17	0
Group 3: 000	17	3
Group 4: 10	20	2
Group 5:	22	0

	Start	Length
Match 1: **CANON0031**	0	9
Group 1: CANON	0	5
Group 2:	5	0
Group 3: 00	5	2
Group 4: 31	7	2
Group 5:	9	0

	Start	Length
Match 1: **LookingGood 126355**	0	18
Group 1: LookingGood	0	11
Group 2:	11	0
Group 3:	11	1
Group 4: 126355	12	6
Group 5:	18	0

	Start	Length
Match 1: **Lesson 25 DNA**	0	13
Group 1: Lesson	0	6
Group 2:	6	0
Group 3:	6	1
Group 4: 25	7	2
Group 5: DNA	9	4

	Start	Length
Match 1: **Chapter1 Lesson2 DNA**	0	20
Group 1: Chapter	0	7
Group 2: 1	7	1
Group 3: Lesson	8	7
Group 4: 2	15	1
Group 5: DNA	16	4

	Start	Length
Match 1: **Test0456067800**	0	14
Group 1: Test	0	4
Group 2:	4	0
Group 3: 0	4	1
Group 4: 456067800	5	9
Group 5:	14	0

	Start	Length
Match 1: **a1**	0	2
Group 1:	0	0
Group 2:	0	0
Group 3: a	0	1
Group 4: 1	1	1
Group 5:	2	0

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

CANON0031	a1	Lesson 1	TEST0001456
CANON0009	CANON002500	Lesson 1 DNA	Test0456067800
CANON0002	Chapter1 Lesson2 DNA	Lesson 25 DNA	Test056430
05	Lesson 0	LookingGood 126355	The Rain in Spain00010

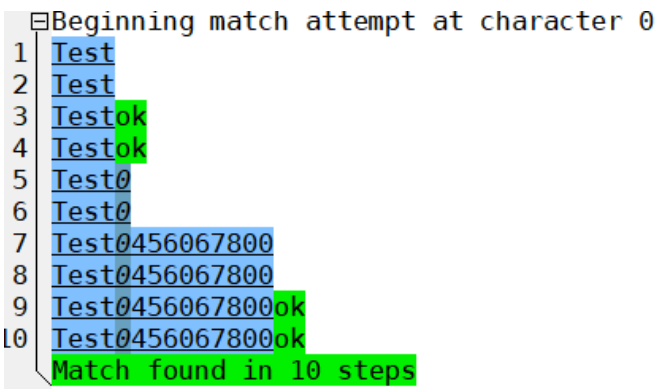
Example #5 – Using a Negated Class to isolate Zeros: cont.

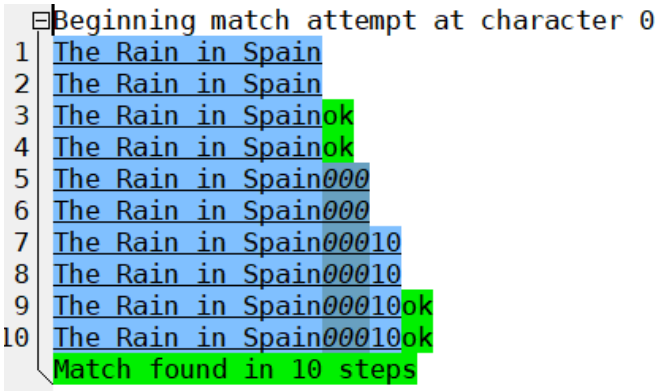
Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)

Tim Notes:

1. Group 1 captures any text preceding Zeros.
   a. If there is a <space> after text preceding numeric digits, it is captured in Group 3. i.e., 'Lesson 25 DNA'
   b. If there is a numeric other than a zero at the end of the first word in a multi-word precedent, this is captured by Group 2. i.e., 'Chapter1 Lesson2 DNA'
      1. The other text that follows up to a numeric digit is captured by Group 3. See same example.
2. Group 2 captures any numeric in the first word of a multi-word precedent. Otherwise Zero Occurrence Match results in null value, i.e. 'CANON0031'
3. Group 3 captures any preceding Zeros (isolates them). If no Zeros, will capture other text that follows after a numeric matched in the first word, i.e., 'Chapter1 Lesson2 DNA'
4. Group 4 will isolate the number after any Zeros. i.e., 'The Rain in Spain00010'
5. Group 5 will capture any text after the numeric digits captured in Capture Group 4. If none, a Zero Occurrence Match will occur resulting in a null value.

Less backtracking normally when using a Negated Class (but not always – depends on the RegEx and the sample matched):





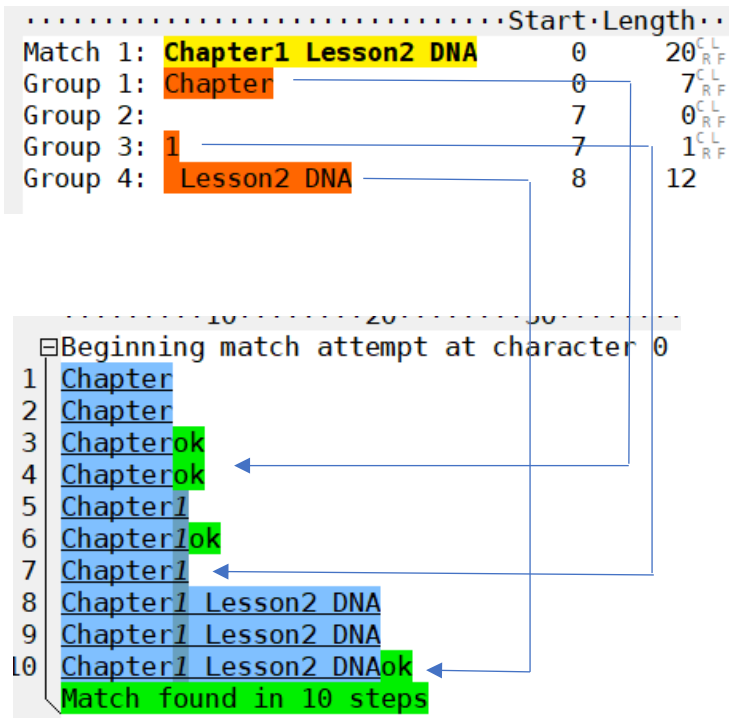
Remove

Remove all Except last Numeric Digits greater than zero cont.

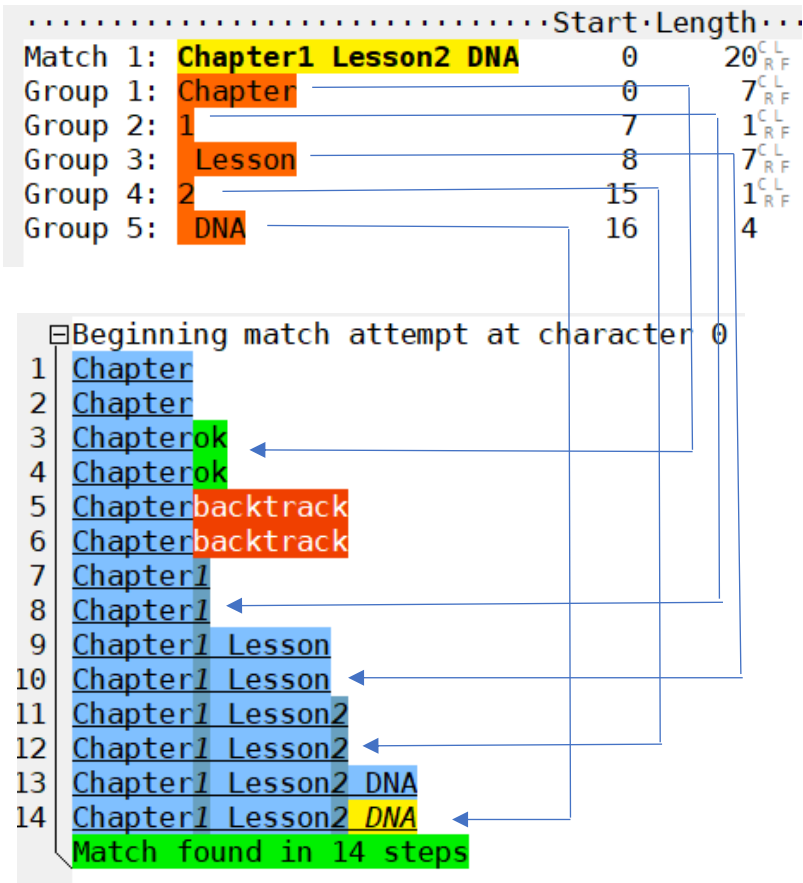
Tim Contribution – cont.

Example #5 – Using a Negated Class to isolate Zeros: cont.

Example #4a: ([A-Za-z]*)(.*?)([1-9]+[0-9]*)(.*)$



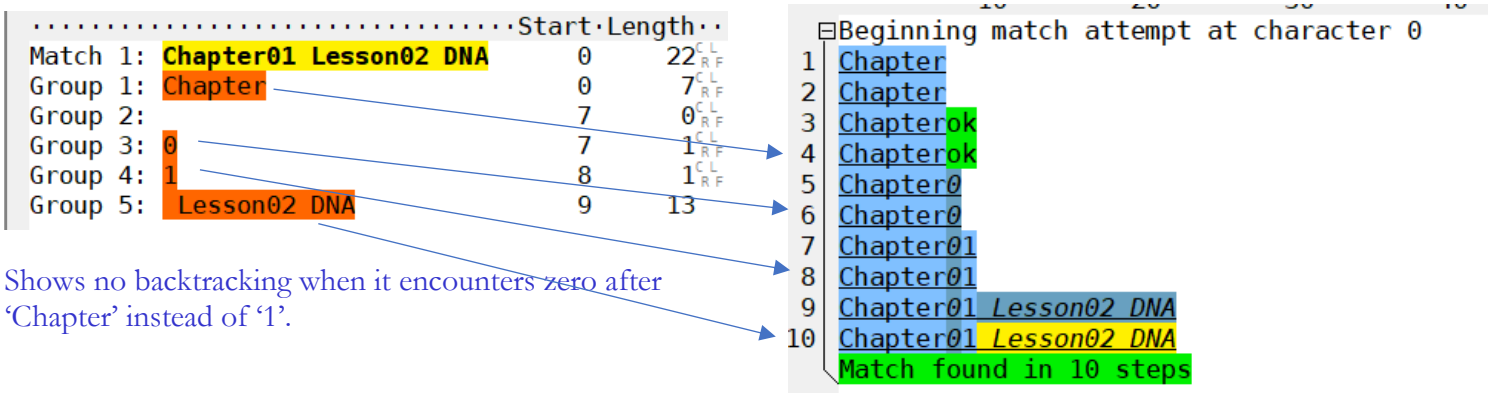
Example #5: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)



Here you can see backtracking in Example #5 where Example #4a had none. Conclusion – Example #4a did not backtrack and although there was little backtracking in Example #5, it still matched a larger sample group than Example #4a.

Added this – not part of original sample group – using Example #5 RegEx

Chapter01 Lesson02 DNA



Shows no backtracking when it encounters zero after 'Chapter' instead of '1'.

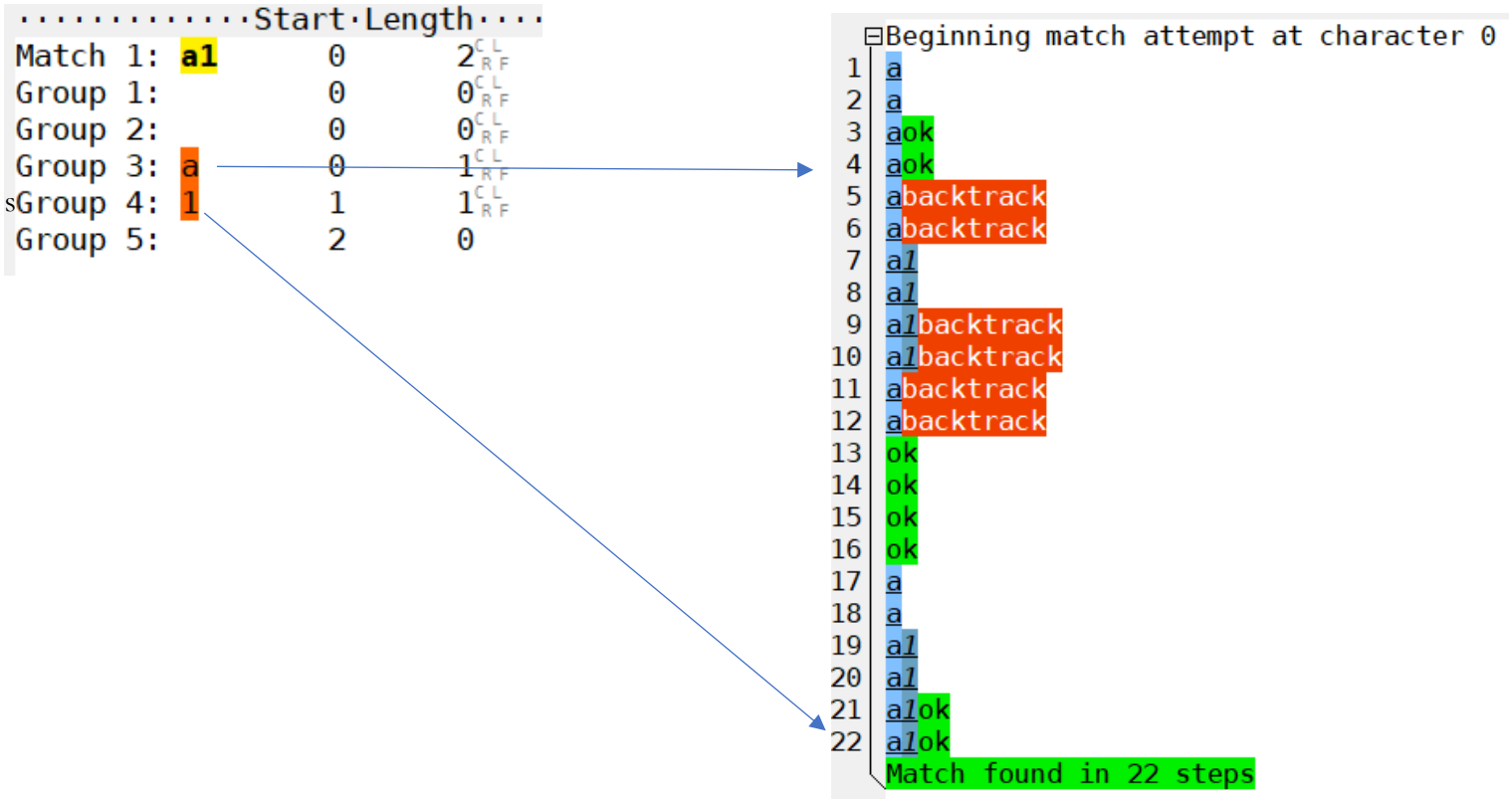
Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Example #5 – Using a Negated Class to isolate Zeros: cont.

Then there is this. It was hard to match up in previous RegEx examples, but when it did in Example #5, it ended up capturing the text in Capture Group 3 with quite a bit of backtracking.



Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Example #5 – Using a Negated Class to isolate Zeros: cont.

This is to show the capturing of the <space> in Capture Group 3:

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: The Rain in Spain00010

Example #5 – Using a Negated Class to isolate Zeros: cont.

Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)

Where:

\1 = The Rain in Spain \2 = null \3 = 000 \4 = 10 \5 = null

Observations:

1. Group 1 captures any text preceding Zeros.
2. Group 2 captures any numeric in the first word of a multi-word precedent. Otherwise Zero Occurrence Match results in null value.
3. Group 3 captures any preceding Zeros (isolates them).
4. Group 4 will isolate the number after any Zeros.
5. Group 5 will capture any text after the numeric digits captured in Capture Group 4. If none, a Zero Occurrence Match will occur resulting in a null value.

Breaks down:

(\D*)	(\d*?)	([^1-9]+)	([0-9]+)	(.*)
The Rain in Spain	null	000	10	null

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: The Rain in Spain00010

Example #5 – Using a Negated Class to isolate Zeros: cont.

Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)

Where:

\1 = The Rain in Spain \2 = null \3 = 000 \4 = 10 \5 = null

Tim Analysis:

1. \D	Match against any Non-Numeric Digit	Capture Group 1 = T
2. *	Make it Greedy Captures until it encounters a numeric	Capture Group 1= The Rain in Spain
3. \d	Match against Numeric Digit	Capture Group 2 = 0
4. *	Make it Greedy	Capture Group 2 = 00010
5. ?	But not Too Greedy Position = right after 'Spain'	Capture Group 2 = null
6. [^1-9]	Match against a Negated Class consisting of a numeric digit 1-9. This is how it will isolate the leading Zeros. If there are no Zeros but some text, then this gets captured here.	Capture Group 3 = 0
7. +	Make it Greedy	Capture Group 3 = 000
8. [0-9]	Match against a class consisting of a numeric digit 0-9. This is how it will capture numbers that contain Zeros within. This group captures the number greater than zero (the original intent of the RegEx).	Capture Group 4 = 1
9. +	Make it Greedy	Capture Group 4 = 10
10. .*	Match against remaining string Matches to EOL. Null value resulting from Zero Occurrence Match.	Capture Group 5 = null

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: a1

Example #5 – Using a Negated Class to isolate Zeros: cont.

Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)

Where:

\1 = null \2 = null \3 = a \4 = 1 \5 = null

Tim Analysis:

1. \D	Match against any Non-Numeric Digit	Capture Group 1 = a
2. *	Make it Greedy	Capture Group 1= Unchanged
3. \d	Match against Numeric Digit	Capture Group 2 = 1
4. *	Make it Greedy	Capture Group 2 = Unchanged
5. ?	But not Too Greedy Position = right after 'a'	Capture Group 2 = null
6. [^1-9]	Match against a Negated Class consisting of a numeric digit 1-9.	Capture Group 3 = a Capture Group 2 = Unchanged Changes Capture Group 1 = null

Tim Notes:

1. The RegEx Engine would skip over any numeric digits 1-9, but because there is only a '1' in the string it searches to EOL then backtracks back to the 'a' which is captured in Capture Group 3. This forces Capture Group 1 to give up the value resulting in a Zero Occurrence Match with a null value.

7. +	Make it Greedy	Capture Group 3 = No Change
8. [0-9]	Match against a class consisting of a numeric digit 0-9. This group captures the number greater than zero (the original intent of the RegEx).	Capture Group 4 = 1
9. +	Make it Greedy	Capture Group 4 = Unchanged
10. .*	Match against remaining string Matches to EOL. Null value resulting from Zero Occurrence Match.	Capture Group 5 = null

Remove

Remove all Except last Numeric Digits greater than zero cont.

Tim Contribution – cont.

Using: Chapter1 Lesson2 DNA

Example #5 – Using a Negated Class to isolate Zeros: cont.

Match: (\D*)(\d*?)([^1-9]+)([0-9]+)(.*)

Where:

\1 = Chapter \2 = 1 \3 = <space>Lesson \4 = 2 \5 = <space>DNA

Tim Analysis:

1. \D	Match against any Non-Numeric Digit	Capture Group 1 = C
2. *	Make it Greedy	Capture Group 1= Chapter
3. \d	Match against Numeric Digit	Capture Group 2 = 1
4. *	Make it Greedy	Capture Group 2 = No Change
5. ?	But not Too Greedy Position = just after 'Chapter'	Capture Group 2 = null
6. [^1-9]	Match against a Negated Class consisting of a numeric digit 1-9. This is the <space> after 'Chapter'	Capture Group 3 = <space> Capture Group 2 = 1
7. +	Make it Greedy	Capture Group 3 = <space> Lesson
8. [0-9]	Match against a class consisting of a numeric digit 0-9. This group captures the number greater than zero (the original intent of the RegEx).	Capture Group 4 = 2
9. +	Make it Greedy	Capture Group 4 = No Change
10. .*	Match against remaining string	Capture Group 5 = <space> DNA

Final Observations:

1. Universal - Will isolate text preceding Zeros. (Capture Group 1)
2. Isolates any numeric digit associated with first word of precedent text. (Capture Group 2)
3. No dependence on Zeros.
4. Complete parsing of all larger sample base
5. Isolates Zeros (Capture Group 3)
6. Isolates numbers greater than zero (Capture Group 4)
7. Captures any text following (Capture Group 5)

Remove

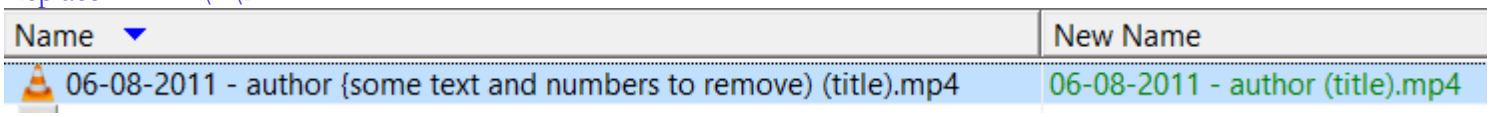
Remove All Text Between { and)

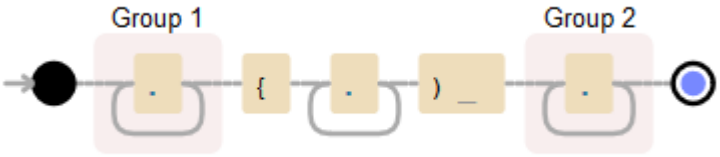
06-08-2011 - author {some text and numbers to remove) (title).mp4 to 06-08-2011 - author (title).mp4

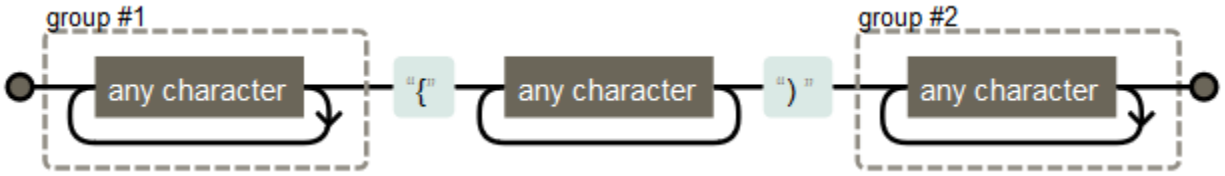
Observations:

1. Remove all between the brace (curly bracket) and the first closing parenthesis.

Match: (.+){.+?\) (.+)
Replace: \1\2







Sample:

```
.....................................................................................
Match 1: 06-08-2011 - author {some text and numbers to remove) (title)
Group 1: 06-08-2011 - author
Group 2: (title)
```

Contributor Analysis:

1. .+ Match everything before the brace { and capture in Group 1
2. .+? Match everything non-greedy till…

3. \) .. first closing parenthesis, which we have to escape by an backslash because it is a RegEx Metacharacter

4. <space> .. followed by one <space>.

   Non-greedy not needed because RegEx indicates there has to be something after the closing parenthesis, so it can only mean the first one in this example is being referenced.

5. .+ Match remainder of string to EOL (End of Line) and capture in Group 2

Remove

Remove All Text Between { and) cont.

Tim Notes:

1. This one is a little strange. The ' {' is considered a Special Metacharacter. Here it is used as a literal within the RegEx. Why this is strange is because literals are defined as those characters that can be reproduced on the keyboard. There are, however, certain characters the RegEx engine reserves because they are used to represent Metacharacters. These are sometimes referred to as 'Special' Metacharacters and the braces are one on them.

   When a Special Metacharacter is used as a literal it must be preceded by an 'escape' character – the backslash ' \ '. This indicates to the RegEx engine to treat the character as a literal and not as a Metacharacter. Here, there is no escape character – just the left brace. It should fail because it requires a right brace ' } ' Metacharacter to complete the syntax, but it doesn't. If I Added the Escape character ' \} ' it would not affect the RegEx, as it should, because now the RegEx would be in the correct syntax.

Forum Contributor: Stefan

Remove

Remove All Text Between { and) cont.

A short Word on Special Characters

As with most programming languages, certain characters are reserved for use by the program. This means that when the program encounters one of these characters it identifies it as a function of the program rather than simply a character. These characters are called 'Metacharacters' or 'Special Metacharacters' while regular characters are referred to as 'Literals'.

The following 'literals' are reserved by Regular Expressions as Metacharacters that have reserved status:

backslash	\	Escape character
caret	^	Beginning or start of string aka BOL (Beginning of Line)
dollar sign	$	EOL (End of Line) indicates end of string
period or dot	.	Any character
vertical bar or pipe symbol	|	Alternation aka Branching
question mark	?	Lazy aka Non-Greedy, Optional, Reluctant (when applied against a Greedy Quantifier)
asterisk or star	*	Greedy Quantifier (Zero or More)
plus sign	+	Greedy Quantifier (One or More)
opening parenthesis	(	Beginning of Group, Capture Group, Atomic or Non-marking Group
closing parenthesis	)	End of Group, Capture Group, Atomic or Non-marking Group
opening square bracket	[	Indicates a Class
opening curly brace	{	Indicates a Quantifier

Why are the closing square bracket and closing curly bracket not considered Metacharacters? Because when the program encounters the opening character of a square bracket or curly bracket, it only checks the syntax to ensure that there is a corresponding closing character. It does not need the closing characters to be Metacharacters. This is unlike the closed parenthesis that can stand for other things – the closure of Capture Groups, Non Marking or Atomic Groups, etc.

If you want to use one of these Metacharacters listed above in the RegEx, you have to precede it with a backslash character. The backslash is a special Metacharacter, called an 'Escape', that indicates to the program that the following character will be treated as a literal. Even the backslash character can be used in the RegEx in this manner – i.e., \\

Question:

So down to our question from the previous example, Why did the RegEx Engine allow the Opening Curly Bracket to be used as a literal NOT preceded by an Escape Metacharacter?

Answer:

Most regular expression flavors treat the opening curly brace { as a literal character, unless it is part of a repetition operator like a {1,3}. So it is made optional to use the Escape Metacharacter in this circumstance.

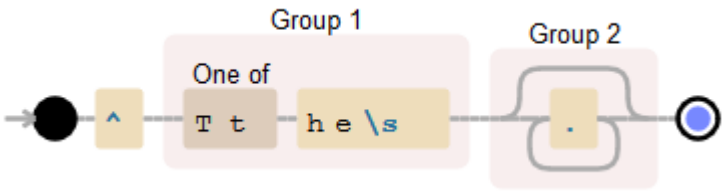
Remove

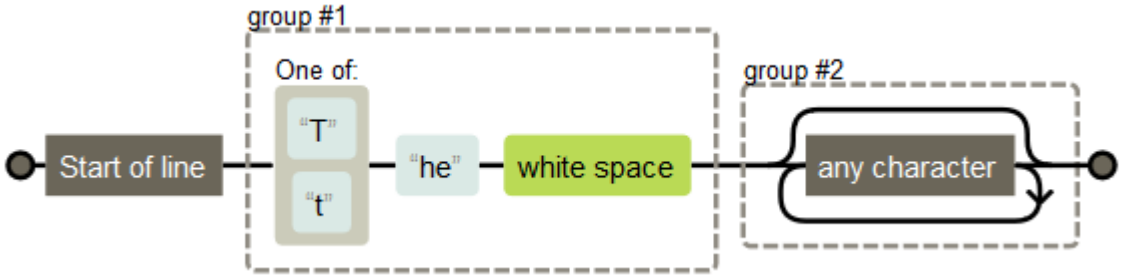
Remove 'The' only at the start of a Filename

The Who to Who

Match: ^([Tt]he\s)(.*)
Replace: \2

Name	New Name
The Who	Who





Sample:

```
.....................
Match 1: The Who
Group 1: The
Group 2: Who
```

Tim Notes:

1. This could have been done many ways, but I thought it was important show a demonstration of how classes could be used to match against a particular word in upper or lowercase.

2. This has been discussed previously. See 'Remove 'The' from beginning to end of title (not end of filename)', and 'Rearrange word following comma to beginning of string (Beginning of Line or BOL)'.

Forum Contributor: therube

Remove

Remove 'The' only at the start of a Filename cont.

Using: The Who

Match: ^([Tt]he\s)(.*)

Where:

\1 = The <space> \2 = Who

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [Tt]	Match against a class consisting of an uppercase 'T' or a lowercase 't'.	Capture Group 1 = T
3. he	Match against a string, 'he'	Capture Group 1 = The
4. \s	Match against the <space> character	Capture Group 1= The<space>
5. .*	Match against remaining string	Capture Group 2 = Who

Remove

Removing multiple instances of the same character

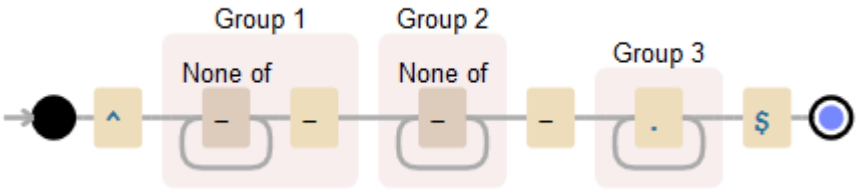
01 - this is a song.mp3	to	Ignore
02 - hey.mp3	to	Ignore
03 - hey-hey.mp3	to	03 – hey hey.mp3

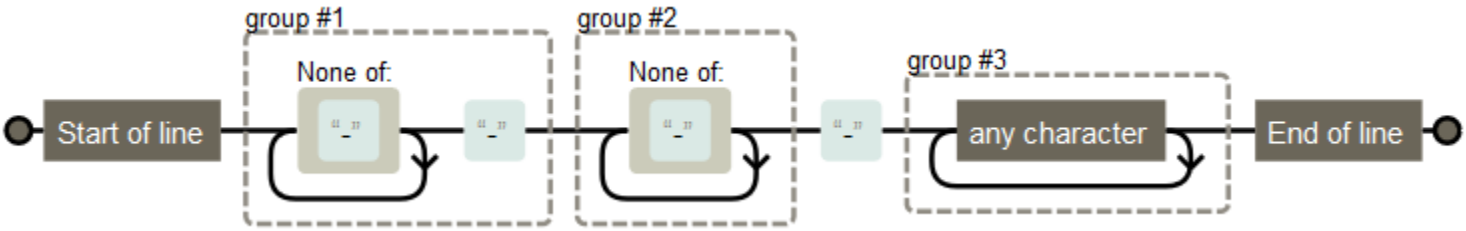
Observations:

1. Replace all instances of the "-" or <hyphen> character, except the first one, to a <space>.
2. Files with only one <hyphen> would not be changed.

Match: ^([^-]+-)([^-]+)-(.+)$
Replace: \1\2 \3 The <space> between \2 and \3 replaces the <hyphen>.

Name	New Name
01 - this is a song.mp3	01 - this is a song.mp3
02 - hey.mp3	02 - hey.mp3
03 - hey-hey.mp3	03 - hey hey.mp3





Sample:

```
.......................
Match 1: 03 - hey-hey
Group 1: 03 -
Group 2:  hey
Group 3: hey
```

Remove

Removing multiple instances of the same character cont.

Contributor Analysis:

1. ^([^-]+-) Captures everything from the start, up to and including, the first <hyphen> and saves it in \1

2. ([^-]+) Captures everything after the first <hyphen> up to the next <hyphen> and saves it in \2

3. - Matches the second <hyphen>

4. (.+)$ Captures everything from after the second <hyphen> (including any more <hyphens>) up to the end and saves it in \3

Contributor Notes:

1. This will only match filenames with more than one <hyphen>.

2. It replaces the second <hyphen> with a <space>.

3. If you have filenames with more than 2 <hyphens> you simply run it more than once.

i.e.,
Filenames with a range of 0 to 4 <hyphens>.

a. It won't match those with 0 or one <hyphen> (and won't change them).

b. Will match those with 2, 3, and 4.

c. Will replace the second <hyphen> on the first pass.

1. All those filenames with 2 <hyphens> become 1.
2. All those filenames with 3 <hyphens> become 2.
3. All those filenames with 4 <hyphens> become 3.

d. A couple of passes and all are changed.

Forum Contributor: bitmonger

Remove

Removing multiple instances of the same character cont.

Using: 03 - hey-hey.mp3

Match: ^([^-]+-)([^-]+)-(.+)$

Where:

\1 = 03 <space> - \2 - <space>hey \3 = hey

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^-]	Match against a Negated Class consisting of a <hyphen>. Tests character is not a <hyphen>	Capture Group 1 = 0
3. +	Make it Greedy	Capture Group 1 = 03
4. -	Match against <hyphen> This is the <hyphen> after '03<space>' Captures the <space> as it matches the <hyphen>	Capture Group 1 = 03 <space>-
5. [^-]	Match against a Negated Class consisting of a <hyphen>. Tests character is not a <hyphen> Captures the <space> after the <hyphen> of '03<space><hyphen>'	Capture Group 2 = <space>
6. +	Make it Greedy	Capture Group 2 = <space>hey
7. -	Match against <hyphen> (Not Captured) = - This is the <hyphen> after 'hey'	
8. (.+)	Match against remaining string This is the second 'hey' in the string.	Capture Group 3 = hey

Remove

Remove parts 'a' and 'b' from filename (Demonstrate the \w)

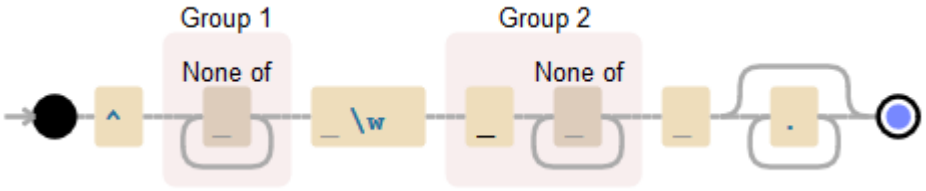
first a_second b.mp3 to first_second.mp3

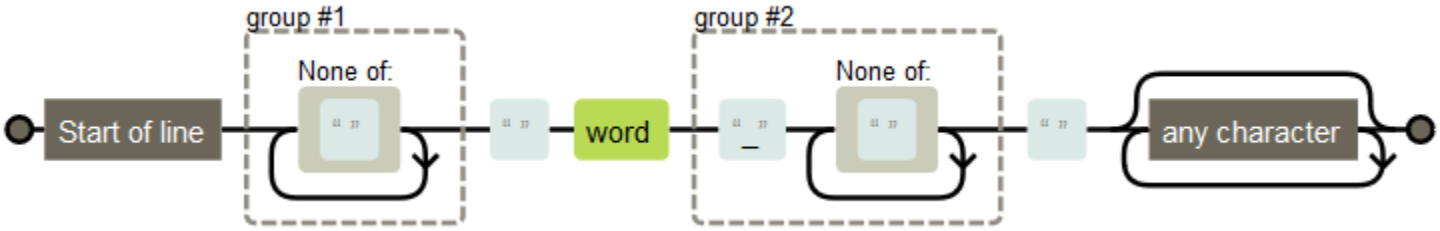
Observations:

1. Pattern established as <text> <space> 'a' <underscore> <text> <space> 'b'
2. Remove the first <space> and single character before the first underscore
3. Remove the next <space> and any character following it up to the EOL (End of Line)

Match: ^([^]+) \w(_[^]+) .*
Replace: \1\2

Name	New Name
first a_second b.mp3	first_second





Sample:

```
.............................
Match 1: first a_second b.mp3
Group 1: first
Group 2: _second
```

Forum Contributor: Glenn

Remove

Remove parts 'a' and 'b' from filename (Demonstrate the \w) cont.

Using: first a_second b.mp3

Match: ^([^]+) \w(_[^]+) .*

Where:

\1 = first \2 = _second

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. [^]	Negated class. Match any character Except the <space> character	Capture Group 1 = f
3. +	Make it Greedy until …	Capture Group 1 = first
4. <space>	… <space> character (Not Captured), followed by .. This is the <space> following 'first'	
5. \w	Match against any single letter, number or underscore (Not Captured) = 'a', followed by …	
6. _	.. underscore	Capture Group 2 = _
7. [^]	Negated class. Match any character Except the <space> character	Capture Group 2 = _s
8. +	Make it Greedy until …	Capture Group 2 = _second
9. <space>	… <space> (Not Captured)	
10. .*	Match against remainder of string (Not Captured) = 'b'	

Tim Notes:

1. \w (Word Metacharacter) matches any single letter, number or underscore in case some files contain a number at that position instead of a letter.

2. I slightly modified the original RegEx of the Contributor, ^([^]+) \w(_[^]+) .\

   a. Would only work IF the 'Rename File Extensions as Being Part of File Name' was set in Rename Options Menu.

   b. Restricted to exact pattern - would not handle extra characters after the second <space> unless an additional RegEx was used - ^([^]+) \w(_[^]+) .+\.

   c. My modification will handle part 'b' made up of more than a single letter.

Remove

Remove parts 'a' and 'b' from filename (Demonstrate the \w) cont.

Tim Contribution –

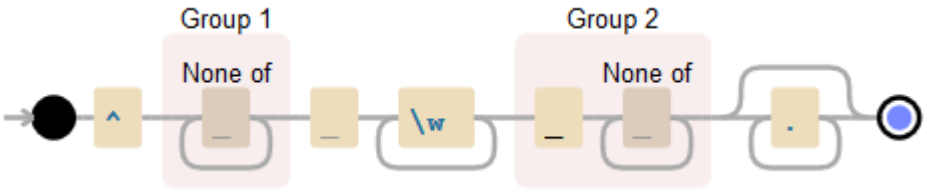
Samples:

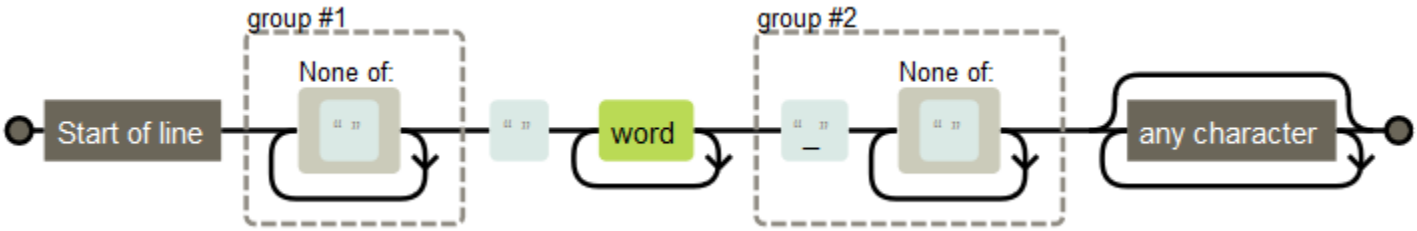
first a_second b.mp3
first agggg_second bhhhh hhhtt ggh.mp3
first abc_second def.mp3

This will handle parts 'a' and 'b' that are made up of more than single letters

Match: ^([^]+)\D\w+(_[^\s]+)\D(.*)
Replace; \1\2

Name	New Name
first a_second b.mp3	first_second.mp3
first abc_second def.mp3	first_second.mp3
first agggg_second bhhhh hhhtt ggh.mp3	first_second.mp3





Sample:

```
.....................................
Match 1: first a_second b.mp3
Group 1: first
Group 2: _second
```

Sample:

```
.....................................
Match 1: first abc_second def.mp3
Group 1: first
Group 2: _second
```

Sample:

```
.................................................
Match 1: first agggg_second bhhhh hhhtt ggh.mp3
Group 1: first
Group 2: _second
```

Remove

Remove parts 'a' and 'b' from filename (Demonstrate the \w) cont.

Tim Contribution – cont.

Using: first agggg_second bhhhh hhhtt ggh.mp3

```
.........................................
Match 1: first agggg_second bhhhh hhhtt ggh
Group 1: first
Group 2:  agggg
Group 3: _second
Group 4: bhhhh hhhtt ggh
```

Tim Notes:

1. I added a Capture Group around the second sub-expression to make the Analysis more comprehensive. This will use a Replacement String of \1\3

Match: ^([^]+)(\D\w+)(_[^\s]+)\D(.*)

Where:

\1 = first \2 = <space> agggg \3 = _second \4 = bhhhh hhhtt ggh

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. [^]	Negated class. Match any character Except the <space> character	Capture Group 1 = f
3. +	Make it Greedy until …	Capture Group 1 = first
4. \D	Match against any Non-Numeric Digit character, followed by .. This is the <space> following 'first'	Capture Group 2 = <space>
5. \w	Match against any single letter, number or underscore	Capture Group 2 = <space> a
6. +	Make it Greedy	Capture Group 2 = <space> agggg_second
7. _	.. underscore	Capture Group 3 = _ Changes Capture Group 2 = <space> agggg
8. [^]	Negated class. Match any character Except the <space> character	Capture Group 3 = _s
9. +	Make it Greedy until …	Capture Group 3 = _second
10. \D	Match against any Non-Numeric Digit = <space> (Not Captured) This is the <space> after 'second'	
11. .*	Match against remainder of string	Capture Group 4 = bhhhh hhhtt ggh

Remove

Remove hyphens from string (Demonstrate Greedy vs Lazy)

aaa-bbb	to	aaabbb
1234-ddd	to	1234ddd
eee---wwww	to	eeewwww

Greedy vs Lazy

Geeedy –

Match: (.*)(-+)(.*)
Replace: \1#\3

1. The first "(.+)" captures all till the last <hyphen>.

Test-1.txt	to	Test#1.txt
Test--2Zwei.txt	to	Test-#2Zwei.txt
Test---3Drei.txt	to	Test--#3Drei.txt
Test----4Vier.txt	to	Test---#4Vier.txt

Lazy –

Match: (.*?)(-+)(.*)
Replace: \1\3
or
Match: (.*?)-+(.*)
Replace: \1\2

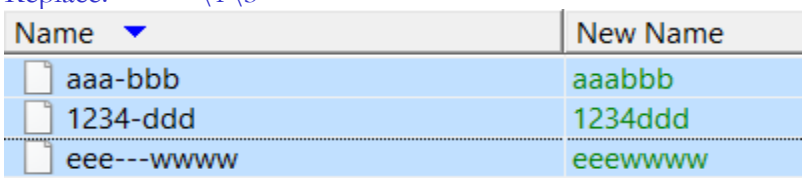
and you will see it works as intended:

Test-1.txt	to	Test1.txt
Test--2Zwei.txt	to	Test2Zwei.txt
Test---3Drei.txt	to	Test3Drei.txt
Test----4Vier.txt	to	Test4Vier.txt

The "?" switch to non-greedy (Lazy) makes the first expression ".*" stop at the first occurrence of the next expression "-+".

A Negated Character Class could also be used -

Match: (.*[^-])(-+)(.*)
Replace: \1\3



[^-] which means: none-or-more of any sign till … (-+) a <hyphen>

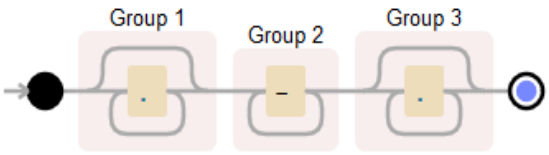
Forum Contributor: Stefan

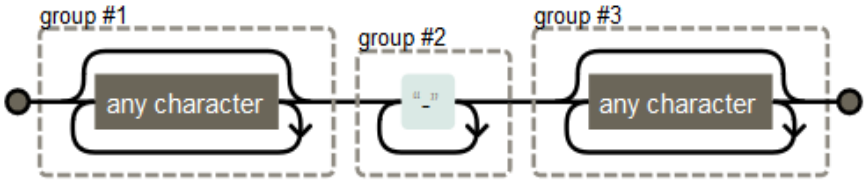
Remove

Remove hyphens from string (Demonstrate Greedy vs Lazy) cont.

Greedy

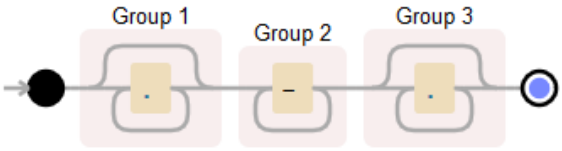
Match: (.*)(-+)(.*)

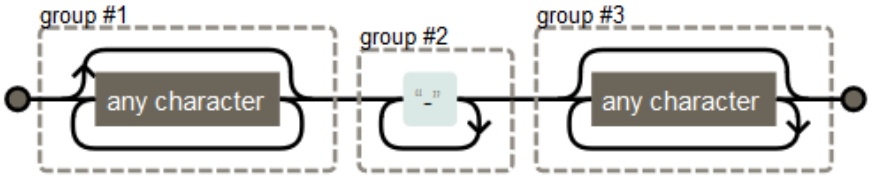




Lazy

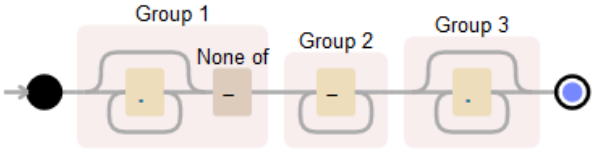
Match: (.*?)(-+)(.*)

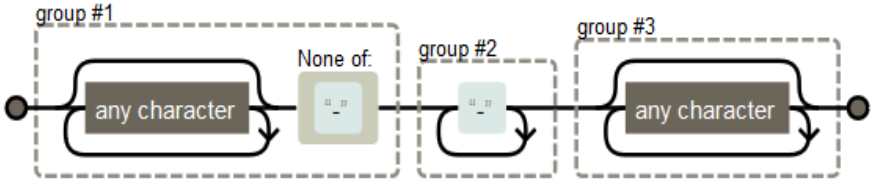




Negated Character Class

Match: (.*[^-])(-+)(.*)





Sample: Greedy

```
.......................
Match 1: Test--2Zwei
Group 1: Test-
Group 2: -
Group 3: 2Zwei
```

Sample: Lazy

```
.......................
Match 1: Test--2Zwei
Group 1: Test
Group 2: --
Group 3: 2Zwei
```

Sample: Negated Class

```
.......................
Match 1: Test--2Zwei
Group 1: Test
Group 2: --
Group 3: 2Zwei
```

Remove

Remove hyphens from string (Demonstrate Greedy vs Lazy) cont.

Using: Test--2Zwei.txt

Greedy

Match: (.*)(-+)(.*)

Where:

\1 = Test- \2 = - \3 = 2Zwei

Tim Analysis:

1. .	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen> This is the first <hyphen> in the string	Capture Group 2 = - Changes Capture Group 1 = Test-
4. +	Make it Greedy	Capture Group 2 = No Change
5. .*	Match against remaining string	Capture Group 3 = 2Zwei

Lazy

Match: (.*?)(-+)(.*)

Where:

\1 = Test \2 = -- \3 = 2Zwei

Tim Analysis:

1. .	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
3. -	Match against <hyphen> This is the first <hyphen> in the string	Capture Group = - Changes Capture Group 1 = Test-
4. +	Make it Greedy	Capture Group 2 = --
5. .*	Match against remaining string	Capture Group 3 = 2Zwei

Remove

Remove hyphens from string (Demonstrate Greedy vs Lazy) cont.

Using: Test--2Zwei.txt

Negated Character Class

Match: (.*[^-])(-+)(.*)

Where:

\1 = Test \2 = -- \3 = 2Zwei

Tim Analysis:

1. .	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [^-]	Match against a Negated Class consisting of a <hyphen> character Still at EOL at this point Current position = EOL	Capture Group 1 = Unchanged
3. -	Match against <hyphen> This is the first <hyphen> in the string	Capture Group 2 = - Changes Capture Group 1 = Test

Tim Notes:

1. The current position was EOL. When it matches against the <hyphen> it backtracks testing each character for the <hyphen> but still also testing against the Negated Class as well. When it encounters the two <hyphens> while still moving backwards, it ignores these until it reaches the BOL. The RegEx Engine then moves forward capturing the first <hyphen) after 'Test'.

   This in turn changes the value of Capture Group 1 = 'Test' because as Capture Group 2 captures the <hyphen>, Capture Group 1 (.*) reevaluates to capture the preceding text.

4. +	Make it Greedy	Capture Group 2 = --
5. .*	Match against remaining string	Capture Group 3 = 2Zwei

Remove

Remove Email addresses from string

152 Email 05124 CLOSED (Mezzanine Framing at Escalators) Kile Quinn kileq@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (04-03-2011).pdf
to
2152 Email 05124 CLOSED (Mezzanine Framing at Escalators) Kile Quinn - Alana Zwanziger (04-03-2011).pdf

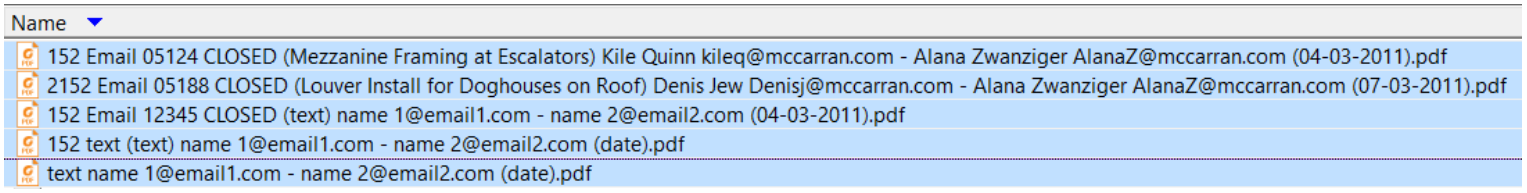
2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011).pdf
to
2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew - Alana Zwanziger (07-03-2011).pdf

Observations:

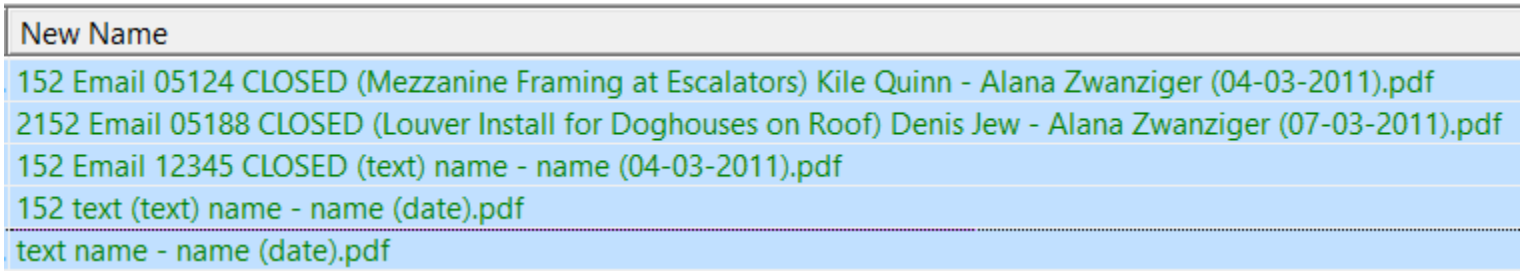
1. Simplify as.. 152 Email 05124 CLOSED (text) name 1@email1.com - name 2@email2.com (04-03-2011).pdf
2. Simplified further as .. 152 text (text) name 1@email1.com - name 2@email2.com (date).pdf
3. or even further.. text name 1@email1.com - name 2@email2.com (date).pdf

Match: (.+).+@.+? (.+).+@.+ (.+)
Replace: \1\2\3

Name:



New Name:



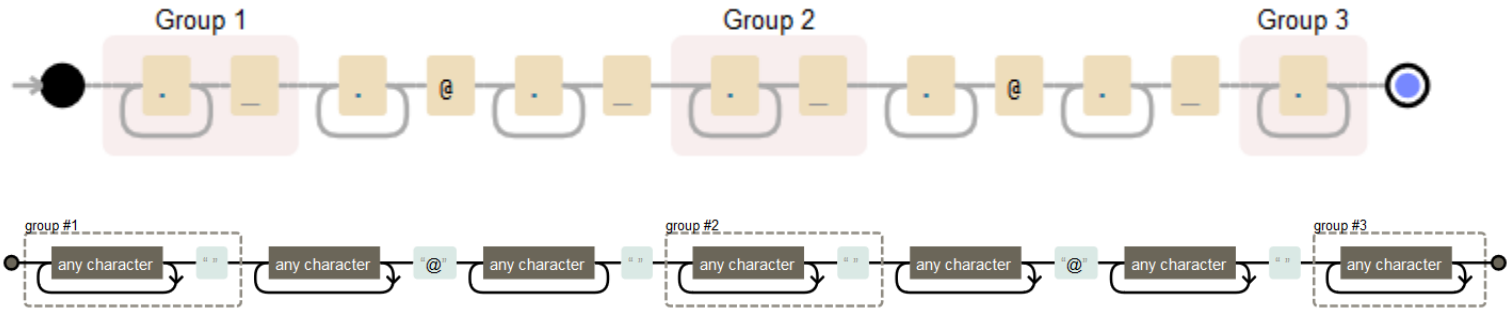
Tim Notes:

1. To the best of my knowledge these email addresses are fake and were submitted by the poster as examples only.

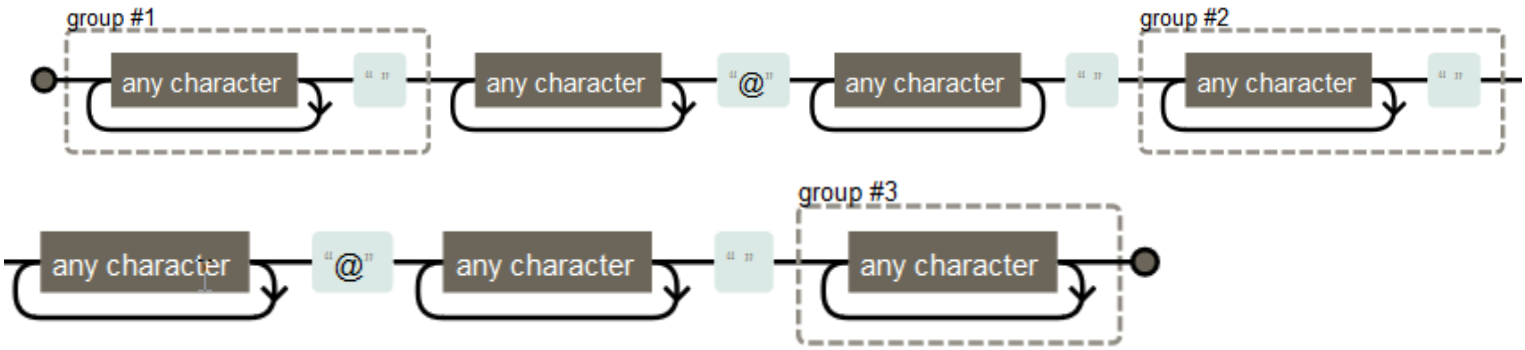
Remove

Remove Email addresses from string cont.

Match: (.+).+@.+? (.+).+@.+ (.+)



Expanded:



Sample:

```
Match 1: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011
Group 1: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew
Group 2: - Alana Zwanziger
Group 3: (07-03-2011
```

Sample Expanded:

```
Group 1: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew
Group 2: - Alana Zwanziger
Group 3: (07-03-2011
```

Contributor Notes:

1. Only for the samples provided.
2. If other files are named different this solution may fail.

Forum Contributor: Stefan

Remove

Remove Email addresses from string cont.

Using: 152 Email 05124 CLOSED (Mezzanine Framing at Escalators) Kile Quinn kileq@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (04-03-2011).pdf

Where:

\1 = 2152 Email 05124 CLOSED (Mezzanine Framing at Escalators) Kyle Quinn
\2 = - Alana Z@ \3 = (04-03-2011)

Tim Analysis:

1. (.+<space>)	Match till a <space> = '2152 Email 05124 CLOSED (Mezzanine Framing at Escalators) Kyle Quinn'	Capture Group 1
2. .+@	Match until an @-sign but don't capture = 'kileq@'	
3. .+?<space>	Match until next expression but don't capture = 'mccarran.com'	
4. .+ <space>	Match till a <space> = '- Alana Zwanziger'	Capture Group 2
5. .+@	Match until an @-sign but don't capture = 'Alana Z@'	
6. .+ <space>	Match till a <space> but don't capture = ;mccarran.com'	
7. (.+)	Capture remainder of string = '(04-03-2011)'	Capture Group 3

Remove

Remove Email addresses from string cont.

Tim Contribution –

Less restrictive:

Acceptable Pattern: text<space>@<space>Text<space>@<space>

i.e., textxx @ xx @ .jpg
text name 1name@.com name 2name@.com .jpg

and the last <space> left in after @ will still match:

textxx @ xx @ (date).jpg

Match: (.*).*(@.*?)(.*).*(@.*)(.*)$
Replace: \1\3\5

Name	New Name
textxx @ xx @ .jpg	textxx xx .jpg
textxx @ xx @ (date).jpg	textxx xx (date).jpg
text name 1name@.com name 2name@.com .jpg	text name name .jpg
text name 1@email1.com - name 2@email2.com (date).pdf	text name - name (date).pdf

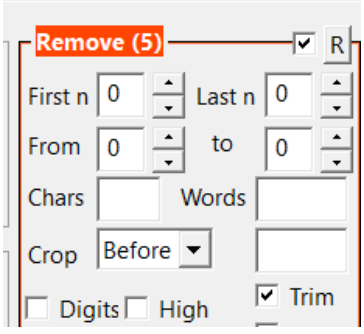
as well as the previous samples:

New Name
152 Email 05124 CLOSED (Mezzanine Framing at Escalators) Kile Quinn - Alana Zwanziger (04-03-2011).pdf
2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew - Alana Zwanziger (07-03-2011).pdf
152 Email 12345 CLOSED (text) name - name (04-03-2011).pdf
152 text (text) name - name (date).pdf
text name - name (date).pdf

Tim Notes:

1. A trailing <space> occurs when there is nothing after the last @ in the string. If you want to remove the trailing <space>:

   'Section #5: Remove' –



Set to Trim.

Tim Notes:

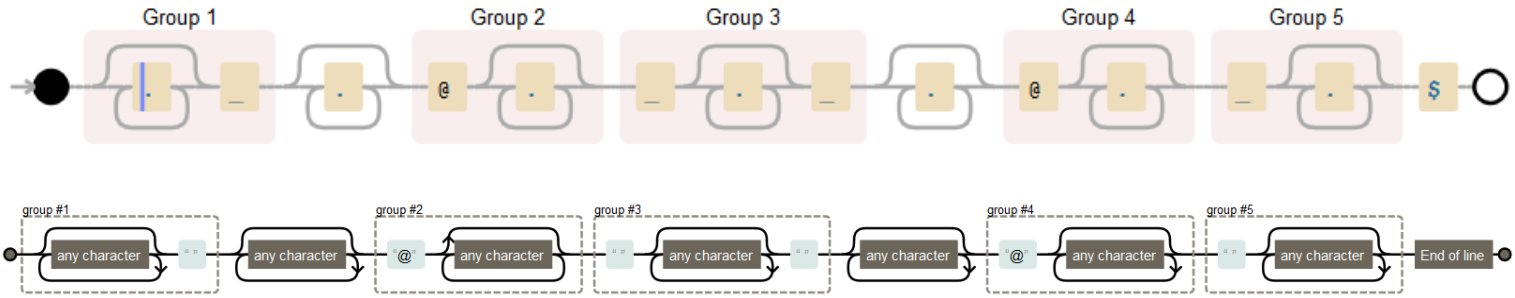
1. This can be done in the same run.

Remove

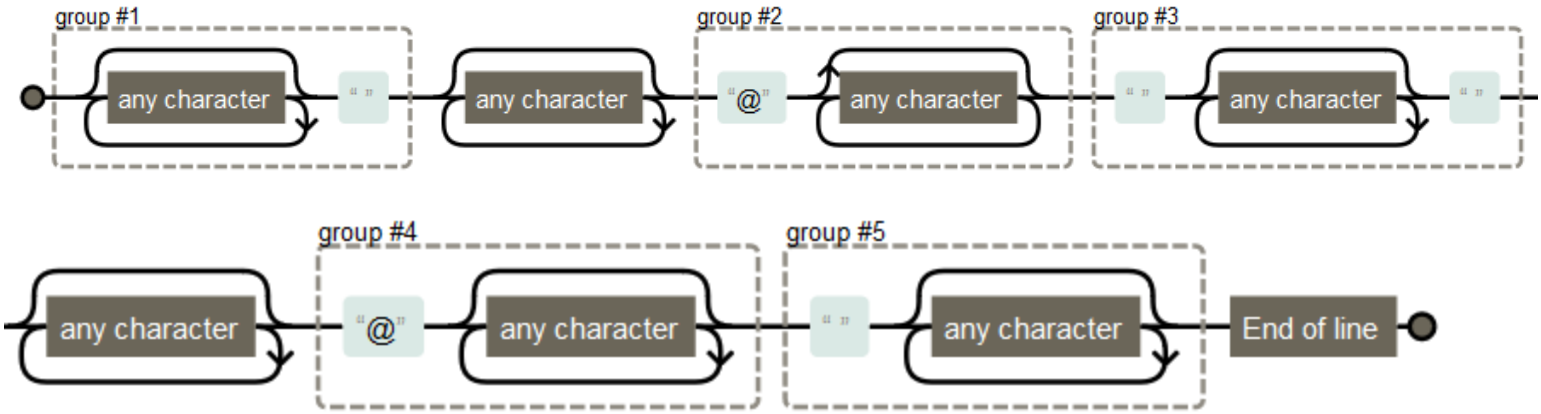
Remove Email addresses from string cont.

Tim Contribution – cont.

Match: (.*).*(@.*?)(.*).*(@.*)(.*)$



Expanded:



Sample:

```
Match 1: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011
Group 1: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew
Group 2: @mccarran.com
Group 3:  - Alana Zwanziger
Group 4: @mccarran.com
Group 5:  (07-03-2011
```

Sample Expanded:

```
Group 1: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew
Group 2: @mccarran.com
Group 3:  - Alana Zwanziger
Group 4: @mccarran.com
Group 5:  (07-03-2011
```

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011).pdf

Match: (.*).*(@.*?)(.*).*(@.*)(.*)$

Where:

\1 = 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew <space>
\2 = @mccarran.com \3 = <space> - Alana Zwanziger<space>
\4 = @mccarran.com \5 = <space> (07-03-2011)

Tim Analysis:

1..	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1 = <entire string>
3..	Match against any character (Not Captured) = ' (' This is the left parenthesis before '(07'	Changes Capture Group 1 = 2152 Email 05188 CLOSED (Louver Install <space>for Doghouses on Roof) Denis Jew <space> Denisj@mccarran.com – <space> Alana Zwanziger <space> AlanaZ@mccarran.com<space>
4. *	Make it Greedy (Not Captured) = '(07-03-2011)'	
5. @	Match against the '@' sign This is the '@' before the second 'mccarran.com' Changes Not Captured in step 4 to 'Alana Z' just before '@mccarran.com'	Capture Group 2 = @ Changes capture Group 1 = 2152 Email 05188 CLOSED (Louver Install <space>for Doghouses on Roof) Denis Jew <space> Denisj@mccarran.com – <space> Alana Zwanziger <space>
6..	Match against any character	Capture Group 2 = @m
7. *	Make it Greedy	Capture Group 2 = @mccarran.com (07-03-2011)
8. ?	But not too Greedy Backtracks back to the '@' before the second 'mccarran.com'	Capture Group 2 = @
9. <space>	Match against a <space> character This is the <space> before '(07' This is the second '@mccarran.com'	Capture Group 3 =<space> Changes Capture Group 2 = @mccarran.com
10..	Match against any character This is the left parenthesis before '07'	Capture Group 3 = (
11. *	Make it Greedy Current position = EOL	Capture Group 3 = (07-03-2011)

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011).pdf

Match: (.*).*(@.*?)(.*).*(@.*)(.*)$

Where:

\1 = 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew <space>
\2 = @mccarran.com \3 = <space> - Alana Zwanziger<space>
\4 = @mccarran.com \5 = <space> (07-03-2011)

Tim Analysis: cont.

12. <space>	Match against a <space> character Backtracks to match <space> before '- Alana' This changes Capture Group 2 to the first '@mccarran.com' in the string because now the second '@mccarran.com' is held by Capture Group 3	Capture Group 3 = <space>- Alana Zwanziger AlanaZ@mccarran.com Changes Capture Group 2 = @mccarran.com Changes Capture Group 1 = 2152 Email 05188 CLOSED (Louver Install <space>for Doghouses on Roof) Denis Jew <space>

Tim Notes:

1. Why didn't it match against the <space> after 'Zwanziger'? Because it is still matching against the expression, (.*), so it is looking for a match where there is a '<space> text <space>' and the first match is the '<space> - <space>' after the first '@mccarran.com' in the string.

13..	Match against any character (Not Captured) = ' (' This is the left parenthesis before '07'	
14. *	Make it Greedy (Not Captured) = (07-03-2011) Current position = EOL	
15. @	Match against the '@' sign This is the '@' before the second 'mccarran.com' Changes Not Captured in step 14 to 'Alana Z' just before '@mccarran.com'	Capture Group 4 = @ Changes Capture Group 3 = <space>- Alana Zwanziger <space> Capture Group 2 = Unchanged Capture Group 1 = Unchanged
16..	Match against any character	Capture Group 4 = @m
17. *	Make it Greedy	Capture Group 4 = @mccarran.com <space> (07-03-2011)

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011).pdf

Match: (.*).*(@.*?)(.*).*(@.*)(.*)$

Where:

\1 = 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew <space>
\2 = @mccarran.com \3 = <space> - Alana Zwanziger<space>
\4 = @mccarran.com \5 = <space> (07-03-2011)

Tim Analysis: cont.

18. <space>	Match against a <space> character This is the <space> character before '07'. This is the second '@mccarran.com' in the string.	Capture Group 5 = <space> Changes Capture Group 4 = @mccarran.com Capture Group 3 = Unchanged Capture Group 2 = Unchanged Capture Group 1 = Unchanged
19. .*	Match against remaining string	Capture Group 5 = <space> (07-03-2011)
20. $	Must be at EOL	

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Here is an alternative solution if the email portion is always preceded by a (text).

Using: 2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew Denisj@mccarran.com - Alana Zwanziger AlanaZ@mccarran.com (07-03-2011).pdf

Match: (.*?)(\))((\D\w+){2})
Replace: \1\2\3

Name	New Name
2152 Email 05188 CLOSED (Louver...	2152 Email 05188 CLOSED (Louver Install for Doghouses on Roof) Denis Jew.pdf

Breaks Down:

Capture Group 3

Capture Group 1 Capture Group 2 Capture Group 4

(.*?) (\)) ((\D \w+) {2})

Tim Analysis: (Changed Color key for this analysis based on the above so that you may better follow nested groups)

1..	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1= BOL
4. \)	Match against right parenthesis This is the right parenthesis after 'Roof'	Capture Group 2 =) Changes Capture Group 1 = 2152 Email 05188 CLOSED <space> (Louver Install for Doghouses on Roof)
5. \D	Match against any Non-Numeric Digit This is the <space> after 'Roof)'	Capture Group 3 = <space>
6.\w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = <space>D
7. +	make it Greedy	Capture Group 3 = <space> Denis
8. {2}	Repeat last expression to match 2 iterations	Capture Group 4 = <space>Jew Capture Group 3 =<space> Denis Jew

Tim Notes:

1. This is another example of a nested Capture Group. By nesting a Capture Group, the Inner Capture Group's value is 'grouped' together with the Outer Capture Group. Therefore if Group 3 has a value A and Group 4 has a value B, then Group 3's value becomes AB. In other words, a nested Capture Group takes its value inclusive of all values of Capture Groups within.

2. The Outer Group is always evaluated first in RegEx, unlike in mathematics where the inner group is evaluated first, thus, the Outer Group is given the Group 3 designation while the Inner Group is assigned the next designation, 4.

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

So this got me thinking (uh-uh watch out, he's at it again!)… **Using Nested Capture Groups to Parse String**

Using: This is a test

Match: ((\D\w+){4})
Replace: \1 <space>

Name	New Name
This is a test	This is a test

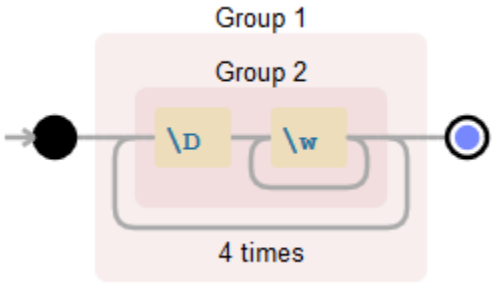
Not amazing in itself because you could have used many variations to just capture the string to EOL, but that is not what I am doing. I am capturing each word of the string separately with every iteration of the Quantifier.

Because the Quantifier is outside of the inner parentheses ('Nested') that separates Capture Group 4 from Capture Group 3, the entire sub-expression (\D\w+) is repeated and not just the last Metacharacter, i.e., (\D\w{4}) would repeat the \w and would end up only matching against the fourth word in the string = 'test'

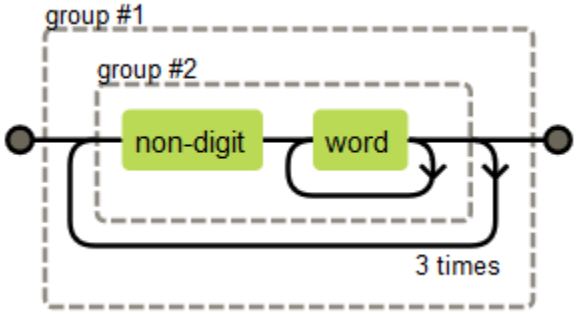
This is what it looks like:

Sample:

```
.........................
Match 1: This is a test
Group 1: This is a test
Group 2:  test
```




```
⊟Beginning match attempt at character 0
1  T
2  This
3  This
4  This is
5  This is
6  This is a
7  This is a
8  This is a test
9  This is a test
10 This is a test
   Match found in 10 steps
```



No backtracking!

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

So this got me thinking (uh-uh watch out, he's at it again!)… **Using Nested Capture Groups to Parse String** cont.

Using: This is a test

More importantly, I can control just how many words I wish to capture:

Replace: \1 <space>

Match: ((\D\w+){4})

Name	New Name
This is a test	This is a test

Match: ((\D\w+){3})

Name	New Name
This is a test	This is a

Match: ((\D\w+){2})

Name	New Name
This is a test	This is

Match: ((\D\w+){1})

Name	New Name
This is a test	This

So next time you have a string that contains separate words you wish to include or exclude, remember this, especially where \D* (without the nested group) would gather up too much of the string.

Tim Notes:

1. If you have more than one <space> or other character between words, use \D+ instead of just \D

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

So this got me thinking (uh-uh watch out, he's at it again!)… **Using Nested Capture Groups to Parse String** cont.

Using: This is a test

If you don't want to gather up each word, this isolates each word as well (with <space>):

Replace: \2

Match: ((\D\w+){4})

Name	New Name
This is a test	test

Match: ((\D\w+){3})

Name	New Name
This is a test	a

Match: ((\D\w+){2})

Name	New Name
This is a test	is

Match: ((\D\w+){1})

Name	New Name
This is a test	This

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

So this got me thinking (uh-uh watch out, he's at it again!)… **Using Nested Capture Groups to Parse String** cont.

Using: This is a test

Match: ((\D\w+){4})

Where:

\1 = This is a test \2 = <space> test

Nested Capture Groups -

Capture Group 1

Capture Group 2

((\D \w+) {4})

Tim Notes:

1. As already discussed, when you have nested Capture Groups, the outer group containing the entire expression is evaluated first (the opposite of what is done in mathematics), then the inner group, and that is why the inner group is assigned a later designation, '2' as opposed to '1'.

2. Any values of sub-Capture Groups within the Outer Group are 'grouped' together. So if Capture Group 1 has value A and Capture Group 2 has value B, Capture Group 1's value is AB. In other words, a nested Capture Group takes its value inclusive of all values of the Capture Groups within.

Tim Analysis:

1.\D	Match against any Non-Numeric Digit	Capture Group 1 = T
2. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = Th
3. +	Make it Greedy	Capture Group 1 = This
4. {4}	Repeat last expression to match 4 iterations	Capture Group 1 = This is a test Capture Group 2 = <space> test

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: This is a test

Alternatively you can use this to (Only) isolate words (without or without <space>):

Match: (.*?)((\s+)(\w+)){3}

Where

\1 = first word \2 = each additional word controlled by {n} with <space> \3 = <space>

\4 = each additional word controlled by {n} without <space>

In order to capture the first word, had to add the (.*?) otherwise matching against the <space> would match against the second word skipping the first. So Capture Group 1 will always have the value of the first word in the string and is unaffected by {n}.

Using a Replace String of \1

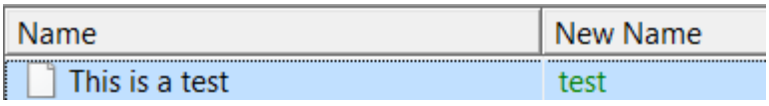
otherwise…

If you want the second word:

Match: (.*?)((\s+)(\w+)){1}
Replace \2

If you want the fourth word:

Match: (.*?)((\s+)(\w+)){3}
Replace \2



If you want the third word:

Match: (.*?)((\s+)(\w+)){2}
Replace \2

Tim Notes:

1. Using (.*?) does perform a little backtracking because it matches to EOL then backtracks to BOL.

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: This is a test

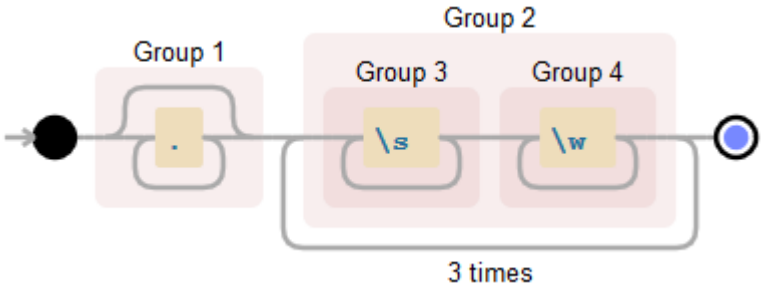
Alternatively you can use this to (Only) isolate words (without or without <space>): cont.

Match: (.*?)((\s+)(\w+)){3}

Where

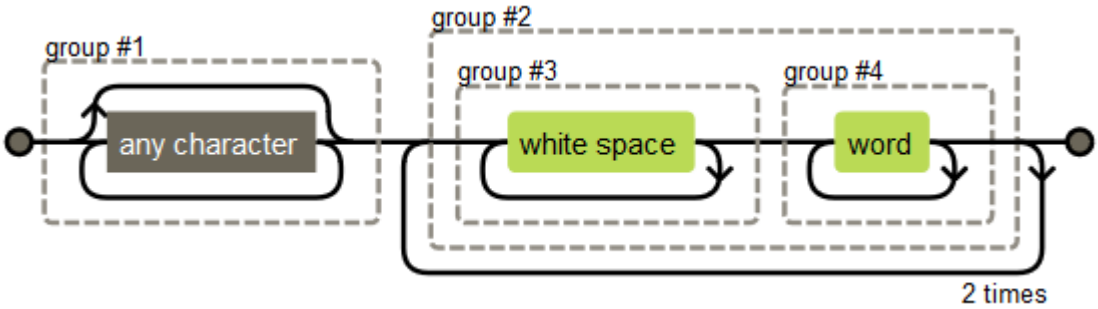
\1 = first word \2 = each additional word controlled by {n} with <space> \3 = <space>

\4 = each additional word controlled by {n} without <space>



Sample:

		Start	Length
Match 1:	This is a test	0	14
Group 1:	This	0	4
Group 2:	test	9	5
Group 3:		9	1
Group 4:	test	10	4



Tim Notes:

1. The sample run above is used with Quantifier set at {3}
2. Unfortunately, there is backtracking involved now. Anytime you have (.*) or (.*?) there will be backtracking.

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: This is a test

Alternatively you can use this to (Only) isolate words (without or without <space>): cont.

Match: (.*?)((\s+)(\w+)){3}

Where:

\1 = This \2 = <space> test \3 = <space> \4 = test

Nested Capture Groups -

		Capture Group 2		
Capture Group 1		Capture Group 3	Capture Group 4	
(.*?)	(	(\s+)	(\w+)	) {3}

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \s	Match against <space> character This is the <space> after 'This'	Capture Group 2 = <space> Capture Group 3 = <space> Changes Capture Group 1 = This
5. +	Make it Greedy	All Capture Groups = Unchanged
6. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = Unchanged Capture Group 2 = <space> i
7. +	Make it Greedy	Capture Group 2 = <space>is
8. {3}	Repeat last expression to match 3 iterations	Capture Group 4 = test Capture Group 2 = <space> test

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #1:

Match: ((\s*(\D*)\w*){3})
Replace: \1 <space>

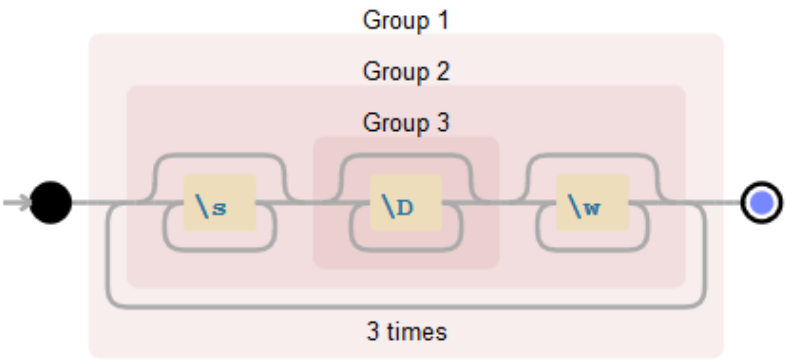
Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

Where:

\1 = gathers up each word based on quantifier \2 isolates each word (with <space>) based on quantifier with numeric
\3 = isolates each word (without <space>) based on quantifier but doesn't include numeric portion

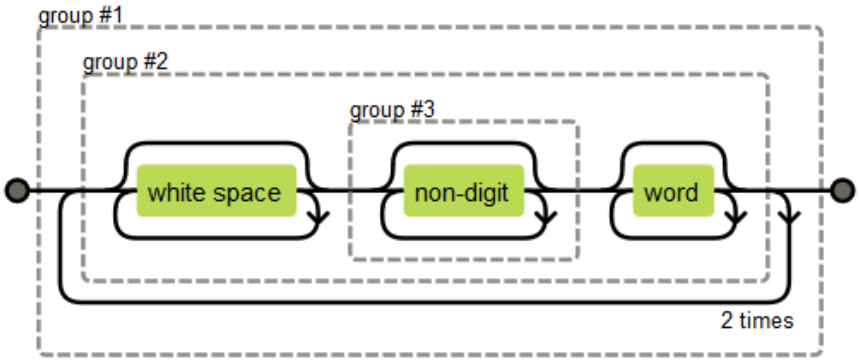
Sample:

```
..............................
Match 1: Chapter1  Lesson2 DNA
Group 1: Chapter1  Lesson2 DNA
Group 2:  DNA
Group 3: DNA
```




```
Beginning match attempt at character 0
1  ok
2  Chapter
3  Chapter
4  Chapter1
5  Chapter1
6  Chapter1  Lesson
7  Chapter1  Lesson
8  Chapter1  Lesson2
9  Chapter1  Lesson2
10 Chapter1  Lesson2 DNA
11 Chapter1  Lesson2 DNA
12 Chapter1  Lesson2 DNAok
13 Chapter1  Lesson2 DNA
14 Chapter1  Lesson2 DNA
Match found in 14 steps
```

No backtracking!



Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #1: cont.

Match: ((\s*(\D*)\w*){3})

So how does this work?
Just as before, it is based on the value of the Quantifier and the Replace String used. There are just more options this time.

Using a Replace String of \1 <space>

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1
Lesson 25 DNA	Lesson 25

{2}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2
Lesson 25 DNA	Lesson 25 DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

Using a Replace String of \2

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1
Lesson 25 DNA	Lesson 25

{2}

Name	New Name
Chapter1 Lesson2 DNA	Lesson2
Lesson 25 DNA	DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	DNA
Lesson 25 DNA	Lesson 25 DNA

Using a Replace String of \3

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter
Lesson 25 DNA	Lesson

{2}

Name	New Name
Chapter1 Lesson2 DNA	Lesson
Lesson 25 DNA	DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	DNA
Lesson 25 DNA	Lesson 25 DNA

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: Chapter1 Lesson2 DNA

These will work where numeric digits are present in the string Example #1: cont.

Match: ((\s*(\D*)\w*){3})

Where:

\1 = Chapter1 Lesson2 DNA \2 = <space>DNA \3 = DNA

Nested Capture Groups -

C a p t u r e G r o u p 1

Capture Group 2

Capture Group 3

((\s* (\D*) \w*) {3})

Tim Analysis:

1. \s	Match against <space> character This is the <space> after 'Chapter1'	Capture Group 1 = <space> Capture Group 2 = <space>
2. *	Make it Greedy Continued to match to EOL, then BOL	Capture Group 1 = BOL
3. \D	Match against any Non-Numeric Digit	Capture Group 3 = C Changes Capture Group 2 = C Changes Capture Group 1 = C
4. *	Make it Greedy	Capture Group 3 = Chapter Changes Capture Group 2 = Chapter Changes Capture Group 1 = Chapter
5. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = Unchanged Changes Capture Group 2 = Chapter1 Changes Capture Group 1 = Chapter1
6. *	Make it Greedy	All Capture Groups = Unchanged
7. {3}	Repeat last expression to match 3 iterations	Changes Capture Group 3 = DNA Changes Capture Group 2= <space> DNA Changes Capture Group 1 = Chapter1 Lesson2 DNA

Tim Notes:

1. When you think about it, steps 1-6 captured one section of the string. There are three sections – so you could say, {1} = 'Chapter1', {2} = '<space>Lesson2', and {3} = '<space>DNA', but the analysis isn't that simple.

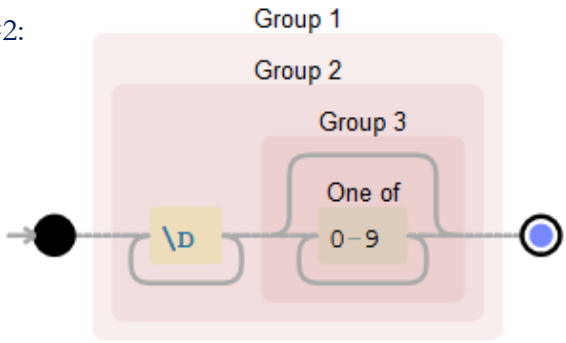
Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #2:



Match: ((\D*([0-9]*)){1})
Replace: \1 <space>

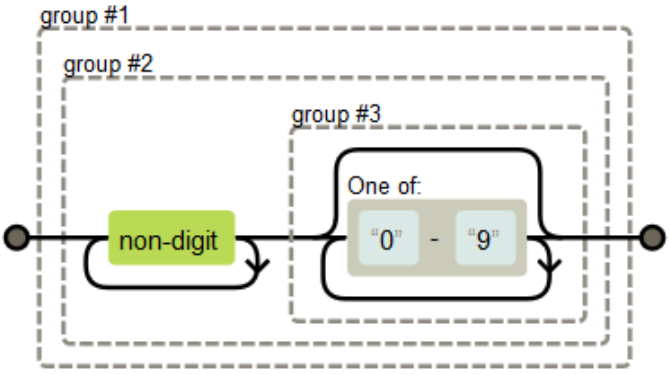
Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

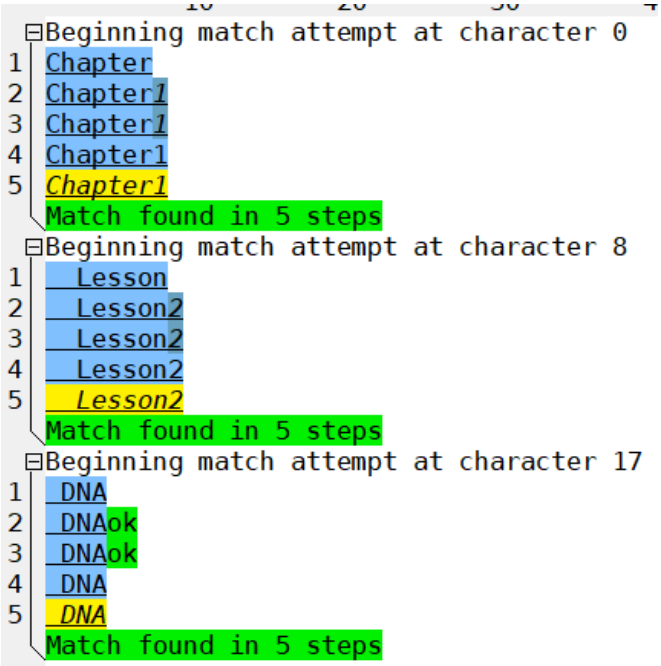
Where:

\1 = gathers up each word based on quantifier
\2 = isolates each word **(with <space>)** based on quantifier and includes numeric portion
\3 = numeric portion only

Sample:

	Start	Length
Match 1:	Chapter1	0	8
Group 1:	Chapter1	0	8
Group 2:	Chapter1	0	8
Group 3:	1	7	1
Match 2:	Lesson2	8	9
Group 1:	Lesson2	8	9
Group 2:	Lesson2	8	9
Group 3:	2	16	1
Match 3:	DNA	17	4
Group 1:	DNA	17	4
Group 2:	DNA	17	4
Group 3:		21	0





Tim Notes:

1. Although the photos show no backtracking, depending on the Quantifier value used, there will be some, but it is minimal as compared with other methods.

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #2: cont.

Match: ((\D*([0-9]*)){1})

Using a Replace String of \1<space>

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1
Lesson 25 DNA	Lesson 25

{2}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2
Lesson 25 DNA	Lesson 25 DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

Using a Replace String of \2

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1
Lesson 25 DNA	Lesson 25

{2}

Name	New Name
Chapter1 Lesson2 DNA	Lesson2
Lesson 25 DNA	DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	DNA
Lesson 25 DNA	Lesson 25 DNA

Using a Replace String of \3

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	1
Lesson 25 DNA	25

{2}

Name	New Name
Chapter1 Lesson2 DNA	2
Lesson 25 DNA	Lesson 25 DNA

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: Chapter1 Lesson2 DNA

These will work where numeric digits are present in the string Example #2: cont.

Match: ((\D*([0-9]*)){2})

Where:

\1 = Chapter1 Lesson2 \2 = <space>Lesson2 \3 = 2

Nested Capture Groups -

C a p t u r e G r o u p 1

Capture Group 2

Capture Group 3

((\D* ([0-9]*)) {1})

Tim Analysis:

1. \D	Match against any Non-Numeric Digit	Capture Group 1 = C Capture Group 2 = C
2. *	Make it Greedy	Capture Group 1 = Chapter Capture Group 2 = Chapter
3. [0-9]	Match against a class consisting of a Numeric digit 0-9	Capture Group 3 = 1 Capture Group 2 = Chapter1 Capture Group 1 = Chapter1
4. *	Make it Greedy This would match against additional numeric digits in the string, i.e. '25' using sample, 'Lesson 25 DNA'	All Capture Groups = Unchanged
5. {2}	Repeat last expression to match 2 iterations.	Capture Group 3 = 2 Capture Group 2 = <space> Lesson2 Capture Group 1 = Chapter1 Lesson2

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #3:

Match: ((\s*(\D*)([0-9]*)\w*){3})
Replace: \1 <space>

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

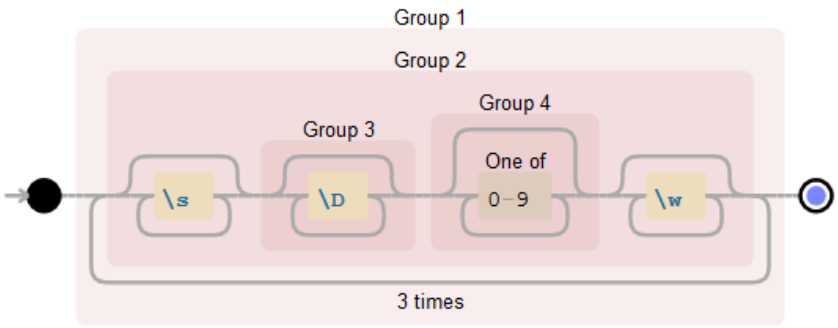
Where:

\1 = gathers up each word based on quantifier
\2 = isolates each word **(with <space>)** based on quantifier and includes numeric portion
\3 = isolates each word **(without <space>)** based on quantifier and includes numeric portion
\4 = numeric portion only

Sample:

```
                                  Start  Length
Match 1:  Chapter1  Lesson2 DNA       0      21
Group 1:  Chapter1  Lesson2 DNA       0      21
Group 2:   DNA                       17       4
Group 3:  DNA                        18       3
Group 4:                             21       0
```

```
Beginning match attempt at character 0
 1  ok
 2  Chapter
 3  Chapter
 4  Chapter1
 5  Chapter1
 6  Chapter1ok
 7  Chapter1
 8  Chapter1  Lesson
 9  Chapter1  Lesson
10  Chapter1  Lesson2
11  Chapter1  Lesson2
12  Chapter1  Lesson2ok
13  Chapter1  Lesson2
14  Chapter1  Lesson2 DNA
15  Chapter1  Lesson2 DNA
16  Chapter1  Lesson2 DNAok
17  Chapter1  Lesson2 DNAok
18  Chapter1  Lesson2 DNAok
19  Chapter1  Lesson2 DNA
20  Chapter1  Lesson2 DNA
Match found in 20 steps
```



Remove

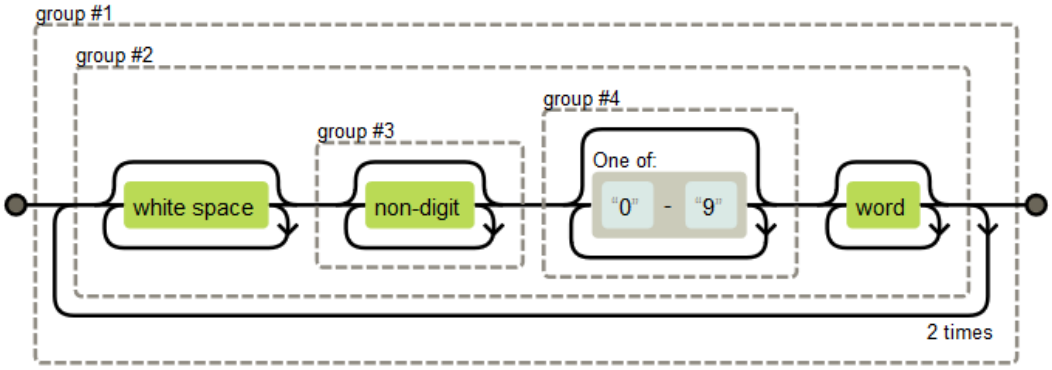
Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #3: cont.

Match: ((\s*(\D*)([0-9]*)\w*){3})



Using a Replace String of \1 <space>

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1
Lesson 25 DNA	Lesson 25

{2}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2
Lesson 25 DNA	Lesson 25 DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1 Lesson2 DNA
Lesson 25 DNA	Lesson 25 DNA

Using a Replace String of \2

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter1
Lesson 25 DNA	Lesson 25

{2}

Name	New Name
Chapter1 Lesson2 DNA	Lesson2
Lesson 25 DNA	DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	DNA
Lesson 25 DNA	Lesson 25 DNA

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Chapter1 Lesson2 DNA
Lesson 25 DNA

These will work where numeric digits are present in the string Example #3: cont.

Match: ((\s*(\D*)([0-9]*)\w*){3})

Using a Replace String of \3

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	Chapter
Lesson 25 DNA	Lesson

{2}

Name	New Name
Chapter1 Lesson2 DNA	Lesson
Lesson 25 DNA	DNA

{3}

Name	New Name
Chapter1 Lesson2 DNA	DNA
Lesson 25 DNA	Lesson 25 DNA

Using a Replace String of \4

Quantifier Value:

{1}

Name	New Name
Chapter1 Lesson2 DNA	1
Lesson 25 DNA	25

{2}

Name	New Name
Chapter1 Lesson2 DNA	2
Lesson 25 DNA	Lesson 25 DNA

Remove

Remove Email addresses from string cont.

Tim Contribution – cont.

Using: Chapter1 Lesson2 DNA

These will work where numeric digits are present in the string Example #3: cont.

Match: ((\s*(\D*)([0-9]*)\w*){3})

Where:

\1 = Chapter1 Lesson2 DNA \2 = <space>DNA \3 = DNA \4 = null

Nested Capture Groups –

C a p t u r e G r o u p 1

C a p t u r e G r o u p 2

Capture Group 3 Capture Group 4

((\s* (\D*) ([0-9]*) \w*) {3})

Tim Analysis:

1. \s	Match against a <space> character This is the <space> after 'Chapter1'	Capture Group 1 = <space> Capture Group 2 = <space>
2. *	Make it Greedy	All Capture Groups = BOL
3. \D	Match against any Non-Numeric Digit	Capture Group 1-3 = C
4. *	Make it Greedy	Capture Group 1-3 = Chapter
5. [0-9]	Match against a class consisting of a numeric digit 0-9	Capture Group 1-2 = Chapter1 Capture Group 3 = Unchanged Capture Group 4 = 1
6. *	Make it Greedy This would capture any additional numeric digits. i.e., for the sample, 'Lesson 25 DNA' this would be '25'	All Capture Groups = Unchanged
7. \w	Word Metacharacter. Match against any single letter, number or underscore.	Capture Groups 1-3 = Unchanged Capture Group 4 = null
8. *	make it Greedy	Capture Groups 1-3 = Unchanged Capture Group 4 = 1
9. {3}	Repeat last expression to match 3 iterations	Capture Group 1 = Chapter1 Lesson2 DNA Capture Group 2 = <space> DNA Capture Group 3 = DNA Capture Group 4 = null

Remove

Renaming files to only initials

jumpman one1 to j o1
jumpman one2 to j o2
jumpman two1 to j t1

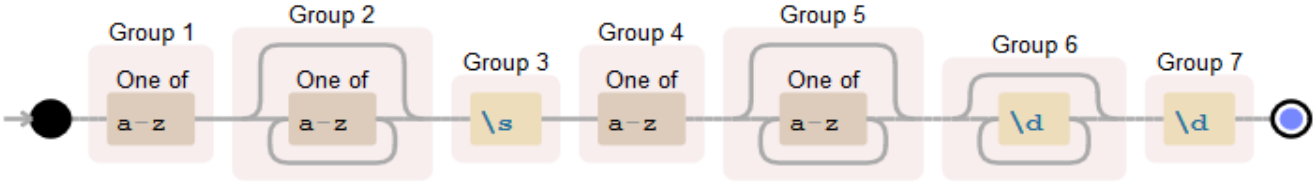
Observations:

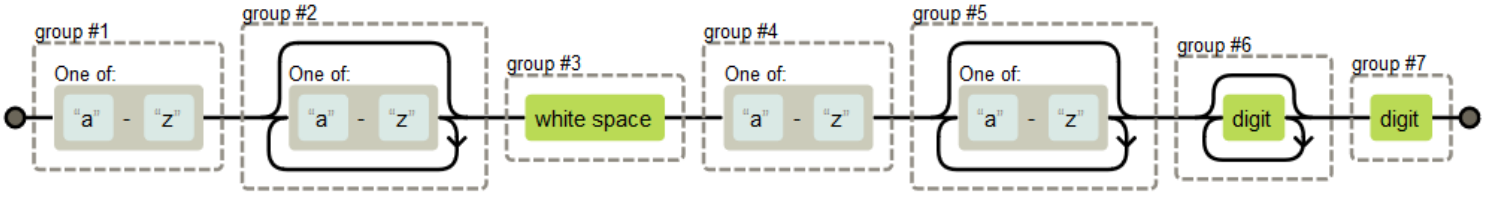
1. Take initials of First text, i.e., j (umpman) and from last text, i.e. o (ne).
2. Add <space> between initials.
3. No <space> between last initial and remaining text (a numeric digit in these samples)
4. It's always the first initial of the two words, but the numeric digits are variable length between 1 to 8 or so.

Example #1:

Match: ([a-z])([a-z]*)(\s)([a-z])([a-z]*)(\d*)(\d)
Replace: \1 \4 \7

Name	New Name
jumpman one1	j o 1
jumpman two1	j t 1
jumpman one2	j o 2





Sample:

```
.....................Start·Length···
Match 1: jumpman one1      0      12
Group 1: j                 0       1
Group 2: umpman            1       6
Group 3:                   7       1
Group 4: o                 8       1
Group 5: ne                9       2
Group 6:                  11       0
Group 7: 1                11       1
```

Tim Notes:

1. Change all instances of [a-z] class to [A-Za-z] to handle both uppercase and lowercase words in string.

Remove

Renaming files to only initials cont.

Enlarged Sample Group:

Jumpman one1	jumpman Three00080
jumpman One2	jumpman Four45 Four46
jumpman Four23 45	The Rain in Spain00010
jumpman two1	Chapter1 Lesson2 DNA

But you don't have to (Capture) what you don't want to back reference in the Replace String (Be aware of <spaces> in both RegEx and Replace strings).

Example #2:

Result gives the very last number:

Match: (.).+ (.).+(\d)
Replace: \1 \2\3

Name	New Name
Jumpman one1	J o1
jumpman One2	j O2
jumpman Four23 45	j F5
jumpman two1	j t1
jumpman Three00080	j T0
jumpman Four45 Four46	j F6
The Rain in Spain00010	T S0
Chapter1 Lesson2 DNA	C L2

Tim Notes:

1. Takes the initial from the first and last word in the string coupled with the last numeric digit found in the string.
2. Recognizes both upper and lowercase words.
3. The '0' of 'jumpman Three00080' is the last '0' of '00080'. The same with 'The Rain in Spain00010'.

Example #3:

Result gives the very first number:

Match: (.).+ (.).+?(\d)
Replace: \1 \2\3

Name	New Name
Jumpman one1	J o1
jumpman One2	j O2
jumpman Four23 45	j F2
jumpman two1	j t1
jumpman Three00080	j T0
jumpman Four45 Four46	j F4
The Rain in Spain00010	T S0
Chapter1 Lesson2 DNA	C L2

Tim Notes:

1. Takes the initial from the very first and last word in the string coupled with the first numeric digit found in the string.
2. Recognizes both upper and lowercase words.
3. The '0' of 'jumpman Three00080' is the first '0' of '00080'. The same with 'The Rain in Spain00010'. The '4' of 'jumpman Four45 Four46' is the first four of '45'

Remove

Renaming files to only initials cont.

Enlarged Sample Group:

Jumpman one1
jumpman One2
jumpman Four23 45
jumpman two1
jumpman Three00080
jumpman Four45 Four46
The Rain in Spain00010
Chapter1 Lesson2 DNA

Example #4:

Result gives the whole number:

Match: (.).+ (.).+?(\d+)
Replace: \1 \2\3

Name	New Name
Jumpman one1	J o1
jumpman One2	j O2
jumpman Four23 45	j F23
jumpman two1	j t1
jumpman Three00080	j T00080
jumpman Four45 Four46	j F46
The Rain in Spain00010	T S00010
Chapter1 Lesson2 DNA	C L2

Tim Notes:

1. Takes the first and last word of the string coupled with the whole number immediately following the last word in the string.
2. Has to be words, not just numbers by themselves – i.e., 'jumpman Four23 45' takes the words 'jumpman' and 'Four' and the numeric '23'.

Contributor Analysis:

1. The dot . means find one char.
2. .+ means find as many characters till there is an blank.
3. \d means search for numbers
4. \d+ means search for one or more numbers
5. question mark ? means find only as less as it's enough (Non Greedy or Lazy)
6. \s for blank signs is correct RegEx syntax, but not needed within BRU

Forum Contributor: jimwillsher (Original Publisher BRU), Stefan

Remove

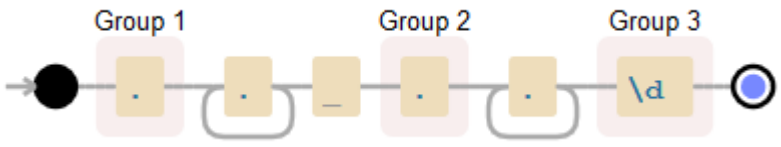
Renaming files to only initials cont.

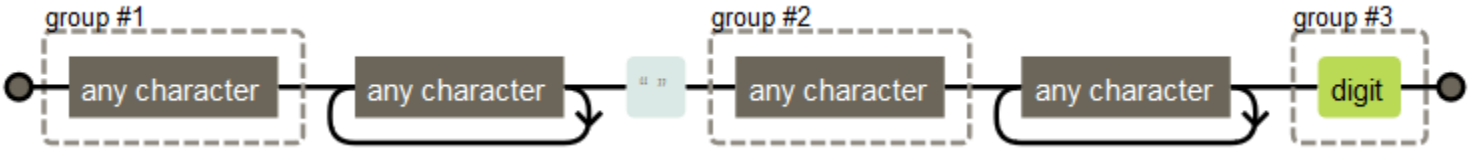
Result gives the very last number:

Match: (.).+ (.).+(\d)

Sample:

```
..............................
Match 1: jumpman Four45 Four46
Group 1: j
Group 2: F
Group 3: 6
```



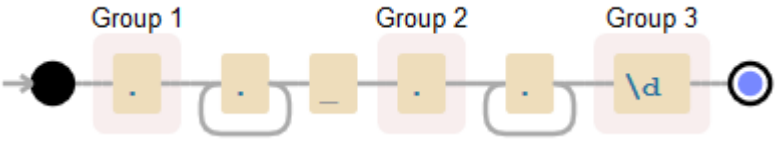


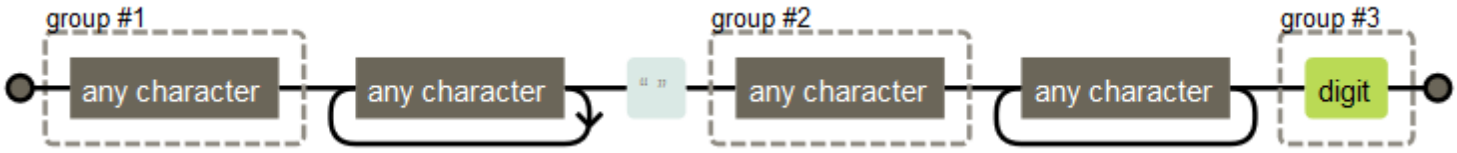
Result gives the very first number:

Match: (.).+ (.).+?(\d)

Sample:

```
..............................
Match 1: jumpman Four45 Four4
Group 1: j
Group 2: F
Group 3: 4
```



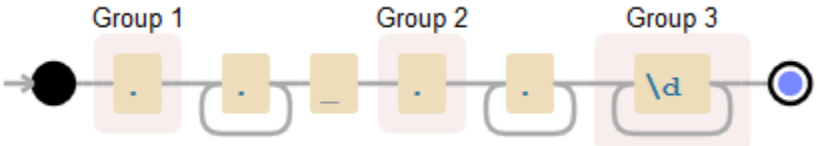


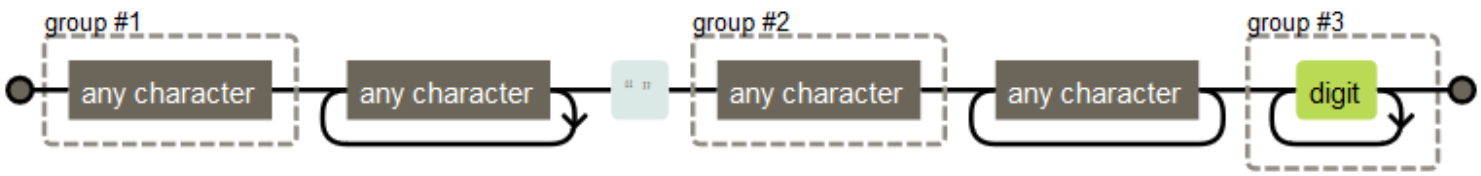
Result gives the whole number:

Match: (.).+ (.).+?(\d+)

Sample:

```
..............................
Match 1: jumpman Four45 Four46
Group 1: j
Group 2: F
Group 3: 46
```





Remove

Renaming files to only initials cont.

Example #1:

Using: Jumpman one1

Match: ([a-z])([a-z]*)(\s)([a-z])([a-z]*)(\d*)(\d)

Tim Analysis:

1. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 1 = j
2. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 2= u
3. *	Make it Greedy	Capture Group 2 = umpman
4. \s	Match against <space>	Capture Group 3 = <space>
5. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 4 = o
6. [a-z]	Match against a class consisting of a lowercase letter	Capture Group 5= n
7. *	Make it Greedy	Capture Group 5 = ne
8. \d	Match against a numeric digit	Capture Group 6 = 1
9. *	Make it Greedy	Capture Group 6 = Unchanged
10. \d	Match against a numeric digit	Capture Group 7 = 1
		Changes Capture Group 6 = null

Example #2: Result gives the very last number

Using: jumpman Four45 Four46

Match: (.).+ (.).+(\d)

Tim Analysis:

1. .	Match against any character	Capture Group 1 = j
2. .	Match against any character (Not Captured) = 'u'	
3. +	Make it Greedy (Not Captured) = 'umpman Four45 Four46'	
4. <space>	Match against <space> (Not Captured) = 'umpman Four45'	
5. .	Match against any character	Capture Group 2 = F
6. .	Match against any character (Not Captured) = 'o'	
7. +	Make it Greedy (Not Captured) = 'our46'	
8. \d	Match against numeric digit	Capture Group 3 = 6

Tim Notes:

1. Remember that these examples take the first and last word so the very last number refers to the last number of the last word, 'Four46' and ignores 'four45'.

Remove

Renaming files to only initials cont.

Example #3: Result gives the very first number

Using: jumpman Four45 Four46

Match: (.).+ (.).+?(\d)

Tim Analysis:

1. .	Match against any character	Capture Group 1 = j
2. .	Match against any character (Not Captured) = 'u'	
3. +	Make it Greedy (Not Captured) = 'umpman Four45 Four46'	
4. <space>	Match against <space> (Not Captured) = 'umpman Four45'	
5. .	Match against any character	Capture Group 2 = F
6. .	Match against any character (Not Captured) = 'o'	
7. +	Make it Greedy (Not Captured) = 'our46'	
8. ?	But not too Greedy = o This is the 'o' of 'Four46'	
8. \d	Match against numeric digit	Capture Group 3 = 4

Example #4: Result gives the whole number

Using: jumpman Four45 Four46

Match: (.).+ (.).+?(\d+)

Tim Analysis:

1. .	Match against any character	Capture Group 1 = j
2. .	Match against any character (Not Captured) = 'u'	
3. +	Make it Greedy (Not Captured) = 'umpman Four45 Four46'	
4. <space>	Match against <space> (Not Captured) = 'umpman Four45'	
5. .	Match against any character	Capture Group 2 = F
6. .	Match against any character (Not Captured) = 'o'	
7. +	Make it Greedy (Not Captured) = 'our46'	
8. ?	But not too Greedy = o. This is the 'o' of 'Four46'	
9. \d	Match against numeric digit	Capture Group 3 = 4
10. +	Make it Greedy	Capture Group 3 = 46

Tim Notes:

1. Remember that these examples take the first and last word so the very first number refers to the first number of the last word, 'Four46' and ignores 'four45'.

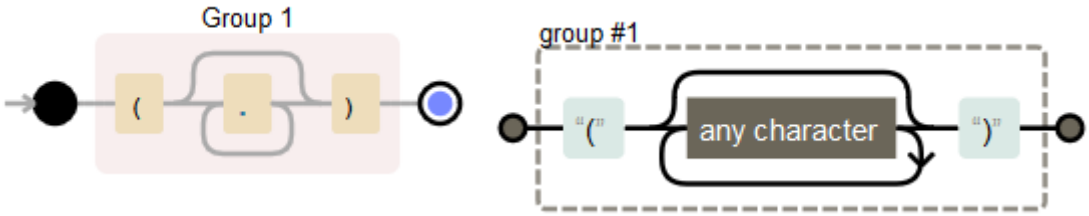
Remove

A simple Lesson on RegEx Remove

Contributor Example #1: This isolates the (1234)

filename (1234).exe to filename.exe

Match: (\(.*\))
Replace: \1



Sample:

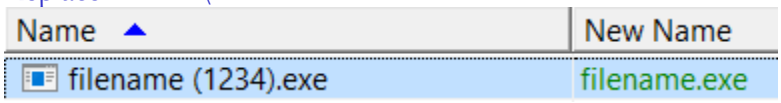
```
.................
Match 1: (1234)
Group 1: (1234)
```

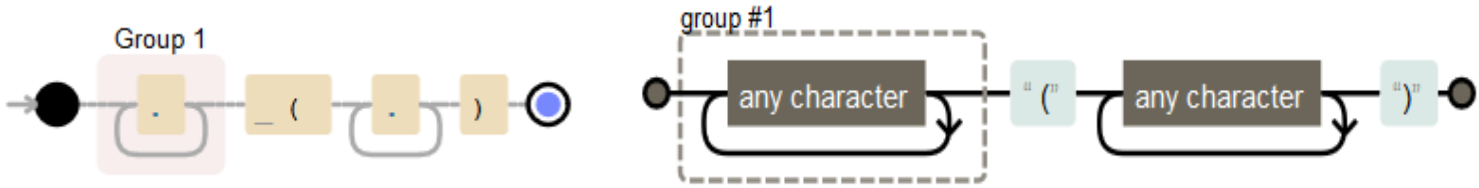
But you have to search and find and replace what you want to keep as well:

Contributor Example #2:

filename (1234).exe to filename

Match: (.+) \(.+\)
Replace: \1





Sample:

```
.........................
Match 1: filename (1234)
Group 1: filename
```

Remove

A simple Lesson on RegEx Remove cont.

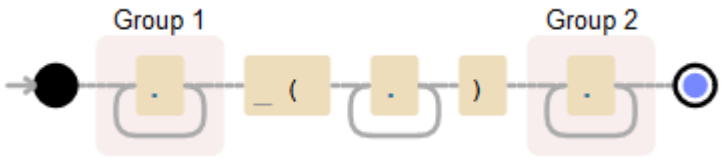
Contributor Example #3:

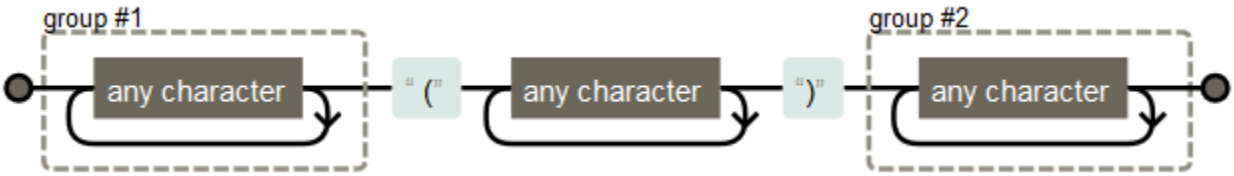
filename (1234).exe	to	filename
filename (1234) somestring.exe	to	filename

Match: (.+) \(.+\)(.+)
Replace: \1

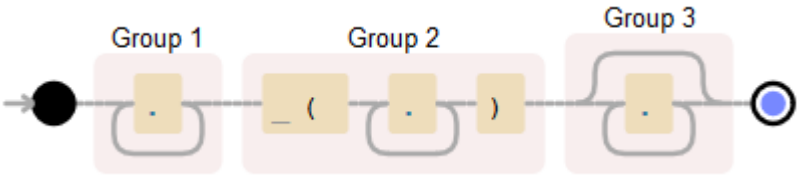
See Example #3 Sample #2

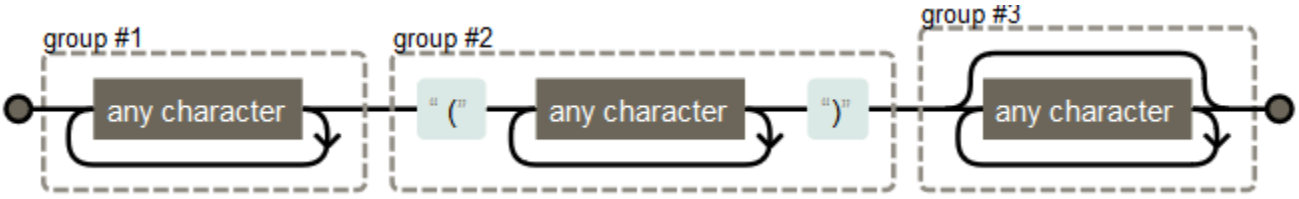
Name	New Name
filename (1234).exe	filename.exe
filename (1234) somestring.exe	filename.exe





Example #3 Sample #1: Using (.+)(\(.+\))(.*)



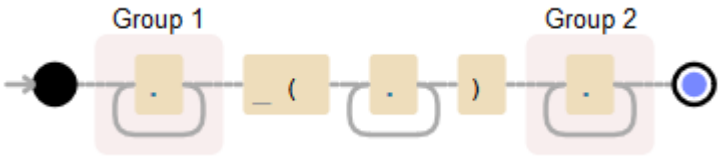


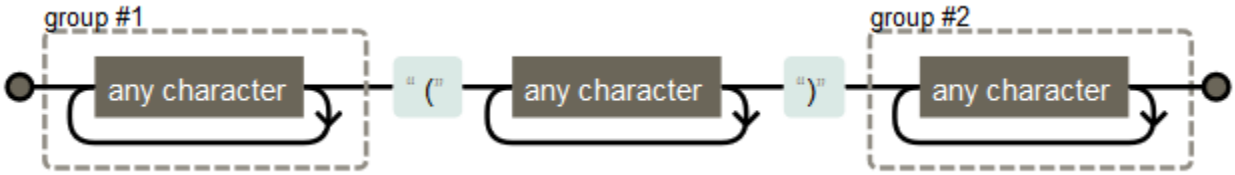

```
·····························Start·Length··
Match 1: filename (1234)          0      15
Group 1: filename                 0       8
Group 2:  (1234)                  8       7
Group 3:                         15       0
```

Remove

A simple Lesson on RegEx Remove cont.

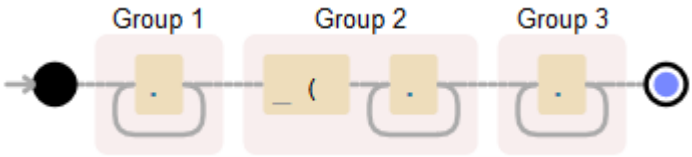
Example #3 Sample #2: Using (.+) \(.+\)(.+)

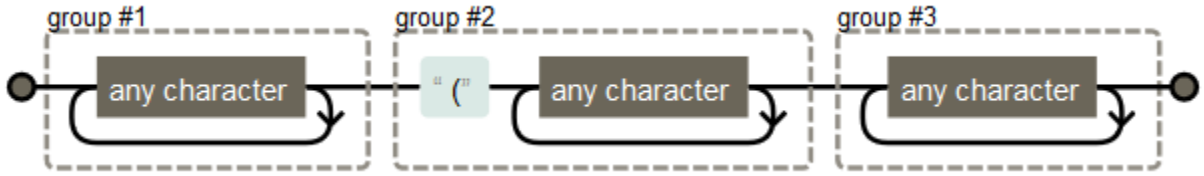





```
.....................................
Match 1: filename (1234) somestring
Group 1: filename
Group 2:  somestring
```

Example #3 Sample #3: Using (.+)(\(.+)(.+)






```
.....................................
Match 1: filename (1234) somestring
Group 1: filename
Group 2:  (1234) somestrin
Group 3: g
```

Remove

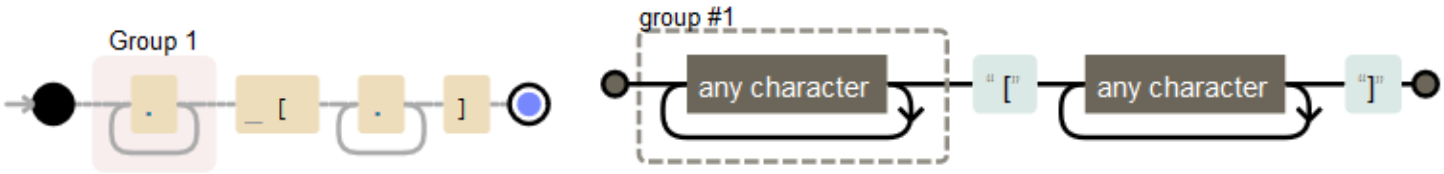
A simple Lesson on RegEx Remove cont.

Contributor Example #4:

filename (1234).exe	to	Ignore
filename (1234) somestring.exe		Ignore
filename [1234].exe	to	filename

Match: (.+) \[.+\]
Replace: \1

Name	New Name
filename (1234).exe	filename (1234).exe
filename (1234) somestring.exe	filename (1234) somestring.exe
filename [1234].exe	filename.exe



Sample:

```
.........................
Match 1: filename [1234]
Group 1: filename
```

Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against <space> (Not Captured) Not Captured = <space>	Capture Group 1 = filename
4. \[	Match against left bracket (Not Captured) Not Captured = '<space>['	Capture Group 1 = Unchanged
5. .	Match against any character (Not Captured) Not Captured = '<space>[1'	Capture Group 1 = Unchanged
6 +	Make it Greedy (Not Captured) Not Captured = '<space>[1234]'	Capture Group 1 = Unchanged
7. \]	Match against right bracket (Not Captured) Not Captured = Unchanged	Capture Group 1 = Unchanged

Tim Notes:

1. You can see that if the goal is to isolate 'filename' by itself, this was done by step 3. The rest of the RegEx is unnecessary.

Remove

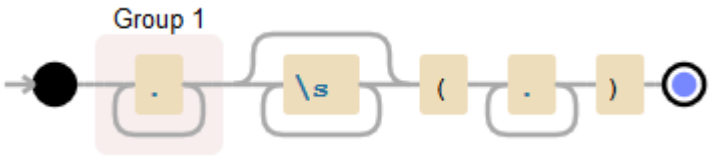
A simple Lesson on RegEx Remove cont.

Contributor Example #5:

filename (1234).exe	to	filename
filename (1234) somestring.exe	to	filename
filename [1234].exe	to	Ignore
filename(1234).exe	to	filename

Match: (.+)\s*\(.+\)
Replace: \1

Name	New Name
filename (1234).exe	filename .exe
filename (1234) somestring.exe	filename .exe
filename [1234].exe	filename [1234].exe
filename(1234).exe	filename.exe



Sample:

```
.........................
Match 1: filename(1234)
Group 1: filename
```

Contributor Notes:

1. \s* search for zero or more blanks.

Remove

A simple Lesson on RegEx Remove cont.

Using: filename (1234).exe

Contributor Example #1: This isolates the (1234)

Match: (\(.*\))

Tim Analysis:

1. (	Match against left parenthesis	Capture Group 1 = (
2. .	Match against any character	Capture Group 1 = (1
3. *	Make it Greedy	Capture Group 1 = (1234)
4.)	Match against right parenthesis	Capture Group 1 = Unchanged

Tim Notes:

1. The Contributor tends to include a match against the right parenthesis in several of these examples – most of which are not needed since the .* captured what was required by itself. In fact in one of the examples, this caused a problem.

Using: filename (1234).exe

Contributor Example #2:

Match: (.+) \(.+\)

Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against <space> (Not Captured) Not Captured = <space>	Capture Group 1 = filename
4. (	Match against left parenthesis (Not Captured) Not Captured = '<space>('	Capture Group 1 = Unchanged
5. .	Match against any character (Not Captured) Not Captured = '<space>(1'	Capture Group 1 = Unchanged
6 +	Make it Greedy (Not Captured) Not Captured = '<space>(1234)'	Capture Group 1 = Unchanged
7.)	Match against right parenthesis (Not Captured) Not Captured = Unchanged	Capture Group 1 = Unchanged

Tim Notes:

1. If the goal is to isolate 'filename' by itself, this was done by step 3. The rest is unnecessary.

Remove

A simple Lesson on RegEx Remove cont.

filename (1234).exe
filename (1234) somestring.exe

Contributor Example #3:

Match: (.+) \(.+\)(.+)

Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string)
3. <space>	Match against <space> (Not Captured) Backtracks to first <space> after 'filename'	Changes Capture Group 1 = filename
4 (	Match against right parenthesis (Not Captured)	
5. .	Match against any character (Not Captured) Not Captured = '<space>(1'	
6. +	Make it Greedy Not Captured = '<space>(1234)'	
7.)	Match against left parenthesis (Not Captured) Not Captured = Unchanged – at EOL	

Tim Notes:

1. This is where a problem comes in – the original RegEx has (.+), however because the current position is at EOL, this will fail and the RegEx ends using the string 'filename (1234).exe'. This *will* work with 'filename (1234) somestring.exe' which has additional text to capture after the '(1234)'. The RegEx is too restrictive to be useful.

2. There are two workarounds:
   a. Enable 'Rename File Extensions as Being Part of File Name' so that the '.exe' will be treated as additional text after the '(1234)' of 'filename (1234) and will match.
   b. Change the RegEx one of two ways:

1. Match:	(.+)(\(.+)(.+)	The addition of \) is not required to perform the match. See Sample #3
2. Match:	(.+)(\(.+\))(.*)	By changing the Quantifier from Match one or More times (+) to Match zero or More times (*), it will match. See Sample #1

8. .	Match against any character	Capture Group 2 = No match or No Value
9. *	Match against remainder of string In the case of 'filename (1234)' a Zero Occurrence Match takes place. If the string has additional text after '(1234)' it would be captured in Capture Group 2.	Capture Group 2 = null

Remove

A simple Lesson on RegEx Remove cont.

Using: filename(1234).exe

Contributor Example #5:

Match: (.+)\s*\(.+\)

Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \s	Match against <space> (Not Captured) Not Captured = <space> This is the <space> after 'filename'	Capture Group 1 = filename
4. *	Make it Greedy (Not Captured) Not Captured = null Current position = EOL Backtracks	Changes Capture Group = <entire string>
4. (	Match against left parenthesis (Not Captured) Not Captured = '('	Changes Capture Group 1 = filename
5. .	Match against any character (Not Captured) Not Captured = '(1'	Capture Group 1 = Unchanged
6 +	Make it Greedy (Not Captured) Not Captured = '(1234)'	Capture Group 1 = Unchanged
7.)	Match against right parenthesis (Not Captured) Not Captured = Unchanged	Capture Group 1 = Unchanged

Tim Notes:

1. This uses the Whitespace Metacharacter to 'stand-in' for the <space> character. In BRU and any program that uses a single line RegEx, the Whitespace character can only match against a <space> character. It isn't until you get into programs that support multiline RegEx that it makes a difference.

2. If the goal is to isolate 'filename' by itself, this was done by step 3. The rest of the RegEx becomes unnecessary.

Forum Contributor: AbteriX

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution:

Want to match them all?

The goal here is to make it as universal as possible without matching against <spaces> or individual parentheses or brackets. I start off with various, almost convoluted, methods to demonstrate various tactics such as the use of Negated Classes, Word Boundary Metacharacters, Alpha and Numeric classes and Negated Word Characters before I simplify it down.

Completely parsed

Tim Contribution Example #1:

Match: (.+)([^([])(\b.+)(\b[0-9]+)(\W)(.*)
Replace: \1\2

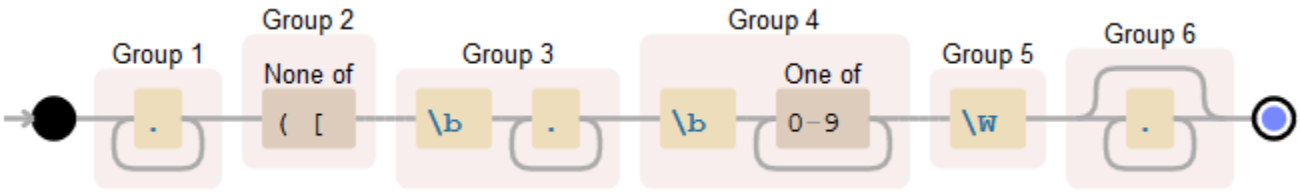
Name	New Name
filename (1234).exe	filename.exe
filename (1234) somestring.exe	filename.exe
filename [1234].exe	filename.exe
filename(1234).exe	filename.exe

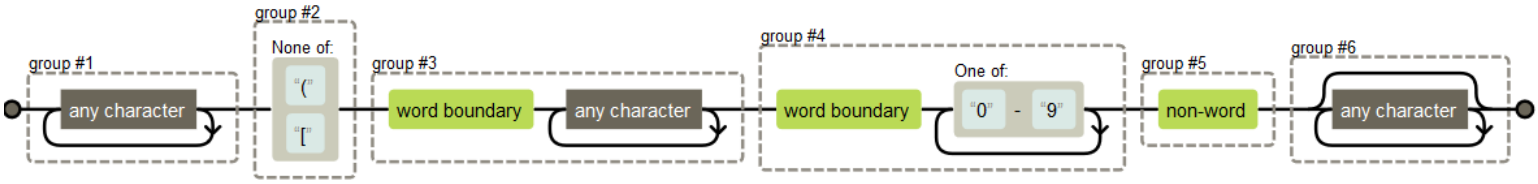
Sample:

		Start	Length
Match 1:	filename(1234)	0	14
Group 1:	filenam	0	7
Group 2:	e	7	1
Group 3:	(	8	1
Group 4:	1234	9	4
Group 5:	)	13	1
Group 6:		14	0

Where:

\1 = filenam \2 = e \3 = <space> <left bracket> or <space> <left parenthesis> or <left parenthesis>
\4 = 1234 \5 = <right bracket> or <right parenthesis> \6 = null or '<space>somestring'





Tim Notes:

1. Group 6 is reserved for additional text after the numeric. If no text then a Zero Occurrence Match takes place resulting in a null value.

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution: cont.

Completely parsed cont.

Tim Contribution Example #2:

Match: ([a-z]*)(\W+)(\d*)(\W)(.*)
Replace: \1

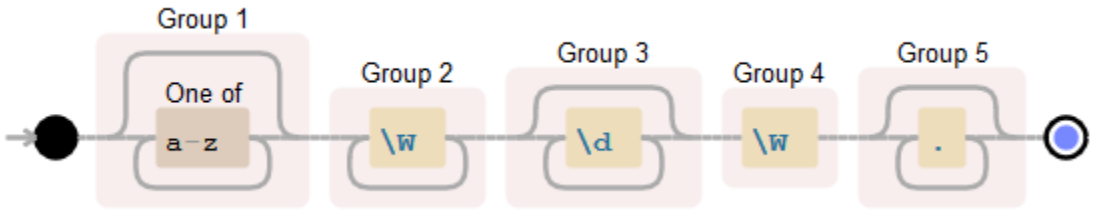
Name	New Name
filename (1234).exe	filename.exe
filename (1234) somestring.exe	filename.exe
filename [1234].exe	filename.exe
filename(1234).exe	filename.exe

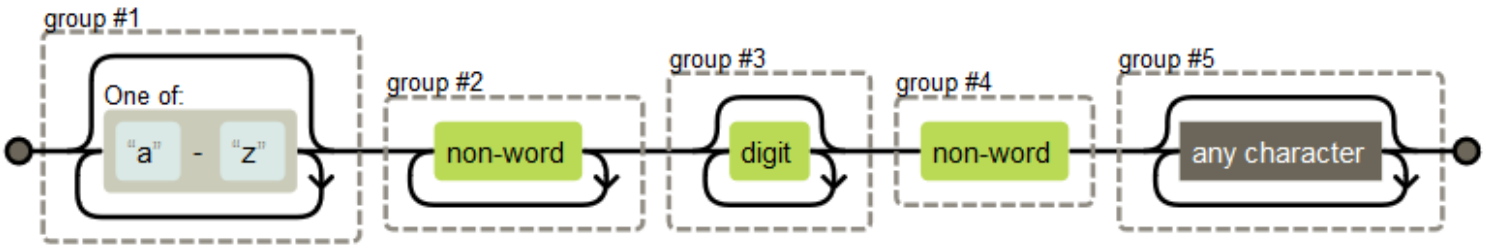
Sample:

```
                        Start  Length
Match 1: filename(1234)     0      14
Group 1: filename           0       8
Group 2: (                  8       1
Group 3: 1234               9       4
Group 4: )                 13       1
Group 5:                   14       0
```

Where:

\1 = Filename \2 = <space> <left bracket> or <space> <left parenthesis> or <left parenthesis>
\3 = 1234 \4 = <right bracket> or <right parenthesis> \5 = null or '<space>somestring'





Tim Notes:

1. Group 5 is reserved for additional text after the numeric. If no text then a Zero Occurrence Match takes place resulting in a null value.

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution: cont.

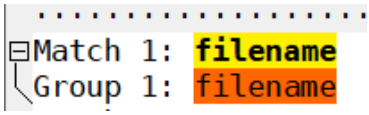
Or simplify it without capturing and parsing everything

Tim Contribution Example #3:

Match: ([A-Za-z]*)
Replace: \1

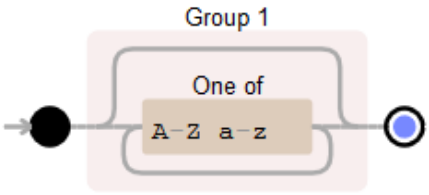
Name	New Name
filename (1234).exe	filename.exe
filename (1234) somestring.exe	filename.exe
filename [1234].exe	filename.exe
filename(1234).exe	filename.exe

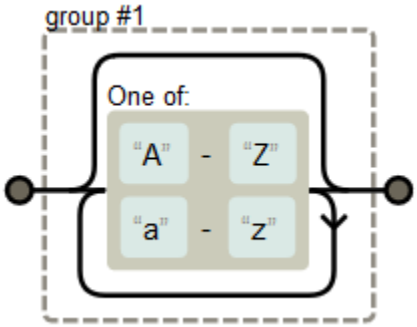
Sample:



Where:

\1 = filename





Using: filename (1234).exe

Tim Analysis:

1. [A-Za-z])	Match against a class consisting of an uppercase or lowercase letter	Capture Group 1 = f
2. *	Make it Greedy	Capture Group 1= filename

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution: cont.

Or simplify it without capturing and parsing everything cont.

Tim Contribution Example #4:

Match: (.*?)\W
Replace: \1

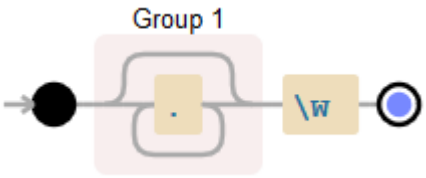
Name	New Name
filename (1234).exe	filename.exe
filename (1234) somestring.exe	filename.exe
filename [1234].exe	filename.exe
filename(1234).exe	filename.exe

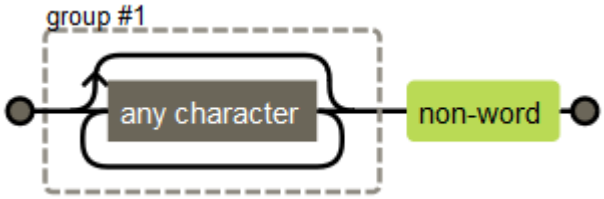
Sample:

```
.....................
Match 1: filename(
Group 1: filename
```

Where:

\1 = filename





Using: filename (1234).exe

Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = Beginning of string
4. \W	Match against Negated Word Character (Not Captured)	Changes Capture Group 1 = filename
	Matches a Non-Word Character.	
	Not Captured = <space> after filename	

Tim Notes:

1. Wherever 'beginning of string' appears, it refers to Beginning of Line or BOL.

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution: cont.

How about just the numeric by itself?

Tim Contribution Example #5:

Match: ([0-9]+)
Replace: \1

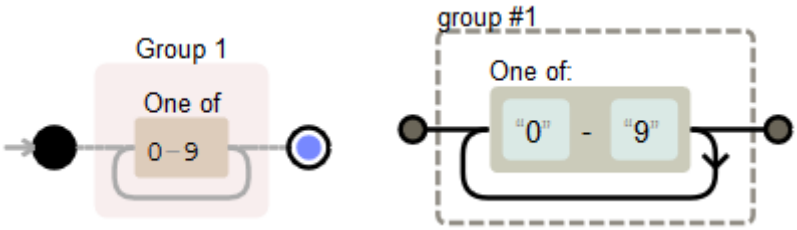
Name	New Name
filename (1234).exe	1234.exe
filename (1234) somestring.exe	1234.exe
filename [1234].exe	1234.exe
filename(1234).exe	1234.exe

Sample:

```
Match 1: 1234
Group 1: 1234
```

Where:

\1 = 1234



Using: filename (1234).exe

Tim Analysis:

1. [0-9]	Match against a class consisting of a numeric digit	Capture Group 1 = 1
2. +	Make it Greedy	Capture Group 1 = 1234

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution: cont.

Using: filename (1234).exe

Tim Contribution Example #1:

Match: (.+)([^([])(\b.+)(\b[0-9]+)(\W)(.*)

Where:

\1 = filenam \2 = e \3 = <space> <left bracket> or <space> <left parenthesis> or <left parenthesis>
\4 = 1234 \5 = <right bracket> or <right parenthesis> \6 = null or '<space>somestring'

Tim Analysis:

1. .	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. [^ ([ ]	Ignore the <left parenthesis> or <left bracket> in a Negated Class	Changes Capture Group 1 = filename (1234 Capture Group 2 =)
4. \b	Match against a Word Boundary Current position = EOL	Capture Group 3 = null

Tim Notes:

1. In this sample, step 4, the current position = EOL, but in the sample, 'filename (1234) somestring' step 4, the current position = <space> after (1234).

5. .	Match against any character	Capture Group 3 =) Changes Capture Group 2 = 4 Changes Capture Group 1 = filename (123
6. +	Make it Greedy	No Changes
7. \b	Match against a Word Boundary Current position = EOL	Capture Group 4 = null Changes Capture Group 3 = <space>1234 Changes Capture Group 2 = e Changes Capture Group 1 = filenam
8. [0-9]	Match against a class consisting of a numeric digit	Capture Group 4 = 1 Changes Capture Group 3 = (All Others = No Change
9. +	Make it Greedy	Capture Group 4 = 1234 All Others = No Change
10. \W	Match against a Negated Word Character	Capture Group 5 =) All Others = No Change
11. .*	Match against remainder of string Zero Occurrence Match results in null value	Capture Group 6 = null

Remove

A simple Lesson on RegEx Remove cont.

Tim Contribution: cont.

Using: filename (1234).exe

Tim Contribution Example #2:

Match: ([a-z]*)(\W+)(\d*)(\W)(.*)

Where:

\1 = Filename \2 = <space> <left bracket> or <space> <left parenthesis> or <left parenthesis>
\3 = 1234 \4 = <right bracket> or <right parenthesis> \5 = null or '<space>somestring'

Tim Analysis:

1. [a-z])	Match against a class consisting of a lowercase letter	Capture Group 1 = f
2. *	Make it Greedy	Capture Group 1 = filename
3. \W	Match against a Negated Word Character This is the <space> after 'filename'	Capture Group 2 = <space>
4. +	Make it Greedy	Capture Group 2 = <space>(
5. \d	Match against numeric digit	Capture Group 3 = 1
6. *	Make it Greedy	Capture Group 3 = 1234
7. \W	Match against a Negated Word Character	Capture Group 4 =)
8. .*	Match against remainder of string Zero Occurrence Match results in null value	Capture Group 5 = null

Tim Notes:

1. Throughout the book, I typically refer to 'null' rather than the alternative "", but it is the same thing. Usually the result of a Zero Occurrence Match or if the RegEx Engine has positioned itself to a character position that holds no value. These would include BOL, EOL or the positions in between the characters – what I refer to as 'just before' or 'just after' a certain character.

2. No Value, on the other hand refers to No Match was made at that point.

3. Invalid refers to a possible match, but may be because of a syntax error, or because of an expression made optional, or by other means, that may have made the previous Capture Group's value, invalid. In the case of an optional expression, the RegEx Engine reverts back to the point before the expression was evaluated as if it never existed, thereby possibly invalidating previous captured values.

Remove

Rename to remove UTC (Coordinated Universal Time) Date/Time/Zone

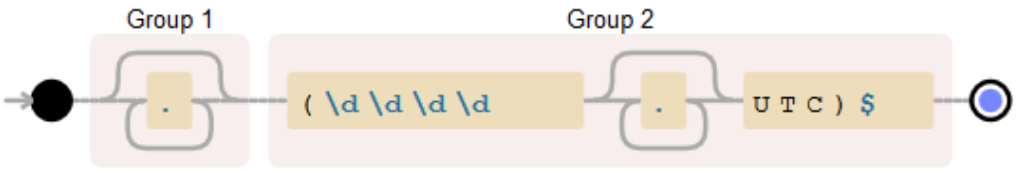
File Name (2015_12_21 19_28_46 UTC).docx	to	File Name.docx
Longer_File_name (2016_01_30 07_16_32 UTC).jpg	to	Longer_File_name.jpg
File-name (9-5) XXXXXXX (1)(2014-09-02 09-27-03 UTC).xlsx	to	File-name (9-5) XXXXXXX (1).xlsx
File Name.pdf	to	Ignore

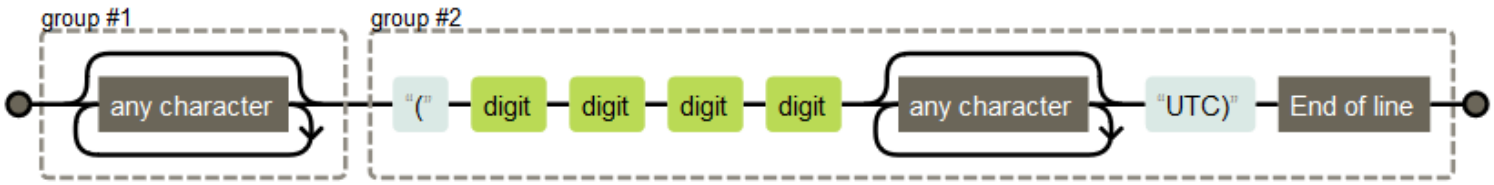
Observations:

1. Recovered files from crash, data recovery software appended UTC timestamps.
2. Remove UTC Timestamps from only those files affected.
3. Some files have leading <spaces> also as a result.
4. Some UTC Timestamps use <hyphen>, others use underscores

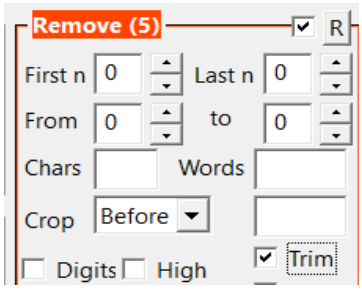
1. RegEx –

Match: (.*)(\(\d\d\d\d.*UTC\)$)
Replace: \1

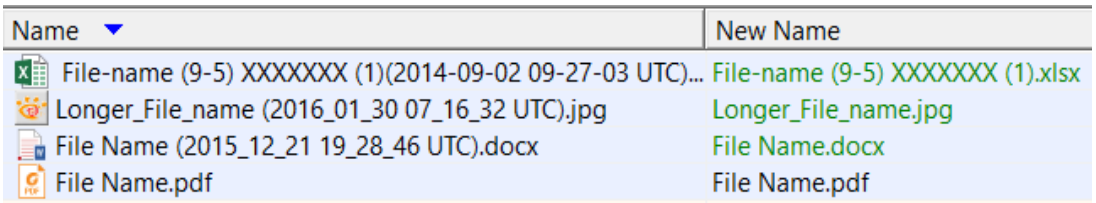




2. 'Section #5: Replace' –



Trim: Enabled



Sample:

```
.....................................................
Match 1: File Name (2015_12_21 19_28_46 UTC)
Group 1: File Name
Group 2: (2015_12_21 19_28_46 UTC)
```

Forum Contributor: therube

Remove

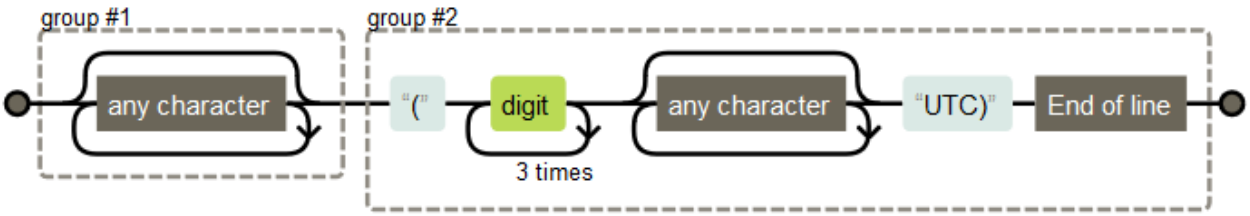
Rename to remove UTC (Coordinated Universal Time) Date/Time/Zone cont.

Tim Notes:

1. The RegEx: (.*)(\(\d\d\d\d.*UTC\)$)

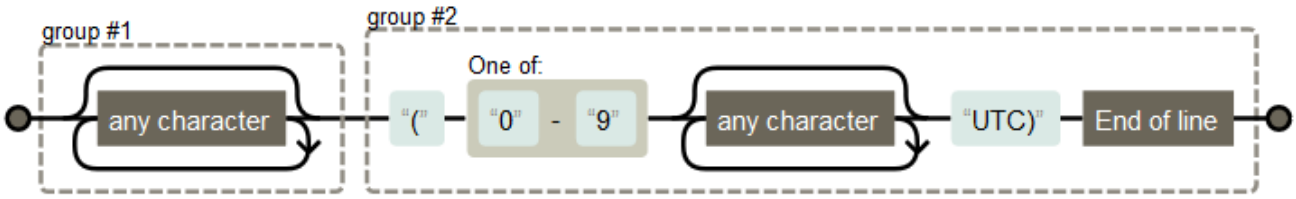
Could also have been expressed as:

Match: (.*)(\(\d{4}.*UTC\)$)



or

Match: (.*)(\([0-9].*UTC\)$)



Tim Tidbit – courtesy earthsky.org

a. UTC based on 24 hour military time. To convert to … from…

1. Eastern Daylight Time (EDT) | subtract 4 hours from UTC

2. Eastern Standard Time (EST) | subtract 5 hours from UTC
3. Central Daylight Time CDT) | subtract 5 hours from UTC

4. Central Standard Time (CST) | subtract 6 hours from UTC
5. Mountain Daylight Time MDT) | subtract 6 hours from UTC

6. Mountain Standard Time (MST) | subtract 7 hours from UTC
7. Pacific Daylight Time PDT) | subtract 7 hours from UTC

8. Pacific Standard Time (PST) | subtract 8 hours from UTC

b. When converting zone time to or from UTC, dates must be properly taken into account.

For example, 10 March at 02 UTC is the same as 9 March at 21 EST (U.S.).

1. 12 midnight is hour 24.
2. 02 UTC is 2 am.
3. 5 hours (EST) subtracted from 2am = hour 21 or 9 pm the previous day.

Remove

Rename to remove UTC (Coordinated Universal Time) Date/Time/Zone cont.

Using: File Name (2015_12_21 19_28_46 UTC).docx

Match: (.*)(\(\d\d\d\d.*UTC\)$)

Where:

\1 = File Name <space> \2 = (2015_12_21 19_28_46 UTC)

First of all, I like to simplify down multiple Metacharacters if possible.

Match: (.*)(\(\d{4}.*UTC\)$)

Tim Analysis:

1..	Match against any character	Capture Group 1 = F
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \(	Match against literal left parenthesis	Capture Group 2 = (Changes Capture Group 1 = File Name <space>
4. \d {4}	Match against 4 numeric digits	Capture Group 2 = (2015
5..	Match against any character	Capture Group 2 = (2015_
6. *	Make it Greedy Until …	Capture Group 2 = (2015_12_21 19_28_46 UTC)
7. UTC	Match against string, 'UTC'	Capture Group 2 = (2015_12_21 19_28_46 UTC
8. \)	Match against literal right parenthesis	Capture Group 2 = (2015_12_21 19_28_46 UTC)
9. $	Must be at EOL	

Tim Notes:

1. It looks redundant but it's not. Just because on this sample, all that was needed was, (.*)(\(\d{4}.*), this would not have worked for all of the samples. Be aware of this when you see my analysis – remember I usually base the analysis on one sample but it may not be representative of all the samples the RegEx is able to match against.

Remove

Rename to remove UTC (Coordinated Universal Time) Date/Time/Zone cont.

Using: Longer_File_name (2016_01_30 07_16_32 UTC).jpg

Match: (.*)(\([0-9].*UTC\)$)

Sample:

```
.....................................................
Match 1: Longer_File_name (2016_01_30 07_16_32 UTC)
Group 1: Longer_File_name
Group 2: (2016_01_30 07_16_32 UTC)
```

Where:

\1 = Longer_File_name \2 = (2016_01_30 07_16_32 UTC)

Tim Analysis:

1..	Match against any character	Capture Group 1 = L
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \(	Match against literal left parenthesis	Capture Group 2 = (Changes Capture Group 1 = Longer_File name <space>
4. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 2 = (2
5..	Match against any character	Capture Group 2 = (20
6. *	Make it Greedy Until …	Capture Group 2 = (2016_01_30 07_16_32 UTC)
7. UTC	Match against string, 'UTC'	Capture Group 2 = (2016_01_30 07_16_32 UTC
8. \)	Match against literal right parenthesis	Capture Group 2 = (2016_01_30 07_16_32 UTC)
9. $	Must be at EOL	

Remove

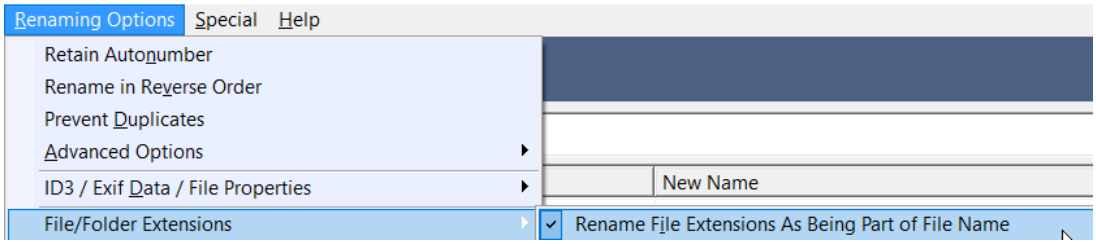
Remove all text after the dot and keep the second Underscore

3052503155104.Bourjois_FDTAirMat_01_1.png	to	3052503155104_1.png
3052503155104.Bourjois_FDTAirMat_01_2.png	to	3052503155104_2.png
3052503155104.Bourjois_FDTAirMat_01_3.png	to	3052503155104_3.png

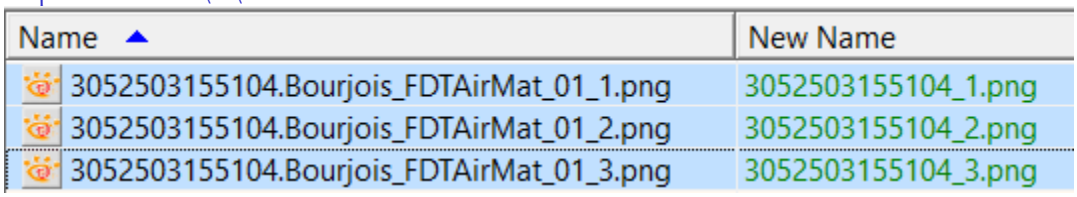
Observations:

1. Keep everything before the first '.' (dot character)
2. Remove everything until the last _
3. Retain the last underscore

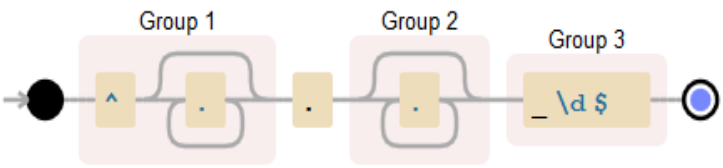
Because there are dot characters in the samples, normally you would enable, 'Rename File Extensions As Being part of File Name', but this example does not, and would in fact not work if enabled.



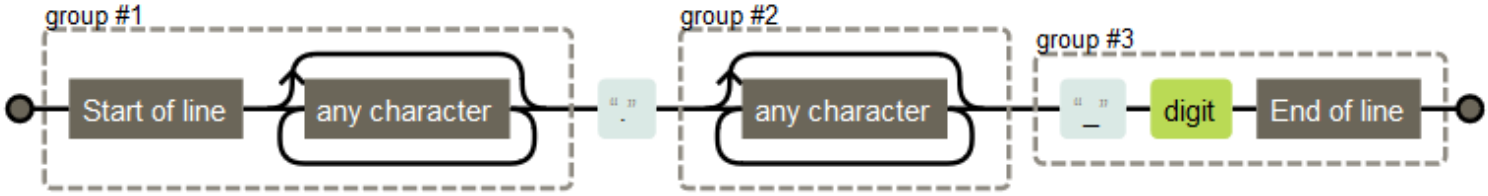
Match: (^.*?)\.(.*?)(_\d$)
Replace: \1\3



Sample:




```
Match 1: 3052503155104.Bourjois_FDTAirMat_01_1
Group 1: 3052503155104
Group 2: Bourjois_FDTAirMat_01
Group 3: _1
```



Forum Contributor: Tolek12

Remove

Remove all text after the dot and keep the second Underscore cont.

Using: 3052503155104.Bourjois_FDTAirMat_01_1.png

Match: (^.*?)\.(.*?)(_\d$)

Where:

\1 = 3052503155104 \2 = Bourjois_FDTAirMat_01 \3 = _1

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .	Match against any character	Capture Group 1 = 3
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. \.	Match against literal <dot> (Not Captured) =<dot> This is the first <dot> following '3052503155104'	Capture Group 1 = 3052503155104
6. .	Match against any character	Capture Group 2 = B Capture Group 1 = Unchanged
7. *	Make it Greedy	Capture Group 2 = Bourjois_FDTAirMat_01_1 Capture Group 1 = Unchanged
8. ?	But not too Greedy	Capture Group 2 = EOL All other Capture Groups = Unchanged
9. _	Match against <underscore> This is the <underscore> preceding 'FD'	Capture Group 3 = _ Capture Group 2 = Bourjois Capture Group 1 = Unchanged
10. \d	Match against numeric digit This is the '_0' of '_01'	Capture Group 3 = _0 Capture Group 2 = Bourjois_FDTAirMat Capture Group 1 = Unchanged
11. $	Must be at EOL (End of Line) This changes to the '_1' at end of string	Capture Group 3 = _1 Changes Capture Group 2 = Bourjois_FDTAirMat_01 Capture Group 1 = Unchanged

Remove

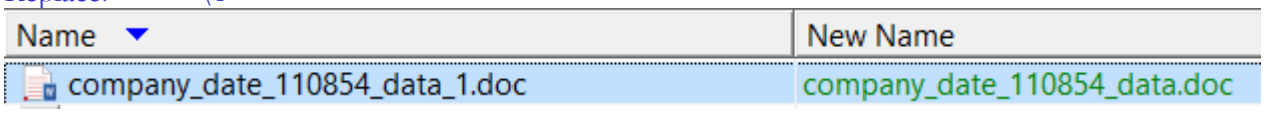
From End… Remove last Underscore with any Following text at end of Filename

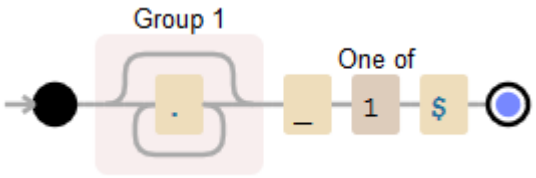
company_date_110854_data_1.doc to company_date_110854_data.doc

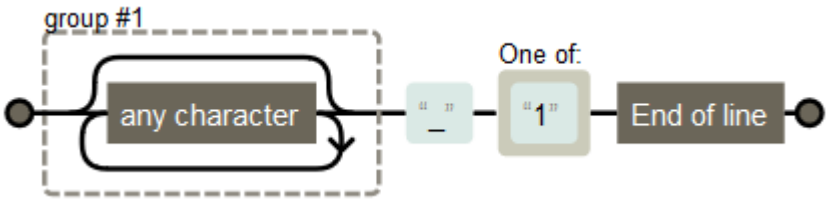
Observations:

1. Remove the last underscore with any following text.

Match: (.*)_[1]$
Replace: \1







Sample:

```
........................................
Match 1: company_date_110854_data_1
Group 1: company_date_110854_data
```

Tim Notes:

1. The $ is an anchor that specifies the End of the String or EOL
2. Restrictive – Specifically matches the Underscore followed by a class defined as containing the numeric digit '1'

Forum Contributor: KenP

Remove

From End… Remove last Underscore with any Following text at end of Filename cont.

Using: company_date_110854_data_1.doc

Match: (.*)_[1]$

Where:

\1 = company_date_110854_data

Tim Analysis:

1..	Match against any character	Capture Group 1 = c
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. _	Match against underscore character (Not Captured)	Changes Capture Group 1 = company_date_110854_data
4. [1]	Match against a class consisting of a '1' (Not Captured) = 1. This is the '1' of 'data_1'	
5. $	Must be at EOL	

Remove

To Beginning… Remove blank space in front of file name

<space> company_date_110854_data.doc to company_date_110854_data.doc

Method #1

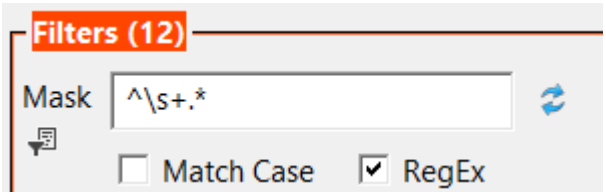
1. 'Section #5: Remove' –

First n: 1

Caveat: Removes any First character of a selected filename not just a <space> character.

Used in conjunction with:

2. 'Section #12: Filters' –



Mask: ^\s+.*
RegEx: Enabled

This would only display files that started with a <space> character

Name	New Name
company_date_110854_data.doc	company_date_110854_data.doc

Now you can perform the Rename safely without worrying about other files consequently affected.

Name	New Name
company_date_110854_data.doc	company_date_110854_data.doc

Remove

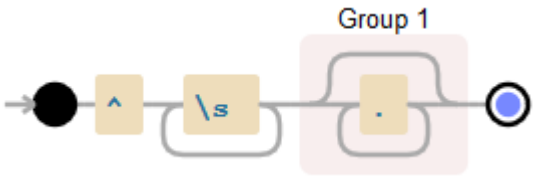
To Beginning… Remove blank space in front of file name cont.

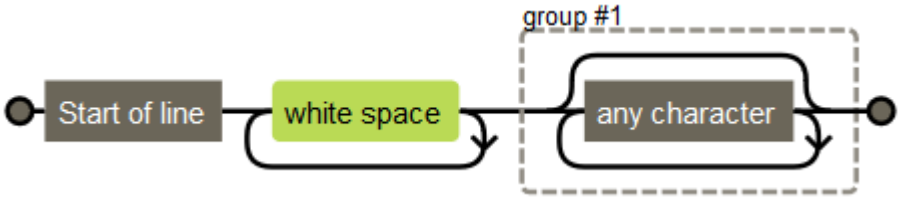
<space> company_date_110854_data.doc to company_date_110854_data.doc

Method #2

Match: ^\s+(.*)
Replace: \1

Name	New Name
company_date_110854_data.doc	company_date_110854_data.doc
Slide of Mt.Rushmore 738.jpg	Slide of Mt.Rushmore 738.jpg
1863_22_10_2019.txt	1863_22_10_2019.txt





Sample:

```
.....................................
Match 1:  company_date_110854_data
Group 1: company_date_110854_data
```

This will only affect filenames that start with a blank <space>.

Tim Notes:

1. The ^ is an anchor that specifies the Beginning of the string (Beginning of Line or BOL)

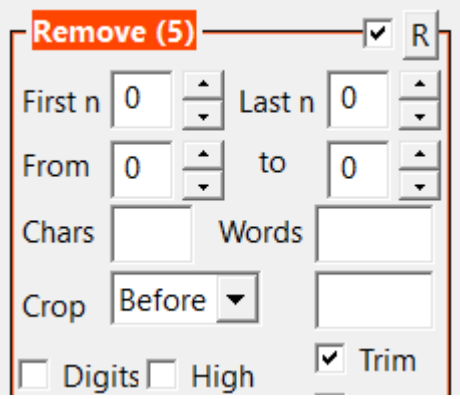
Remove

To Beginning… Remove blank space in front of file name cont.

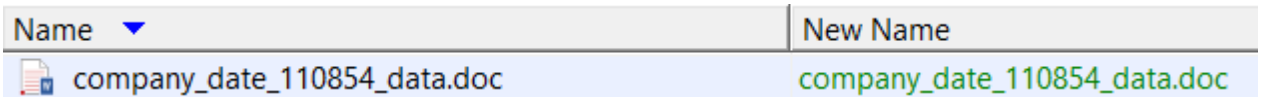
<space> company_date_110854_data.doc to company_date_110854_data.doc

Method # 3

1. 'Section #5: Remove' –



Trim: Enable This removes both trailing and leading <space> characters



Forum Contributor: KenP, jrstech

Remove

To Beginning… Remove blank space in front of file name cont.

Using: <space> company_date_110854_data.doc

Match: ^\s+(.*)

Where:

\1 = company_date_110854_data

Tim Analysis: Method #2

1. ^	Start at beginning of line (BOL)	
2. \s	Match against <space> character (Not Captured) This is the first <space> in the string located just after BOL	
3. +	Make it Greedy (Not Captured) Would match against additional <spaces> at beginning of string.	
4. .*	Match against remaining string	Capture Group 1 = company_date_110854_data

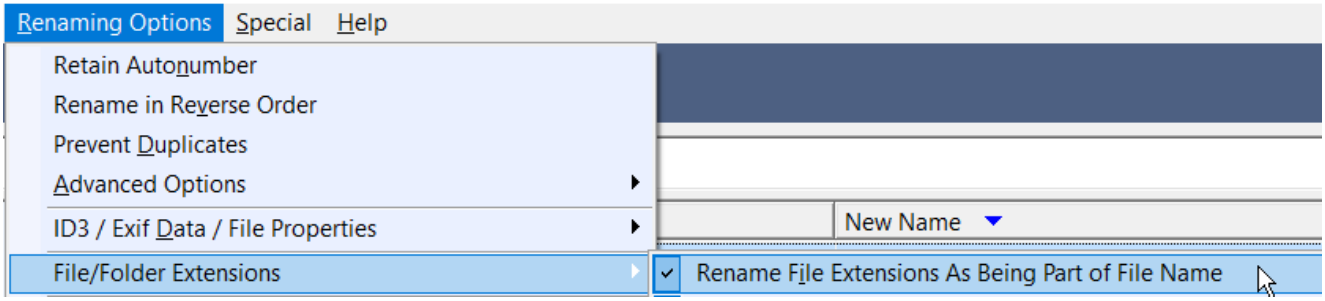
Remove

Remove text between a word and a date

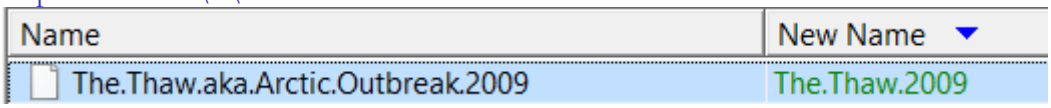
Tim Notes:

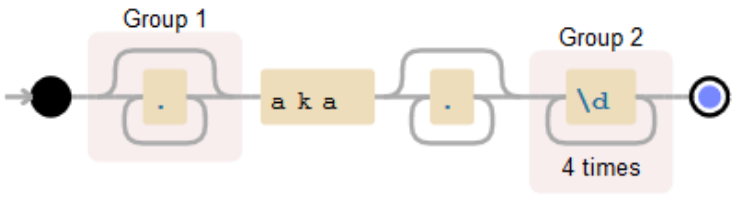
1. Here is a good example to illustrate how the RegEx engine matches a pattern as it moves through the string.

The.Thaw.aka.Arctic.Outbreak.2009 to The.Thaw.2009



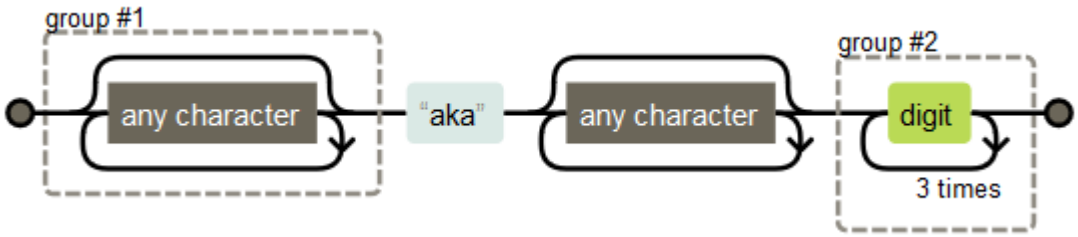
Match: (.*)aka.*(\d{4})
Replace: \1\2





Sample:

```
Match 1: The.Thaw.aka.Arctic.Outbreak.2009
Group 1: The.Thaw.
Group 2: 2009
```



Forum Contributor: KenP

Tim Analysis:

1. (*.)aka.*	a. Match and Capture against all characters until 'aka.' b. 'aka' is matched but Not Captured c. Continue matching until ….	Capture Group 1 = The.Thaw.
2. \d{4}	.. match against and capture 4 numeric digits	Capture Group 2 = 2009

Remove

Remove all text prior to song title at Beginning of String

00-songname.mp3	to	songname.mp3
00 - songname.mp3	to	songname.mp3
00- songname.mp3	to	songname.mp3
00 -songname.mp3	to	songname.mp3

Observations:

1. Pattern is numeric digits {<hyphen> or <spaces>} <hyphen><optional spaces>
2. Remove all text before song title
3. prefixes are:
   a. 00- (Prefix <hyphen>)
   b. 00 - (Prefix <space> <hyphen>)
   c. 00 - (Prefix <space> <hyphen> <space>)
   d. 00- (Prefix <hyphen> <space>)

4. These possibilities will be offered through alternates using the ' | ' (the 'Or' metacharacter)

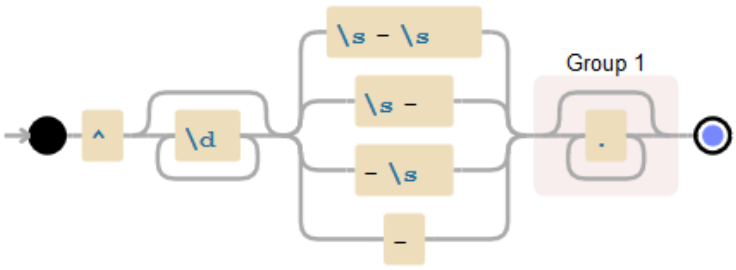
Match: ^(?:\d*)(?:\s-\s|\s-|-\s|-)(.*)
Replace: \1

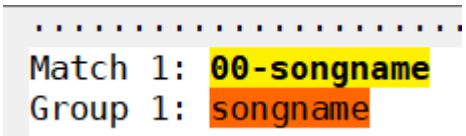
Name	New Name
00-songname.mp3	songname.mp3
00- songname.mp3	songname.mp3
00 - songname.mp3	songname.mp3
00 -songname.mp3	songname.mp3

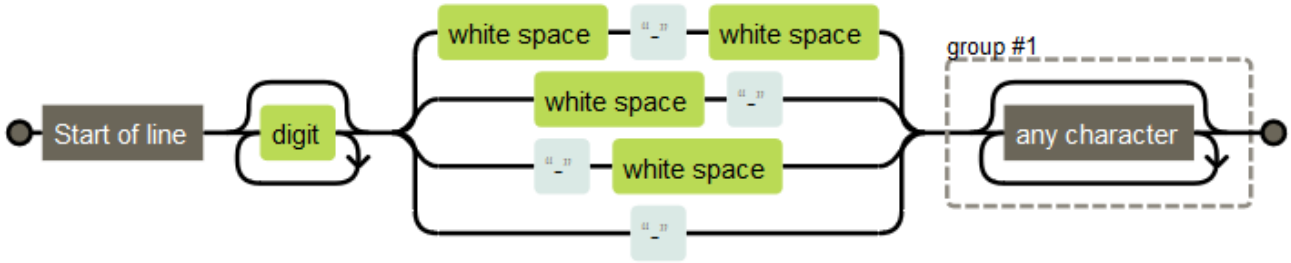
Tim Notes:

1. This is only an exercise, but consider enabling 'Prevent Duplicates' from Rename Options Menu

Sample:







Tim Notes

1. The ^(?:\d*) is not needed. The (?:\s-\s|\s-|-\s|-)(.*) would be enough by itself.

Remove

Remove all text prior to song title at Beginning of String cont.

Contributor Notes:

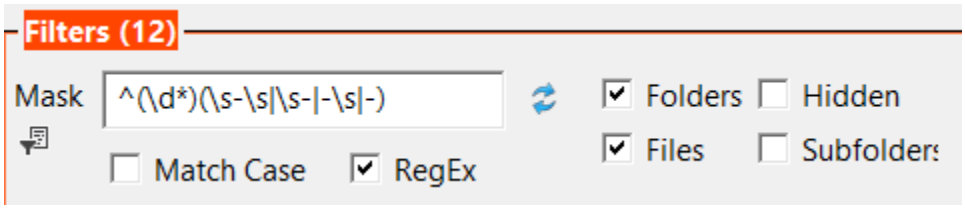
1. Works with any files of that pattern

   i.e.,

   2000 -04 03 Standing - Burning Empires to 04 03 Standing Burning Empires

2. If you want to filter only those files that start with numeric digits, followed by a <hyphen> or <spaces> and a <hyphen> in any order:

1. 'Section #12: Filters' –



Mask: ^(\d*)(\s-\s|\s-|-\s|-)
RegEx: Enabled

Forum Contributor: KenP

Remove

Remove all text prior to song title at Beginning of String cont.

Using: 00-songname.mp3

Match: ^(?:\d*)(?:\s-\s|\s-|-\s|-)(.*)

Where:

\1 = songname

Tim Analysis:

1. ^ Start at beginning of string (Beginning of Line or BOL)
2. (?:\d*) Non-Marking Group to skip over any numeric digits and move forward until…

3. (?:\s-\s |\s- | -\s | -) Alternates (Branches).

Tim Notes:

1. This is interesting and requires further explanation:

   a. This sets up another Non-Marking Group.
   b. It uses a group of 4 alternates:

This	Or	This	Or	This	Or	This
<space><hyphen><space>		<space><hyphen>		<hyphen><space>		<hyphen>

   c. If the next characters match the above alternates, then = TRUE and continue.. else evaluation Fails.

4. .* Capture remainder of string Capture Group 1 = songname

Tim Notes:

1. A Non-Marking Group is not a Capture Group. The syntax is (?:). For more information, refer to, 'Volume 1 Bulk Rename Utility Operations Manual' of the RegEx Manual found in the Appendix.

2. The ^(?:\d*) is not needed. The (?:\s-\s|\s-|-\s|-)(.*) would be enough by itself.

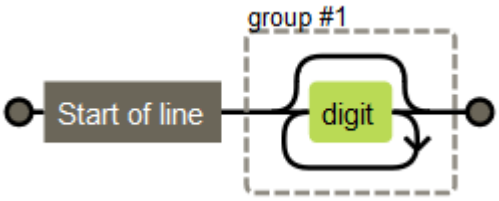
Remove

Remove all text prior to song title at Beginning of String cont.

00-songname.mp3	to	songname.mp3
00 - songname.mp3	to	songname.mp3
00- songname.mp3	to	songname.mp3
00 -songname.mp3	to	songname.mp3

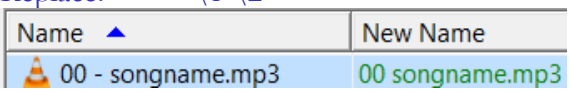
Tim Contribution –

So what does the ' ^(?:\d*)' do?



If we capture the value then you can see:

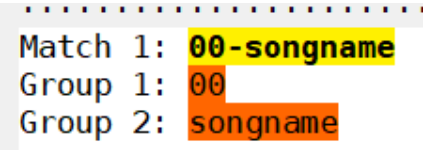
Match: ^((?:\d*))(?:\s-\s|\s-|-\s|-)(.*)
Replace: \1 \2

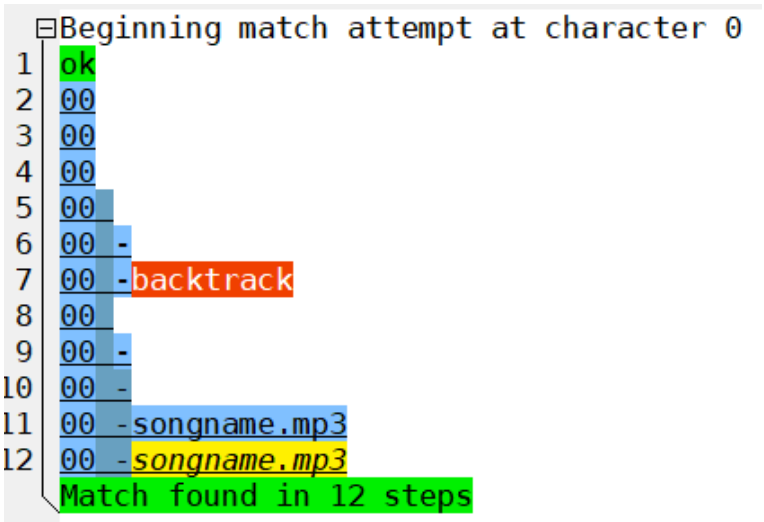


Where:

\1 = 00 \2 = songname

Sample:



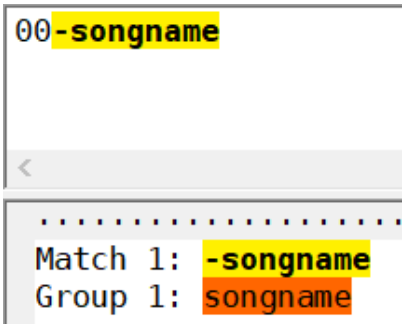


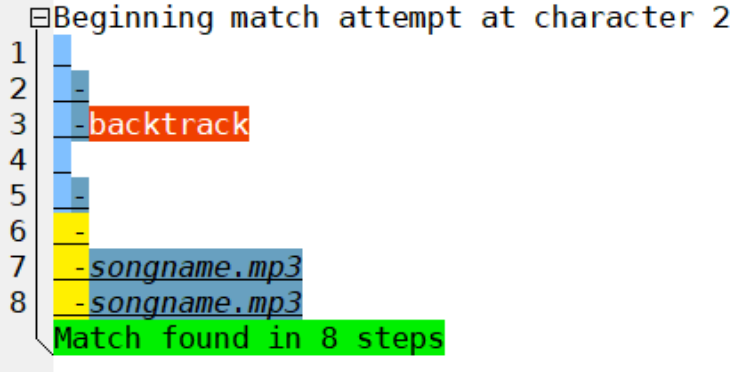
Without using the ' ^(?:\d*)'

Where:

\1 = songname

Sample:





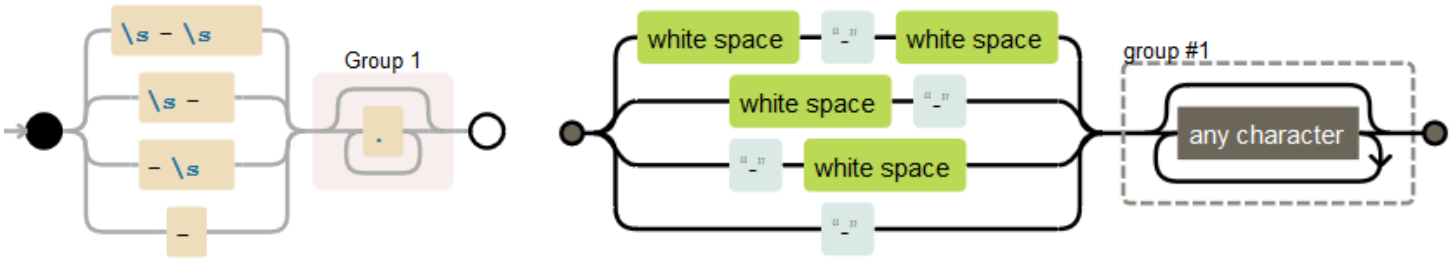
Remove

Remove all text prior to song title at Beginning of String cont.

00-songname.mp3	to	songname.mp3
00 - songname.mp3	to	songname.mp3
00- songname.mp3	to	songname.mp3
00 -songname.mp3	to	songname.mp3

Tim Contribution – cont.

Without using the ‘ ^(?:\d*)’ cont.



Without using the anchor, it would match files beginning with text as well:

Using: t00 -songname.mp3

Match: ((?:\d*))(?:\s-\s|\s-|-\s|-)(.*)
Replace: \2

Name	New Name
t00 -songname.mp3	songname.mp3

Where:

1\ = 00 \2 = songname

Sample:

t00 -songname

```
.....................
Match 1: 00 -songname
Group 1: 00
Group 2: songname
```

```
Beginning match attempt at character 1
 1  00
 2  00
 3  00
 4  00
 5  00 -
 6  00 -backtrack
 7  00
 8  00 -
 9  00 -
10  00 -songname.mp3
11  00 -songname.mp3
Match found in 11 steps
```

Remove

Remove all text prior to song title at Beginning of String cont.

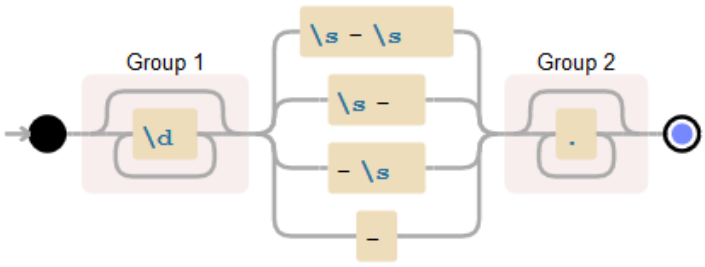
00-songname.mp3	to	songname.mp3
00 - songname.mp3	to	songname.mp3
00- songname.mp3	to	songname.mp3
00 -songname.mp3	to	songname.mp3

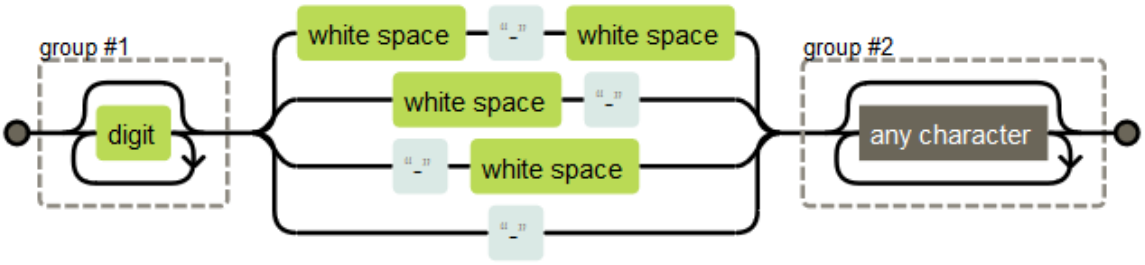
Tim Contribution – cont.

Without using the anchor, it would match files beginning with text as well: cont.

It would now match these samples:

Name	New Name
00 - songname.mp3	songname.mp3
00 -songname.mp3	songname.mp3
00- songname.mp3	songname.mp3
00-songname.mp3	songname.mp3
t00 -songname.mp3	songname.mp3
this is a test 01-Whip it Devo.wav	Whip it Devo.wav
this01 - is a test	is a test





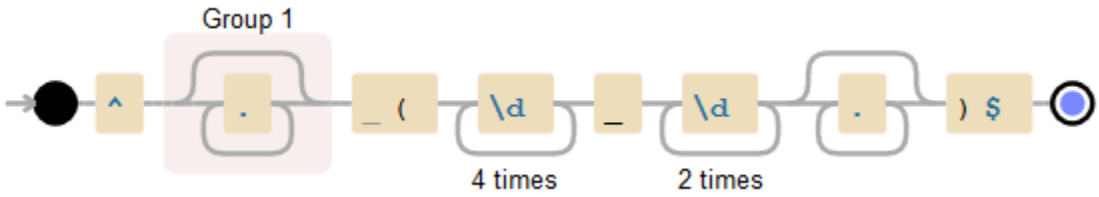
Remove

Remove everything after last space

ratios (2016_11_12 18_50_08 UTC)

Match: ^(.*) \(\d{4}_\d{2}.*\)$
Replace: \1

Name	New Name
ratios (2016_11_12 18_50_08 UTC)	ratios



Sample:

```
.........................................
Match 1: ratios (2016_11_12 18_50_08 UTC)
Group 1: ratios
```

Forum Contributor: KenP

Remove

Remove everything after last space cont.

Using: ratios (2016_11_12 18_50_08 UTC)

Match: ^(.*) \(\d{4}_\d{2}.*\)$

Where:

\1 = ratios

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .*	Match against all characters until…	Capture Group 1 = ratios
3. <space>	.. <space> (Not Captured) .. must be followed by …	= <space>
4. \(	.. left parenthesis, (Not Captured) .. followed by …	= (
5. \d {4}	.. 4 numeric digits, (Not Captured) .. followed by …	= 2016
6. _	.. underscore, (Not Captured) … followed by …	= _
7. \d{2}	.. 2 numeric digits, (Not Captured)	= 11
8. .*	Match against characters, (Not Captured), until ….	= _12 18_50_08 UTC
9. \)	... right parenthesis, (Not Captured)	=)
10. $	Last expression must have matched at EOL (must be at End of Line)	

Tim Notes:

1. Although \(\d{4}_\d{2}.*\) looks like a Capture Group, it is in fact not. This is because the parentheses give it that appearance but what it actually is are separate characters that are matched.

   The parentheses are also just characters in this example. Because RegEx reserves these Metacharacters, in order to use them as characters or 'literals', they have to be preceded with what is called an 'Escape' character. The Escape character is the back slash '\'. This indicates to the RegEx engine that the following is to be treated as a literal and not as a Metacharacter. For more information, please refer to the RegEx Manual found in the Appendix of 'Volume I Bulk Rename Utility Operations Manual.'

Breaks down:

\(	\d{4}	_	\d{2}	.*	\)
left or open parenthesis		underscore			right or closed parenthesis

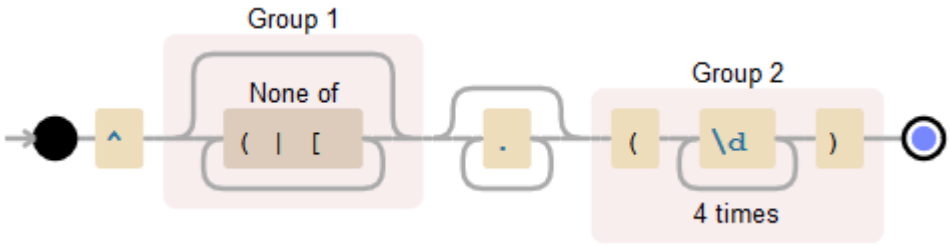
Remove

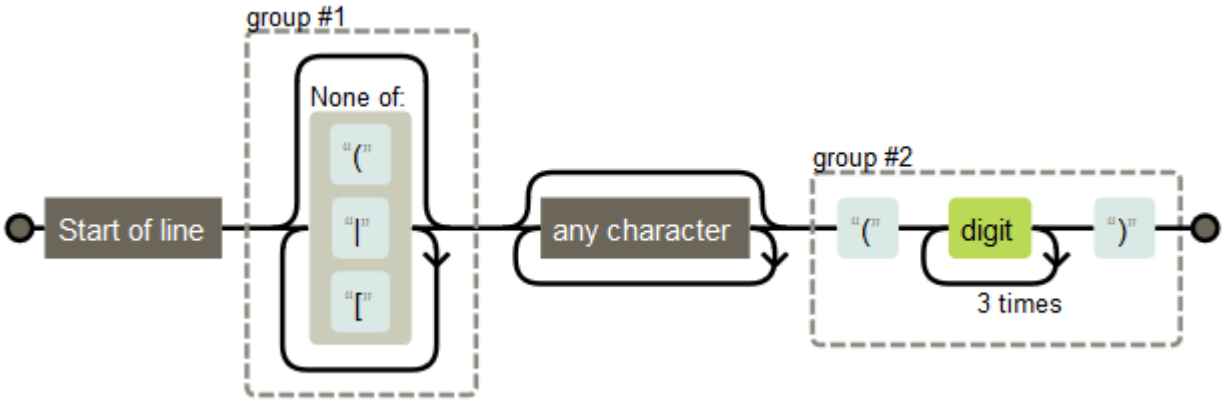
Remove all Except preserve filename and date in parentheses

FILENAME (1991)(text)	to	FILENAME (1991)
FILENAME (1983)(text - a to z)[text -a to z]	to	FILENAME (1983)
FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	to	FILENAME (1974)
FILENAME (1994)[text]	to	FILENAME (1994)

Match: ^([^(|[]*).*(\(\d{4}\))
Replace: \1\2

Name	New Name
FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	FILENAME (1974)
FILENAME (1994)[text]	FILENAME (1994)
FILENAME (1991)(text)	FILENAME (1991)
FILENAME (1983)(text - a to z)[text -a to z]	FILENAME (1983)





Sample:

```
.............................
Match 1: FILENAME (1991)
Group 1: FILENAME
Group 2: (1991)
```

It can remove both the brackets and any text inside the brackets without specifying the exact text or brackets/parentheses.

Forum Contributor: KenP

Remove

Remove all Except preserve filename and date in parentheses cont.

FILENAME (1991)(text)	to	FILENAME (1991)
FILENAME (1983)(text - a to z)[text -a to z]	to	FILENAME (1983)
FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	to	FILENAME (1974)
FILENAME (1994)[text]	to	FILENAME (1994)

Using: FILENAME (1983)(text - a to z)[text -a to z]

Match: ^([^(|[]*).*(\(\d{4}\))

Where:

\1 = FILENAME \2 = 1983

Tim Analysis:

Spaces have been added for clarification and are Not part of the RegEx.
This was complicated – any time backtracking is involved - so I am not 100% sure of my analysis.

1. ^ Start at Beginning of string (Beginning of Line or BOL)

2. [^ (| [ ] Sets up a Negated Class – identified as [^]

   a. Within this class it sets a Capture Group that starts within the class and ends outside of it.
   [( ])

   b. Within the class it includes the left parenthesis (also called the 'open parenthesis'), the pipe character ('| ') and the left bracket (also called the 'open bracket').
   1. What the ^ means is the negated class vs positive class:

      i.e.,
      Negated ([^(|[]*) Capture Group 1 = FILENAME

      Positive ([(|[]*) Capture Group 1 = null leaving: Capture Group 2 = (1983)

   c. ^([^(|[]) Backtracks to the beginning of the string where it Matches the first character, = 'F'

   d. ^([^(|[]*) The * is greedy and pulls in the rest of the word Capture Group 1 = FILENAME

3. .* Greedy – pulls in the remainder of the string to End of Line (EOL), but Not Captured

Remove

Remove all Except preserve filename and date in parentheses cont.

Tim Analysis: cont.

4. \(Backtracks to match against first left parenthesis followed by … Capture Group 2 = (

a. But which left parenthesis, can be verified by adding (…….) to the end ^([^(|[]*).*(\()(.....)

FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	text - a to
FILENAME (1994)[text]	1994)[text]
FILENAME (1991)(text)	1991)(text)
FILENAME (1983)(text - a to z)[text -a to z]	text - a to

use \3 to see the result

5. \d{4} … 4 numeric digits followed by …. Capture Group 2 =

FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	(1974
FILENAME (1994)[text]	(1994
FILENAME (1991)(text)	(1991
FILENAME (1983)(text - a to z)[text -a to z]	(1983

a. Notice that 'text – a to' are no longer included in Capture Group 2. This is because the RegEx engine for those filenames moved forward until it found the match for the parentheses followed by the numeric digits and discarded the previous value.

6. \) .. a right parenthesis Capture Group 2 =

FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	(1974)
FILENAME (1994)[text]	(1994)
FILENAME (1991)(text)	(1991)
FILENAME (1983)(text - a to z)[text -a to z]	(1983)

Tim Notes:

1. Brackets in this example, refer to either parentheses or square brackets.

Remove

Filename should Have One <space> Preceding

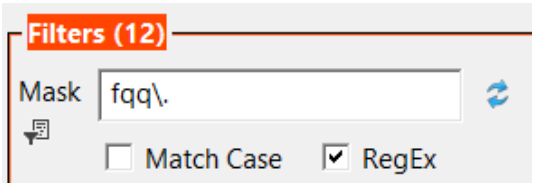
fqq.xxx	to	<space>fqq.xxx
<space>fqq.xxx	to	Ignore
<space><space>fqq	to	<space> fqq.xxx
<space><space><space>fqq	to	<space> fqq.xxx

Observations:

1. Only one <space> character before filename

1. Filter out files to ensure that other files do not inadvertently be renamed.

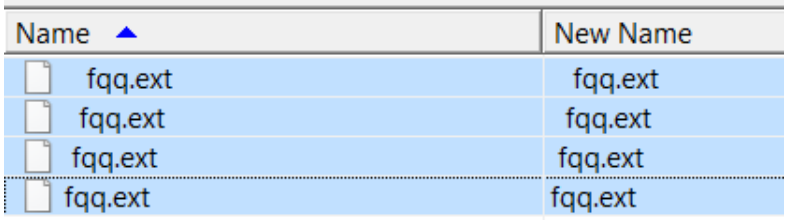
'Section #12: Filters' –



Enter: fqq\.
Enable: RegEx

2. RegEx

Match: (.*)(\w)(| | | *)(fqq)
Replace: \1\2 \4



Didn't work

Match: (.*)(\w)(*)(fqq)
Replace: \1\2 \4

Didn't work

Forum Contributor: KenP

Remove

Filename should Have One <space> Preceding cont.

Tim Contribution –

fqq.xxx	to	<space>fqq.xxx
<space>fqq.xxx	to	Ignore
<space><space>fqq	to	<space> fqq.xxx
<space><space><space>fqq	to	<space> fqq.xxx

These will – slight modifications of Contributor's RegEx for the first two.

Match: (.*)(\w*)(| | | *)(fqq)
Replace: <space>\4

	New Name
fqq.ext	fqq.ext
fqq.ext	fqq.ext
fqq.ext	fqq.ext
fqq.ext	fqq.ext

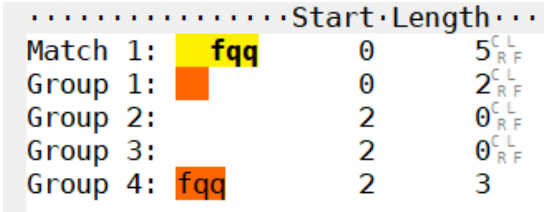
Where:

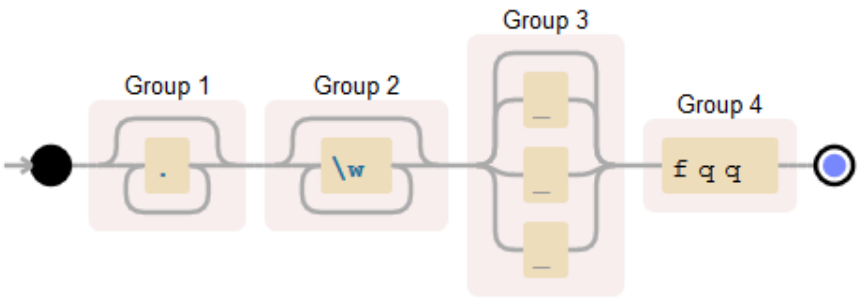
\1 =

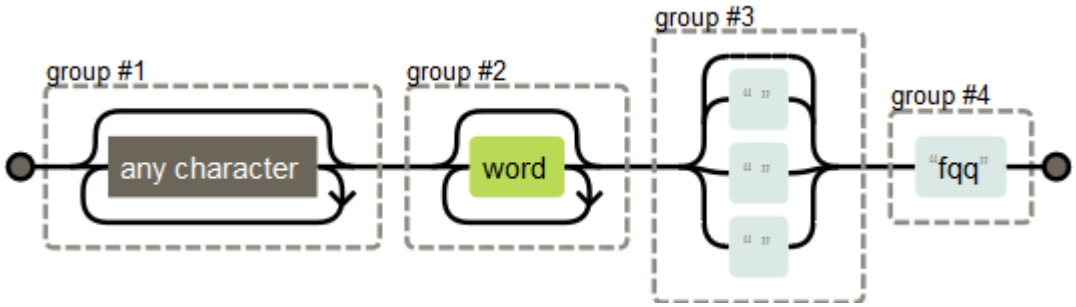
spacefqq.xxx	space.xxx
spacefqq.xxx	space.xxx
spacefqq.xxx	space.xxx

\2 = null \3 = null \4 = fqq

Sample:







Remove

Filename should Have One <space> Preceding cont.

Tim Contribution – cont.

Using: <space><space>fqq

Match: (.*)(\w*)(| | | *)(fqq)

Where:

\1 = <space><space> \2 = null \3 = null \4 = fqq

Tim Analysis:

1. .	Match against any character	Capture Group 1 = <space>
2. *	Make it Greedy	Capture Group 1 = <entire string>
3.\w	Match against Word Character	Capture Group 2 = q Changes Capture Group 1 = <space><space>fq
4. *	Make it Greedy	Capture Group 2 = <entire string> Changes Capture Group 2 = null
5. <no char> | <space> | <space>*		Capture Group 3 = null Capture Group 2 = Unchanged Capture Group 1 = Unchanged

Tim Notes:

1. This will account for those filenames that..

   a. Do not have any leading <space> characters
   b. Have one leading <space> character
   c. Have more than one leading <space> characters

6. fqq	Match against string 'fqq'	Capture Group 4 = fqq Changes Capture Group 1 = <space><space> All Other Groups = Unchanged

It will also handle:

Name	New Name
fqq	fqq
this is a test fqq	fqq
this is a testfqq	fqq
thisisatestfqq	fqq

Remove

Filename should Have One <space> Preceding cont.

Tim Contribution – cont.

fqq.xxx	to	<space>fqq.xxx
<space>fqq.xxx	to	Ignore
<space><space>fqq	to	<space> fqq.xxx
<space><space><space>fqq	to	<space> fqq.xxx

Match: (.*)(\w*)(fqq)
Replace: <space>\3

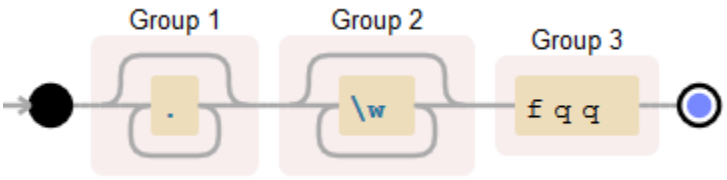
	New Name
fqq.ext	fqq.ext
fqq.ext	fqq.ext
fqq.ext	fqq.ext
fqq.ext	fqq.ext

Where:

\1 =

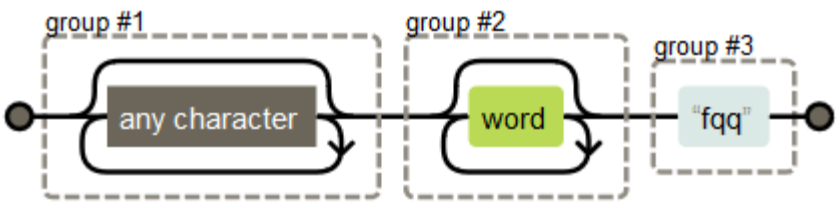
spacefqq.xxx	space.xxx
spacefqq.xxx	space.xxx
spacefqq.xxx	space.xxx

\2 = null \3 = fqq



Sample:

		Start	Length
Match 1:	fqq	0	5
Group 1:		0	2
Group 2:		2	0
Group 3:	fqq	2	3



Remove

Filename should Have One <space> Preceding cont.

Tim Contribution – cont.

Using: <space><space>fqq

Match: (.*)(\w*)(fqq)

Where:

\1 = <space><space> \2 = null \3 = fqq

Tim Analysis:

1. .	Match against any character	Capture Group 1 = <space>
2. *	Make it Greedy	Capture Group 1 = <entire string>
3.\w	Match against Word Character	Capture Group 2 = q
		Changes Capture Group 1 = <space><space>fq
4. *	Make it Greedy	Capture Group 2 = <entire string>
		Changes Capture Group 2 = null
5. fqq	Match against string 'fqq'	Capture Group 3 = fqq
		Changes Capture Group 1 = <space><space>
		All Other Groups = Unchanged

It will also handle:

Name	New Name
fqq	fqq
this is a test fqq	fqq
this is a testfqq	fqq
thisisatestfqq	fqq

Remove

Filename should Have One <space> Preceding cont.

Tim Contribution – cont.

fqq.xxx	to	<space>fqq.xxx
<space>fqq.xxx	to	Ignore
<space><space>fqq	to	<space> fqq.xxx
<space><space><space>fqq	to	<space> fqq.xxx

Match: ^([ ]+)(.*)(fqq)$
Replace: <space>\3

Name	New Name
spacefqq.xxx	fqq.xxx
spacefqq.xxx	fqq.xxx
spacefqq.xxx	fqq.xxx

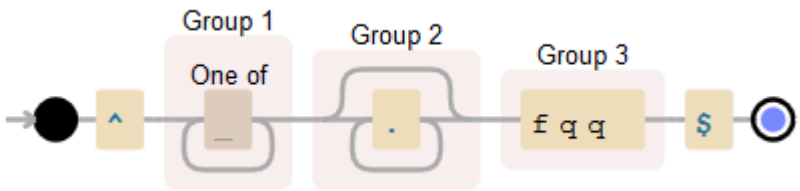
Where:

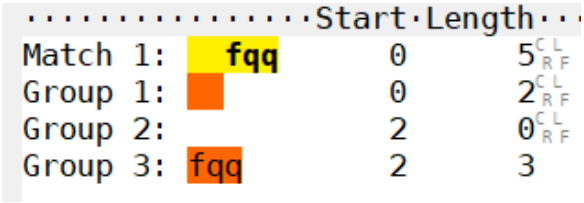
\1 = <all spaces captured> \2 = any text before fqq \3 = fqq

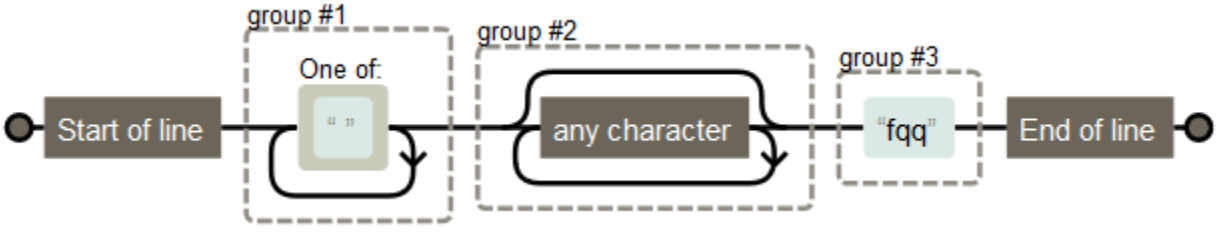
It will also handle:

Name	New Name
fqq	fqq
this is a test fqq	fqq
this is a testfqq	fqq
thisisatestfqq	fqq

Sample:







Remove

Filename should Have One <space> Preceding cont.

Tim Contribution – cont.

Using: <space><space>fqq

Match: ^([ ]+)(.*)(fqq)$

Where:

\1 = <space><space> \2 = null \3 = fqq

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. [<space>]	Match against a class consisting of a <space> character	Capture Group 1 = <space>
3. +	Make it Greedy	Capture Group 1 = <space> <space>
4. .	Match against any character	Capture Group 2 = f
5. *	Make it Greedy	Capture Group 2 = fqq Capture Group 1 = Unchanged

Tim Notes:

1. At this point, it would also capture any preceding text before 'fqq', for example, ' this is a test fqq', Capture Group 2 would equal 'this is a test fqq'.

6. fqq	Match against string 'fqq'	Capture Group 3 = fqq Changes Capture Group 2 = null

Tim Notes:

1. If there were preceding text, then, using for example, ' this is a test fqq', Capture Group 2 would equal 'this is a test'

7. $	Must be at EOL (End Of Line)

Remove

Moving articles (a, an, the) from the end of the filename

Author - Story Title, A.epub to Author - A Story Title.epub
Author - Story Title, An.epub to Author - An Story Title.epub
Author - Story Title, The.epub to Author - The Story Title.epub

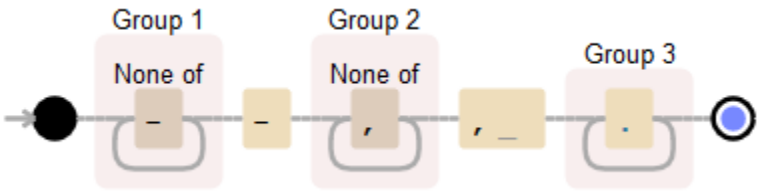
Observations:

1. Remove comma<space>
2. Move the article in front of 'Story Title'
3. Add <space> as separator between article and 'Story Title'.
4. Pattern – Author <space> <hyphen> <space> Story Title <comma> <space> <article>

Where :
article = A, An, The

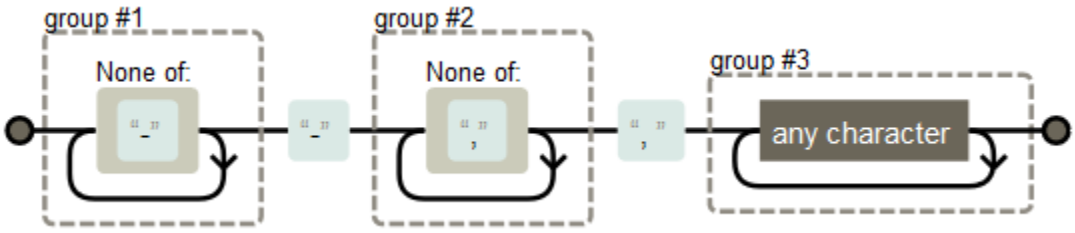
Match: ([^-]+)-([^,]+), (.+)
Replace: \1- \3\2

Name	New Name
Author - Story Title, A.epub	Author - A Story Title.epub
Author - Story Title, An.epub	Author - An Story Title.epub
Author - Story Title, The.epub	Author - The Story Title.epub



Sample:

```
Match 1: Author - Story Title, A
Group 1: Author
Group 2:  Story Title
Group 3: A
```



Remove

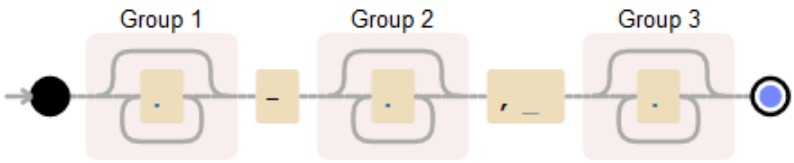
Moving articles (a, an, the) from the end of the filename cont.

Author - Story Title, A.epub	to	Author - A Story Title.epub
Author - Story Title, An.epub	to	Author - An Story Title.epub
Author - Story Title, The.epub	to	Author - The Story Title.epub

or you could use…

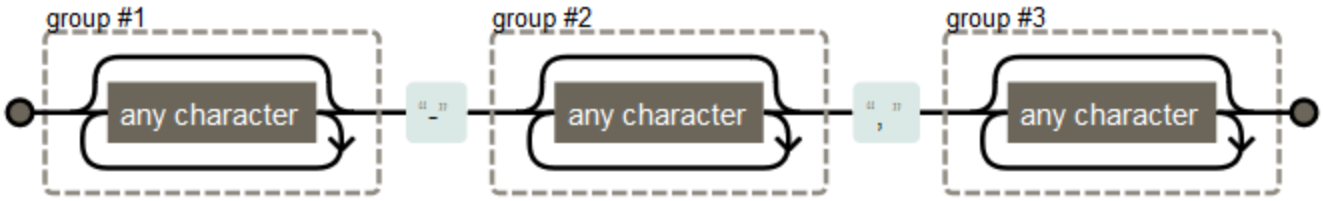
Match: (.*)-(.*), (.*)
Replace: \1- \3\2

Name	New Name
Author - Story Title, A.epub	Author - A Story Title.epub
Author - Story Title, An.epub	Author - An Story Title.epub
Author - Story Title, The.epub	Author - The Story Title.epub



Sample:

```
Match 1: Author - Story Title, A
Group 1: Author
Group 2:  Story Title
Group 3: A
```



Contributor Notes:

1. Using (.*) is not my first choice because (.*) always matches everything to the EOL (End of Line) and then does a lot of backtracking to allow following parts of the RegEx to match. Because of all the backtracking, the expression is solved for each file in about 80 steps. This can cause a significant difference in speed if you are renaming thousands of files.

2. In this case, using [^-] and [^,] solves the RegEx in 10 steps.

Forum Contributor: Glenn

Remove

Moving articles (a, an, the) from the end of the filename cont.

Using sample: Author - Story Title, A.epub

Match: (.*)-(.*), (.*)

Where:

\1 = Author<space> /2 = <space>Story Title \3 = A

Tim Analysis:

Tim Notes:

1. This is a simple parsing. I decided to include the analysis so it can be compared against the other RegEx the Contributor offers.

1. .	Match against any character	Capture Group 1 = A
2. *	Make it Greedy until ..	Capture Group 1 = entire string to EOL
3. -	.. <hyphen> (Not Captured)..followed by ..	Changes Capture Group 1 = Author<space>
	RegEx engine is backtracking	
4. .	.. Match against next character	Capture Group 2 = <space>
	This is the <space> following the <hyphen>	
	RegEx engine moving forward	
5. *	Make it Greedy until…	Capture Group 2 = <space>Story Title, A
6. , .	. <comma> followed by …	Capture Group 2 = <space>Story Title
	RegEx engine is backtracking	
7. <space>	.. <space> (Not Captured) followed by …	
	This is the <space> following 'Title'	
8. .	.. Match against next character	Capture Group 3 = A
9. *	Make it Greedy	Capture Group 3 = A

Tim Notes:

1. In step 9, this would have captured the rest of the string – for other filenames, Capture Group 3 = 'An', and 'The'

Remove

Moving articles (a, an, the) from the end of the filename cont.

Using sample: Author - Story Title, A.epub

Match: ([^-]+)-([^,]+), (.+)

Where:

\1 = Author<space> /2 = <space>Story Title \3 = A

Tim Analysis:

1. [^-]	Ignore <hyphen> character and move forward Moves to last character of string but not to EOL	Capture Group 1 = A
2. +	Make it Greedy .. until .. RegEx engine is backtracking	Capture Group 1 = Author <space>
3. -	.. <hyphen> (Not Captured) RegEx engine moving forward	Capture Group 1 = Unchanged
4. [^,]	Ignore <comma> This is the <space> following the <hyphen>	Capture Group 2 = <space>
5. +	Make it Greedy until …	Capture Group 2 = <space> Story Title
6. ,	<comma> (Not Captured) followed by ... This is the <comma> following 'Title'	
7. <space>	.. <space> (Not Captured) This is the <space> following the <comma>	
8. .+	Capture remainder of string	Capture Group 3 = A

Tim Notes:

1. You can see that there is less backtracking with this RegEx than the other.

Remove

Remove hyphens between date at Beginning of String

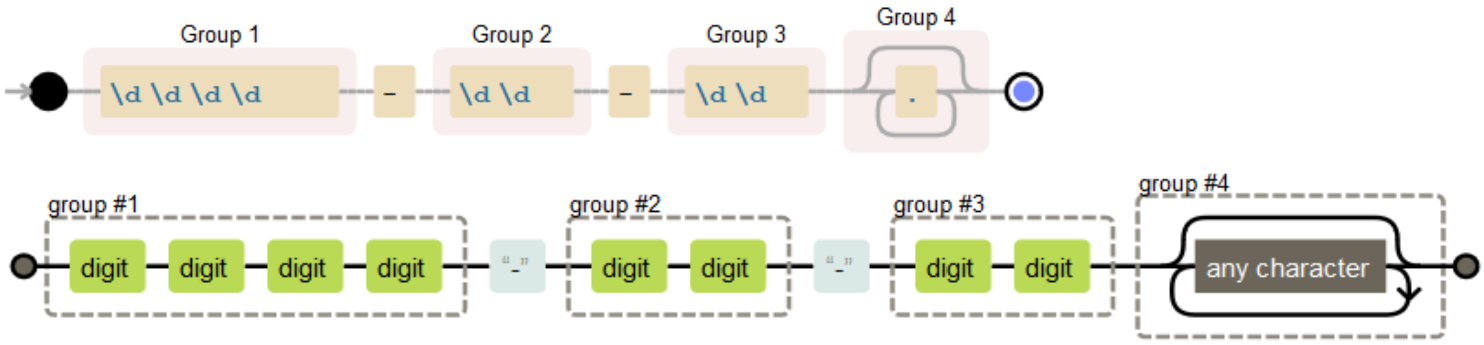
2020-01-01_this-is a test.jpg to 20200101_this-is a test.jpg

Observations:

1. Remove the <hyphen> from only the first 10 characters of the filename (the date)
2. Date format YYYY-MM-DD
2. Other <hyphens> after the date, do not get removed.
3. Pattern established: <4 numeric digits> <hyphen> <2 numeric digits> <hyphen> <2 numeric digits>

Match: (\d\d\d\d)-(\d\d)-(\d\d)(.*)
Replace: \1\2\3\4

Name	New Name
2020-01-01_this-is a test.jpg	20200101_this-is a test.jpg



Sample:

```
.....................................
Match 1: 2020-01-01_this-is a test
Group 1: 2020
Group 2: 01
Group 3: 01
Group 4: _this-is a test
```

If you want a delimiter between the dates, other than a <hyphen>, then include this in the Replacement String.

i.e.,

Replace: \1^^\2^^\3\4

Name	New Name
2020-01-01_this-is a test.jpg	2020^^01^^01_this-is a test.jpg

Tim Notes:

1. Could also have been expressed as, (\d{4})-(\d{2})-(\d{2})(.*)

Remove

Remove hyphens between date at Beginning of String cont.

Using: 2020-01-01_this-is a test.jpg

Match: (\d\d\d\d)-(\d\d)-(\d\d)(.*)

Where:

\1 = 2020 \2 = 01 \3 = 01 \4 = _this-is a test

The First thing I like to do when a RegEx contains multiple Metacharacters if possible is simplify it down:

Match: (\d{4})-(\d{2})-(\d{2})(.*)

Tim Analysis:

1. \d{4}	Match against 4 numeric digits	Capture Group 1 = 2020
2. -	Match against <hyphen> (Not Captured) This is the <hyphen> after '2020'	
3. \d{2}	Match against 2 numeric digits This is the first '01' in the string.	Capture Group 2 = '01'
4. -	Match against <hyphen> (Not Captured) This is the <hyphen> after the first '01'	
5. \d{2}	Match against 2 numeric digits This is the second '01' in the string.	Capture Group 3 = '01'
6. .*	Match against remaining string	Capture Group 4 = _this-is a test

Remove

Remove hyphens between date at Beginning of String cont.

01-01-2020_this-is-1-test	to	01012020_this-is-1-test
05-10-2020_this-is_2-test.jpg	to	05102020_this-is_2-test.jpg
25032020_this-should be-ignored.pdf	to	Ignore
2020-01-01_this-is a test.jpg	to	20200101_this-is a test.jpg

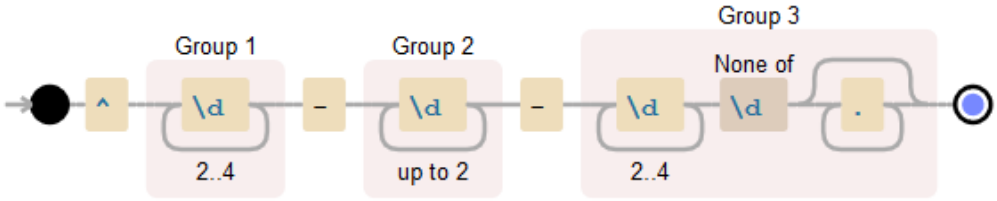
Observations:

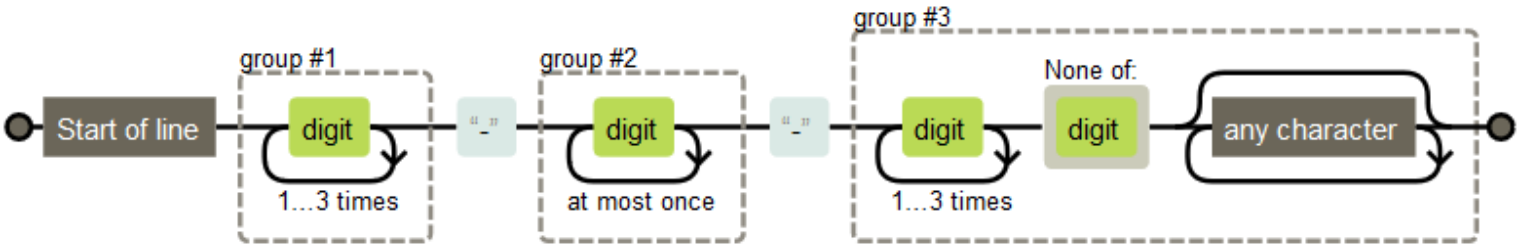
1. Pattern changes to a, '<hyphen> anywhere within the first 10 characters' (still date format).
2. Date no longer restricted to YYYY-MM-DD
3. Ignore files that have no <hyphens> within first 10 characters.

This will match against dates in any order as long as the pattern uses a date that has the year as 4 numeric digits, Year = YYYY, the month as 2 numeric digits, Month = MM, and the day as 2 numeric digits, Day = YY.

Match: ^(\d{2,4})-(\d{1,2})-(\d{2,4}[^\d].*)
Replace: \1\2\3

Name	New Name
01-01-2020_this-is-1-test	01012020_this-is-1-test
05-10-2020_this-is_2-test.jpg	05102020_this-is_2-test.jpg
25032020_this-should be-ignored.pdf	25032020_this-should be-ignored.pdf
2020-01-01_this-is a test.jpg	20200101_this-is a test.jpg





Sample:

```
.........................................
Match 1: 05-10-2020_this-is_2-test.jpg
Group 1: 05
Group 2: 10
Group 3: 2020_this-is_2-test.jpg
```

Remove

Remove hyphens between date at Beginning of String cont.

Using: 05-10-2020_this-is_2-test.jpg

Match: ^(\d{2,4})-(\d{1,2})-(\d{2,4}[^\d].*)

Where:

\1 = 05 \2 = 10 \3 = 2020_this-is_2-test

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. \d{2,4}	Specifies a minimum of 2 and maximum of 4 match repetitions until…	Capture Group 1 = 05
3. -	.. Match against <hyphen> (Not Captured) This is the <hyphen> after '05'	
4. \d{1,2}	Specifies a minimum of 1 and maximum of 2 match repetitions until…	Capture Group 2 = 10
5. -	.. Match against <hyphen> (Not Captured) This is the <hyphen> after '10'	
6. \d{2,4}	Specifies a minimum of 2 and maximum of 4 match repetitions until…	Capture Group 3 = 2020
7. [^\d]	Match against a Negated Class consisting of a numeric digit	Capture Group 3 = 2020_
6. .*	Match against remaining string	Capture Group 4 = 2020_this-is_2-test

Tim Notes:

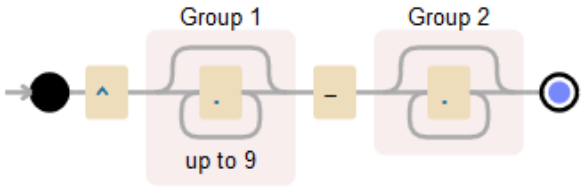
1. Doing a self-analysis also allows you to see where you could isolate more elements of the string if desired. Here you can see the '2020' could easily be isolated from the remainder of the string in step 6 by just placing it in a separate Capture Group. You could also isolate the underscore in step 7 the same way.

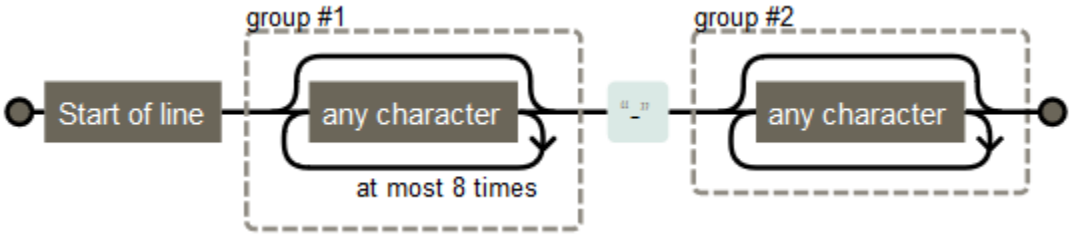
Remove

Remove hyphens between date at Beginning of String: Example #3

If you need to remove/replace <hyphens> within any – not just dates – first 10 characters of string:

Match: ^(.{0,9})-(.*)
Replace: \1\2





Sample:

```
.....................................
Match 1: 01-01-2020_this-is-1-test
Group 1: 01-01
Group 2: 2020_this-is-1-test
```

```
Beginning match attempt at character 0
1  ok
2  01-01-202
3  01-01-202
4  01-01-202backtrack
5  01-01-20
6  01-01-20
7  01-01-20backtrack
8  01-01-2
9  01-01-2
10 01-01-2backtrack
11 01-01-
12 01-01-
13 01-01-backtrack
14 01-01
15 01-01
16 01-01-
17 01-01-2020_this-is-1-test
18 01-01-2020_this-is-1-test
   Match found in 18 steps
```

This is meant for multiple runs. It removes only the last <hyphen> from within the first 10 characters on each subsequent run.

Remove

Remove hyphens between date at Beginning of String: Example #3 cont.

If you need to remove/replace <hyphens> within any – not just dates – first 10 characters of string: cont.

Match: ^(.{0,9})-(.*)

Using: 01-01-2020_this-is-1-test

Where:

\1 = 01-01 \2 = 2020_this-is-1-test

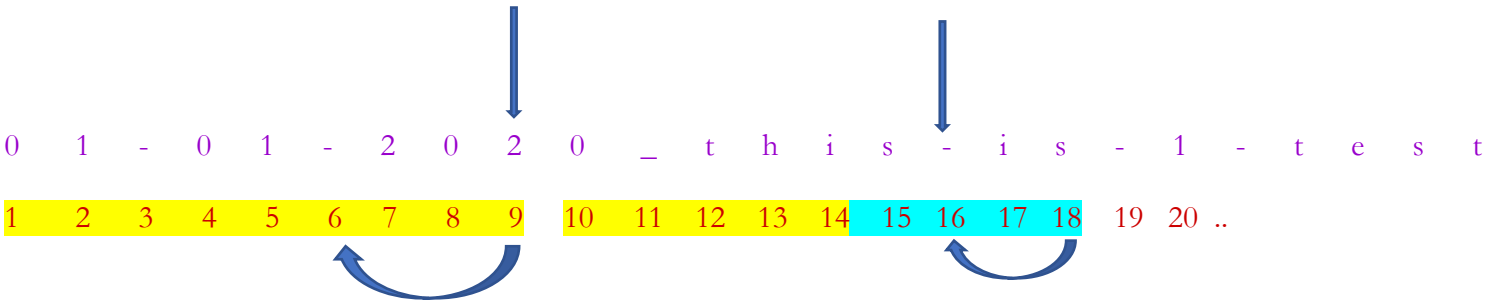
Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = 0
3. {0,9}	Specifies a minimum of 0 and maximum of 9 match repetitions until…	Capture Group 1 = 01-01-202
4. -	.. Match against <hyphen> (Not Captured) = - Backtracked. This is the <hyphen> preceding '2020'	Changes Capture Group 1 = 01-01

Tim Notes:

This requires further explanation:

Why didn't the RegEx engine move forward to match against the <hyphen> after 'this' ? Because it is still matching within the 0-9 character positions. In step 3 it reached the 9th position in the string:



Now it needs to match against the <hyphen> within those 9 character positions, so it backtracks. This range value could have been from {0,9} to {0,14} , where 0 = no repetitions, and still match the previous <hyphen> without matching against the <hyphen > after 'this' .

In order to include the <hyphen> after 'this' the RegEx engine position would have had to have been anywhere from {0,15} to {0,18}.

5. .*	Match against remaining string	Capture Group 2 = 2020_this-is-1-test

Remove

Remove hyphens between date at Beginning of String cont.

First run:

Name	New Name
01-01-2020_this-is-1-test	01-012020_this-is-1-test
05-10-2020_this-is_2-test.jpg	05-102020_this-is_2-test.jpg
25032020_this-should be-ignored.pdf	25032020_this-should be-ignored.pdf
2020-01-01_this-is a test.jpg	2020-0101_this-is a test.jpg

```
.....................................
Match 1: 2020-01-01_this-is a test.jpg
Group 1: 2020-01
Group 2: 01_this-is a test.jpg
```

Second run:

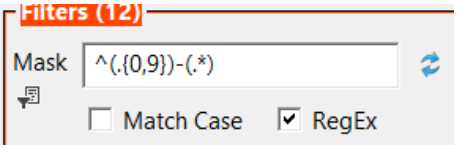
Name	New Name
01-012020_this-is-1-test	01012020_this-is-1-test
05-102020_this-is_2-test.jpg	05102020_this-is_2-test.jpg
25032020_this-should be-ignored.pdf	25032020_this-should be-ignored.pdf
2020-0101_this-is a test.jpg	20200101_this-is a test.jpg

```
.................................
Match 1: 01-012020_this-is-1-test
Group 1: 01
Group 2: 012020_this-is-1-test
```

After second run – there are no more matches:

Name	New Name
01012020_this-is-1-test	01012020_this-is-1-test
05102020_this-is_2-test.jpg	05102020_this-is_2-test.jpg
25032020_this-should be-ignored.pdf	25032020_this-should be-ignored.pdf
20200101_this-is a test.jpg	20200101_this-is a test.jpg

Another thing you can do is to display only those files that match the criteria so you don't need to select them individually. You can do this through, 'Section #12: Filters'



Mask: ^(.{0,9})-(.*)
RegEx: Enabled

Name	New Name
01-012020_this-is-1-test	01-012020_this-is-1-test
05-10-2020_this-is_2-test.jpg	05-10-2020_this-is_2-test.jpg
2020-01-01_this-is a test.jpg	2020-01-01_this-is a test.jpg

Press F5 or use the blue Filter Refresh button to the right of the Mask data field to display the files.

Forum Contributors: Admin, therube, RegexNinja

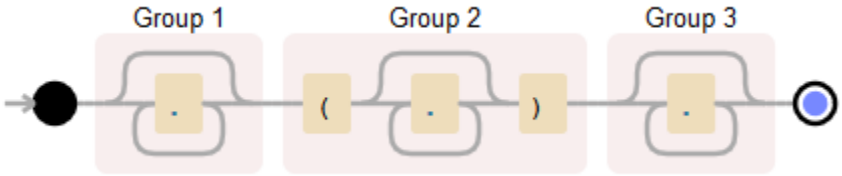
Remove

Remove Text in Parentheses or Brackets

test (U) test	to	test test
(iND) test	to	test
test (Xenophobia)	to	test

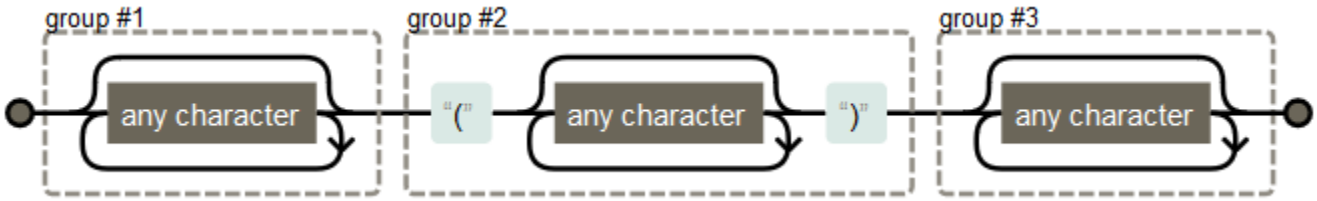
Match: (.*)(\(.*\))(.*)
Replace: \1\3

Name	New Name
test (Xenophobia)	test
test (U) test	test test
(iND) test	test



Sample:

```
.......................
Match 1: test (U) test
Group 1: test
Group 2: (U)
Group 3:  test
```



Tim Notes:

1. Just change to (.*)(\[.*\])(.*) for brackets.

Forum Contributor: Admin

Remove

Remove Text in Parentheses or Brackets cont.

Using: test (U) test

Match: (.*)(\(.*\))(.*)

Where:

\1 = test<space> \2 = (U) \3 = <space> test

Tim Analysis:

1..	Match against any character	Capture Group 1 = t
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \(	Match against literal left parenthesis	Capture Group 2 = (Changes Capture Group 1 = test<space>
4..	Match against any character	Capture Group 2 = (U
5. *	Make it Greedy	Capture Group 2 = (U) test
6. \)	Match against literal right parenthesis	Capture Group 2 = (U) Changes Capture Group 1 = Unchanged
7. .*	Match against remaining string	Capture Group 3 = <space>test

Remove

Remove Text in Parentheses or Brackets cont.

test (U) test	to	test test
(iND) test	to	test
test (Xenophobia)	to	test

Tim Contribution –

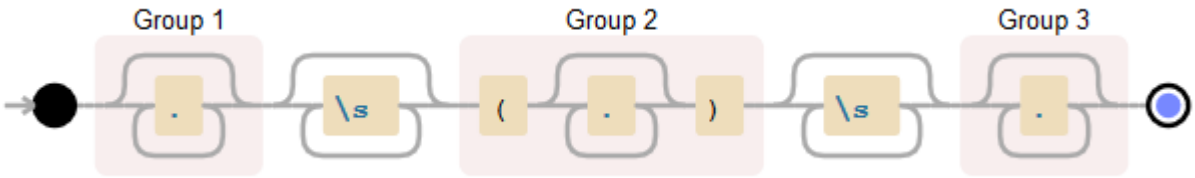
Get rid of any trailing excess <spaces> after parentheses.

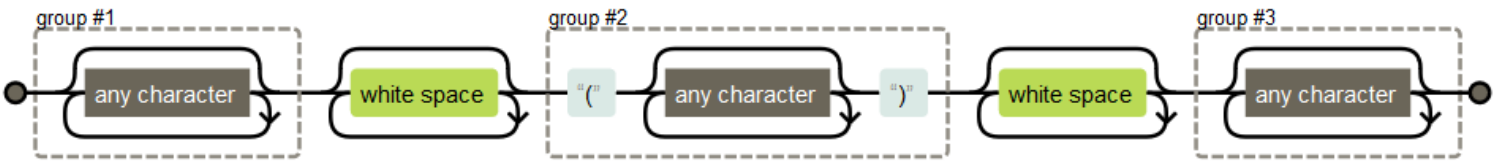
Match (.*)\s*(\(.*\))\s*(.*)
Replace: \1\3

Name	New Name
test (Xenophobia)	test
test (U) test	test test
(iND) test	test

Sample:

```
Match 1: test (U) test
Group 1: test
Group 2: (U)
Group 3: test
```





Remove

Remove Text in Parentheses or Brackets cont.

Using: test (U) test

Tim Contribution – cont.

Get rid of any trailing excess <spaces> after parentheses. cont.

Match (.*)\s*(\(.*\))\s*(.*)

Where:

\1 = test <space> \2 = (U) \3 = test

Tim Analysis:

1..	Match against any character	Capture Group 1 = t
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. \s	Match against <space> character, followed by … (Not Captured). This is the <space> after '(U)'.	Changes Capture Group 1 = test (U)
4. *	Make it Greedy (Not Captured). Matches to EOL	Changes Capture Group 1 = <entire string>
5. \(	Match against literal left parenthesis Backtracks.	Capture Group 2 = (Changes Capture Group 1 = test<space>
6. \s	Match against <space> character (Not Captured). This is the <space> after '(U)'.	Change Capture Group 2 = (U)
7..*	Match against remaining string	Capture Group 3 = test

Remove

Remove Text in Parentheses or Brackets cont.

test (U) test	to	test test	
(iND) test	to	Ignore	(if first word using \1, otherwise if \5 = test)
test (Xenophobia)	to	test	
test (U) test	to	test test	

Tim Contribution – cont.

Get rid of any leading or trailing excess <spaces> before or after parentheses.

It can handle one or multiple <spaces>:

Match: (\w*)(\s*)(\D*\))(\s*)(.*)

Where:

\1 = First word in string without parentheses
\2 = captured <spaces> before word in parentheses
\3 = word in parentheses
\4 = <spaces> after word in parentheses
\5 = last word in string without parentheses

Using a Replacement String of \1

Name	New Name
test (U) test	test
(iND) test	(iND) test
test (U) test	test
test (Xenophobia)	test

This displays the captured *first word without parentheses* of each sample.

Since 'test' is the *last word without parentheses* of the 'iND) test' sample, it is not included in capture Group 1, but is included in Capture Group 5 (the (.*) at the end of the RegEx).

Remove

Remove Text in Parentheses or Brackets cont.

test (U) test	to	test test	
(iND) test	to	Ignore	(if first word using \1, otherwise if \5 = test)
test (Xenophobia)	to	test	
test (U) test	to	test test	

Tim Contribution – cont.

Get rid of any leading or trailing excess <spaces> before or after parentheses. cont.

Match: (\w*)(\s*)(\D*\))(\s*)(.*)

Using a Replacement String of \5:

Name	New Name
test (U) test	test
(iND) test	test
test (U) test	test
test (Xenophobia)	test (Xenophobia)

This displays the captured *last word without parentheses* of each sample.
The sample 'test (Xenophobia)' would not be included because it has no words without parentheses after '(Xenophobia)'

Using a Replacement String of \1\5 displays as:

Name	New Name
test (U) test	testtest
(iND) test	test
test (U) test	testtest
test (Xenophobia)	test

To add the <space> between 'test' (first word without parentheses of string – Capture Group 1) and 'test' (last word without parentheses of string – Capture Group 5), use a Replacement String of, '\1 <space> \5' :

Name	New Name
test (U) test	test test
(iND) test	test
test (U) test	test test
test (Xenophobia)	test

The odd <space> for the sample, '(iND) test' is because it is the last word without parentheses (Capture Group 5). There are no other like words, so the <space> is added before not after.

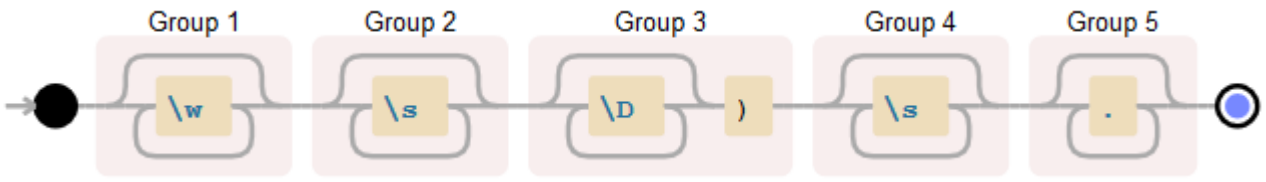
Remove

Remove Text in Parentheses or Brackets cont.

Tim Contribution – cont.

Get rid of any leading or trailing excess <spaces> before or after parentheses. cont.

Match: (\w*)(\s*)(\D*\))(\s*)(.*)



Sample: Single <spaces> between words

		Start	Length
Match 1:	test (U) test	0	13
Group 1:	test	0	4
Group 2:		4	1
Group 3:	(U)	5	3
Group 4:		8	1
Group 5:	test	9	4

Sample: Multiple <spaces> between words

		Start	Length
Match 1:	test (U) test	0	16
Group 1:	test	0	4
Group 2:		4	3
Group 3:	(U)	7	3
Group 4:		10	2
Group 5:	test	12	4

Sample: No last words in string without parentheses

		Start	Length
Match 1:	test (Xenophobia)	0	17
Group 1:	test	0	4
Group 2:		4	1
Group 3:	(Xenophobia)	5	12
Group 4:		17	0
Group 5:		17	0

Sample: No first word in string without parentheses

		Start	Length
Match 1:	(iND) test	0	10
Group 1:		0	0
Group 2:		0	0
Group 3:	(iND)	0	5
Group 4:		5	1
Group 5:	test	6	4

Remove

Remove Text in Parentheses or Brackets cont.

Using: test (U) test (test <3 spaces> (U) <2 spaces> test

Tim Contribution – cont.

Get rid of any leading or trailing excess <spaces> before or after parentheses. cont.

Match: (\w*)(\s*)(\D*\))(\s*)(.*)

Where:

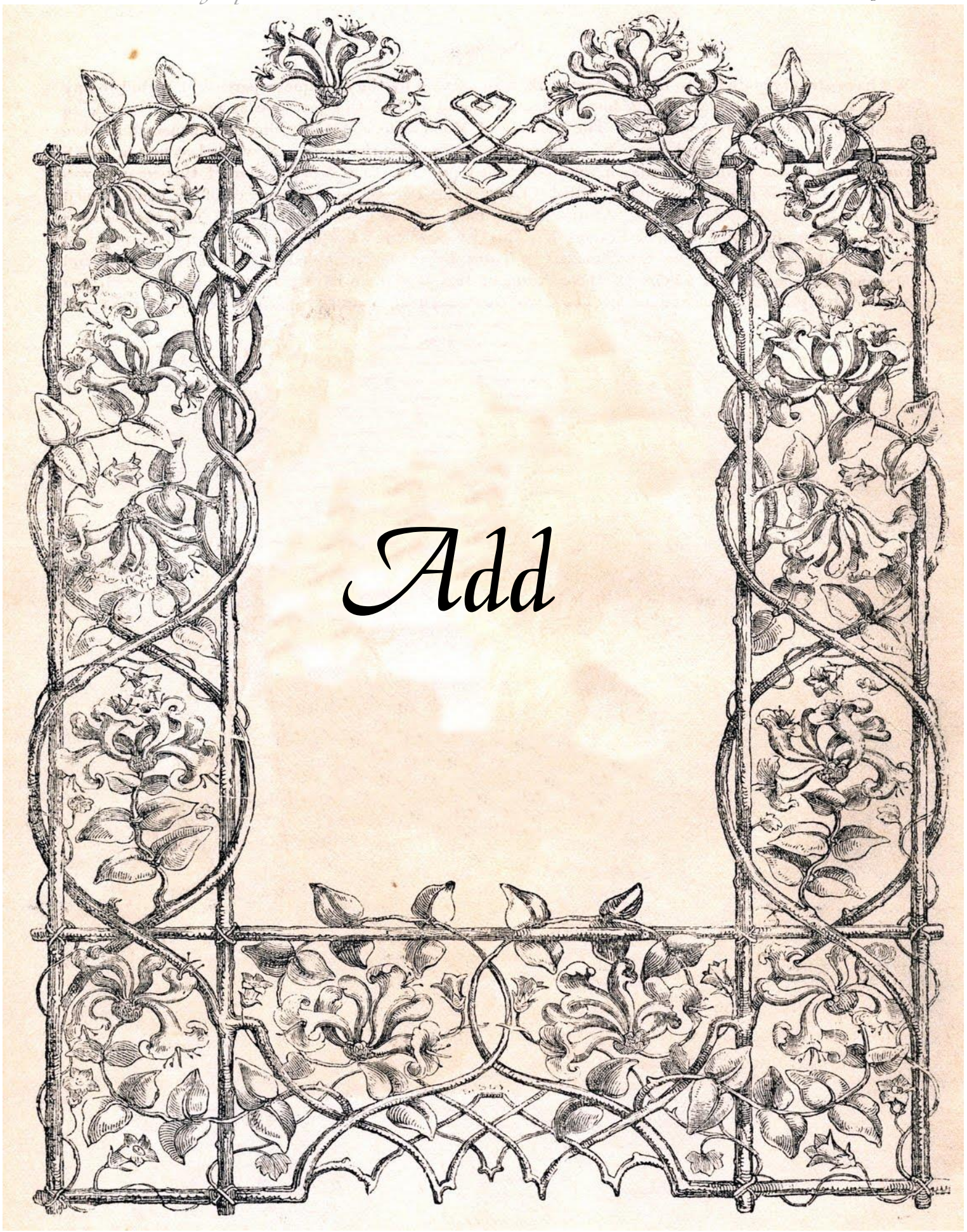
\1 = test \2 = multiple <spaces> \3 = (U) \4 = multiple <spaces> \5 = test

Breaks down:

Capture Group 1	Capture Group 2	Capture Group 3	Capture Group 4	Capture Group 5
		literal		
(\w*)	(\s *)	(\D* \))	(\s*)	(.*)
test	<spaces>	(U)	<spaces>	test

Tim Analysis:

1. \w	Word Metacharacter. Match against any single letter, number or underscore.	Capture Group 1 = t
2. *	Make it Greedy	Capture Group 1 = test
3. \s	Match against a <space> character This is the <space> after the first 'test' in the string	Capture Group 2 = <space>
4. *	Make it Greedy Matches against multiple <space> characters	Capture Group 2 = multiple <spaces>
5. \D	Match against any Non-Numeric Digit	Capture Group 3 = (
6. *	Make it Greedy	Capture Group 3 = (U) test
7. \)	Match against a literal right parenthesis	Capture Group 3 = (U)
8. \s	Match against a <space> character This is the <space> after the '(U)' in the string	Capture Group 4 = <space>
9. *	Make it Greedy Matches against multiple <space> characters	Capture Group 4 = multiple <spaces>
10. .*	Match against remaining string	Capture Group 5 = test


Add

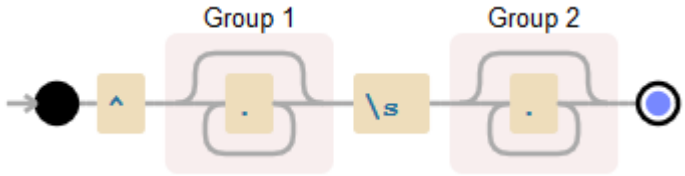
Add

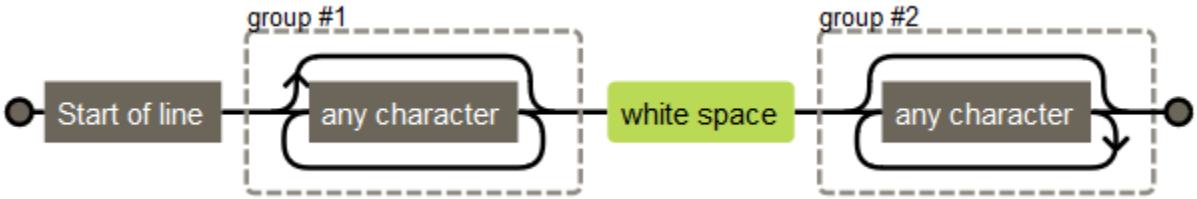
Add Comma after first item, i.e., Last Name

William George Steven to William, George Steven

Match: ^(.*?)\s(.*)
Replace: \1, \2

Name	New Name
William George Steven.pdf	William, George Steven.pdf





Sample:

```
Match 1: William George Steven
Group 1: William
Group 2: George Steven
```

Contributor Analysis:

1. Non-greedily match everything up to the first <space>
2. Eat the <space>
3. Match everything else
4. Add them back together with a ,<space> in between

Forum Contributor: therube

Add

Add Comma after first item, i.e., Last Name cont.

Using: William George Steven

Match: ^(.*?)\s(.*)

Where:

\1 = William \2 = George Steven

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = W
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy Current position = BOL	Capture Group 1 = null
5. \s	Match against a <space> character (Not Captured) This is the <space> after 'William'	Changes Capture Group 1 = William
6. .*	Match against remaining string	Capture Group 2= George Steven

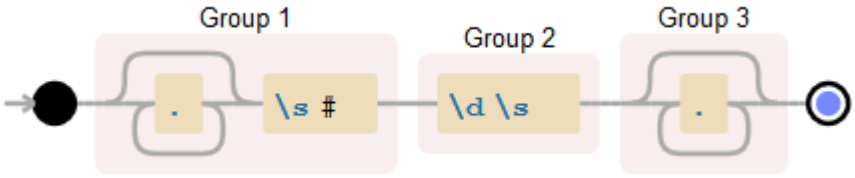
Add

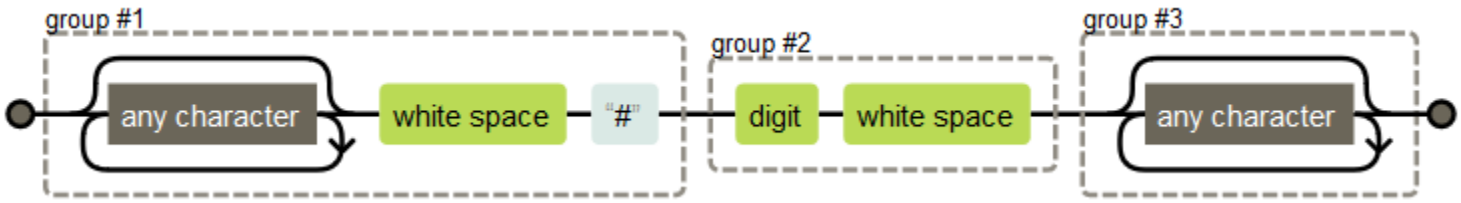
Add Leading zero after

Jessica Jones V2016 #1 (2016).cbz to Jessica Jones V2016 #01 (2016).cbz

Match: (.*\s#)(\d\s)(.*)
Replace: \10\2\3

Name	New Name
Jessica Jones V2016 #1 (2016).cbz	Jessica Jones V2016 #01 (2016).cbz





Sample:

```
.....................................
Match 1: Jessica Jones V2016 #1 (2016)
Group 1: Jessica Jones V2016 #
Group 2: 1
Group 3: (2016)
```

Forum Contributor: therube

Add

Add Leading zero after # cont.

Using: Jessica Jones V2016 #1 (2016).cbz

Match: (.*\s#)(\d\s)(.*)

Where:

\1 = Jessica Jones V2016 # \2 = 1 <space> \3 = (2016)

Tim Analysis:

1..	Match against any character	Capture Group 1 = J
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. \s	Match against a <space> character Backtracks.	Capture Group 1 = Jessica Jones V2016 #1<space>
4. #	Match against '#'	Capture Group 1 = Jessica Jones V2016 #
5. \d	Match against Numeric Digit	Capture Group 2 = 1
6. \s	Match against <space> character	Capture Group 2 = 1<space>
7. .*	Match against remaining string	Capture Group 3 = (2016)

Add

Padding with Zeros

05m01	to	0005_001	05 is padded with 2 Zeros and the 01 is padded with 1 zero
23m34	to	0023_034	23 is padded with 2 Zeros and the 34 is padded with 1 zero
712m182	to	0712_182	712 is padded with 1 zero, and the 182 already equals 3 places so it is ignored

This one requires:

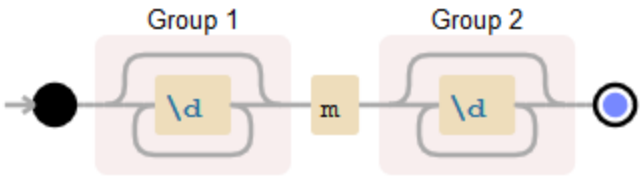
1. Pad the numerals before m with Zeros to equal 4 places
2. Pad the numeral after m with Zeros to equal 3 places.
3. Obviously do not add any padding to elements that are already at maximum places

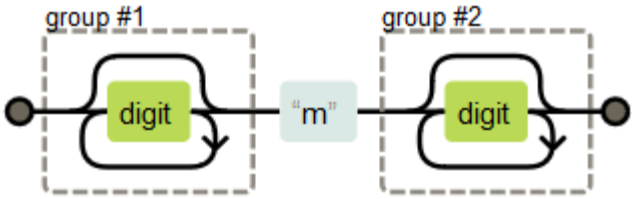
Will be done in two separate runs:

First Run -

Match: (\d*)m(\d*)
Replace: 000\1_00\2

Name	New Name
712m182	000712_00182
23m34	00023_0034
05m01	00005_0001





Sample:

```
.................
Match 1: 05m01
Group 1: 05
Group 2: 01
```

This First run adds the additional Zeros not taking into account the current numeric places.

Add

Padding with Zeros cont.

Using: 05m01

First Run – cont.

Match: (\d*)m(\d*)

Where:

\1 = 05 \2 = 01

Tim Analysis:

1. \d	Match against Numeric Digit	Capture Group 1 = 0
2. *	Make it Greedy	Capture Group 1 = 05
3. m	Match against 'm' (Not Captured) = m	
4. \d	Match against Numeric Digit	Capture Group 2 = 0
5. *	Make it Greedy	Capture Group 2 = 01

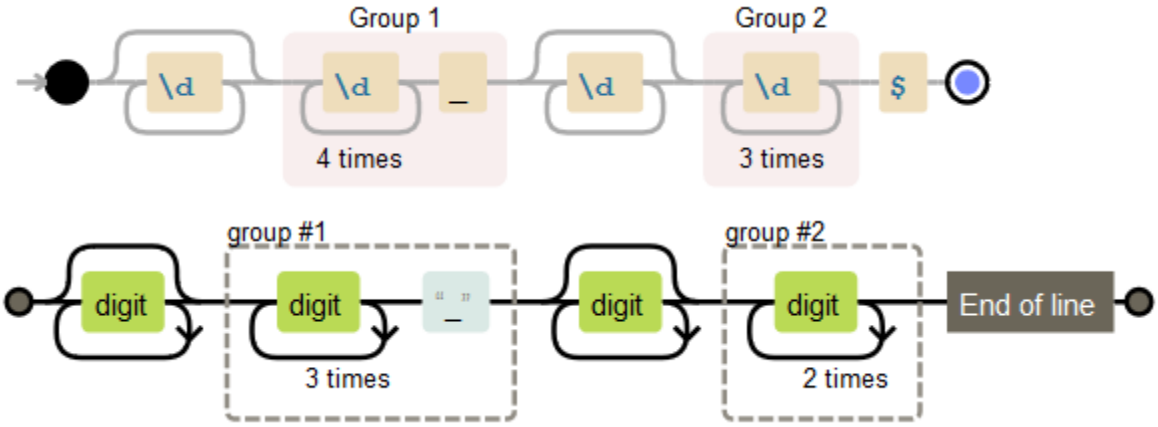
Add

Padding with Zeros cont.

Second Run -

Match: \d*(\d{4}_)\d*(\d{3})$
Replace: \1\2

Name	New Name
000712_00182	0712_182
00023_0034	0023_034
00005_0001	0005_001



Sample:

```
.....................
Match 1: 00005_0001
Group 1: 0005_
Group 2: 001
```

This Second run limits each of the numeric values to their designated digital places – 4 and 3

Forum Contributor: KenP

Add

Padding with Zeros cont.

Using: 00005_0001

Second Run – cont.

Match: \d*(\d{4}_)\d*(\d{3})$

Where:

\1 = 0005_ \2 = 001

Tim Analysis:

1. \d	Match against Numeric Digit (Not Captured) = '0'	
2. *	Make it Greedy (Not Captured) = '00005'	
3. \d{4}	Match against 4 numeric digits Changes the uncaptured value to '0'.	Capture Group 1 = 0005

Tim Notes:

1. The current position is the '5'. The RegEx Engine tries to match looking for 4 numeric digits, but it encounters the underscore as the next character so it backtracks and matches the '0005'. This in turn changes the value of the uncaptured in step 2 to '0'.

4. _	Match against underscore	Capture Group 1 = 0005_
5. \d	Match against Numeric Digit (Not Captured) = '0'	
6. *	Make it Greedy (Not Captured) = '001'	
7. \d{3}	Match against 3 numeric digits Changes Not Captured value to '0'	Capture Group 2 = 001

Tim Notes:

1. Same as before. The current position is the '1'. The RegEx Engine tries to match looking for 3 numeric digits, but it encounters the EOL so it backtracks and matches the '001'. This in turn changes the value of the uncaptured in step 6 to '0'.

2. These two procedures of, 'matching against the numeric digits but not capturing' and then 'backtracking to match against previously captured numeric digits', is how the excess leading Zeros are removed.

8. $ Must be EOL

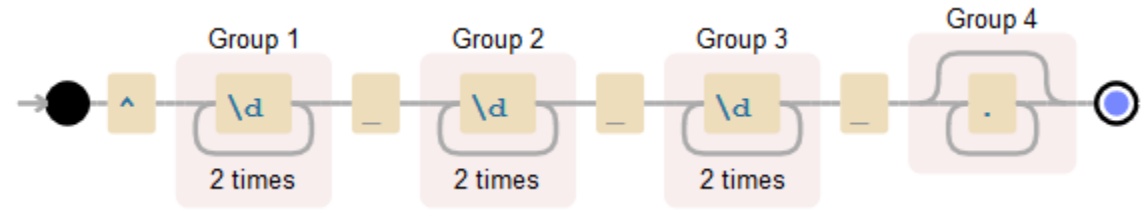
Add

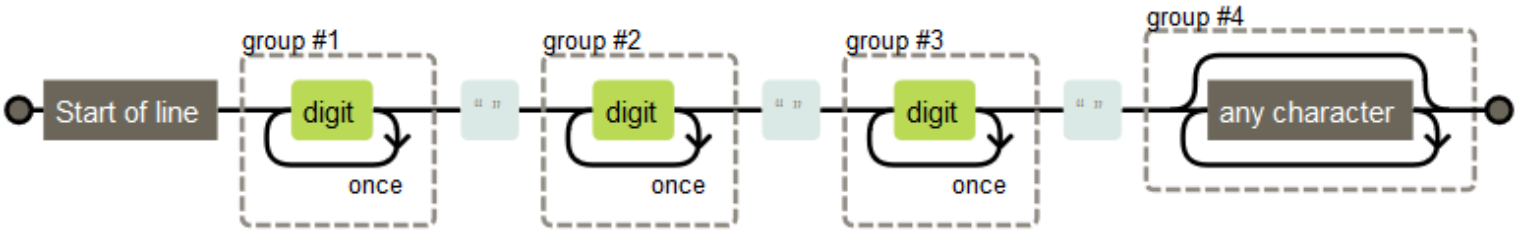
Add the Century to YY in string and move to end

16 05 07 Protocol V7_rev3 to Protocol V7_rev3_2016_05_07

Match: ^(\d{2}) (\d{2}) (\d{2}) (.*)
Replace: \4_20\1_\2_\3

Name	New Name
16 05 07 Protocol V7_rev3	Protocol V7_rev3_2016_05_07





Sample:

```
.....................................
Match 1: 16 05 07 Protocol V7_rev3
Group 1: 16
Group 2: 05
Group 3: 07
Group 4: Protocol V7_rev3
```

Contributor Notes:

1. Change 20 in 'Replace' as needed to reflect century to add, i.e. 19 for 1900's, etc.

Forum Contributor: KenP

Add

Add the Century to YY in string and move to end cont.

Using: 16 05 07 Protocol V7_rev3

Match: ^(\d{2}) (\d{2}) (\d{2}) (.*)

Where:

\1 = 16 \2 = 05 \3 = 07 \4 = Protocol V7_rev3

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. \d{2}	Match against 2 numeric digits followed by…	Capture Group 1 = 16
3. <space>	.. Match against a <space> character (Not Captured) followed by…	
4. \d{2}	.. Match against 2 numeric digits followed by…	Capture Group 2 = 05
5. <space>	.. Match against a <space> character (Not Captured) followed by…	
6. \d{2}	.. Match against 2 numeric digits followed by…	Capture Group 3 = 07
7. <space>	.. Match against a <space> character (Not Captured)	
8. .*	Match against remaining string	Capture Group 4 = Protocol V7_rev3

Add

Add additional 'Year' as second item, remove hyphen

Name - Text [typ,year,info,info] to Name year Text [typ,year,info,info]

Observations:

1. Remove first <hyphen>.
2. First pattern is Name <space> - <space>
3. Year can be YY or YYYY
4. The second 'info' in brackets may or may not be present
5. There could also be a second 'typ' within the brackets
6. Capture year from brackets and add after 'Name'

Tim Conclusion:

1. Since the information in the brackets doesn't change, the number of variable text between them is not relevant.
2. Since the year is always present, and needs to be captured, it is best to use a universal \d rather than one that selects between YY and YYYY.

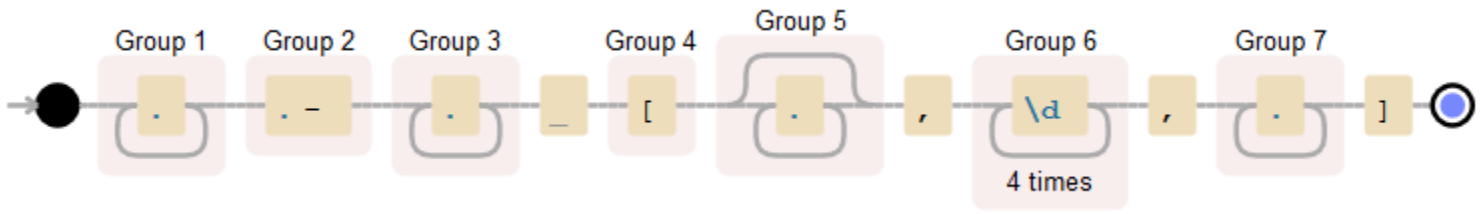
Pattern is established.

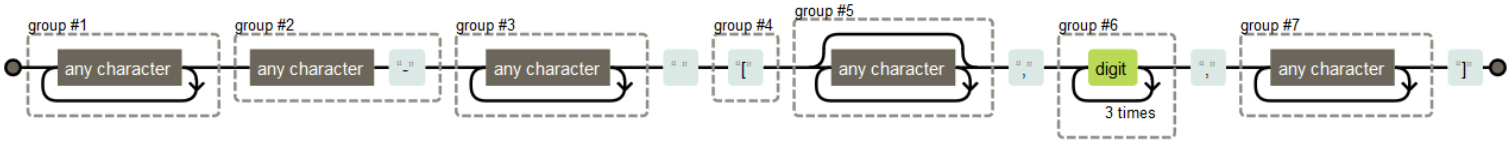
Samples:

Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb] to Name 2014 Text [sddadds,2014,ewrvvegfvvf,yvbvbcb]
Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff] to Name 2016 Text2 [zuii,tgfgdf,2016,wqxccff]

Match: (.+)(.\-)(.+) (\[)(.*),(\d{4})\,(.+)\]
Replace: \1 \6\3 [\5,\6,\7]

Name	New Name
Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]	Name 2016 ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]
Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]	Name 2014 abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]





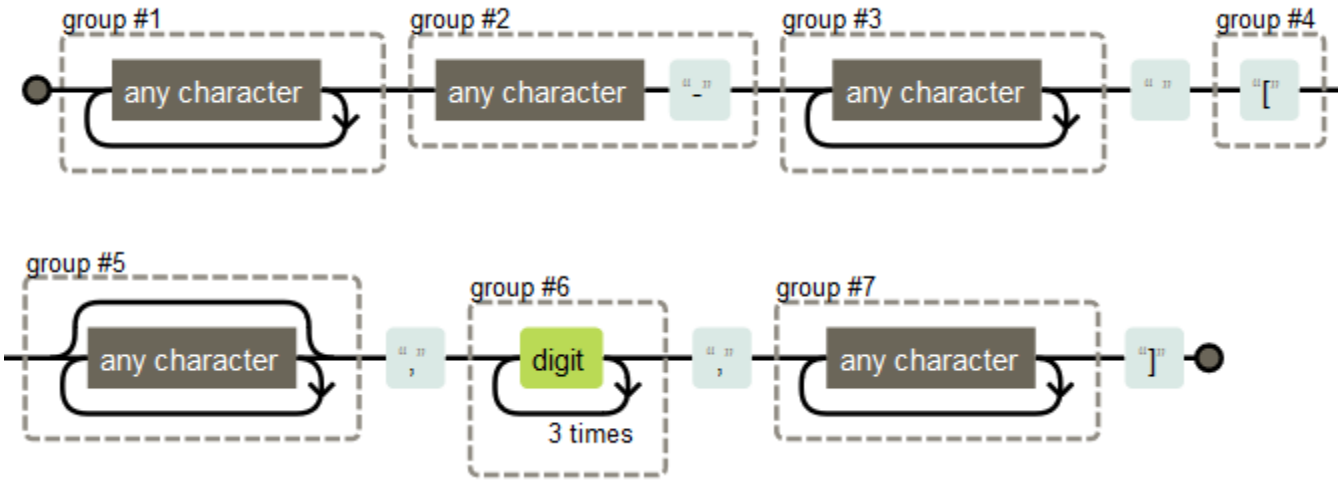
Add

Add additional 'Year' as second item, remove hyphen cont.

Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb] to Name 2014 Text [sddadds,2014,ewrvvegfvvf,yvbvbcb]
Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff] to Name 2016 Text2 [zuii,tgfgdf,2016,wqxccff]

Match: (.+)(.\-)(.+) (\[)(.*),(\d{4})\,(.+)\]

Expanded:



Sample:

```
.....................................................
Match 1: Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]
Group 1: Name
Group 2:  -
Group 3:  ewrwrsfwefwer
Group 4: [
Group 5: zuii,tgfgdf
Group 6: 2016
Group 7: wqxccff
```

Where:

\1 = Name \2 = <space> - \3 = <space> ewrwrsfwefwer \4 = [
\5 = zuii,tgfgdf \6 = 2016 \7 = wqxccff

Add

Add additional 'Year' as second item, remove hyphen cont.

Using: Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]

Match: (.+)(.\-)(.+) (\[)(.*),(\d{4})\,(.+)\]

Tim Analysis:

1. .	Match against any character	Capture Group 1 = N
2. +	Make Greedy	Capture Group 1= <entire string>
3. .	Match against any character	Capture Group 2 =] Changes Capture Group 1 = Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff
4. -	Match against <hyphen> followed by …	Capture Group 2 = <space><hyphen> Changes Capture Group 1 = Name

Tim Notes:

1. <hyphen> does not need an Escape Metacharacter.

5. .	.. Match against any character	Capture Group 3 = <space>

Tim Notes:

1. This accounts for the <space> that follows <space><hyphen>

6. +	Make it Greedy	Capture Group 3 = <space> ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]
7. <space>	Match against <space> (Not Captured)	Capture Group 3 = <space> ewrwrsfwefwer
8. [	Match against left bracket	Capture Group 4 = [
9. .	Match against any character	Capture Group 5 = z
10. *	Make it Greedy	Capture Group 5 = zuii,tgfgdf,2016,wqxccff]
11. ,	Match against <comma> (Not Captured) This is the <comma> following '2016'	Changes Capture Group 5 = zuii,tgfgdf,2016 All Others = Unchanged
12. \d	Match against numeric digit Changes the <comma> to the <comma> preceding '2016'	Capture Group 6 = 2 Changes Capture Group 5 = zuii,tgfgdf
13. {4}	Repeat match of last expression 4 iterations (\d)	Capture Group 6 = 2016
14. ,	Match against <comma> (Not Captured) This is the comma following '2016'	
15. .	Match against any character	Capture Group 7 = w
16. +	Make Greedy	Capture Group 7= wqxccff]
17. \]	Match against right bracket (Not Captured)	Changes Capture Group 7 = wqxccff

Add

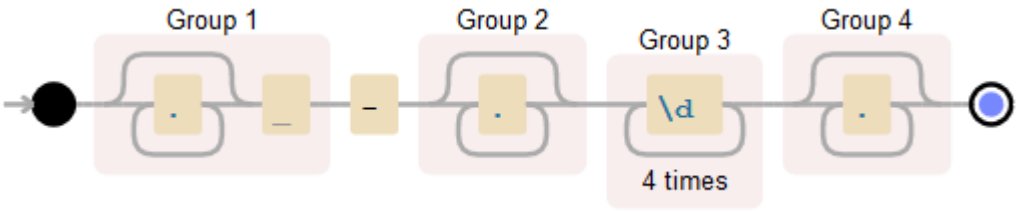
Add additional 'Year' as second item, remove hyphen cont.

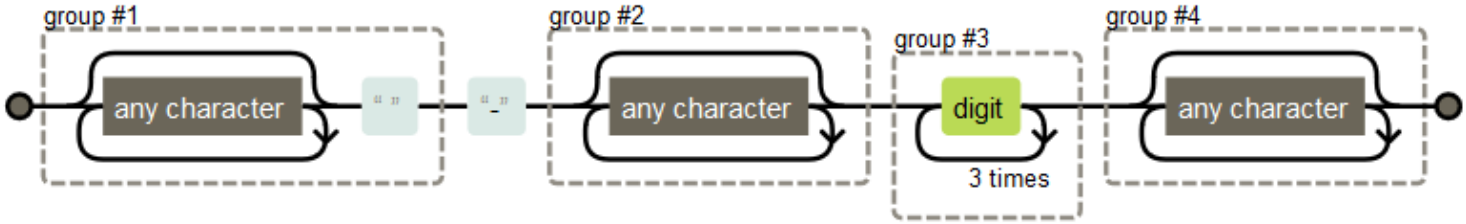
Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb] to Name 2014 Text [sddadds,2014,ewrvvegfvvf,yvbvbcb]
Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff] to Name 2016 Text2 [zuii,tgfgdf,2016,wqxccff]

And another way:

Match: (.*)-(.*)(\d{4})(.*)
Replace: \1\3\2\3\4

Name	New Name
Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]	Name 2016 ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]
Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]	Name 2014 abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]





Sample:

```
.....................................................................
Match 1: Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]
Group 1: Name
Group 2:  ewrwrsfwefwer [zuii,tgfgdf,
Group 3: 2016
Group 4: ,wqxccff]
```

Forum Contributor: therube

Add

Add additional 'Year' as second item, remove hyphen cont.

Using: Name - ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]

Match: (.*)-(.*)(\d{4})(.*)

Where:

\1 = Name \2 = <space> ewrwrsfwefwer [zuii,tgfgdf, \3 = 2016 \4 = ,wqxccff]

Tim Analysis:

1. .	Match against any character	Capture Group 1 = N
2. *	Make Greedy	Capture Group 1= <entire string>
3. <space>	Match against <space>	Capture Group 1 = Name - ewrwrsfwefwer
4. -	Match against <hyphen> (Not Captured)	Changes Capture Group 1 =Name<space>
5. .	Match against any character	Capture Group 2 = <space>
6. *	Make Greedy	Capture Group 2 = <space> ewrwrsfwefwer [zuii,tgfgdf,2016,wqxccff]
7. \d	Match against numeric digit	Capture Group 3 = 6 Changes Capture Group 2 = <space> ewrwrsfwefwer [zuii,tgfgdf,201
8. {4}	Repeat match of last expression 4 iterations (\d)	Capture Group 3 = 2016 Changes Capture Group 2 = <space> ewrwrsfwefwer [zuii,tgfgdf,
9. .	Match against any character This is the <comma> after '2016'	Capture Group 4 = ,
10. *	Make Greedy	Capture Group 4 = ,wqxccff] All Others = Unchanged

Add

Add additional 'Year' as second item, remove hyphen cont.

Tim Contribution –

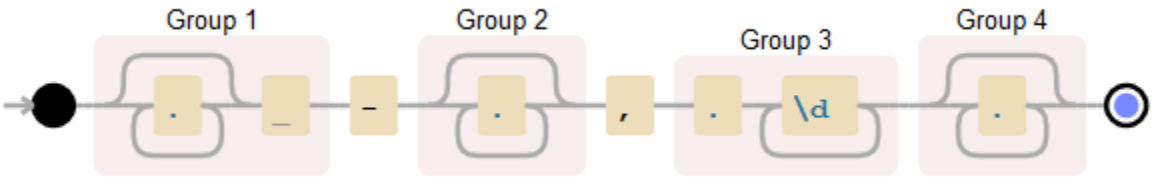
Tim Notes regarding Contributor RegEx:

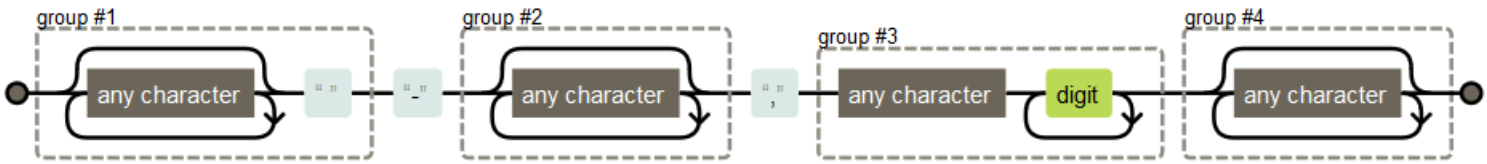
1. Matches samples
2. Does not account for a two digit year.

This does –

Match: (.*)-(.*),(.\d+)(.*)
Replace: \1\3\2\4

Name	New Name
Name - ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]	Name 16 ewrwrsfwefwer [zuii,tgfgdf,wqxccff]
Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]	Name 2014 abcdeferr [sddadds,ewrvvegfvvf,yvbvbcb]





Sample:

```
.....................................................
Match 1: Name - ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]
Group 1: Name
Group 2:  ewrwrsfwefwer [zuii,tgfgdf
Group 3: 16
Group 4: ,wqxccff]
```

Add

Add additional 'Year' as second item, remove hyphen cont.

Tim Contribution – cont.

Using: Name - ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]

Match: (.*)-(.*),(.\d+)(.*)

Where:

\1 = Name<space> \2 = <space> ewrwrsfwefwer [zuii,tgfgdf \3 = 16 \4 = ,wqxccff]

Tim Analysis:

1. .	Match against any character	Capture Group 1 = N
2. *	Make Greedy	Capture Group 1= <entire string>
3. <space>	Match against <space>	Capture Group 1 = Name - ewrwrsfwefwer
4. -	Match against <hyphen> (Not Captured)	Changes Capture Group 1 =Name<space>
5. .	Match against any character This is the <space> after 'Name -'	Capture Group 2 = <space>
6. *	Make Greedy	Capture Group 2= <space> ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]
7. ,	Match against <comma> (Not Captured) This is the <comma> after '16'	Changes Capture Group 2 = <space> ewrwrsfwefwer [zuii,tgfgdf,16
8. \d	Match against numeric digit The <comma> changes to the <comma> before '16'	Capture Group 3 = 1 Changes Capture Group 2 = <space> ewrwrsfwefwer [zuii,tgfgdf
9. +	Make Greedy	Capture Group 3 = 16 All Others = Unchanged
10. .*	Capture remainder of string	Capture Group 4 = ,wqxccff]

Add

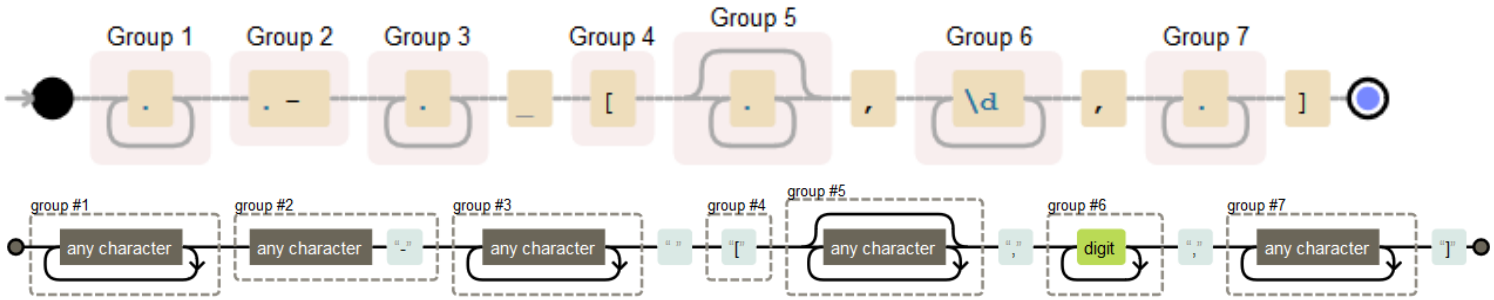
Add additional 'Year' as second item, remove hyphen cont.

Tim Contribution – cont.

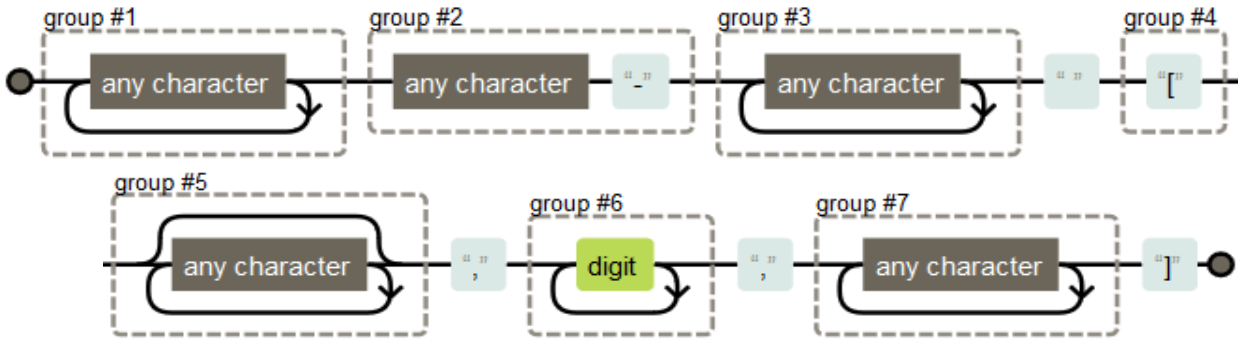
As does this –

Match: (.+)(.\-)(.+) (\[)(.*),(\d+)\,(.+)\]
Replace: \1 \6\3 [\5,\6,\7]

Name	New Name
Name - ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]	Name 16 ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]
Name - abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]	Name 2014 abcdeferr [sddadds,2014,ewrvvegfvvf,yvbvbcb]



Expanded:



Sample:

```
.....................................................
Match 1: Name - ewrwrsfwefwer [zuii,tgfgdf,16,wqxccff]
Group 1: Name
Group 2:  -
Group 3:  ewrwrsfwefwer
Group 4: [
Group 5: zuii,tgfgdf
Group 6: 16
Group 7: wqxccff
```

Tim Analysis:

This is the same as the Contributor's RegEx except one minor revision changes it from restrictive to universal when it comes to the Year. Changing from (\d{4}) to (\d+). That's all it took.

Add

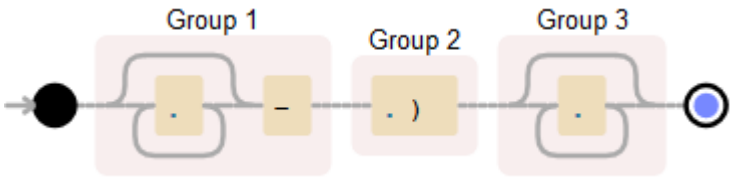
Pad Single Digit Day with Zero

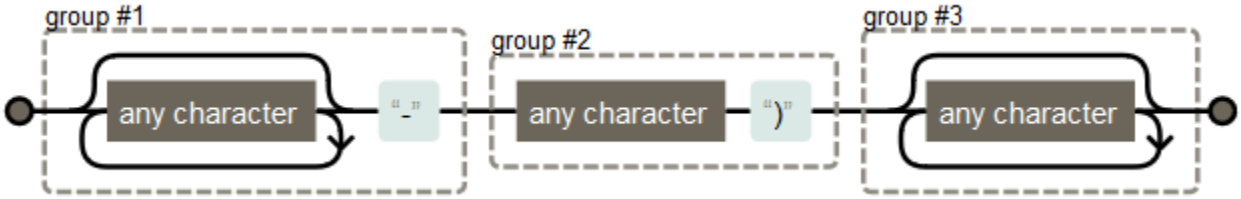
TIC-(YYYY-MM-D)_XXXX.pdf to TIC-(YYYY-MM-0D)_XXXX.pdf

TIC-(YYYY-MM-DD)_XXXX.pdf to Ignore

Match: (.*-)(.\))(.*)
Replace: \10\2\3

Name	New Name
TIC-(YYYY-MM-DD)_XXXX.pdf	TIC-(YYYY-MM-DD)_XXXX.pdf
TIC-(YYYY-MM-D)_XXXX.pdf	TIC-(YYYY-MM-0D)_XXXX.pdf





Sample:

```
..............................
Match 1: TIC-(YYYY-MM-D)_XXXX
Group 1: TIC-(YYYY-MM-
Group 2: D)
Group 3: _XXXX
```

Tim Notes:

1. The Replace \10 is \1 with a Zero following – there is no Capture Group 10

Forum Contributor: therube

Add

Pad Single Digit Day with Zero cont.

Using: TIC-(YYYY-MM-D)_XXXX.pdf

Match: (.*-)(.\))(.*)

Where:

\1 = TIC-(YYYY-MM- \2 = D) \3 = _XXXX

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen>	Capture Group 1 = TIC-(YYYY-MM-
	Backtracks to match against <hyphen> after 'MM'	
4. .	Match against any character	Capture Group 2 = D
5. \)	Match against a literal right parenthesis	Capture Group 2 = D)
6. .*	Match against remaining string	Capture Group 3 = _XXXX

Add

Add a hyphen before first numeric digit in string

Felix Publishing - Planes 01 - P-51 Mustang Vs Mitsubishi Zero - Pacific Theater 1944-45
to:
Felix Publishing - **Planes - 01** - P-51 Mustang Vs Mitsubishi Zero - Pacific Theater 1944-45

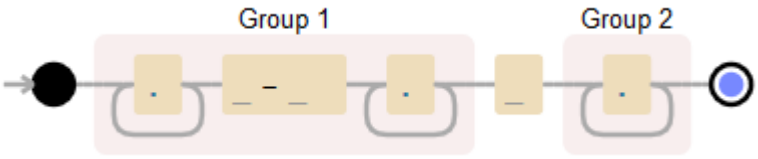
Osprey Publishing - Campaign 108 - Marathon 490 Bc - The First Persian Invasion Of Greece
to:
Osprey Publishing - **Campaign - 108** - Marathon 490 Bc - The First Persian Invasion Of Greece

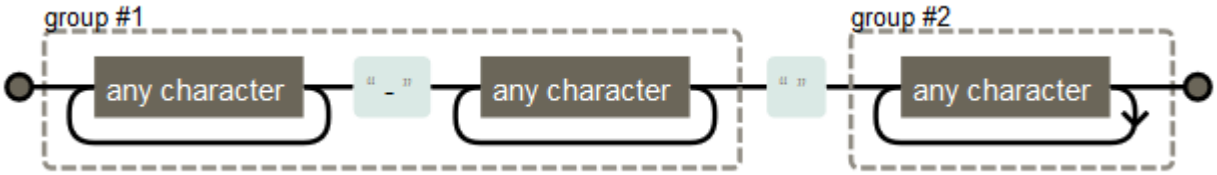
Osprey Publishing - Campaign 222 - Salamis 480 Bc - The Naval Campaign That Saved Greece
to:
Osprey Publishing - **Campaign - 222** - Salamis 480 Bc - The Naval Campaign That Saved Greece

Osprey Publishing - Command 11 – Hannibal
to:
Osprey Publishing - **Command - 11** - Hannibal

Match: (.+? - .+?) (.+)
Replace: \1 - \2

Name	New Name
Osprey Publishing - Command 11 – Hannibal	Osprey Publishing - Command - 11 – Hannibal
Osprey Publishing - Campaign 222 - Salamis 480 Bc - Th...	Osprey Publishing - Campaign - 222 - Salamis 480 Bc - The Naval Campaign That ...
Osprey Publishing - Campaign 108 - Marathon 490 Bc - ...	Osprey Publishing - Campaign - 108 - Marathon 490 Bc - The First Persian Invasion...
Felix Publishing - Planes 01 - P-51 Mustang Vs Mitsubishi...	Felix Publishing - Planes - 01 - P-51 Mustang Vs Mitsubishi Zero - Pacific Theater 1...





Sample:

```
Match 1: Osprey Publishing - Command 11 – Hannibal
Group 1: Osprey Publishing - Command
Group 2: 11 – Hannibal
```

Tim Notes:

1. **Bold** was used to illustrate change in Complex samples and is not part of renaming.

Forum Contributor: therube

Add

Add a hyphen before first numeric digit in string cont.

Using: Osprey Publishing - Command 11 - Hannibal

Match: (.+? - .+?) (.+)

Where:

\1 = Osprey Publishing - Command \2 = 11 - Hannibal

Tim Analysis:

1. .	Match against any character	Capture Group 1 = O
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1= O
4. <space>	Match against a <space> character followed by …	Capture Group 1= Osprey<space>
5. -	.. Match against <hyphen> followed by …	Capture Group 1 = Osprey Publishing -
6. <space>	Match against a <space> character	Osprey Publishing -<space

Tim Notes:

1. In step 4, the <space> was matched after 'Osprey' but this was changed to the <space> following Publishing in step 5 when it was matched against the <hyphen>

7. .	Match against any character	Capture Group 1 = Osprey Publishing - C
8. +	Make it Greedy	Capture Group 1 = <entire string>
9. ?	But not too Greedy	Capture Group 1= Osprey Publishing - C
10. <space>	Match against a <space> character Not Captured. This is the <space> after 'Command'	Changes Capture Group 1= Osprey Publishing - Command
11. .+	Match against remaining string	Capture Group 2 = 11 – Hannibal

Add

Pad random number prefix with Zeros

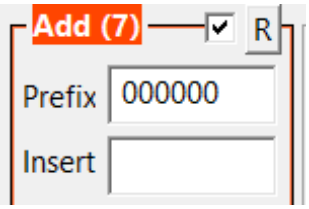
323 -this is file1.pdf	to	0000323 -this is file1.pdf
489834 - this is file two.pdf	to	0489834 - this is file two.pdf
9273 - this is file 3.pdf	to	0009273 - this is file3.pdf
2837261 -this is another file.pdf	to	2837261 -this is another file.pdf

Observations:

1. Take the largest number value in all of the filenames = 2837261
2. The number of numeric digits represented by that number minus one will equal the number of padded Zeros = (7-1) = 6
   a. The reason 1 is subtracted is because the highest value will not need padding, therefore the maximum number of padded places is 6 and not 7.

Two runs:

First run – 'Section #7: Add' -



Will be used to add 6 Zeros to the prefix of each filename

After first run:

Name	New Name
323 -this is file1.pdf	000000323 -this is file1.pdf
489834 - this is file two.pdf	000000489834 - this is file two.pdf
9273 - this is file 3.pdf	0000009273 - this is file 3.pdf
2837261 -this is another file.pdf	0000002837261 -this is another file.pdf

Each file contains a equal number of Zeros in the prefix.

Add

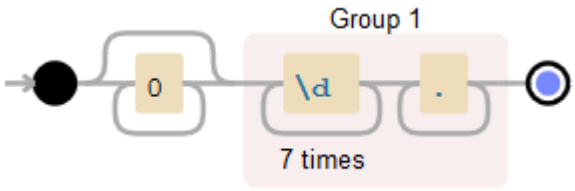
Pad random number prefix with Zeros cont.

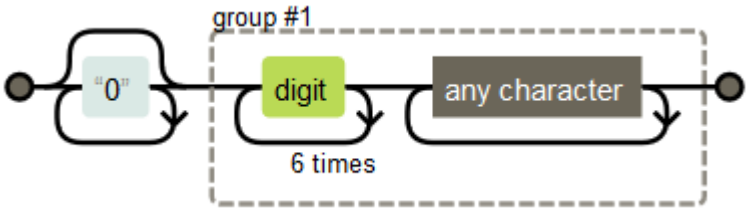
323 -this is file1.pdf	to	0000323 -this is file1.pdf
489834 - this is file two.pdf	to	0489834 - this is file two.pdf
9273 - this is file 3.pdf	to	0009273 - this is file3.pdf
2837261 -this is another file.pdf	to	2837261 -this is another file.pdf

Second run – RegEx –

Match: 0*(\d{7}.+)
Replace: \1

Name	New Name
000000323 -this is file1.pdf	0000323 -this is file1.pdf
000000489834 - this is file two.pdf	0489834 - this is file two.pdf
0000009273 - this is file 3.pdf	0009273 - this is file 3.pdf
0000002837261 -this is another file.p...	2837261 -this is another file.pdf





Sample:

```
.....................................
Match 1: 0000323 -this is file1
Group 1: 0000323 -this is file1
```

Forum Contributor: Stefan

Add

Pad random number prefix with Zeros cont.

Using: 0000000323 -this is file1.pdf

Second run – RegEx – cont.

Match: 0*(\d{7}.+)

Where:

\1 = 0000323 -this is file1

Tim Analysis:

1. 0	Match against zero (Not Captured) = '0'	
2. *	Make it Greedy (Not Captured) This gathers up all of the Zeros = '0000000'	
3. \d{7}	Match against 7 numeric digits Not Captured value in step 2 changes to '000'.	Capture Group 1 = 0000323

Tim Notes:

1. What happens here is that the Zeros have already been matched but Not Captured. Think of them as open prey still in the game and up for grabs. Current position = the first '3' of '323. In step 3, the RegEx matches against 7 numeric digits. The RegEx Engines moves forward capturing the '323' but hits the <space>. It then backtracks to match the '0000323'.

4. .+	Match against remaining string	Capture Group 1 = 0000323 -this is file1

Add

Pad Digits at end of filename

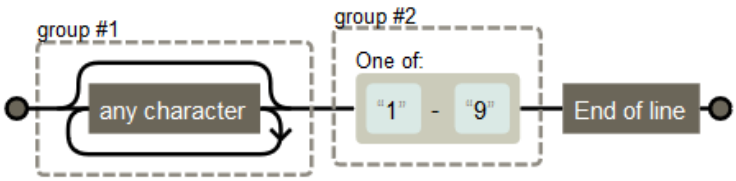
file1 .ext	to	file01.ext
file 1.ext	to	file 01.ext
file 21.ext	to	file 201.ext

Observations:

1. Does NOT match file0.ext or file.ext
2. Add zero in front of the numeric digit 1-9 in the position next to the extension dot.

Match: (.*)([1-9])$
Replace: \10\2

Name	New Name
file1 .ext	file1 .ext
file0.ext	file0.ext
file.ext	file.ext
file 21.ext	file 201.ext
file 1.ext	file 01.ext



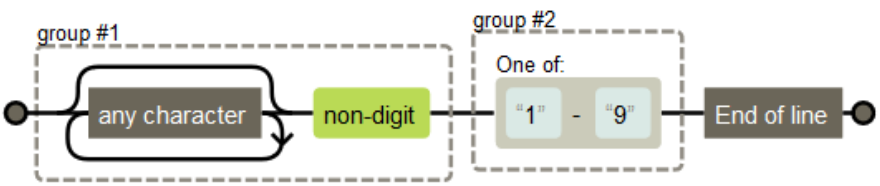
Didn't work

Contributor Analysis:

1. This will match the numbers 1-9 and won't match 0 or a non number.
2. If you don't want it to match any filenames having more than a single number at the end of the filename, use:

Match: (.*\D)([1-9])$
Replace: \10\2

Name	New Name
file1 .ext	file1 .ext
file0.ext	file0.ext
file.ext	file.ext
file 21.ext	file 21.ext
file 1.ext	file 01.ext



Didn't work

Add

Pad Digits at end of filename cont.

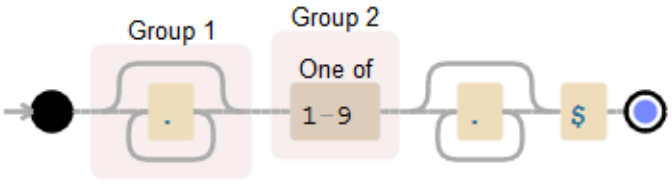
Tim Contribution –

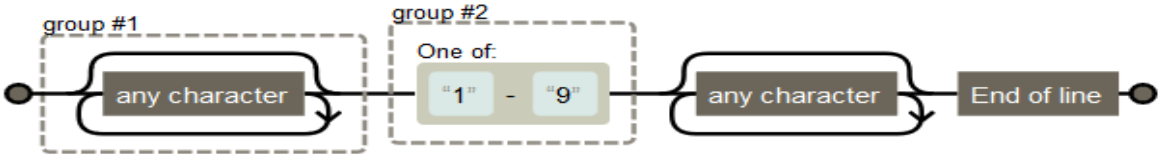
#1 First RegEx Does not account for the <space> character in File1 .ext

This will:

Match: (.*)([1-9]).*$
Replace: \10\2

Name	New Name
file1 .ext	file01.ext
file0.ext	file0.ext
file.ext	file.ext
file 21.ext	file 201.ext
file 1.ext	file 01.ext





Sample:

```
.....................
Match 1: File1 .ext
Group 1: File
Group 2: 1
```

Sample:

```
.................
Match 1: File 21
Group 1: File
Group 2:
Group 3: 2
Group 4: 1
```

Using: File1 .ext

Tim Analysis:

1. .	Match against any character	Capture Group 1 = F
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [1-9]	Match against a class consisting of a numeric digit 1-9	Capture Group 2 = 1
		Changes Capture Group 1 = File
4. .	Match against any character (Not Captured) = <space>	
5. *	Make it Greedy (captures any other text) (Not Captured)	
6. $	Must be at EOL (End of Line)	

Add

Pad Digits at end of filename cont.

Tim Contribution – cont.

Second Contributor RegEx has same problem.

#2

- If you don't want it to match any filenames having more than a single number at the end of the filename -

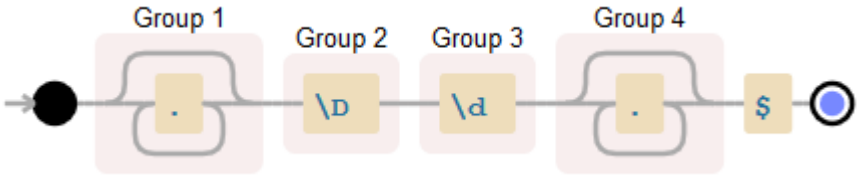
This one really stumped me. The only way I found to accomplish this given the samples, is in two runs:

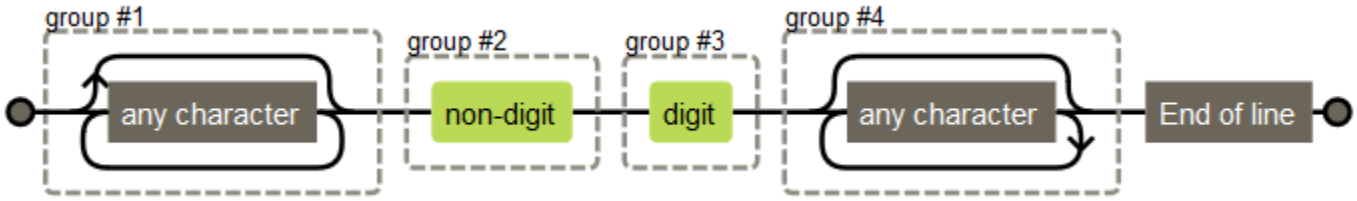
First run:

1. RegEx

Match: (.*?)(\D)(\d)(.*)$
Replace: \1\20\3\4

Name	New Name
file1.ext	file01.ext
file0.ext	file00.ext
file.ext	file.ext
file 21.ext	file 021.ext
file 1.ext	file 1.ext





Sample:

```
·················Start·Length····
Match 1: File1       0      5
Group 1: Fil         0      3
Group 2: e           3      1
Group 3: 1           4      1
Group 4:             5      0
```

Sample:

```
···················
Match 1: File 21
Group 1: File
Group 2:
Group 3: 2
Group 4: 1
```

Add

Pad Digits at end of filename cont.

Tim Contribution – cont.

#2 cont.

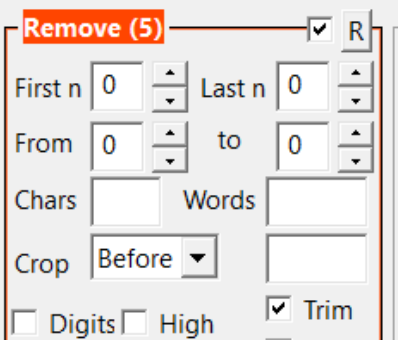
First run: cont.

2. 'Section #3: Replace' –

Replace any filenames containing double Zeros with a single zero. This is for those filenames that originally just contained a zero. These filenames are not meant to be included in the renaming process. By initiating this within the same run as the RegEx, it disqualifies those filenames from being included.

Name	New Name
file1 .ext	file01 .ext
file0.ext	file0.ext
file.ext	file.ext
file 21.ext	file 021.ext
file 1.ext	file 01.ext

3. 'Section #5: Remove' –



Gets rid of any extra <spaces> at end of filename.

Name	New Name
file1 .ext	file01 .ext

After first run:

Name	New Name
file1 .ext	file01.ext
file0.ext	file0.ext
file.ext	file.ext
file 21.ext	file 021.ext
file 1.ext	file 01.ext

'file 21.ext' was unfortunately included in the renaming. Any file that held a numeric value greater than zero would have been included as well. This is why a second run is required.

Add

Pad Digits at end of filename cont.

Tim Contribution – cont.

#2 cont.

First run: cont.

Using: File 21.ext

Match: (.*?)(\D)(\d)(.*)$

Where:

\1 =File \2 = <space> \3 = 2 \4 = 1

Tim Analysis:

1. .	Match against any character	Capture Group 1 = F
2. *	Make It Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = null
4. \D	Match against Non-Numeric Digit	Capture Group 2 = F Capture Group 1 = Unchanged
5. \d	Match against numeric digit	Capture Group 3 = 2 Changes Capture Group 2 = <space> Changes Capture Group 1 = File
6. .*	Match against remainder of string	Capture Group 4 = 1 All Others = Unchanged
7. $	Must be EOL (End of Line)	

Add

Pad Digits at end of filename cont.

Tim Contribution – cont.

#2 cont.

Second run:

This restores files that had two or more numeric digits that were inadvertently changed in the first run.

1. RegEx

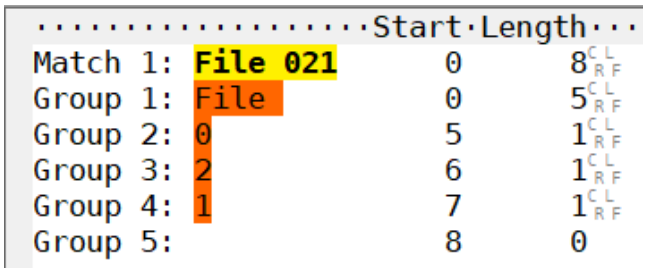
Match: ^(.+?)(0)(\d)(\d)(.*)
Replace: \1\3\4\5

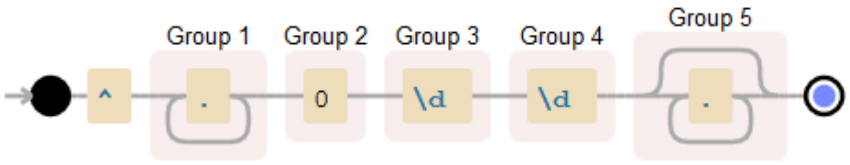
After second run:

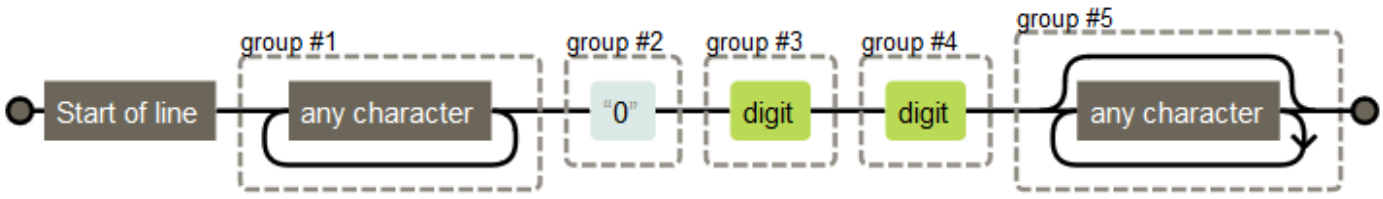
Name	New Name
file123.ext	file123.ext
file01.ext	file01.ext
file 1243.ext	file 1243.ext
file 021.ext	file 21.ext
file 01.ext	file 01.ext

I included filenames that are not part of the original samples just to make certain it was working as I wanted.

Sample:







Tim Notes:

1. In the second run I tested with all combinations of multiple numeric digits that were not part of the original samples.
2. Second RegEx from original Contributor was meant not to include any files in the renaming except for single numeric digits. This is why I had to restore the 'file 021.ext' back to 'file 21.ext'.

Add

Pad Digits at end of filename cont.

Tim Contribution – cont.

#2 cont.

Second run: cont.

Using: File 021.ext

Match: ^(.+?)(0)(\d)(\d)(.*)

Where:

\1 =File <space> \2 = 0 \3 = 2 \4 = 1 \5 = null

Tim Analysis:

1. ^	Start at beginning	
2. .	Match against any character	Capture Group 1 = F
3. +	Make It Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = F
5. 0	Match against a zero	Capture Group 2 = 0
		Changes Capture Group 1 = File<space>
6. \d	Match against numeric digit	Capture Group 3 = 2
7. \d	Match against numeric digit	Capture Group 4 = 1
8. .*	Match against remaining string If any text follows after the number	Capture Group 5 = EOL (null)

Add

Add Text before Numeric Digits in string (Parses text portion from numeric)

videofile1.avi	to	videofile.CD1.avi
videofile2.avi	to	videofile.CD2.avi
videofile3.avi	to	videofile.CD3.avi
videofile15.avi	to	videofile.CD15.avi
videofile0016.avi	to	videofile.CD0016.avi

Match: (.+)(\d+)$
Replace: \1.CD\2

Name	New Name
videofile3.avi	videofile.CD3.avi
videofile2.avi	videofile.CD2.avi
videofile15.avi	videofile1.CD5.avi
videofile1.avi	videofile.CD1.avi
videofile0016.avi	videofile001.CD6.avi

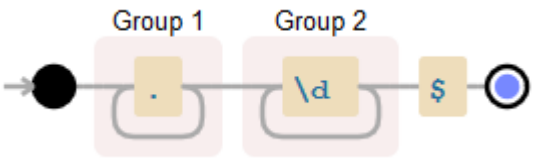
Contributor Analysis:

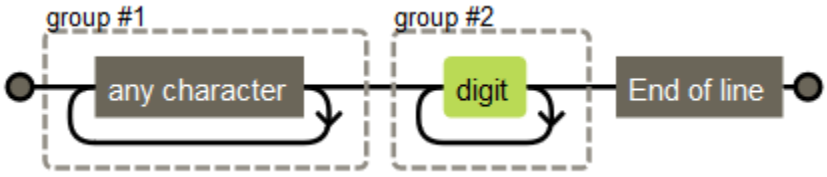
1. .+ find one or more of any character, till
2. \d+ find one or more numeric digits
3. ( ) group the matched content for back reference
4. $ followed by the end of the string/filename (that means: take only numeric digits at the end into account)

Where:

- \1 = content of what is matched in group 1
- . a dot
- CD = 'CD'
- \2 the found numeric digits

Didn't work





Tim Notes:

1. In a two numeric digit filename, it was only able to capture the last numeric digit.

Forum Contributor: Stefan

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Tim Contribution –

No indication in forum that it didn't work, but the results above do not reflect the original poster's request. Any filename containing two or more numeric digits did not have the format of, '.CD#'. Instead it was, '.#CD#'.

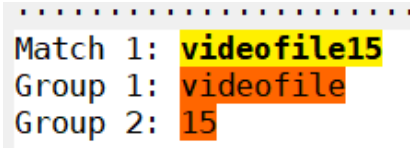
Where:

\1 = videofile for single numeric digit filenames \1 = videofile1 for two numeric digit filenames

\1 = videofile001 for four numeric digit filenames

In other words Capture Group 1 is capturing up until the final numeric digit of string.

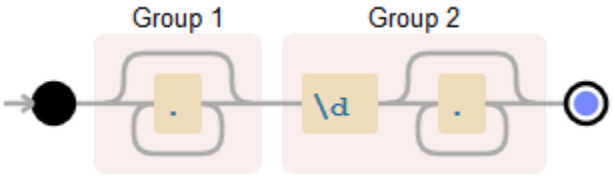
\2 = the last numeric digit (only) in the string at the end of the filename Sample:



This will fix it:

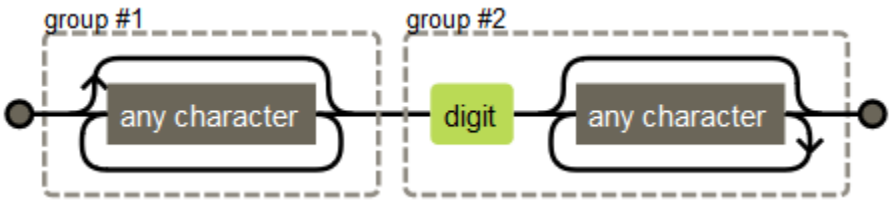
Match: (.*?)(\d.*)
Replace: \1.CD\2

Name	New Name
videofile3.avi	videofile.CD3.avi
videofile2.avi	videofile.CD2.avi
videofile15.avi	videofile.CD15.avi
videofile1.avi	videofile.CD1.avi
videofile0016.avi	videofile.CD0016.avi



Where:

\1 = videofile (any text)
\2 = numeric portion



Tim Disclaimer:

1. Great Minds Think Alike. It should be noted that I developed this RegEx before continuing reading the remainder of the post. This is how this book is written – Testing, analyzing and writing each pertinent section of each posting individually, especially when there are variations included, as there are here. In doing so, Stefan's notes #4 in the 'variation' section on the following pages refer to a (tested and verified working) RegEx almost identical to this. I assure you that it was not plagiarism on my part.

2. Since my correction is based on the error in Stefan's first RegEx and this error was not acknowledged or a correction offered anywhere in the posting regarding this first RegEx, I stand by my contribution.

3. In RegEx there are many ways to skin a cat.

4. Call your lawyers off (or your Solicitors as the Brits say).

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Tim Contribution – cont.

Using: videofile15.avi

Match: (.*?)(\d.*)

Where:

\1 = videofile \2 = 15

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \d	Match against Numeric Digit	Capture Group 2 = 1
		Changes Capture Group 1 = videofile
5. .*	Match against remaining string	Capture Group 2 = 15

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Post Continues -

Variation:

videofile1.avi	to	videofile.CD1.avi
videofile12.avi	to	videofile.CD12.avi
videofile1.en.srt	to	videofile.CD1.en.srt
videofile99.eng.srt	to	videofile.CD99.eng.srt
videofile1.fr.srt	to	videofile.CD1.fr.srt
videofile12.fr.srt	to	videofile.CD12.fr.srt
videofile12.fra.srt	to	videofile.CD12.fra.srt

Observations:

1. Match one or more of any character = .+
2. Group this = (.+)
3. .. but don't be greedy = (.*?)
4. Search for one numeric digit = \d
5. .. followed by one or no numeric digit = \d*
6. And group this too = (\d\d*)

7. For some file names, position is at end of Line (EOL) = $

   a. ..but for others there are additional characters like '.en' or '.eng', resp. '.fr' or '.fra'

   b. So I drop this, "match the end using the anchor $', behavior, to be able to cover all possibilities using one RegEx match:

      Match-or-not a dot and two or three characters

      1. For simplification of the RegEx, I search for one-or-more characters

8. Use a Capture Group = ([\.\w+]*)

Tim Notes:

1. Although there are dot characters as part of the string samples, this does not require, 'Rename File Extensions As Being Part of File Name', enabled.

Add

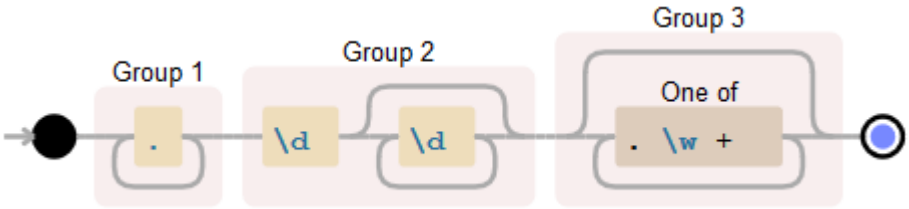
Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

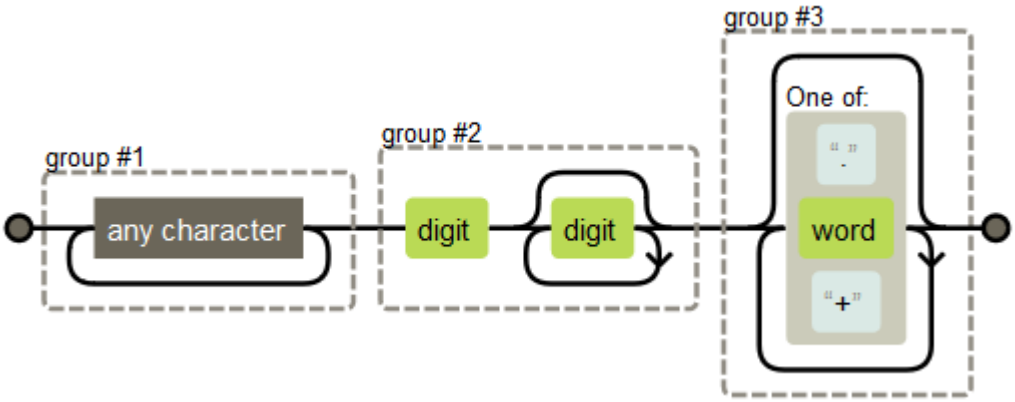
Post Continues – cont.

Variation: cont.

Match: (.+?)(\d\d*)([\.\w+]*)
Replace: \1.CD\2\3

Name	New Name
videofile99.eng.srt	videofile.CD99.eng.srt
videofile3.avi	videofile.CD3.avi
videofile2.avi	videofile.CD2.avi
videofile15.avi	videofile.CD15.avi
videofile12.fra.srt	videofile.CD12.fra.srt
videofile12.fr.srt	videofile.CD12.fr.srt
videofile1.fr.srt	videofile.CD1.fr.srt
videofile1.en.srt	videofile.CD1.en.srt
videofile1.avi	videofile.CD1.avi
videofile0016.avi	videofile.CD0016.avi





Sample:

```
.....................Start·Length···
Match 1: videofile15       0      11
Group 1: videofile         0       9
Group 2: 15                9       2
Group 3:                  11       0
```

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Using: videofile15.avi

Post Continues – cont.

Match: (.+?)(\d\d*)([\.\w+]*)

Where:

\1 = videofile \2 = 15 \3 = null

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = v
4. \d\d	Match against 2 Numeric Digits	Capture Group 2 = 15 Changes Capture Group 1 = videofile
5. *	Make it Greedy	Capture Group 2 = Unchanged
6. [\. \w+]	Match against a class consisting of a literal <dot> a Word character (one or more with '+')	
7.*	Repeat the class match Zero or more times	Capture Group 3 = null Capture Group 2 = Unchanged

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Using: videofile99.eng.srt

```
.........................
Match 1: videofile99.eng
Group 1: videofile
Group 2: 99
Group 3: .eng
```

Post Continues – cont.

Match: (.+?)(\d\d*)([\.\w+]*)

Where:

\1 = videofile \2 = 99 \3 = .eng

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = v
4. \d\d	Match against 2 Numeric Digits	Capture Group 2 = 99 Changes Capture Group 1 = videofile
5. *	Make it Greedy	Capture Group 2 = Unchanged
6. [\. \w+]	Match against a class consisting of a literal <dot>. a Word character (one or more with '+')	Capture Group 3 = .
7.*	Repeat the class match Zero or more times	Capture Group 3 = .eng Capture Group 2 = Unchanged

Tim Notes:

1. The first time this matches the <dot> after '.99'. Since there can only be one match, the /w+ is not evaluated and would not match anyway against the <dot>.
2. * Repeat class. When the class is repeat zero or more times, it will seek out any additional <dots> (That are not the extension because it would hit the EOL before that) and/or word characters in the string.
   a. The second time it repeats it matches the \w+ to 'eng' so this changes Capture Group 3 = .eng
   b. The third time it repeats it doesn't match. It reaches EOL

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Using: videofile99eng.srt.test.last

```
.................................
Match 1: videofile99eng.srt.test
Group 1: videofile
Group 2: 99
Group 3: eng.srt.test
```

Post Continues – cont.

Match: (.+?)(\d\d*)([\.\w+]*)

Where:

\1 = videofile \2 = 99 \3 = .eng.srt.test

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = v
4. \d\d	Match against 2 Numeric Digits	Capture Group 2 = 99 Changes Capture Group 1 = videofile
5. *	Make it Greedy	Capture Group 2 = Unchanged
6. [\. \w+]	Match against a class consisting of a literal <dot>. a Word character (one or more with '+')	Capture Group 3 = .
7.*	Repeat the class match Zero or more times	Capture Group 3 = .eng.srt.test Capture Group 2 = Unchanged

Tim Notes:

1. The first time this matches the <dot> after '.99'. Since there can only be one match, the /w+ is not evaluated and would not match anyway against the <dot>.
2. * Repeat class. When the class is repeat zero or more times, it will seek out any additional <dots> (That are not the extension because it would hit the EOL before that) and/or word characters in the string.
   a. The second time it repeats it matches the \w+ to 'eng' so this changes Capture Group 3 = .eng
   b. The third time it repeats it matches the <dot> after 'eng'. Capture Group 3 = .eng.
   c. The fourth time it repeats it matches the \w+ to 'srt'. Capture Group 3 = .eng.srt
   d. The fifth time it repeats it matches the <dot> after the 'srt'. Capture Group 3 = .eng.srt.
   e. The sixth time it repeats it matches the \w+ to 'test'. Capture Group 4 = .eng.srt.test
   f. The seventh time it repeats it doesn't match. It reaches EOL

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Variation cont.

Contributor Analysis:

1. (.+?) match one-or-more of any sign, but don't be greedy and match too much,
   .. to 'not' match the first numeric digit too (maybe (\w+) would be clearer to understand)
2. (\d\d*) match one numeric digit followed by one-or-no numeric digit
3. ([\.\w+]*) match one dot or one-or-more any characters

Thinking some more this third match could be simplified to (.*)

(.*) = match non-or-more of any sign (if there's an, i.e., '.en' or similar extraneous characters)

Match: (.+?)(\d\d*)(.*)
Replace: \1.CD\2\3

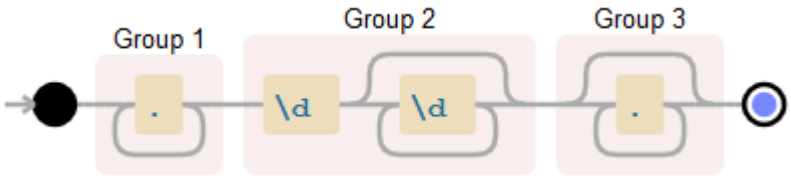
Name	New Name
videofile99.eng.srt	videofile.CD99.eng.srt
videofile3.avi	videofile.CD3.avi
videofile2.avi	videofile.CD2.avi
videofile15.avi	videofile.CD15.avi
videofile12.fra.srt	videofile.CD12.fra.srt
videofile12.fr.srt	videofile.CD12.fr.srt
videofile1.fr.srt	videofile.CD1.fr.srt
videofile1.en.srt	videofile.CD1.en.srt
videofile1.avi	videofile.CD1.avi
videofile0016.avi	videofile.CD0016.avi

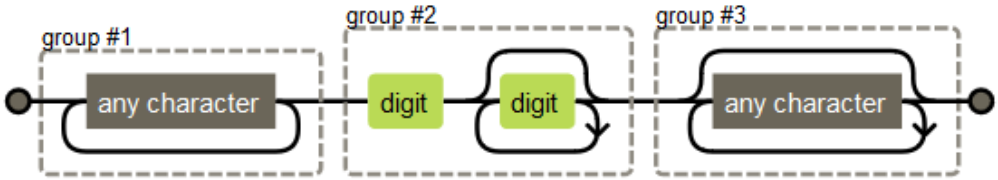
Sample:

	Start	Length
Match 1:	videofile15	0	11
Group 1:	videofile	0	9
Group 2:	15	9	2
Group 3:		11	0

Where:

\1 = will hold and give back "videofile"
.CD = will put in just this: '.CD'
\2 = will hold and give back the numeric digit(s)
\3 = will hold nothing or give back ".en" or ".fra" or like that, without the ".srt"-part





Contributor Notes:

1. But each simplification is a risk to not match what you want.
2. So just try it (that's the key to mastering RegEx: trial-and-error, reading the rules and test, test, test,...).

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Variation cont.

Using: videofile15.avi

Match: (.+?)(\d\d*)(.*)

Where:

\1 = videofile \2 = 15 \3 = null

Tim Analysis:

1..	Match against any character	Capture Group 1 = v
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = v
4. \d\d	Match against 2 Numeric Digits	Capture Group 2 = 15
		Changes Capture Group 1 = videofile
5. *	Make it Greedy	Capture Group 2 = Unchanged
6. .*	Match against remaining string	Capture Group 3 = null

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Variation cont.

videofile1.avi	to	videofile.CD1.avi
videofile12.avi	to	videofile.CD12.avi
videofile1.en.srt	to	videofile.CD1.en.srt
videofile99.eng.srt	to	videofile.CD99.eng.srt
videofile1.fr.srt	to	videofile.CD1.fr.srt
videofile12.fr.srt	to	videofile.CD12.fr.srt
videofile12.fra.srt	to	videofile.CD12.fra.srt

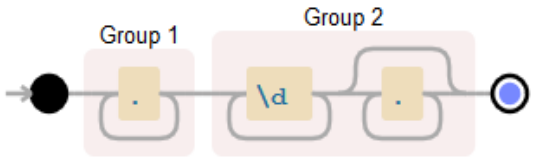
Or this will work here too:

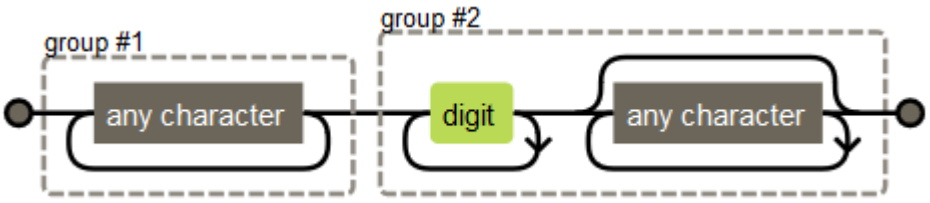
Match: (.+?)(\d+.*)
Replace: \1.DD\2

Name	New Name
videofile99.eng.srt	videofile.CD99.eng.srt
videofile3.avi	videofile.CD3.avi
videofile2.avi	videofile.CD2.avi
videofile15.avi	videofile.CD15.avi
videofile12.fra.srt	videofile.CD12.fra.srt
videofile12.fr.srt	videofile.CD12.fr.srt
videofile1.fr.srt	videofile.CD1.fr.srt
videofile1.en.srt	videofile.CD1.en.srt
videofile1.avi	videofile.CD1.avi
videofile0016.avi	videofile.CD0016.avi

Sample:

```
Match 1: videofile15
Group 1: videofile
Group 2: 15
```





Contributor Analysis:

1. (.+?) match one-or-more any signs non-greedy
2. (\d+.*) match one-or-more numeric digit(s) followed by no-or-more of any signs (dot or other characters)

Add

Add Text before Numeric Digits in string (Parses text portion from numeric) cont.

Variation cont.

Tim Notes:

1. Also my original RegEx I used to correct the Contributor's first RegEx will work here as well:

Match: (.*?)(\d.*)
Replace: \1.CD\2

Name	New Name
videofile99.eng.srt	videofile.CD99.eng.srt
videofile3.avi	videofile.CD3.avi
videofile2.avi	videofile.CD2.avi
videofile15.avi	videofile.CD15.avi
videofile12.fra.srt	videofile.CD12.fra.srt
videofile12.fr.srt	videofile.CD12.fr.srt
videofile1.fr.srt	videofile.CD1.fr.srt
videofile1.en.srt	videofile.CD1.en.srt
videofile1.avi	videofile.CD1.avi
videofile0016.avi	videofile.CD0016.avi

Forum Contributor: Stefan

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename

Tim Contribution –

Perhaps this can be done in JavaScript – but not in RegEx as far as I can tell. But here it is anyway. I thought I would give it a try since it was similar to another challenge I did:

SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv
to
SunFly116_12-The Beatles- I Want To Hold Your Hand-IWTHYH.mkv

Observations:

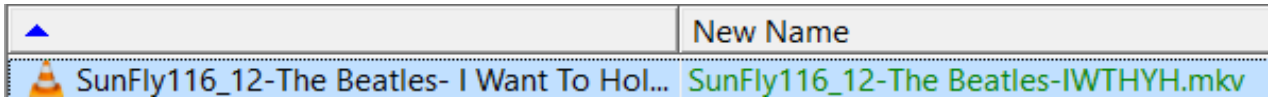
1. Impossible – Yes I think so. Any challengers out there who can do better? I will include it here.
2. The Poster wants to take each initial from each word that makes up the song title and append this to the end of the same filename.

Why impossible?

1. Unknown variables of how many words will make up the title. No pattern.
2. Only have 9 Capture Groups available.
3. After taking each initial and reached EOL, I cannot in my experience think of how to backtrack enough to capture the last part of the string when the expression has already been satisfied.

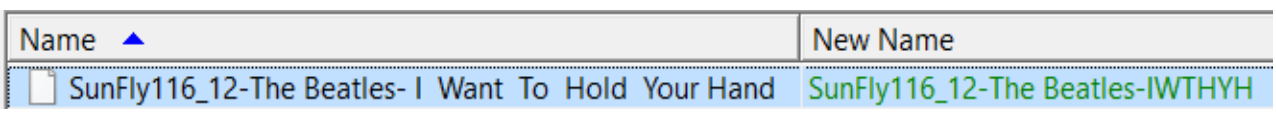
So for S&** and Giggles I base my RegEx on the only (Thank God) sample given.

It is not correct because, although my results are good, they remain incomplete. I am only able to append to the end of the first part of the filename.



It can also handle excess <spaces>:

SunFly116_12-The Beatles- I Want To Hold Your Hand



Add

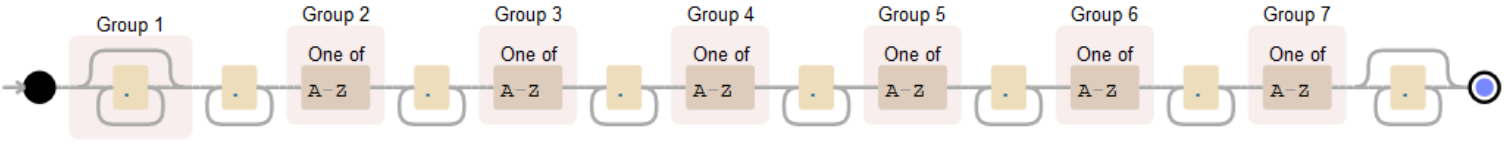
Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Tim Contribution – cont.

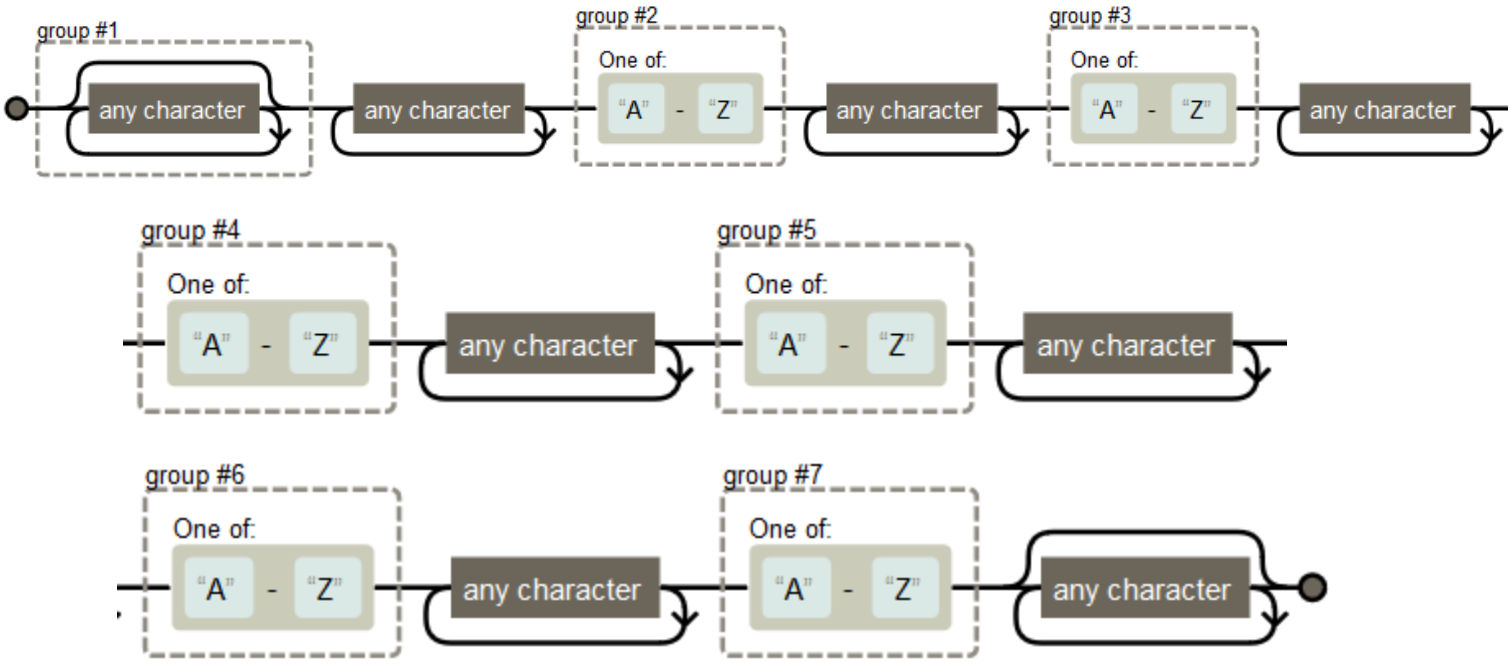
TaDah!

Match: (.*).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).*
Replace: \1\2\3\4\5\6\7

Name	New Name
SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv	SunFly116_12-The Beatles-IWTHYH.mkv



Expanded:



Sample:

```
Match 1: SunFly116_12-The Beatles- I Want To Hold Your Hand
Group 1: SunFly116_12-The Beatles-
Group 2: I
Group 3: W
Group 4: T
Group 5: H
Group 6: Y
Group 7: H
```

Tim Final Notes:

This is **so** limited it can only be matched with a filename that has **exactly** 6 words in the title.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Tim Contribution – cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

Match: (.*).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).*

Where:

\1 = SunFly116_12-The Beatles	\2 = I	\3 = W	\4 = T	\5 = H
\6 = Y	\7 = H	\8 = and		

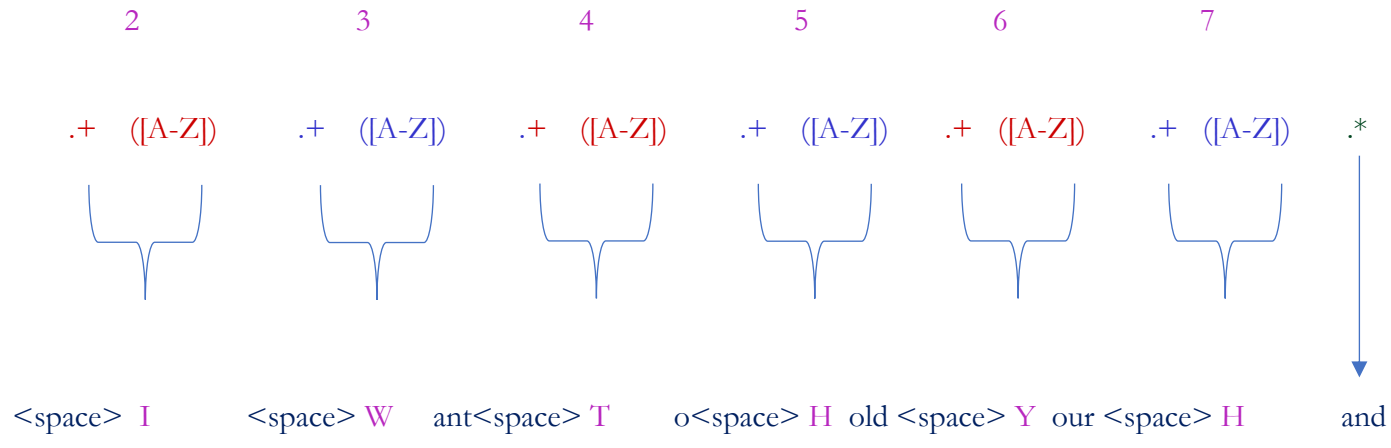
Breaks down:

1. Mostly just repetition – shown in alternating colors of light blue and red for clarity.
2. Green highlights the part of RegEx that is not repetition.
3. I have numbered the Capture Groups.

1

(.*)

SunFly116_12-The Beatles-



Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Tim Contribution – cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

Match: (.*).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).*

Where:

\1 = SunFly116_12-The Beatles \2 = I \3 = W \4 = T \5 = H
\6 = Y \7 = H

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>

Begin Repetition:

3. .	Match against any character (Not Captured) = 'd'	Changes Capture Group 1 = SunFly116_12-The Beatles- I Want To Hold <space> Your Han
4. +	Make it Greedy (Not Captured) = No Change	Capture Group 1 = No Change
5. [A-Z]	Match against a class consisting of an uppercase letter Changes Not Captured to <space> after 'Your'	Capture Group 2 = H Changes Capture Group 1 = SunFly116_12-The Beatles- I Want To Hold <space>Your
6. .	Match against any character (Not Captured) = 'a'	
7. +	Make it Greedy (Not Captured) = 'and' Current position = EOL	
8. [A-Z]	Backtracks to Match against a class consisting of an uppercase letter. This is the 'H' of 'Hand'	Capture Group 3 = H Changes Capture Group 2 = Y Changes Capture Group 1 = SunFly116_12-The Beatles- I Want To Hold
9. .	Match against any character (Not Captured) = 'a'	
10. +	Make it Greedy (Not Captured) = 'and' Current position = EOL	
11. [A-Z]	Backtracks to Match against a class consisting of an uppercase letter. This is the 'H' of Hold'	Capture Group 4 = H Changes Capture Group 3 = Y Changes Capture Group 2 = H Changes Capture Group 1 = SunFly116_12-The Beatles- I Want To

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Tim Contribution – cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

Match: (.*).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).*

Where:

\1 = SunFly116_12-The Beatles \2 = I \3 = W \4 = T \5 = H
\6 = Y \7 = H

Tim Analysis: cont.

Begin Repetition – cont.

12. .	Match against any character (Not Captured) = 'a'	
13. +	Make it Greedy (Not Captured) = 'and' Current position = EOL	
14. [A-Z]	Backtracks to Match against a class consisting of an uppercase letter. This is the 'H' of 'Hand' Capture Group 3 = 'H' of 'Hold'	Capture Group 5 = H Changes Capture Group 4 = Y Changes Capture Group 3 = H Changes Capture Group 2 = T Changes Capture Group 1 = SunFly116_12-The Beatles- I Want
15. .	Match against any character (Not Captured) = 'a'	
16. +	Make it Greedy (Not Captured) = 'and' Current position = EOL	
17. [A-Z]	Backtracks to Match against a class consisting of an uppercase letter. This is the 'H' of 'Hand' Capture Group 4 = 'H' of 'Hold'	Capture Group 6 = H Changes Capture Group 5 = Y Changes Capture Group 4 = H Changes Capture Group 3 = T Changes Capture Group 2 = W Changes Capture Group 1 = SunFly116_12-The Beatles- I

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Tim Contribution – cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

Match: (.*).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).+([A-Z]).*

Where:

\1 = SunFly116_12-The Beatles \2 = I \3 = W \4 = T \5 = H
\6 = Y \7 = H

Tim Analysis: cont.

Begin Repetition – cont.

18. .	Match against any character (Not Captured) = 'a'	
19. +	Make it Greedy (Not Captured) = 'and' Current position = EOL	
20. [A-Z]	Backtracks to Match against a class consisting of an uppercase letter. This is the 'H' of 'Hand' Capture Group 4 = 'H' of 'Hold'	Capture Group 7 = H Changes Capture Group 6 = Y Changes Capture Group 5 = H Changes Capture Group 4 = T Changes Capture Group 3 = W Changes Capture Group 2 = I Changes Capture Group 1 = SunFly116_12-The Beatles-

End Repetition.

21. .* Match against remaining string (Not Captured) = 'and'

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Now the challengers:

Since this was a challenge, individual Contributors will be noted rather than grouped together.

SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv
to
SunFly116_12-The Beatles- I Want To Hold Your Hand-IWTHYH.mkv

BRC by bru

The problem is there's no way for BRC/BRU to know where the SongName begins.
IF you're saying that a final – <hyphen> always precedes SongName and there are no <hyphens> in SongName), you could:

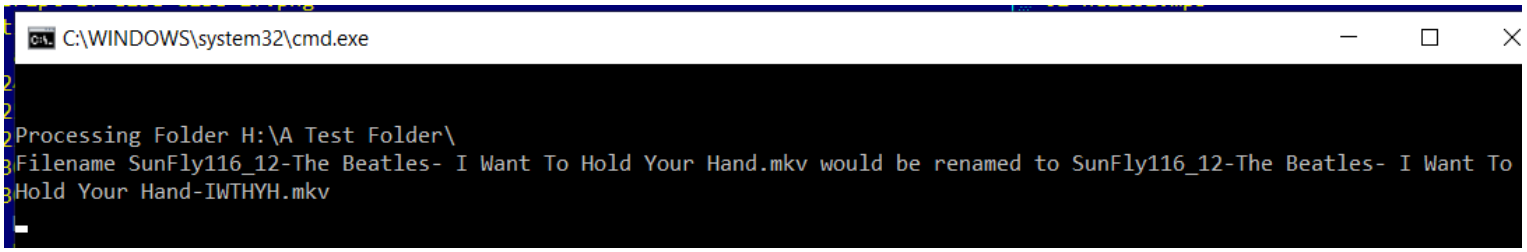
The batch file:

```
@echo off
cd/d "FolderPathToYour.mkv's"

set reg1=/Regexp:"^(.*- *.*?)([A-Z])([^-]+[^A-Z])$:\1\2|\3-\2"

set reg2=/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"

set max10=%reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2%

brc32 /Pattern:*mkv %reg1% %max10% /ReplaceCI:"|":

pause>nul
```

---

i.e., using “H:\ A Test Folder” as the directory that contains my sample -

The Results:



SunFly116_12-The Beatles- I Want To Hold Your Hand-IWTHYH.mkv

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Contributor Notes:

1. This gives you a preview of what will happen. Add the /Execute parameter to the batch file to process:

    brc32 /Pattern:*mkv %reg1% %max10% /ReplaceCI:"|": /Execute

2. This will handle up to 10 words in SongName.

3. Add additional %reg2%'s if more words are required.

4. Reg1 is written to purposely NOT match files ending with Uppercase (assumes previously renamed).
   a. Uses Negated Classes to achieve this.

Alternative:

If you want a single command for testing, go to the folder with your .mkvs, launch a command-prompt & copy this:

---

cd /d “H:\A Test Folder”

```
brc32 /Pattern:*mkv /Regexp:"^(.*- *.*?)([A-Z])([^-]+[^A-Z])$:\1\2|\3-\2"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"
/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"  /ReplaceCI:"|":
```

```
Processing Folder H:\A Test Folder\
Filename SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv would be renamed to SunFly116_12-The Beatles- I Want To
Hold Your Hand-IWTHYH.mkv

H:\A Test Folder>
```

Tim Notes:

1. Substitute your own folder name that contains the samples that you want to rename in the first line. This changes to the specified drive and directory. Otherwise BRC will not be able to find the files and fail.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Contributor Notes:

1. If you don't keep brc64 within 1 of your system's %PATH% folders, i.e., c:\Windows\System32, you'll need to specify the "FullPathTo\brc64" (versus just brc64).

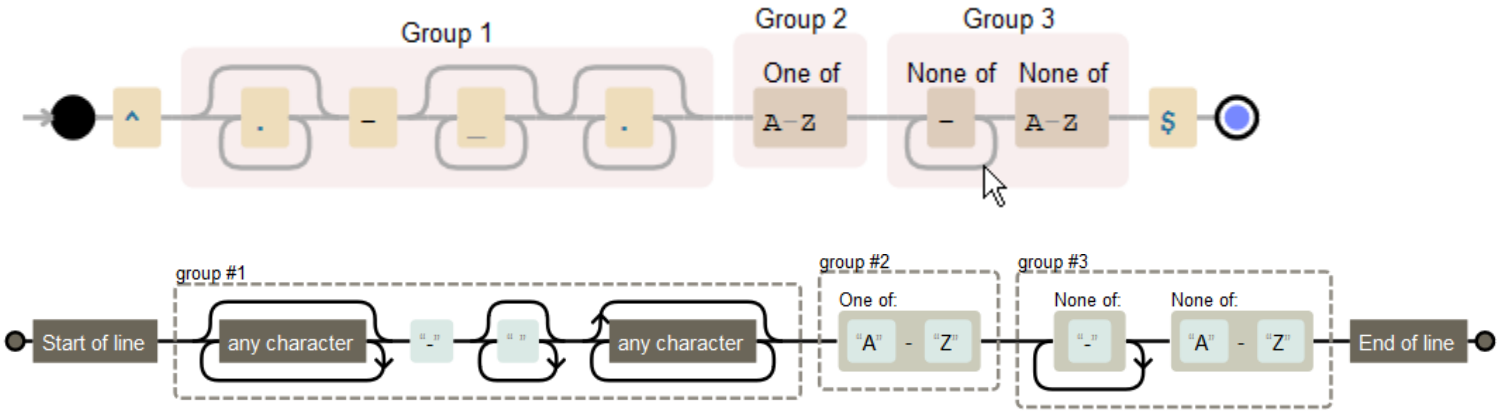
   If for example, the executable is located at H:\BRC_Unicode_64\, then the first line would be changed to:

   H:\BRC_Unicode_64\brc64 /Pattern:*.mkv

2. The /Execute has been stripped to let you preview the would-be rename(s).
   a. In BRC, if you use the /Execute at the end, it will perform the Rename process.

Reg1 – Meant to be run once

Match: ^(.*- *.*?)([A-Z])([^-]+[^A-Z])$
Replace: \1\2|\3-\2



Sample:

```
.....................................................................
Match 1: SunFly116_12-The Beatles- I Want To Hold Your Hand
Group 1: SunFly116_12-The Beatles-
Group 2: I
Group 3:  Want To Hold Your Hand
```

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

BRC by bru cont.

Reg1 – Meant to be run once cont.

Match: ^(.*- *.*?)([A-Z])([^-]+[^A-Z])$

Where:

\1 = SunFly116_12-The Beatles-<space> \2 = I \3 = <space> Want To Hold Your Hand

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = S
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. -	Match against <hyphen> Backtracks.	Capture Group 1 = SunFly116_12-The Beatles-
5. <space>	Match against a <space> character	Capture Group 1 = SunFly116_12-The Beatles-<space>
6. *	Make it Greedy	Capture Group 1 = Unchanged
7..	Match against any character	Capture Group 1 = SunFly116_12-The Beatles- I
8. *	Make it Greedy	Capture Group 1 = <entire string>
9. ?	But not too Greedy	Capture Group 1 = SunFly116_12-The Beatles-<space>
10. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 2 = I
11. [^-]	Match against a Negated Class consisting of a <hyphen>. This is the <space> after 'I'	Capture Group 3 = <space>
12. +	Make it Greedy	Capture Group 3 = <space> I Want To Hold Your Hand
13.	Match against a Negated Class consisting of an uppercase letter.	Capture Group 3 = Unchanged
14. $	Must be at EOL	

Tim Notes:

1. As already mentioned the Contributor added steps 13 and 14 to ensure that the string does not already have a single Uppercase letter at the end. If it does the RegEx will fail as it is presumed that if it does, the file has previously been renamed so it should be ignored.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

BRC by bru cont.

Reg1 – Meant to be run once cont.

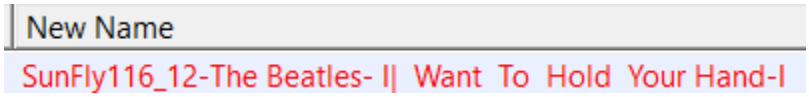
Match: ^(.*- *.*?)([A-Z])([^-]+[^A-Z])$

Where:

\1 = SunFly116_12-The Beatles-<space> \2 = I \3 = <space> Want To Hold Your Hand

Tim Analysis: cont.

Replace: \1\2|\3-\2



Reg1 sets up the format of the string and Reg2 will act upon this.

The format is

Isolating the first initial using a character as a 'placeholder' that couldn't possibly be found in a filename – the pipe character, because it is an illegal character in a Windows filename'. Then append that initial to the end of the string with a preceding <hyphen>.

This is why the BRU photo shows the New Name as Invalid (in red). It won't rename. But that is the beauty of BRC, it doesn't need to rename until the very end with the /Execute parameter. So strings can be processed in memory without having to worry about final evaluation, using various RegEx to 'prepare' them for the renaming. Something BRU is definitely not able to do because BRU can only process a string using one RegEx at a time.

The reason the Contributor uses the pipe character is to ensure that whichever filename gets processed, the character used as the 'placeholder' cannot be evaluated by mistake.. For example, if the exclamation point was used instead, and a file actually contained an exclamation point, it would be misinterpreted by the RegEx amounting to a probable error in the evaluation.

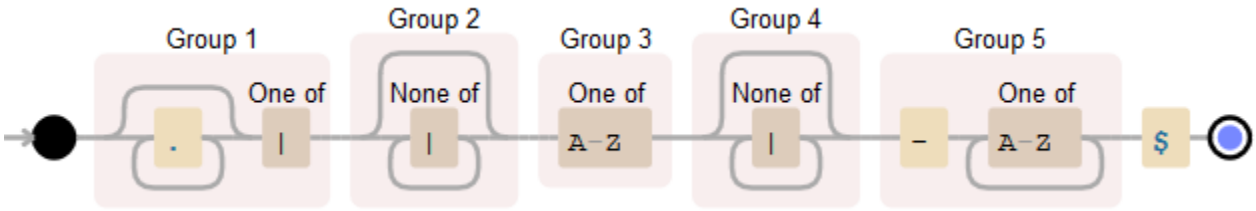
Add

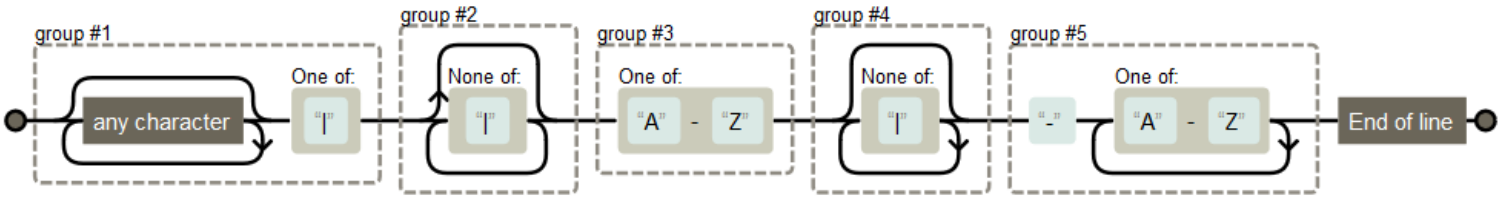
Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$
Replace: \1\2\3|\4\5\3





Here is a single sample run:

```
                                                          Start Length
Match 1: SunFly116_12-The Beatles- I! Want To Hold Your Hand-I     0     53
Group 1: SunFly116_12-The Beatles- I!                              0     28
Group 2:                                                          28      1
Group 3: W                                                        29      1
Group 4: ant To Hold Your Hand                                    30     21
Group 5: -I                                                       51      2
```

Tim Notes:

1. In order to get a valid response from Regex Buddy, I had to temporarily substitute the exclamation point for the pipe character:

   SunFly116_12-The Beatles- I! Want To Hold Your Hand-I

   Match: (.*[!])([^!]*?)([A-Z])([^!]*)(-[A-Z]+)$

2. Capture Group3 will be 'reserved' for the captured initial, while Capture Group 5 will be 'reserved' for the accumulated initials.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

First Run:

Using: SunFly116_12-The Beatles-I| Want To Hold Your Hand-I

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Where:

\1 = SunFly116_12-The Beatles- I| \2 = <space> \3 = W \4 = ant To Hold Your Hand \5 = -I

Breaks down:

Capture Group 1		Capture Group 2		Capture Group 3	Capture Group 4		Capture Group 5		
Class		Negated Class		Class	Negated Class		Class		anchor
(.*	[|])	([^|]	*?)	([A-Z])	([^|]	*)	(-[A-Z]+	)	$

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [|]	Match against a class consisting of a pipe character. Backtracks.	Capture Group 1 = SunFly116_12-The Beatles-I|
4. [^|]	Match against a Negated Class consisting of a pipe character. This is the <space> after the pipe character	Capture Group 2 = <space>
5. *	Make it Greedy	Capture Group 2 = <space> I Want To Hold Your Hand-I
6. ?	But not too Greedy Moves back to position just after pipe character.	Capture Group 2= null
7. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter	Capture Group 3 = W
8. [^|]	Match against a Negated Class consisting of a pipe character.	Capture Group 4 = a Capture Group 3 = Unchanged Changes Capture Group 2 = <space>
9. *	Make it Greedy Current position = EOL	Capture Group 4 = ant To Hold Your Hand-I

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

First Run: cont.

Using: SunFly116_12-The Beatles-I| Want To Hold Your Hand-I

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Tim Analysis: cont.

10. -	Match against <hyphen> Backtracks to match against <hyphen> after 'Hand'	Capture Group 5 = - Changes Capture Group 4 = ant To Hold Your Hand
11. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 5 = -I
12. +	Make it Greedy	Capture Group 5 = Unchanged
13. $	Must be at EOL	

	Start	Length
Match 1: SunFly116_12-The Beatles- I! Want To Hold Your Hand-I	0	53
Group 1: SunFly116_12-The Beatles- I!	0	28
Group 2:	28	1
Group 3: W	29	1
Group 4: ant To Hold Your Hand	30	21
Group 5: -I	51	2

SunFly116_12-The Beatles- I! Want To Hold Your Hand-I

Match: (.*[!])([^!]*?)([A-Z])([^!]*)(-[A-Z]+)$
Replace: \1\2\3|\4\5\3

Where:

\1 = SunFly116_12-The Beatles- I| \2 = <space> \3 = W \4 = ant To Hold Your Hand \5 = -I
1 2 3 |4 5 3
New Name = SunFly116_12-The Beatles- I| W|ant To Hold Your Hand-IW

Tim Notes:

1. I will repeat myself - that the exclamation point has been substituted for the pipe character in Regex Buddy in order to get a response. In BRU, the pipe character is used and produces an invalid result as expected:

SunFly116_12-The Beatles- I! W|ant To Hold Your Hand-IW.mkv

2. On each subsequent run, a pipe character is added after an initial, and that initial is added to the end of the string.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Second Run:

Using: SunFly116_12-The Beatles-I| W|ant To Hold Your Hand-IW

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant To Hold Your Hand-IW
Group 1: SunFly116_12-The Beatles-I! W!
Group 2: ant
Group 3: T
Group 4: o Hold Your Hand
Group 5: -IW
```

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Where:

\1 = SunFly116_12-The Beatles- I| W| \2 = ant <space> \3 = T \4 = o Hold Your Hand \5 = -IW

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [|]	Match against a class consisting of a pipe character. Backtracks.	Capture Group 1 = SunFly116_12-The Beatles-I| W|
4. [^|]	Match against a Negated Class consisting of a pipe character. This is the 'a' of 'Want'	Capture Group 2 = a
5. *	Make it Greedy	Capture Group 2 = ant To Hold Your Hand-IW
6. ?	But not too Greedy Moves back to position just after 'W|'.	Capture Group 2= null
7. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter	Capture Group 3 = T
8. [^|]	Match against a Negated Class consisting of a pipe character.	Capture Group 4 = o Capture Group 3 = Unchanged Changes Capture Group 2 = ant<space>
9. *	Make it Greedy Current position = EOL	Capture Group 4 = o Hold Your Hand-IW

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Second Run: cont.

Using: SunFly116_12-The Beatles-I| W|ant To Hold Your Hand-IW

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Tim Analysis: cont.

10. -	Match against <hyphen> Backtracks to match against <hyphen> after 'Hand'	Capture Group 5 = - Changes Capture Group 4 = o Hold Your Hand
11. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 5 = -I
12. +	Make it Greedy	Capture Group 5 = -IW
13. $	Must be at EOL	

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant To Hold Your Hand-IW
Group 1: SunFly116_12-The Beatles-I! W!
Group 2: ant
Group 3: T
Group 4: o Hold Your Hand
Group 5: -IW
```

SunFly116_12-The Beatles-I! W!ant To Hold Your Hand-IW

Match: (.*[!])([^!]*?)([A-Z])([^!]*)(-[A-Z]+)$
Replace: \1\2\3|\4\5\3

Where:

\1 = SunFly116_12-The Beatles- I|W| \2 = ant<space> \3 = T \4 = o Hold Your Hand \5 = -IW

1 2 3|4 5 3
New Name = SunFly116_12-The Beatles- I|W| ant T|o Hold Your Hand-IWT

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Third Run:

Using: SunFly116_12-The Beatles-I| W|ant T|o Hold Your Hand-IWT

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant T!o Hold Your Hand-IWT
Group 1: SunFly116_12-The Beatles-I! W!ant T!
Group 2: o
Group 3: H
Group 4: old Your Hand
Group 5: -IWT
```

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Where:

\1 = SunFly116_12-The Beatles- I| W|ant T| \2 = o<space> \3 = H \4 = old Your Hand \5 = -IWT

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [|]	Match against a class consisting of a pipe character. Backtracks.	Capture Group 1 = SunFly116_12-The Beatles-I| W|ant T|
4. [^|]	Match against a Negated Class consisting of a pipe character. This is the 'o' of 'To'	Capture Group 2 = o
5. *	Make it Greedy	Capture Group 2 = o Hold Your Hand-IWT
6. ?	But not too Greedy Moves back to position just after 'T|'.	Capture Group 2= null
7. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter	Capture Group 3 = H
8. [^|]	Match against a Negated Class consisting of a pipe character. Capture Group 4 is the 'o' of 'Hold' Capture Group 2 is the 'o' of 'To'	Capture Group 4 = o Capture Group 3 = Unchanged Changes Capture Group 2 = o<space>
9. *	Make it Greedy Current position = EOL	Capture Group 4 = old Your Hand-IWT

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Third Run: cont.

Using: SunFly116_12-The Beatles-I| W|ant T|o Hold Your Hand-IWT

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Tim Analysis: cont.

10. -	Match against <hyphen> Backtracks to match against <hyphen> after 'Hand'	Capture Group 5 = - Changes Capture Group 4 = old Your Hand
11. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 5 = -I
12. +	Make it Greedy	Capture Group 5 = -IWT
13. $	Must be at EOL	

```
Match 1: SunFly116_12-The Beatles-I! W!ant T!o Hold Your Hand-IWT
Group 1: SunFly116_12-The Beatles-I! W!ant T!
Group 2: o
Group 3: H
Group 4: old Your Hand
Group 5: -IWT
```

SunFly116_12-The Beatles-I! W!ant T!o Hold Your Hand-IWT

Match: (.*[!])([^!]*?)([A-Z])([^!]*)(-[A-Z]+)$
Replace: \1\2\3|\4\5\3

Where:

\1 = SunFly116_12-The Beatles- I| W|ant T| \2 = o<space> \3 = H \4 = old Your Hand \5 = -IWT
1 2 3 |4 5 3
New Name = SunFly116_12-The Beatles- I|W|ant T|o H|old Your Hand-IWTH

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Fourth Run:

Using: SunFly116_12-The Beatles-I| W|ant T|o H|old Your Hand-IWTH

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant T!o H!old Your Hand-IWTH
Group 1: SunFly116_12-The Beatles-I! W!ant T!o H!
Group 2: old
Group 3: Y
Group 4: our Hand
Group 5: -IWTH
```

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Where:

\1 = SunFly116_12-The Beatles- I| W|ant T|o H| \2 = old <space> \3 = Y \4 = our Hand \5 = -IWTH

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [|]	Match against a class consisting of a pipe character. Backtracks.	Capture Group 1 = SunFly116_12-The Beatles- I| W|ant T|o H|
4. [^|]	Match against a Negated Class consisting of a pipe character. This is the 'o' of 'Hold'	Capture Group 2 = o
5. *	Make it Greedy	Capture Group 2 = old Your Hand-IWTH
6. ?	But not too Greedy Moves back to position just after 'H|'.	Capture Group 2= null
7. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter	Capture Group 3 = Y
8. [^|]	Match against a Negated Class consisting of a pipe character. Capture Group 4 is the 'o' of 'Your'	Capture Group 4 = o Capture Group 3 = Unchanged Changes Capture Group 2 = old<space>
9. *	Make it Greedy Current position = EOL	Capture Group 4 = our Hand-IWTH

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Fourth Run: cont.

Using: SunFly116_12-The Beatles-I| W|ant T|o H|old Your Hand-IWTH

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Tim Analysis: cont.

10. -	Match against <hyphen> Backtracks to match against <hyphen> after 'Hand'	Capture Group 5 = - Changes Capture Group 4 = our Hand
11. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 5 = -I
12. +	Make it Greedy	Capture Group 5 = -IWTH
13. $	Must be at EOL	

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant T!o H!old Your Hand-IWTH
Group 1: SunFly116_12-The Beatles-I! W!ant T!o H!
Group 2: old
Group 3: Y
Group 4: our Hand
Group 5: -IWTH
```

SunFly116_12-The Beatles-I! W!ant T!o H!old Your Hand-IWTH

Match: (.*[!])([^!]*?)([A-Z])([^!]*)(-[A-Z]+)$
Replace: \1\2\3|\4\5\3

Where:

\1 = SunFly116_12-The Beatles- I| W|ant T|o H| \2 = old <space> \3 = Y \4 = our Hand \5 = -WITH

1 2 3|4 5 3
New Name = SunFly116_12-The Beatles- I|W|ant T|o H|old Y|our Hand-IWTHY

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Fifth Run:

Using: SunFly116_12-The Beatles-I| W|ant T|o H|old Y|our Hand-IWTHY

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant T!o H!old Y!our Hand-IWTHY
Group 1: SunFly116_12-The Beatles-I! W!ant T!o H!old Y!
Group 2: our
Group 3: H
Group 4: and
Group 5: -IWTHY
```

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Where:

\1 = SunFly116_12-The Beatles- I| W|ant T|o H|old Y| \2 = our <space> \3 = H \4 = and \5 = -IWTHY

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [|]	Match against a class consisting of a pipe character. Backtracks.	Capture Group 1 = SunFly116_12-The Beatles- I| W|ant T|o H|old Y|
4. [^|]	Match against a Negated Class consisting of a pipe character. This is the 'o' of 'Your'	Capture Group 2 = o
5. *	Make it Greedy	Capture Group 2 = our Hand-IWTHY
6. ?	But not too Greedy Moves back to position just after 'Y|'.	Capture Group 2= null
7. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter This is the 'H' of 'Hand'	Capture Group 3 = H
8. [^|]	Match against a Negated Class consisting of a pipe character. Capture Group 4 is the 'a' of 'hand'	Capture Group 4 = a Capture Group 3 = Unchanged Changes Capture Group 2 = our<space>
9. *	Make it Greedy Current position = EOL	Capture Group 4 = and-IWTHY

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Fifth Run: cont.

Using: SunFly116_12-The Beatles-I| W|ant T|o H|old Y|our Hand-IWTHY

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Tim Analysis: cont.

10. -	Match against <hyphen> Backtracks to match against <hyphen> after 'Hand'	Capture Group 5 = - Changes Capture Group 4 = and
11. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 5 = -I
12. +	Make it Greedy	Capture Group 5 = -IWTHY
13. $	Must be at EOL	

```
.....................................................................
Match 1: SunFly116_12-The Beatles-I! W!ant T!o H!old Y!our Hand-IWTHY
Group 1: SunFly116_12-The Beatles-I! W!ant T!o H!old Y!
Group 2: our
Group 3: H
Group 4: and
Group 5: -IWTHY
```

SunFly116_12-The Beatles-I! W!ant T!o H!old Y!our Hand-IWTHY

Match: (.*[!])([^!]*?)([A-Z])([^!]*)(-[A-Z]+)$
Replace: \1\2\3|\4\5\3

Where:

\1 = SunFly116_12-The Beatles- I| W|ant T|o H|old Y| \2 = our <space> \3 = H \4 = and \5 = -IWTHY

1 2 3 |4 5 3

New Name = SunFly116_12-The Beatles- I|W|ant T|o H|old Y|our H|and-IWTHYH

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

Reg2 – Meant to be run multiple times cont.

Sixth Run:

Using: SunFly116_12-The Beatles-I| W|ant T|o H|old Y|our H|and-IWTHYH

Match: (.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. [|]	Match against a class consisting of a pipe character. Backtracks.	Capture Group 1 = SunFly116_12-The Beatles-I| W|ant T|o H|old Y|our H|
4. [^|]	Match against a Negated Class consisting of a pipe character. This is the 'a' of 'Hand'	Capture Group 2 = a
5. *	Make it Greedy	Capture Group 2 = and-IWTHYH
6. ?	But not too Greedy Moves back to position just after 'H|' of 'H|and'.	Capture Group 2= null
7. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter This is the 'I' after 'H|and-' Capture Group 2 - This is the 'H' of 'Hand'	Capture Group 3 = I Changes Capture Group 2= and-
8. [^|]	Match against a Negated Class consisting of a pipe character. Capture Group 4 is the 'W' after '-I"	Capture Group 4 = W Capture Group 3 = Unchanged Capture Group 2 = Unchanged
9. *	Make it Greedy Current position = EOL	Capture Group 4 = IWTHY
10. -	Match against <hyphen> = False	

Fails – already at EOL and Capture Group 2 already has the <hyphen> captured. There are no others in the string (after 415 steps of permutations)!

String Exhausted. The New Name remains as:

New Name = SunFly116_12-The Beatles- I|W|ant T|o H|old Y|our H|and-IWTHYH

Time to go into the next phase.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

BRC by bru cont.

In this next phase, the rest of the batch file is analyzed.

set reg1=/Regexp:"^(.*- *.*?)([A-Z])([^-]+[^A-Z])$:\1\2|\3-\2"

set reg2=/Regexp:"(.*[|])([^|]*?)([A-Z])([^|]*)(-[A-Z]+)$:\1\2\3|\4\5\3"

set max10=%reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2% %reg2%

brc32 /Pattern:*mkv %reg1% %max10% /ReplaceCI:"|":

Set max10 is how reg2 is run multiple times. By assigning reg2 to a variable %reg2%, %max10%, another variable, is called to run 10 times because there are 10 %reg2% variables that make up the max10 line . Now that we have the analysis of both reg1 and reg2, this means that a string can potentially have up to 10 capitalized words and be processed.

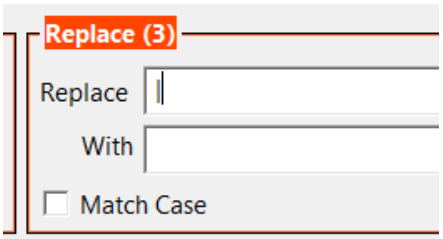
Finally you have the BRC statement. This is the actual run of the BRC program. Everything before this was preliminary.

It breaks down to:

1. brc32 – calls the brc program. There is a 32 bit program and a 64 bit program. The Contributor uses a 32 bit version.
2. /Pattern – this optional parameter establishes any filter – in this case it uses *.mkv
3. %reg1% - calls the reg1 variable. The reg1 variable is set to the reg1 RegEx. This will run the reg1 RegEx. one time.
4. %max10% - calls the max10 variable. The max10 variable is set to run reg2 to a maximum of 10 times or until the string is exhausted.
5. /Replace – this optional parameter is the same as the 'Section #3: Replace' in BRU. The CI at the end indicates that it is Case Insensitive.

   What it does is this:

   a. Remember all of those pipe characters that were used as 'placeholders' before each of the initials in the string? This removes them –



Replace: |

With: null

Match Case: Disabled.

New Name = SunFly116_12-The Beatles- I Want To Hold Your Hand-IWTHYH

Tim Notes:

1. Could also have used /RemoveChars:| -which would essentially be the same as, 'Section #5: Remove'.
2. Remember, to execute the BRC you have to use the /Execute parameter at the end of the BRC statement.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

JavaScript by Admin

JavaScript Outline:

If the title where the initials are to be taken from it is always after the ' - '

a) get part of the name after the ' - '
b) split into an array of words
c) append first letter of each word to name (capitalized if needed)

Here is the script:

```
function getSecondPart(str) {
  return str.substr(str.lastIndexOf("-")+1);
}

function getFirstPart(str) {
  return str.substr(0,str.lastIndexOf("-"));
}

var title = getSecondPart(name);
words = title.split(' ');
var text = "";

for (var i = 0; i < words.length; i++) {
  text += words[i].substr(0,1);
}

newName = getFirstPart(name) + "-" + getSecondPart(name) + "-" + text;
```

Not clear from script above, – but the line 'text += words[i]' is a lowercase letter i in brackets [i].

Name	New Name
SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv	SunFly116_12-The Beatles- I Want To Hold Your Hand-IWTHYH.mkv

Tim Notes:

1. JavaScript has once more proven itself a worthy small investment. Take note readers and support the BRU and BRC program development and purchase a license today.

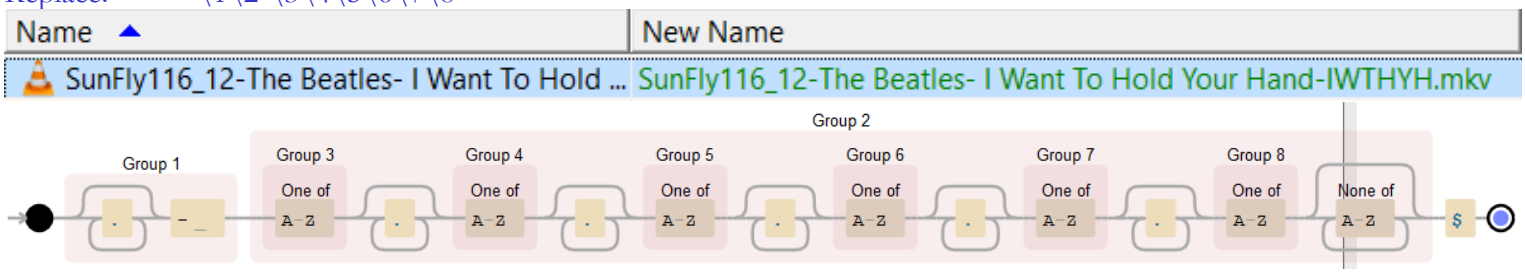
Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

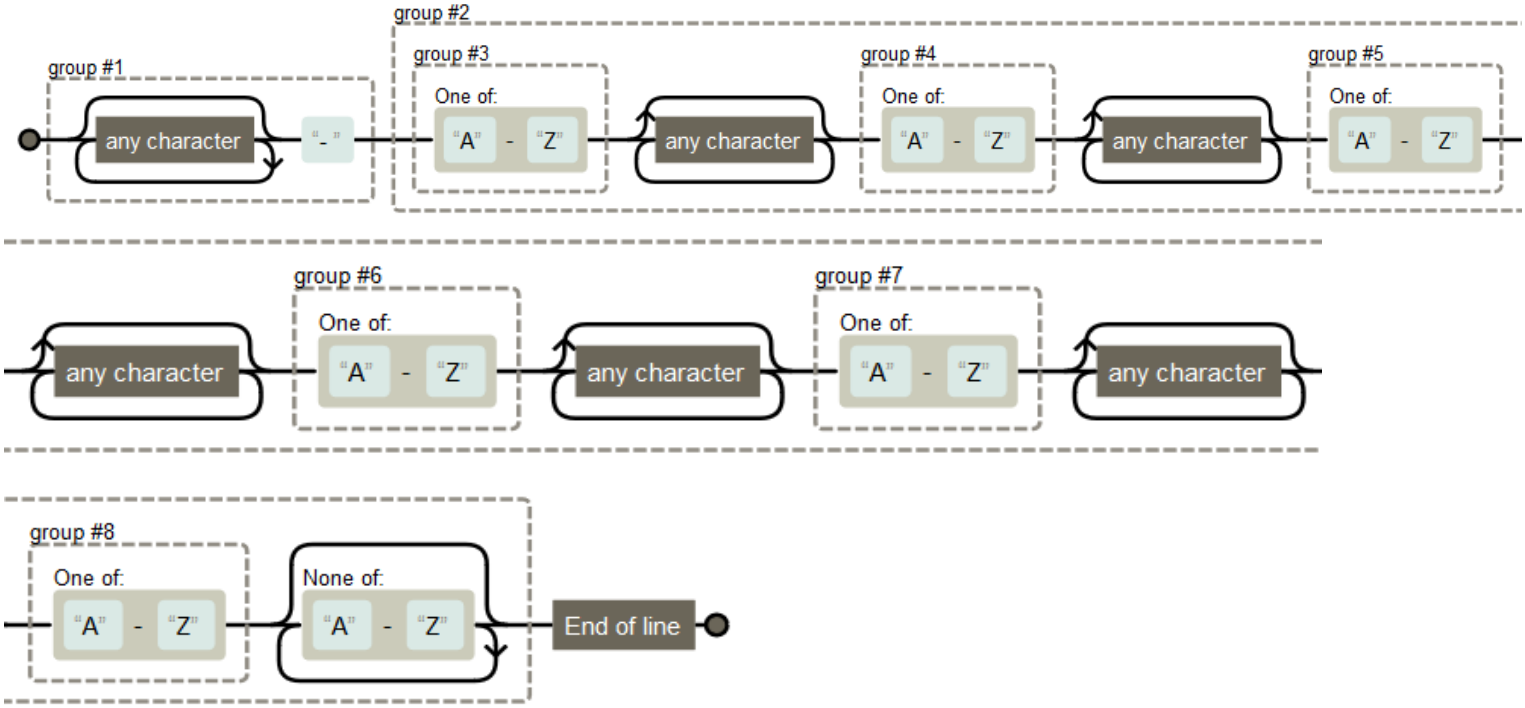
RegEx by bru

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺

Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$
Replace: \1\2-\3\4\5\6\7\8



Expanded:




```
Match 1: SunFly116_12-The Beatles- I Want To Hold Your Hand
Group 1: SunFly116_12-The Beatles-
Group 2: I Want To Hold Your Hand
Group 3: I
Group 4: W
Group 5: T
Group 6: H
Group 7: Y
Group 8: H
```

Contributor and Tim Notes:

1. Be careful in selection. This one isn't fussy about what it renames. It will rename files that have already been renamed. See page 917.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺ cont.

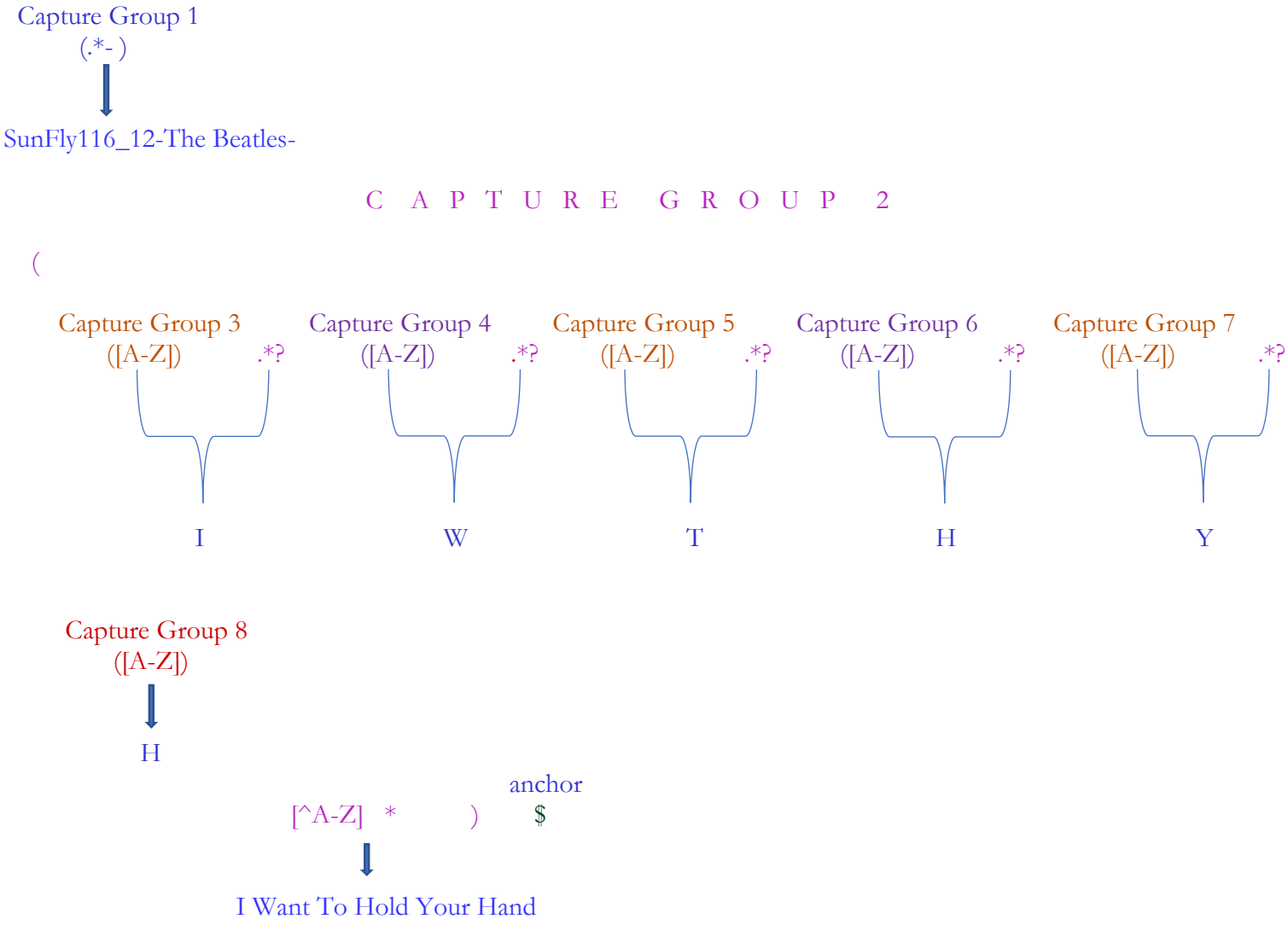
Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$

Where:

\1 = Using: SunFly116_12-The Beatles- <space> \2 = I Want To Hold Your Hand

\3 = I \4 = W \5 = T \6 = H \7 = Y \8 = H

Breaks down:



Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺ cont.

Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$

Where:

\1 = Using: SunFly116_12-The Beatles- <space> \2 = I Want To Hold Your Hand

\3 = I \4 = W \5 = T \6 = H \7 = Y \8 = H

```
Match 1: SunFly116_12-The Beatles- I Want To Hold Your Hand
Group 1: SunFly116_12-The Beatles-
Group 2: I Want To Hold Your Hand
Group 3: I
Group 4: W
Group 5: T
Group 6: H
Group 7: Y
Group 8: H
```

Tim Notes:

1. Similar to many potentially intimidating RegEx, when broken down you can see that a lot of the sub-expressions are just repetitious patterns.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

Before I begin…

A word from our Sponsor:

There was once two RegEx named Tim and bru
who decided on a challenge to a parsing duel.
And when it was over, one remained humbled
While the other was proud, calm and cool.

What?
Did you think it was going to be Bawdy?
I really am sorry,
but I think you will find me no fool.

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺ cont.

Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$

Where:

\1 = Using: SunFly116_12-The Beatles- <space> \2 = I Want To Hold Your Hand

\3 = I \4 = W \5 = T \6 = H \7 = Y \8 = H

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against the <hyphen>	Capture Group 1 = SunFly116_12-The Beatles-
5. <space>	Match against <space> character	Capture Group 1 = Unchanged

Begin Repetition –

6. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 3 = I Capture Group 2 = I Capture Group 1 = Unchanged
7. .	Match against any character	Changes Capture Group 2 = I <space>
8. *	Make it Greedy	Changes Capture Group 2 = I Want To Hold Your Hand
9. ?	But not too Greedy	Changes Capture Group 2 = I
10. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 4 = W Changes Capture Group 2 = I W
11. .	Match against any character	Changes Capture Group 2 = I Wa
12. *	Make it Greedy	Changes Capture Group 2 = I Want To Hold Your Hand
13. ?	But not too Greedy	Changes Capture Group 2 = I W

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺ cont.

Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$

Where:

\1 = Using: SunFly116_12-The Beatles- <space> \2 = I Want To Hold Your Hand

\3 = I \4 = W \5 = T \6 = H \7 = Y \8 = H

Tim Analysis: cont.

Begin Repetition – cont.

14. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 5 = T Changes Capture Group 2 = I Want T
15. .	Match against any character	Changes Capture Group 2 = I Want To
16. *	Make it Greedy	Changes Capture Group 2 = I Want To Hold Your Hand
17. ?	But not too Greedy	Changes Capture Group 2 = I Want T
18. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 6 = H Changes Capture Group 2 = I Want To H
19. .	Match against any character	Changes Capture Group 2 = I Want To Ho
20. *	Make it Greedy	Changes Capture Group 2 = I Want To Hold Your Hand
21. ?	But not too Greedy	Changes Capture Group 2 = I Want To H

Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺ cont.

Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$

Where:

\1 = Using: SunFly116_12-The Beatles- <space> \2 = I Want To Hold Your Hand

\3 = I \4 = W \5 = T \6 = H \7 = Y \8 = H

Tim Analysis: cont.

Begin Repetition – cont.

22. [A-Z]	[A-Z] Match against a class consisting of an uppercase letter.	Capture Group 7 = Y Changes Capture Group 2 = I Want To Hold Y
23. .	Match against any character	Changes Capture Group 2 = I Want To Hold Yo
24. *	Make it Greedy	Changes Capture Group 2 = I Want To Hold Your Hand
25. ?	But not too Greedy	Changes Capture Group 2 = I Want To Hold Y

End Repetition.

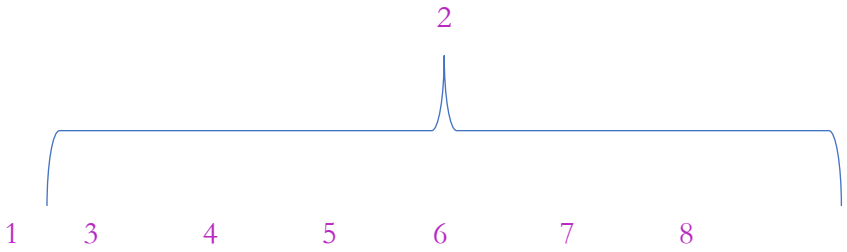
Add

Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru cont.

Using: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

A Challenger comes with TWO Successful BRU RegEx. I am defeated ☺ cont.



Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]*)$

Where:

\1 = SunFly116_12-The Beatles- <space> \2 = I Want To Hold Your Hand

\3 = I \4 = W \5 = T \6 = H \7 = Y \8 = H

Tim Analysis: cont.

26. [A-Z] [A-Z] Match against a class consisting of an uppercase letter.

27. [^A-Z] Match against a Negated Class consisting of an uppercase letter.

28. * Make it Greedy

29. $ Must be at EOL

Capture Group 8 = H
Changes Capture Group 2 =
I Want To Hold Your H
Capture Group 8 = Unchanged
Changes Capture Group 2 =
I Want To Hold Your Ha
Changes Capture Group 2 =
I Want To Hold Your Hand

Add

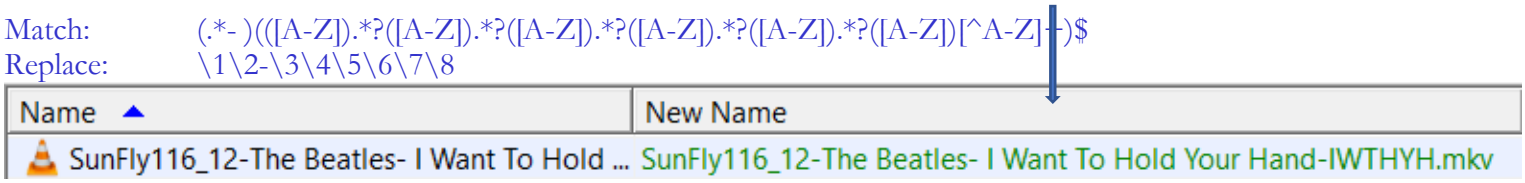
Just for Fun – an Impossible Task – Append Initials of Song Title to end of Filename cont.

RegEx by bru cont.

.. or (from page 909) …

Just change the Greedy Zero or More Quantifier ‘ * ’ at end of string for a Greedy One or More Quantifier ‘ + ’:

Match: (.*-)(([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z]).*?([A-Z])[^A-Z]+)$
Replace: \1\2-\3\4\5\6\7\8



Contributor Notes:

1. By making this change, it won’t evaluate previously renamed files,

   i.e.,

   SunFly116_12-The Beatles- I Want To Hold Your Hand-IWTHYH

Tim Notes:

1. I always encourage you to make your own analysis and see if you can improve upon any of the RegEx in the book. It is all about teaching you to do it yourself, after all. – you have enough to try it yourself or keep reading if you want to learn even more before attempting. Also refer to both RegEx Manual Volume I found in the Appendix of Volume I and the RegEx Manual Volume II found in the appendix of this book.

Victor’s Notes to Tim:

1. I based my answer off of yours, you had it 90% done..
2. Your groups 2-7 were already capturing the uppercase letters but anything captured by ‘ .+ ‘ was lost because it was ‘Not Captured’
3. I grouped around your groups to capture WholeName (Nested Capture Group 2)
4. I changed ‘ .+ ’ to ‘ .* ’ to capture the Zero Occurrence Matches.

Tim Final Notes:

You can see below the Comparison, and based on the Victor’s Notes, I can see the tiny changes that I could have made to have a successful RegEx. I added this under ‘Fixed’:

Comparison –

Mine:	(.*) .+	([A-Z]).+	([A-Z]).+	([A-Z]).+	([A-Z]).+	([A-Z]).+	([A-Z])		.*	
Fixed:	(.*-).+	(([A-Z]).+	([A-Z]).+	([A-Z]).+	([A-Z]).+	([A-Z]).+	([A-Z])		.*)	$
bru:	(.*-)	(([A-Z]).*?	([A-Z]).*?	([A-Z]).*?	([A-Z]).*?	([A-Z]).*?	([A-Z])	[^A-Z]	*)	$

To make my Fixed version work, use a Replacement String of: Replace: \1\2-\3\4\5\6\7\8

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title)

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg
to
SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.cdg

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.mp3
to
SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.mp3

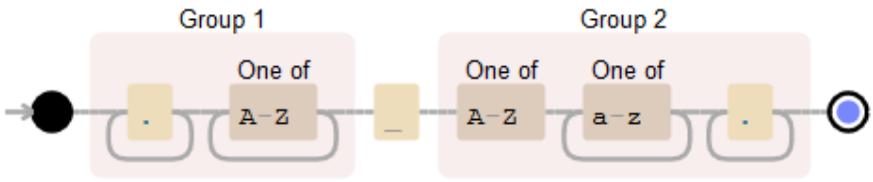
SF175-01 - ANASTACIA Not That Kind .cdg	to	SF175-01 - ANASTACIA - Not That Kind .cdg
SF175-01 - ANASTACIA Not That Kind .mp3	to	SF175-01 - ANASTACIA - Not That Kind .mp3

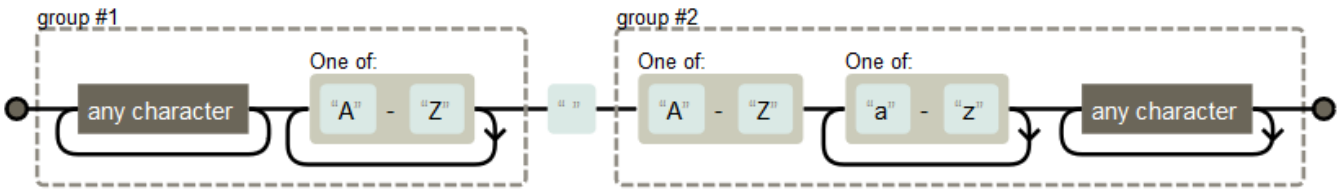
Observations:

1. Add a " - " after each Artist Name but before the Title to separate artist from Title.
2. The name of the artists are all upper case and the title is usually first character upper case and second lower case.
3. Pattern established.

Match: (.+?[A-Z]+) ([A-Z][a-z]+.+)
Replace: \1 - \2

Name	New Name
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.m...	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.mp3
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.cdg
SF175-01 - ANASTACIA Not That Kind .mp3	SF175-01 - ANASTACIA - Not That Kind .mp3
SF175-01 - ANASTACIA Not That Kind .cdg	SF175-01 - ANASTACIA - Not That Kind .cdg





Sample:

```
Match 1: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing
Group 1: SF175-03 - JENNIFER LOPEZ
Group 2: My Love Don't Cost A Thing
```

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Contributor Analysis:

1. Match non-greedy everything, followed by one-or-more upper case characters followed by a blank. — Capture Group 1

2. Match one upper case character followed by one-or-more lower case characters, followed by one-or-more of any text or characters — Capture Group 2

Note: This may only work with non-umlauts characters using standard English.

Tim Notes:

1. umlaut - a mark (¨) used over a vowel, as in German or Hungarian, to indicate a different vowel quality, usually fronting or rounding. Includes - Ü ü Ö ö Ä ä and ß

2. I notice that a couple of the samples (not shown) had a trailing <space> at the end of the filename. This can be suppressed by combining the above RegEx with the 'Trim' function of 'Section #5: Remove'.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg
to
Match: (.+?[A-Z]+) ([A-Z][a-z]+.+)

Where:

\1 = SF175-03 - JENNIFER LOPEZ \2 = My Love Don't Cost A Thing

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = S
4. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 1= SF
5. +	Make it Greedy	Capture Group 1 = Unchanged
6. <space>	Match against <space> character, (Not Captured) followed by … This is the <space> after 'JENNIFER'	Changes Capture Group 1 = SF175-03 - JENNIFER

Tim Notes:

1. The reason it matches against the <space> after "JENNIFER' rather than the <space> after '03' is because it is still matching against the uppercase letter made Greedy in steps 4 and 5.

2. In step 4 it matched against the 'F' and in step 5 it was made greedy, but the next character was a numeric that didn't match, until the <space> was matched.

3. When the <space> was evaluated, it continued to match against the uppercase letters (still Greedy) until it encounters the <space> which was matched; thereby changing Capture Group 1's value to include the additional uppercase letters also matched.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg
to
Match: (.+?[A-Z]+) ([A-Z][a-z]+.+)

Where:

\1 = SF175-03 - JENNIFER LOPEZ \2 = My Love Don't Cost A Thing

Tim Analysis: cont.

7. [A-Z]	.. Match against a class consisting of an uppercase letter, followed by …	Capture Group 2 = L
8. [a-z]	.. Match against a lowercase letter	Capture Group 2 = My

Tim Notes:

1. In step 7, an uppercase letter is matched against the 'L' because the current position after step 6 was the <space> after 'JENNIFER'.

2. In step 8, however, it evaluates for a lowercase letter. The next lowercase letter is the 'y' of 'My' thereby changing Capture Group's 2 value to 'My' because there is no lowercase letter after the 'L' until the 'y'. The match has to be an uppercase letter immediately followed by a lowercase letter.

9. +	Make it Greedy	Capture Group 2 = Unchanged

Tim Notes:

1. No change because the next character is a <space> after 'My' and it is trying to match against a lowercase letter.

10. .+	Match against remaining string	Capture Group 2 = My Love Don't Cost A Thing

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation:

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg
to
SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.cdg

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.mp3
to
SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.mp3

SF175-11 - MARTINE McCUTCHEN On The Radio.mp3
to
SF175-11 - MARTINE McCUTCHEN - On The Radio.mp3

SF999-98 - BOB DYLAN I Can't Get You Off Of My Mind.mp3
to
SF999-98 - BOB DYLAN - I Can't Get You Off Of My Mind.mp3

SF175-01 - ANASTACIA Not That Kind .cdg	to	SF175-01 - ANASTACIA - Not That Kind .cdg
SF175-01 - ANASTACIA Not That Kind .mp3	to	SF175-01 - ANASTACIA - Not That Kind .mp3
SF175-09 - U2 Stuck In a Moment With You.cdg	to	SF175-09 - U2 - Stuck In a Moment With You.cdg
SF175-10 - A1 No More.cdg	to	SF175-10 - A1 - No More.cdg
SF999-99 - BOB DYLAN A Fool Such As I.mp3	to	SF999-99 - BOB DYLAN - A Fool Such As I.mp3

Observations:

1. A more complex example to extend this RegEx to cover additional file names.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)
Replace: \1 - \2

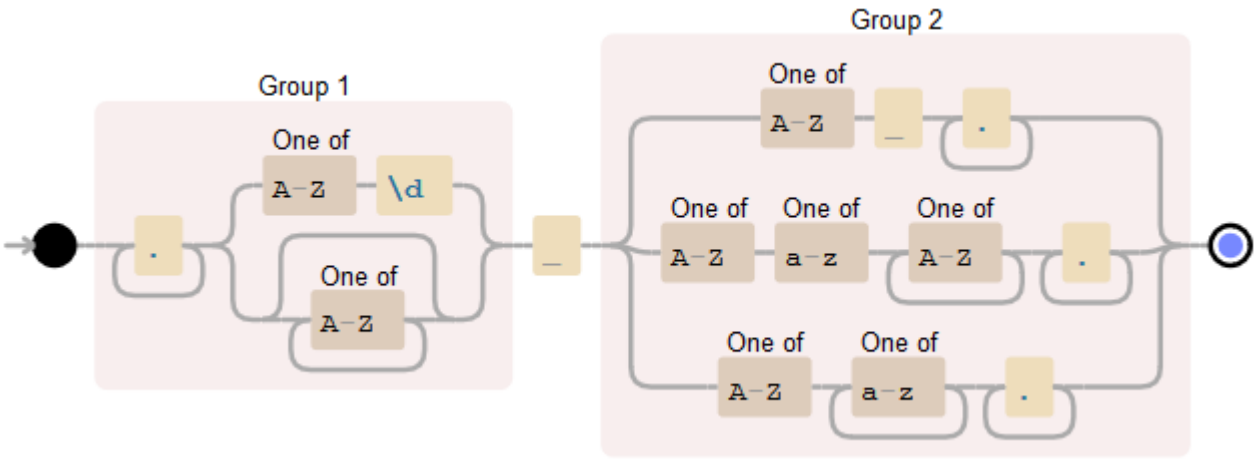
Name	New Name
SF999-99 - BOB DYLAN A Fool Such As I.mp3	SF999-99 - BOB DYLAN - A Fool Such As I.mp3
SF999-98 - BOB DYLAN I Can't Get You Off Of My Mind.mp3	SF999-98 - BOB DYLAN - I Can't Get You Off Of My Mind.mp3
SF175-11 - MARTINE McCUTCHEN On The Radio.mp3	SF175-11 - MARTINE - McCUTCHEN On The Radio.mp3
SF175-10 - A1 No More.cdg	SF175-10 - A1 - No More.cdg
SF175-09 - U2 Stuck In a Moment With You.cdg	SF175-09 - U2 - Stuck In a Moment With You.cdg
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.m...	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.mp3
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.cdg
SF175-01 - ANASTACIA Not That Kind .mp3	SF175-01 - ANASTACIA - Not That Kind .mp3
SF175-01 - ANASTACIA Not That Kind .cdg	SF175-01 - ANASTACIA - Not That Kind .cdg

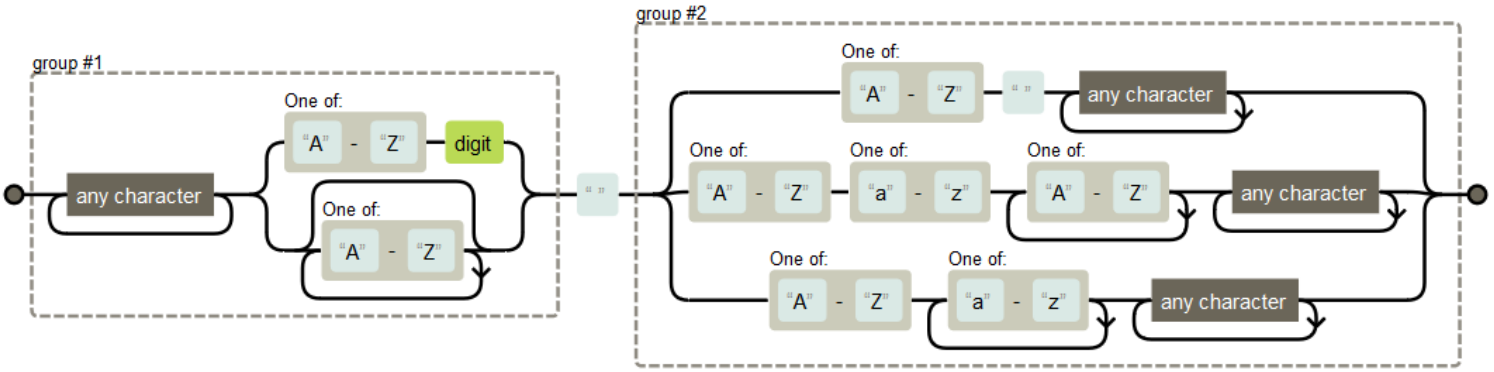
Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)





Tim Notes:

1. Did not work for Martine McCutchen filename.

instead of :

SF175-11 - MARTINE McCUTCHEN - On The Radio.mp3

resulted in this:

SF175-11 - MARTINE - McCUTCHEN On The Radio.mp3

Where:

\1 = SF175-11 – MARTINE \2 = McCUTCHEN On the Radio
\1 = SF999-98 – BOB DYLAN \2 = I Can't Get You Off Of My Mind

I see the problem. Where all other Artists names are all in uppercase, this filename uses both Upper and Lowercase to distinguish the Mc. The RegEx is following a pattern of all uppercase.

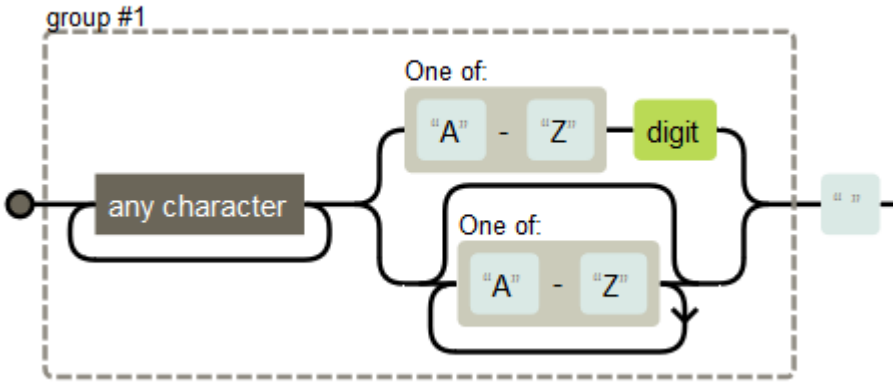
Add

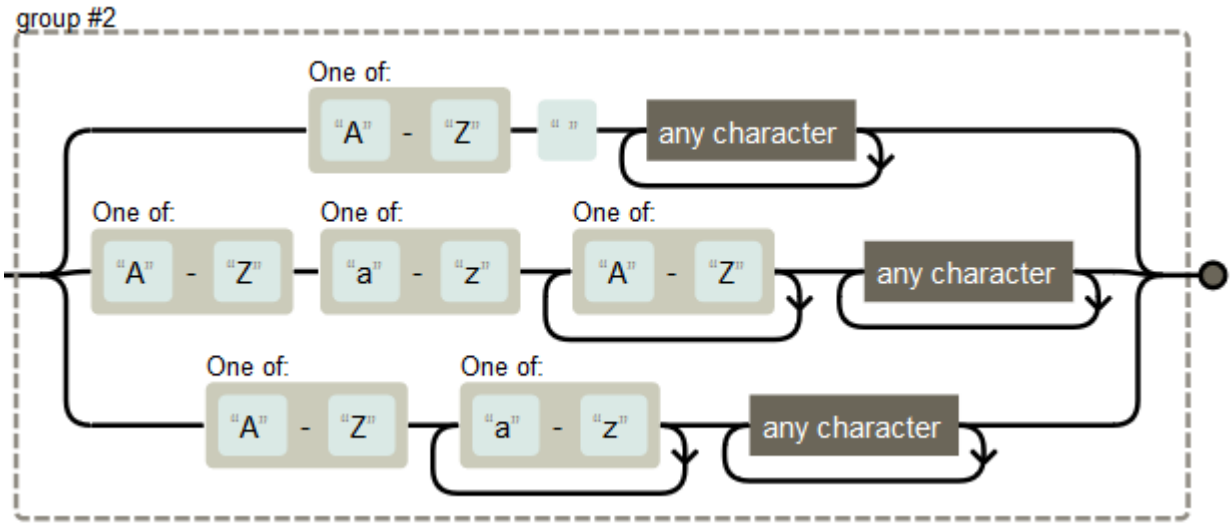
Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Expanded:





Sample:

```
.....................................................................
Match 1: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing
Group 1: SF175-03 - JENNIFER LOPEZ
Group 2: My Love Don't Cost A Thing
```

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Where:

\1 = SF175-03 - JENNIFER LOPEZ \2 = My Love Don't Cost A Thing

Breaks down:

Capture Group 1 Capture Group 2

Non-Marking Group
alternate alternate alternate
(.+? (?: [A-Z] \d | [A-Z]*)) ([A-Z] .+ | [A-Z] [a-z] [A-Z]+ .+ | [A-Z] [a-z]+ .+)

This uses an Alternate for Capture Group 1:

This	OR	This
[A-Z]\d	|	[A-Z]*
Match against an Uppercase letter immediately followed by a numeric digit		Match against all uppercase letters until a non-uppercase letter

This uses an Alternate for Capture Group 2:

This	OR	This	OR	This
[A-Z] .+	|	[A-Z][a-z][A-Z]+.+	|	[A-Z][a-z]+.+
Match against an Uppercase letter followed by a <space> followed by any additional text.		Match against an uppercase letter immediately followed by a lowercase letter immediately followed by additional uppercase letters followed by any additional text.		Match against an uppercase letter immediately followed by additional lowercase letters followed by any additional text.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Where:

\1 = SF175-03 - JENNIFER LOPEZ \2 = My Love Don't Cost A Thing

Tim Notes:

1. A Non-marking Group does not capture unless additional Captures within or without are specified using parentheses. In this case, the entire Non-Marking Group is enclosed in Capture Group 1 so Capture Group 1 will include any values matched in the Non-Marking Group.

2. If a Non-marking Group was not used, then Capture Group 1 would be nested and the Non-marking Group would become Capture Group 2 and Capture Group 2 would become Capture Group 3.

Tim Analysis:

1..	Match against any character	Capture Group 1 = S
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = S

4. [A-Z]\d

Alternate (see Previous Page)

a. [A-Z]	Match against a Class consisting of an uppercase letter.	Capture Group 1 = SF
b. \d	Match against a numeric digit	Capture Group 1 = SF1

or

a. [A-Z]	Match against a Class consisting of an uppercase letter.	Capture Group 1 = SF
b. *	Make it Greedy	Capture Group 1 = Unchanged

Tim Notes:

1. All of the samples presented will *only match against the first alternative* in Capture Group 1. To match against the second alternative of Capture Group 1, the sample will have to be something like, 'SFTTT999-98 - BOB DYLAN', then Capture Group 1 = 'SFTTT'.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Where:

\1 = SF175-03 - JENNIFER LOPEZ \2 = My Love Don't Cost A Thing

Tim Analysis: cont.

5. <space>	Match against a <space> character (Not Captured) This is the <space> after '03'.	Changes Capture Group 1 = SF175-03

6. [A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+

Alternate (see Previous Page)

a. [A-Z]	Match against a Class consisting of an uppercase letter.	Capture Group 2 = J Changes Capture Group 1 = SF175-03 -
b. <space>	Match against a <space> character	Capture Group 2 = A<space> Changes Capture Group 1 = SF175-03 - JENNIFER LOPEZ My Love <space> Don't Cost
c. .	Match against any character	Capture Group 2 = A T Capture Group 1= Unchanged
d. +	Make it Greedy	Capture Group 2 = A Thing

or

a. [A-Z]	Match against a Class consisting of an uppercase letter.	Capture Group 2 = J Changes Capture Group 1 = SF175-03 -
b. [a-z]	Match against a Class consisting of a lowercase letter.	Capture Group 2 = My Changes Capture Group 1 = SF175-03 - JENNIFER LOPEZ
c. [A-Z]	Match against a Class consisting of an uppercase letter. = <space> = False	
d. +	Make it Greedy	
e. .	Match against any character	
f. +	Make it Greedy	

* Will fail for this sample on the first alternate of Capture Group 2 even though it shows a match at the 'A<space>. Explanation will follow.

* Fails for this sample on the second alternate of Capture Group 2.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Where:

\1 = SF175-03 - JENNIFER LOPEZ \2 = My Love Don't Cost A Thing

Tim Analysis: cont.

6. [A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+ cont.

Alternate (see Previous Pages)

a. [A-Z]	Match against a Class consisting of an uppercase letter.	Capture Group 2 = J Changes Capture Group 1 = SF175-03 -
b. [a-z]	Match against a Class consisting of a lowercase letter.	Capture Group 2 = My Changes Capture Group 1 = thSF175-03 - JENNIFER LOPEZ
c. +	Make it Greedy	All Capture Groups = Unchanged
d. .	Match against any character	Capture Group 2 = My <space> Capture Group 1 = Unchanged
e. +	Make it Greedy	Capture Group 2 = My Love Don't Cost A Thing Capture Group 1 = Unchanged

For the sample, Alternate three is the final selection for Capture Group 2.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Tim Analysis: cont.

So, question. 'If the first Alternate matched in Capture Group 2, why would it end up selecting the Third Alternate, if the RegEx Engine goes with First come, first served?'

The simple explanation is that it didn't match- a least not enough to satisfy the match. When the expression is provided with alternates, upon any failure to match, the RegEx Engine will seek out the next alternate without trying to satisfy the match with further permutations. If the expression did not have any alternates, then the RegEx Engine would continue to try and satisfy the match.

Here is my interpretation of a more technical nature:

With the Alternates:

Capture Group 1
(.+?(?:[A-Z]\d|[A-Z]*)) <space>

1. Capture Group 1 = SF175-03

2. Current position is the <space> after '03' that is matched but not captured by the <space> after Capture Group 1.

Capture Group 2 – First Alternate
([A-Z]<space> .+|

1. The sub-expression, [A-Z]<space> is looking for an uppercase letter followed immediately by a <space>.
2. When the first part [A-Z] of the sub-expression is evaluated, the RegEx Engine finds the 'J' of 'JENNIFER' satisfying the first match for the initial part of the expression.
3.. It evaluates the second part of the sub-expression, looking for an immediate <space>.
4. = 'E' = Fails.
5. Instead of trying to match further, given another Alternate available, it moves to the second alternate which, has already been explained, Fails.

Capture Group 2 – Third Alternate
[A-Z][a-z]+.+

1 The RegEx Engine moves on to the third alternate. Now it is looking for an Uppercase letter immediately followed by a lowercase letter.
2. It matches against the 'My' made Greedy using the ' + '
3. 'Love Don't Cost a Thing' captured by the ending ' .+ '

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Tim Analysis: cont.

So now let's see what happens if we don't use any alternates and rewrite the RegEx as:

Capture Group 1	Capture Group 2
(.+?(?:[A-Z]\d|[A-Z]*))<space>	([A-Z]<space> .+)

Without the alternates:

Capture Group 1
(.+?(?:[A-Z]\d|[A-Z]*)) <space>

1. Capture Group 1 = SF175-03

2. Current position is the <space> after '03' that is matched but not captured by the <space> after Capture Group 1.

Capture Group 2 – First Alternate
([A-Z]<space> .+|

1. The sub-expression, [A-Z]<space> is looking for an uppercase letter followed immediately by a <space>.
2. When the first part [A-Z] of the sub-expression is evaluated, the RegEx Engine finds the 'J' of 'JENNIFER' satisfying the first match of the initial part of the expression.
3.. It evaluates the second part of the sub-expression, looking for an immediate <space>.
4. = 'E' = Fails.
5. The RegEx Engine attempts to satisfy the match by moving forward.
6. Because of the (.+?)<space> of Capture Group 1 which it is still matching against, the RegEx Engine is looking for a match of a <space> - this would be the <space> after Capture Group 1 that was not captured, followed by the match of [A-Z] <space> of Capture Group 2. In other words, <space> Uppercase Letter <space>
7. It will continue to search through the string until it hits the EOL.
8. It then backtracks matching against the '<space> A <space>' of 'Cost A Thing'.
   a. Changes Capture Group 1 = SF175-03 - JENNIFER LOPEZ My Love Don't Cost
9. The First <space> is not captured. This would be the <space> after Capture Group 1 –

(.+?(?:[A-Z]\d|[A-Z]*))<space>

The 'A <space>' is captured in Capture Group 2 satisfying the match of [A-Z]<space>

([A-Z]<space> .+)

10. 'Thing' captured by the ending ' .+ ' Capture Group 2 = A Thing

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Match: (.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

Using: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg

Tim Analysis: cont.

Using only the first and last Alternates (for clarification):

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing

(.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+|[A-Z][a-z]+.+)

```
.....................................................................
Match 1: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing
Group 1: SF175-03 - JENNIFER LOPEZ
Group 2: My Love Don't Cost A Thing
```

Without the Alternates:

(.+?(?:[A-Z]\d|[A-Z]*)) ([A-Z] .+)

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing

```
.....................................................................
Match 1: SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing
Group 1: SF175-03 - JENNIFER LOPEZ My Love Don't Cost
Group 2: A Thing
```

This can further be simplified to:

(.+?) ([A-Z] .+)

(.+?) ([A-Z] .+)

SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing

Same result.

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Post continues -

Contributor and *Tim Analysis:*

1. Capture Group 1: (.+?(?:[A-Z]\d|[A-Z]*))

consists of (for clarification):

(.+? (?: [A-Z]\d | [A-Z]*))

a. Match non-greedy one-or-more of any characters, followed by one uppercase character followed by one numeric digit,

OR

b. One-or-more of uppercase characters until a <space>, to Capture Group 1.

2. Capture Group 2: ([A-Z] .+|[A-Z][a-z][A-Z]+.+|[A-Z][a-z]+.+)

consists of (for clarification):

([A-Z] .+ | [A-Z][a-z][A-Z]+ .+ | [A-Z][a-z]+ .+)

a. Match one uppercase character followed by one blank followed by one-or-more characters, OR –

b. … one uppercase one lowercase one-or-more uppercase characters followed by one-or-more characters, OR -

c. … one uppercase character followed by one-or-more lowercase characters followed by one blank followed by one-or-more characters.

Add

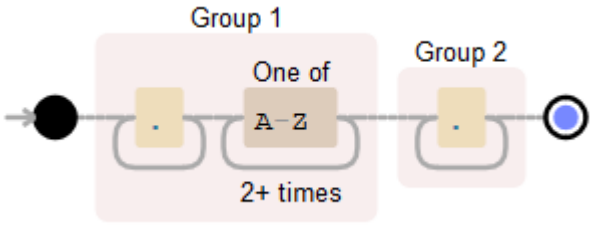
Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

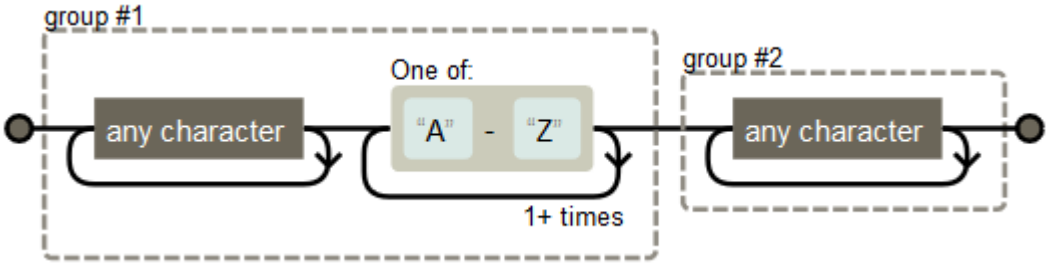
Variation: cont.

Another way is to match everything up to and inclusive of the 'last two uppercase letters':

Match: (.+[A-Z]{2,})(.+)
Replace: \1 -\2

Name	New Name
SF999-99 - BOB DYLAN A Fool Such As I.mp3	SF999-99 - BOB DYLAN - A Fool Such As I.mp3
SF999-98 - BOB DYLAN I Can't Get You Off Of My Mind.mp3	SF999-98 - BOB DYLAN - I Can't Get You Off Of My Mind.mp3
SF175-11 - MARTINE McCUTCHEN On The Radio.mp3	SF175-11 - MARTINE McCUTCHEN - On The Radio.mp3
SF175-10 - A1 No More.cdg	SF175-10 - A1 No More.cdg
SF175-09 - U2 Stuck In a Moment With You.cdg	SF175-09 - U2 Stuck In a Moment With You.cdg
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.m...	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.mp3
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.cdg
SF175-01 - ANASTACIA Not That Kind .mp3	SF175-01 - ANASTACIA - Not That Kind .mp3
SF175-01 - ANASTACIA Not That Kind .cdg	SF175-01 - ANASTACIA - Not That Kind .cdg





Sample:

```
.................................................
Match 1: SF175-11 - MARTINE McCUTCHEN On The Radio
Group 1: SF175-11 - MARTINE McCUTCHEN
Group 2:  On The Radio
```

Tim Notes:

1. This matched all except filenames that used an artist name beginning with an uppercase followed by a numeric digit.

Forum Contributor: Stefan, Truth

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation: cont.

Using: SF175-11 - MARTINE McCUTCHEN On The Radio.mp3

Match: (.+[A-Z]{2,})(.+)

Where: \1 = SF175-11 - MARTINE McCUTCHEN \2 = <space> On The Radio.mp3

Tim Analysis:

1. .	Match against first character	Capture Group 1 = S
2. +	Make it Greedy	Capture Group 1 = entire string
3. [A-Z]	Match against a class consisting of an uppercase letter Capture Group 1 = SF175-11 - MARTINE McCUTCHEN On The R RegEx engine is backtracking. It is at the EOL so the first uppercase letter found is the 'R' of Radio	
4. {2}	Limits the iterations to 2. This means that in order to be TRUE, the RegEx must match against two uppercase letters in sequence. Therefore 'EN' is the match: 'SF175-11 - MARTINE McCUTCHEN On The R' Capture Group 1 = SF175-11 - MARTINE McCUTCHEN	
5. .	Match against next character <space> following 'N' of 'McCUTCHEN' RegEx engine is moving forward again.	Capture Group 2 = <space>
6. +	Make it Greedy	Capture Group 2 = <space> On The Radio

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

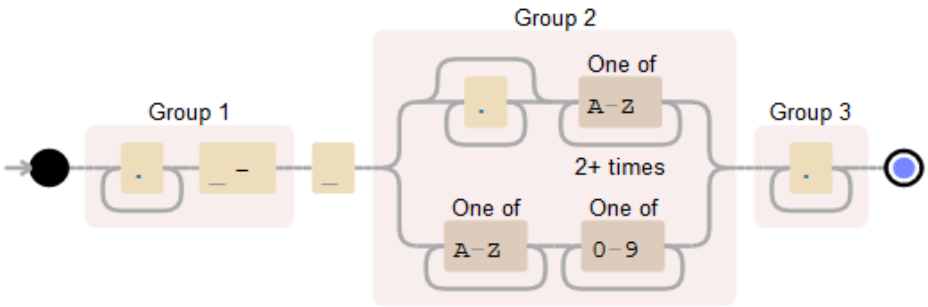
Variation: cont.

To capture Group Names ending like U2, you could match either:

Last two uppercase letters OR Last uppercase followed by Numeric Digits

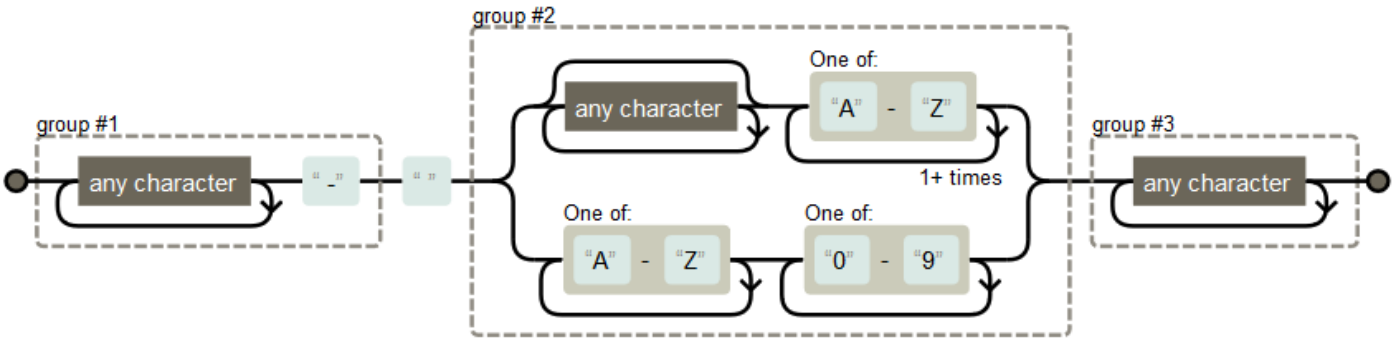
Match: (.+ -) (.*[A-Z]{2,}|[A-Z]{1,}[0-9]{1,})(.+)
Replace: \1 \2 -\3

Name	New Name
SF999-99 - BOB DYLAN A Fool Such As I.mp3	SF999-99 - BOB DYLAN - A Fool Such As I.mp3
SF999-98 - BOB DYLAN I Can't Get You Off Of My Mind.mp3	SF999-98 - BOB DYLAN - I Can't Get You Off Of My Mind.mp3
SF175-11 - MARTINE McCUTCHEN On The Radio.mp3	SF175-11 - MARTINE McCUTCHEN - On The Radio.mp3
SF175-10 - A1 No More.cdg	SF175-10 - A1 - No More.cdg
SF175-09 - U2 Stuck In a Moment With You.cdg	SF175-09 - U2 - Stuck In a Moment With You.cdg
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.m...	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.mp3
SF175-03 - JENNIFER LOPEZ My Love Don't Cost A Thing.cdg	SF175-03 - JENNIFER LOPEZ - My Love Don't Cost A Thing.cdg
SF175-01 - ANASTACIA Not That Kind .mp3	SF175-01 - ANASTACIA - Not That Kind .mp3
SF175-01 - ANASTACIA Not That Kind .cdg	SF175-01 - ANASTACIA - Not That Kind .cdg



Tim Notes:

1. This worked for ALL samples. Mission accomplished.



Sample:

```
.....................................................
Match 1: SF175-09 - U2 Stuck In a Moment With You
Group 1: SF175-09 -
Group 2: U2
Group 3:  Stuck In a Moment With You
```

Forum Contributor: Stefan, Truth

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation:

Using: SF175-10 - A1 No More.cdg

Where:
\1 = SF175-10 - \2 = A1 \3 = <space> No More

Tim Analysis:

(.+ -) (.*[A-Z]{2,}|[A-Z]{1,}[0-9]{1,})(.+)

This uses an Alternate for Capture Group 2:

This	OR	This
.* [A-Z]{2,}	|	[A-Z] {1,} [0-9] {1,}
Capture all until first 2 Uppercase letters minimum to maximum infinity		Capture first uppercase letter minimum to maximum infinity followed by a single numeric minimum

Tim Notes:

1. I added <spaces> above for clarity – there are no <spaces> in the expressions for the alternates.

Where:

	Matches:	Matches:
	SF175-11 - MARTINE McCUTCHEN On The Radio.mp3	SF175-10 - A1 No More.cdg
1. .	Match against first character	Capture Group 1 = S
2. +	Make it Greedy until….	Capture Group 1 = entire string
3. <space>	.. <space> followed by … RegEx engine is backtracking	Capture Group 1 = SF175-10 –A1 No <space>
4. -	… <hyphen> followed by..	Capture Group 1 = SF175-10 –
5. <space>	.. <space> (Not Captured) This is the <space> after 'A1' RegEx engine is moving forward	

Add

Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title) cont.

Variation:

Using: SF175-10 - A1 No More.cdg

Where:
\1 = SF175-10 - \2 = A1 \3 = <space> No More

Tim Analysis: cont.

Alternate 1

(.*[A-Z]{2,})

6. .	Match against next character	Capture Group 2 = A
7. *	Make it Greedy	Capture Group 2 = A1 No More
8. [A-Z]	Match against a class consisting of an uppercase letter RegEx engine is backtracking	Capture Group 2 = A1 No M
9. {2,}	Limits the iterations to 2 or More. This means that in order to be TRUE, the RegEx must match against minimum two uppercase letters in sequence. Sample Fails. There are not two uppercase letters sequentially in the string.	

Alternate 2

([A-Z]{1,}[0-9]{1,})

6. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = A
7. {1,}	Limits the iterations to one or more. This means that in order to be TRUE, the RegEx must match against at least one uppercase letters in sequence. = TRUE. 'A' is the match. Capture Group 2 unchanged.	
8.[0-9]	Match against a class consisting of numeric digits.	Capture Group 2= A1
9. {1,}	Limits the iterations to one or more. This means that in order to be TRUE, the RegEx must match against at least one numeric digit. = TRUE. '1' is the match. Capture Group 2 unchanged.	

RegEx Continues

10. (.+)	Capture remainder of string.	Capture Group 3 = <space> No More

Add

Pad 0 before variable numbers (represented by #)

Word word 01x 7 - variable_word (other words).txt	to	Word word 01x 007 - variable_word (other words).txt
Word word 01x 88 - variable_word (other words).txt	to	Word word 01x 088 - variable_word (other words).txt
Word word 01x 123 - variable_word (other words).txt	to	Ignore

Observations:

1. Pad '0' and '00' in # and ## cases only.

Tim Observations:

As with many of these requests that rely on identifying a certain number of numeric digits in the middle of the string rather than at then end or beginning, it is difficult to do this in BRU RegEx in one single run (although I believe in the command line version, BRC, it can be done since BRC can perform multiple RegEx's in one pass). This is because in BRU PCRE RegEx at least (unsure if this exists in any RegEx), there isn't an expression that can differentiate between the quantity of items (nor is the Global Switch that would allow multiple matches within the same string, as of this date, supported).

If, for example, there are three strings that have one to three numeric digits, the moment you specify one digit in the expression, it will match all three. You can limit an expression to affect certain strings if there are additional patterns that match one string vs another, but in most cases, there will still be unintended strings that have been altered, while other strings that needed to be altered, remain unaffected. This concludes that the expression needs to be universal, matching the pattern and 'Damn the Torpedoes' (meaning inadvertent affected file names be damned).

Therefore, changes such as padding require at least two runs in most[1] cases. The first run usually adds an equal number of padding to all of the matching strings based on the maximum length of the sample or the pattern within the sample.

In this example, the string containing the maximum number of numeric characters is,

i.e.,
3 numeric places for '123' … - 3 numeric digits is therefore the maximum based on the file…
'Word word 01x 123 - variable_word (other words).txt'

The padding added to all of the strings will be two places or, Zeros regardless. This is based on a minimum of 1 numeric digit in the string to equal three numeric digits, will require two additional zeroes.

This will of course, create unintended alterations. Some Files will have too many Zeros added, while other files will be altered that were not supposed to be altered.

The Second run will select only those strings. Any excess padding (Zeros) will be removed, correcting files that had too much padding added in the first run while restoring those files that should not have been altered back to their original filenames.

1. It should be noted that padding numeric digits *can* be done in a single run, and in this book I have fully illustrated techniques that can do just that.

Add

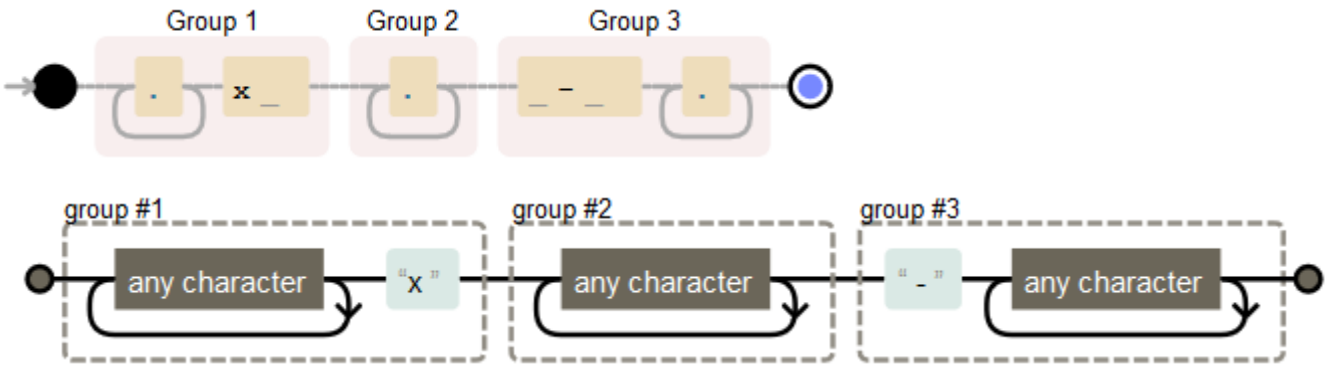
Pad 0 before variable numbers (represented by #) cont.

First run:

Add zeros padded to required length.

Match: (.+x)(.+)(- .+)
Replace: \100\2\3

Name	New Name
Word word 01x 88 - variable_word (other words).txt	Word word 01x 0088 - variable_word (other words).txt
Word word 01x 7 - variable_word (other words).txt	Word word 01x 007 - variable_word (other words).txt
Word word 01x 123 - variable_word (other words).txt	Word word 01x 00123 - variable_word (other words).txt



Sample:

```
.....................................................
Match 1: Word word 01x 7 - variable_word (other words)
Group 1: Word word 01x
Group 2: 7
Group 3:  - variable_word (other words)
```

Contributor Analysis:

1. Match 'anything till x + <space> ' Capture Group 1.
2. .. then anything Capture Group 2
3. .. till <space> <hyphen> <space>
4. .. and the rest. Capture Group 3

Add as many zeros as required in the Replace string

Add

Pad 0 before variable numbers (represented by #) cont.

Using: Word word 01x 7 - variable_word (other words).txt

First run: cont.

Add zeros padded to required length.

Match: (.+x)(.+)(- .+)

Where:

\1 = Word word 01x <space> \2 = 7 \3 = <space> - variable_word (other words)

Tim Analysis:

1..	Match against any character	Capture Group 1 = W
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. x	Match against string 'x'	Capture Group 1 = Word word 01x
4. <space>	Match against a <space> character	Capture Group 1 = Word word 01x <space>
5..	Match against any character	Capture Group 2 = 7
6. +	Make it Greedy	Capture Group 2 = 7 - variable_word (other words)
7. <space>	Match against a <space> character This is the <space> after 'other'	Capture Group 3 = <space> Changes Capture Group 2= 7 - variable_word (other
8. <hyphen>	Match against <hyphen>	Capture Group 3 = <space><hyphen> Changes Capture Group 2 = 7
9. .	Match against any character	Capture Group 3 = <space>
10. +	Make it Greedy	Capture Group 3 = <space> - variable_word (other words)

Replace: \100\2\3

New Name = Word word 01x 007 - variable_word (other words).txt

Add

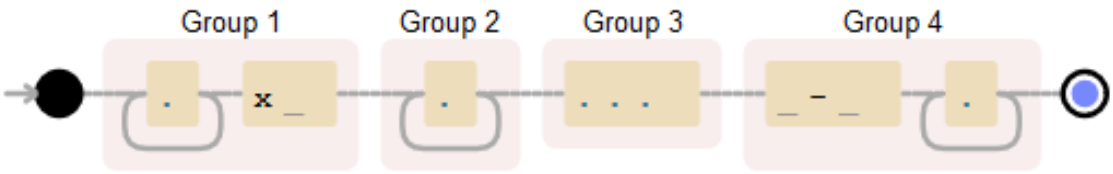
Pad 0 before variable numbers (represented by #) cont.

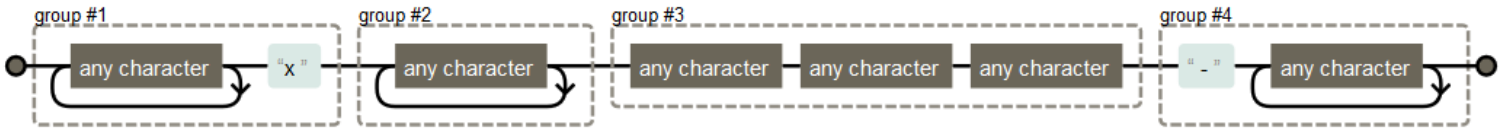
Second run:

Correct any unintended files. Ignore any files that do not require correction.

Match: (.+x)(.+)(...)(- .+)
Replace: \1\3\4

Name	New Name
Word word 01x 0088 - variable_word (other words).txt	Word word 01x 088 - variable_word (other words).txt
Word word 01x 007 - variable_word (other words).txt	Word word 01x 007 - variable_word (other words).txt
Word word 01x 00123 - variable_word (other words).txt	Word word 01x 123 - variable_word (other words).txt





Sample:

```
Match 1: Word word 01 x 0088 - variable_word (other words)
Group 1: Word word 01 x
Group 2: 0
Group 3: 088
Group 4:  - variable_word (other words)
```

Contributor Analysis:

1. Match 'anything till x + <space> '	Capture Group 1
2. .. then anything	Capture Group 2
3. .. the last three signs from that anything	Capture Group 3
4. .. till <space> <hyphen> <space>	
5. .. and the rest	Capture Group 4

Forum Contributor: Stefan

Add

Pad 0 before variable numbers (represented by #) cont.

Using: Word word 01x 0088 - variable_word (other words).txt

Second run: cont.

Match: (.+x)(.+)(...)(- .+)

Where:

\1 = Word word 01x <space> \2 = 0 \3 = 088 \4 = <space> - variable_word (other words)

Tim Analysis:

1..	Match against any character	Capture Group 1 = W
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. x	Match against string 'x'	Capture Group 1 = Word word 01x
4. <space>	Match against a <space> character	Capture Group 1 = Word word 01x <space>
5..	Match against any character This is the first zero of '0007'	Capture Group 2 = 0
6. +	Make it Greedy	Capture Group 2 = 0088 - variable_word (other words)
7….	Match against any 3 characters	Capture Group 3 = ds) Changes Capture Group 2 = 0088 - variable_word (other wor
8. <space>	Match against a <space> character This is the <space> after 'other'	Capture Group 4 = <space> Changes Capture Group 3= her Changes Capture Group 2 = <space> - variable_word (ot
9. <hyphen>	Match against <hyphen> Capture Group 2 = '0' is the first zero of '0007'	Capture Group 4 = <space><hyphen> Changes Capture Group 3 = 008 Changes Capture Group 2 = 0
10. .	Match against any character	Capture Group 4 = <space> - v
11. +	Make it Greedy	Capture Group 4 = <space> - variable_word (other words)

Tim Notes:

1. Just because steps 10 and 11 show no change for this sample, there may be different values based on other samples.
2. I am only analyzing one sample from a group of samples.

Replace: \1\3\4

New Name = Word word 01x 088 - variable_word (other words).txt

Add

Pad 0 before variable numbers (represented by #) cont.

Tim Contribution –

Word word 01x 7 - variable_word (other words).txt	to	Word word 01x 007 - variable_word (other words).txt
Word word 01x 88 - variable_word (other words).txt	to	Word word 01x 088 - variable_word (other words).txt
Word word 01x 123 - variable_word (other words).txt	to	Ignore

Here's another one that will also work (again in two runs). It also demonstrates using a 'Section:12 Filter' RegEx.

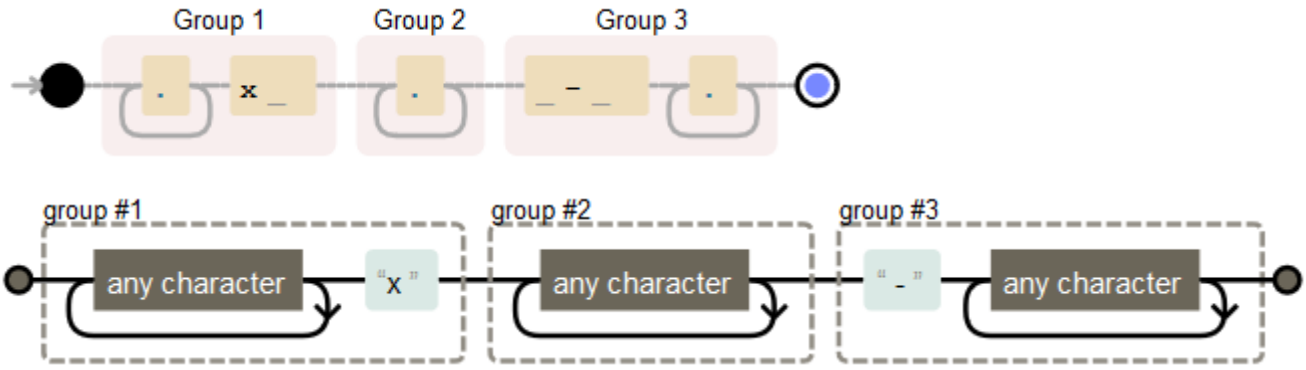
In this method I break against tradition and instead of padding with equal number of Zeros, instead I opt for one. This will still alter the file, 'Word word 01x 123 - variable_word (other words).txt', and will need to be corrected in the second run. Why do this at all? Just to demonstrate a different way to accomplish the same thing. One is neither better than the other. This method only needs to perform a correction to restore unintended files in the second run, not to remove excess padding from previously matched files in the first run.

First run:

1. RegEx

Match: (.*x)([0-9])(.*)
Replace: \10\2\3

Name	New Name
Word word 01x 88 - variable_word (other words).txt	Word word 01x 088 - variable_word (other words).txt
Word word 01x 7 - variable_word (other words).txt	Word word 01x 07 - variable_word (other words).txt
Word word 01x 123 - variable_word (other words).txt	Word word 01x 0123 - variable_word (other words).txt



Sample:

```
.....................................................
Match 1: Word word 01x 7 - variable_word (other words)
Group 1: Word word 01x
Group 2: 7
Group 3:  - variable_word (other words)
```

Add

Pad 0 before variable numbers (represented by #) cont.

Tim Contribution – cont.

Using: Word word 01x 7 - variable_word (other words).txt

First run: cont.

Match: (.*x)([0-9])(.*)

Where:

\1 = Word word 01x <space> \2 = 7 \3 = <space> - variable_word (other words)

Tim Analysis:

1..	Match against any character	Capture Group 1 = W
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. x	Match against string 'x'	Capture Group 1 = Word word 01x
4. <space>	Match against a <space> character	Capture Group 1 = Word word 01x <space>
5. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 2 = 7
6. .*	Match against remaining string	Capture Group 3= <space> - variable_word (other words)

Replace: \10\2\3

New Name = Word word 01x 07 - variable_word (other words).txt

Add

Pad 0 before variable numbers (represented by #) cont.

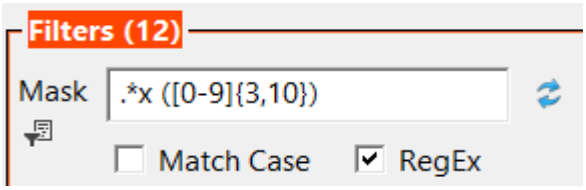
Tim Contribution – cont.

Using: Word word 01x 0123 - variable_word (other words).txt

Second run:

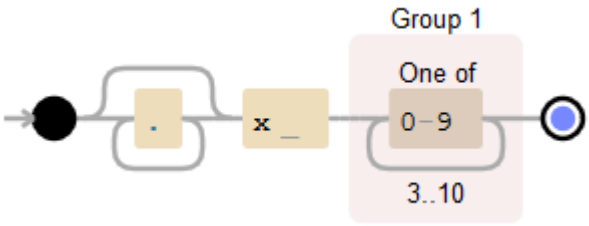
1. 'Section #12: Filter' -

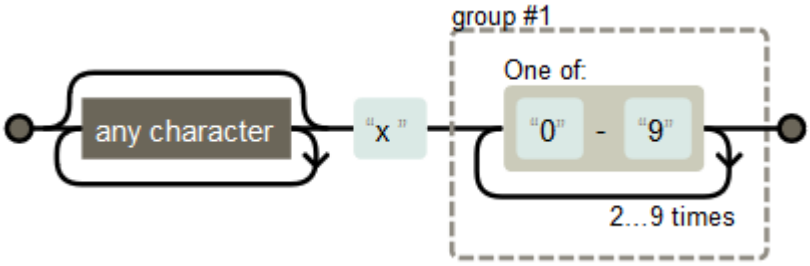
Mask: .*x ([0-9]{3,10})



Use Filter to only display those files that need correction. In this case we select only those files that contain three or more numeric digits. Click on the Refresh Filter button (the blue button to the right of the Mask data field) to see the changes.

Name	New Name
Word word 01x 123 - variable_word (other words).txt	Word word 01x 123 - variable_word (other words).txt





Add

Pad 0 before variable numbers (represented by #) cont.

Tim Contribution – cont.

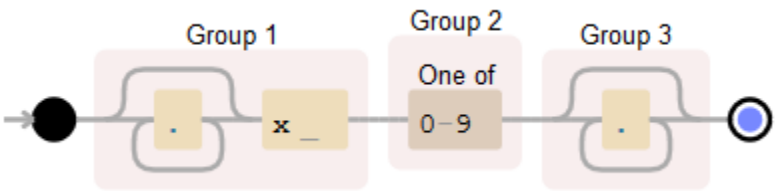
Second run: cont.

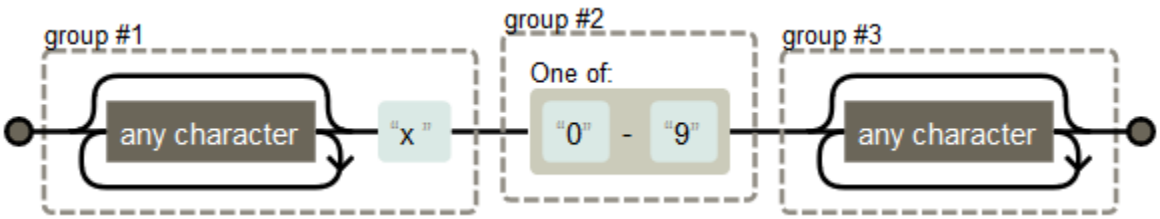
Using: Word word 01x 0123 - variable_word (other words).txt

2. RegEx

Match: (.*x)([0-9])(.*)
Replace: \1\3

Name	New Name
Word word 01x 0123 - variable_word (other words).txt	Word word 01x 123 - variable_word (other words).txt





Sample:

```
.....................................................
Match 1: Word word 01x 0123- variable_word (other words)
Group 1: Word word 01x
Group 2: 0
Group 3: 123- variable_word (other words)
```

Where:

\1 = Word word 01x <space> \2 = 7 \3 = <space> - variable_word (other words)

Tim Notes:

1. By running the exact same RegEx as in the first run, it corrects for the inadvertent zero.

Replace: \1\3

New Name = Word word 01x 123 - variable_word (other words).txt

Add

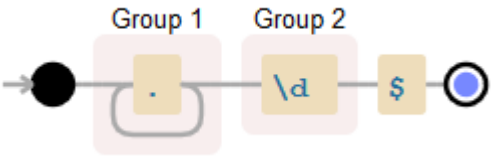
Add space before numeric at end of filename, ignore those that already have

Variable Number of Words Filename1 to Variable Number of Words Filename 1
Variable Number of Words Filename 2 to Ignore

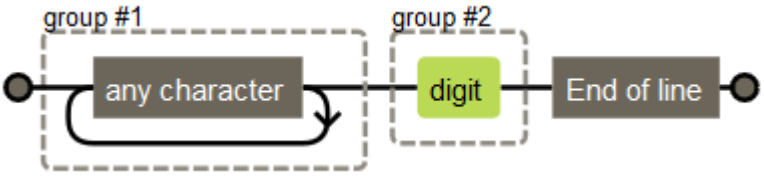
Match: (.+)(\d)$
Replace: \1 \2

Name	New Name
Variable Number of Words Filename1	Variable Number of Words Filename 1
Variable Number of Words Filename 2	Variable Number of Words Filename 2

Sample:




```
Match 1: Variable Number of Words Filename1
Group 1: Variable Number of Words Filename
Group 2: 1
```



Using: Variable Number of Words Filename1

Where:

\1 = Variable Number of Words Filename \2 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = V
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \d	Match against Numeric Digit	Capture Group 2 = 1 Changes Capture Group 1 = Variable Number of Words Filename
4. $	Must be at EOL (End of Line)	

Replace: \1 \2

New Name = Variable Number of Words Filename 1

Add

Add space before numeric at end of filename, ignore those that already have cont.

Using: Variable Number of Words Filename 2

Match: (.+)(\d)$

Where:

\1 = Variable Number of Words Filename <space> \2 = 2

Tim Analysis:

1..	Match against any character	Capture Group 1 = V
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \d	Match against Numeric Digit	Capture Group 2 = 1
		Changes Capture Group 1 = Variable Number of Words Filename <space>
4. $	Must be at EOL (End of Line)	

Replace: \1 \2

New Name = Variable Number of Words Filename 2

Excess <space> created

This filename got inadvertently renamed and now needs to be corrected.

Tim Notes:

1. A filename that already had the <space> was inadvertently altered – has two <spaces> now before the '2'.
2. This RegEx was from the Poster Not the Contributor, which I normally don't do. However, the Poster in this case (along with the Contributor) came up with the solution, so both need to be included.
3. I included this to show the progression to the correction.

To correct, run this –

Match: (.+) (\d)$
Replace: \1\2

Tim Notes:

1. Removes the excess <space> using the <space> (Not Captured) after (.+) … (.+) <space>(\d)$

New Name = Variable Number of Words Filename 2

Add

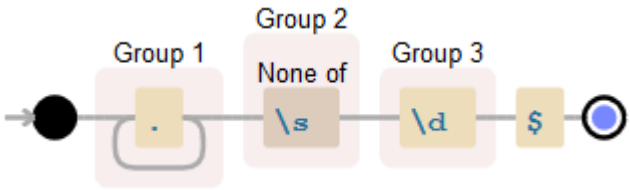
Add space before numeric at end of filename, ignore those that already have cont.

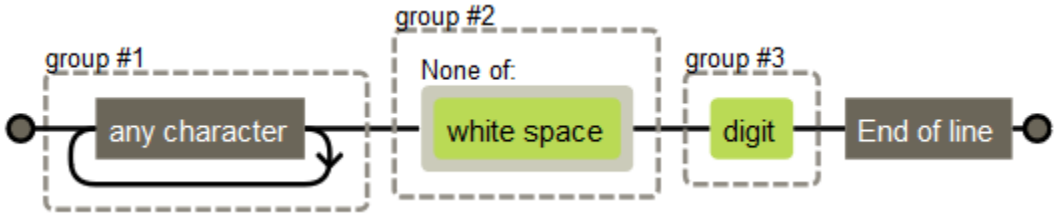
Variable Number of Words Filename1	to	Variable Number of Words Filename 1
Variable Number of Words Filename 2	to	Ignore

This will appropriately ignore the filename that already has the existing <space>

Match: (.+)([^\s])(\d)$
Replace: \1\2 \3

Name	New Name
Variable Number of Words Filename1	Variable Number of Words Filename 1
Variable Number of Words Filename 2	Variable Number of Words Filename 2





Sample:

```
.....................................................
Match 1: Variable Number of Words Filename1
Group 1: Variable Number of Words Filenam
Group 2: e
Group 3: 1
```

Contributor Analysis:

1. ([^\s]) Any non-space character that immediately precedes the numeric digit. Matches a string that doesn't end with a <space>…
2. (\d)$ .. and is immediately followed by a numeric digit at the end of the filename (End of Line or EOL).

Add

Add space before numeric at end of filename, ignore those that already have cont.

Using: Variable Number of Words Filename1

This will appropriately ignore the filename that already has the existing <space> cont.

Match: (.+)([^\s])(\d)$

Where:

\1 = Variable Number of Words Filenam \2 = e \3 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = V
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. [^\s]	Match against a Negated Class consisting of \s <space> Metacharacter.	Capture Group 2 = 1 Changes Capture Group 1 = Variable Number of Words Filename
4. \d	Match against Numeric Digit	Capture Group 3 = 1 Changes Capture Group 2 = e Changes Capture Group 1 = Variable Number of Words Filenam
5. $	Must be at EOL (End of Line)	

Replace: \1\2 \3

New Name = Variable Number of Words Filename 1

Add

Add space before numeric at end of filename, ignore those that already have cont.

Variation:

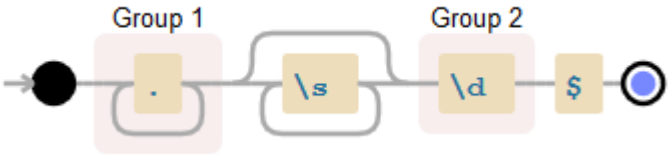
You can also use this RegEx. It has the advantage of replacing zero or more <spaces> by just 1 <space>.
i.e., Files that already have the <space> are ignored no matter how many <spaces> are between "Filename" and "1"

Variable Number of Words Filename 1 Ignore

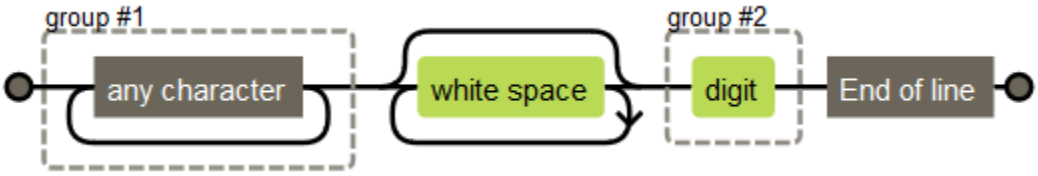
Match: (.+?)\s*(\d)$
Replace: \1 \2

Name	New Name
Variable Number of Words Filename1	Variable Number of Words Filename 1
Variable Number of Words Filename 2	Variable Number of Words Filename 2

Sample:




```
Match 1: Variable Number of Words Filename1
Group 1: Variable Number of Words Filename
Group 2: 1
```



Contributor Analysis:

1. ".+?" Non-greedy match that allows "\s*" to match as many trailing <spaces> as possible before the numeric digit.

Tim Notes:

Remember that inadvertently renamed file that ended up with the extra <spaces> from the first RegEx in this section? It can be restored using this as well. Just substitute the ' + ' Greedy Quantifier to match One or More Times for the ' * ' Greedy Quantifier that matches Zero or More Times and use a Replacement String of \1\3.

Match: (.+?)\s+(\d)$
Replace: \1\3

Name	New Name
Variable Number of Words Filename 2.txt	Variable Number of Words Filename 2.txt

Forum Contributors: CharlesFromVA, TheRegExpMaster

Add

Add space before numeric at end of filename, ignore those that already have cont.

Variation: cont.

Using: Variable Number of Words Filename1

Match: (.+?)\s*(\d)$

Where:

\1 = Variable Number of Words Filename \2 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = V
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = V
4. \s	Match against a <space> character (Not Captured) This is the <space> after 'Filename'	Changes Capture Group 1 = Variable
5. *	Make it Greedy (Not Captured)	Changes Capture Group 1 = V
6. \d	Match against Numeric Digit	Capture Group 2 = 1 Changes Capture Group 1 = Variable Number of Words Filename
7. $	Must be at EOL (End of Line)	

Add

Add 0 to Single numeric digit in string (position does not matter)

fdgfd-10-5-2014-gfd.txt to fdgfd-10-05-2014-gfd.txt
fdgfd-1-5-2014-gfd.txt to fdgfd-01-05-2014-gfd.txt

Tim Notes:

1. Here is an interesting one. Simple but affective. Instead of relying on a limited pattern, it performs one function only – it finds a single numeric digit and prefixes a zero. That's it. It is meant to be run multiple times if there is more than one single numeric digit. Run it as many times as required for the desired results. The position does not matter, it will match from left to right on each run until the string is exhausted (no more single numeric digits found).

2. This is an append rather than a pad although the two terms can get confusing. Padding is about equality and appending is just prefixing or suffixing. Many examples in this book display both outcomes and at times, become interchangeable.

You will have to repeat it as many times as necessary if the filename has many single numeric digits to replace.

i.e.

fdgfd-10-5-2014-gfd.txt to fdgfd-10-05-2014-gfd.txt (one run only needed)
fdgfd-1-5-2014-gfd.txt to fdgfd-1-05-2014-gfd.txt" (first run)
fdgfd-01-05-2014-gfd.txt (second run)

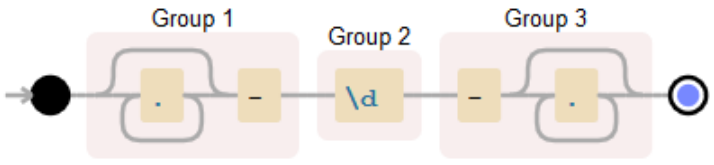
Match: (.*-)(\d)(-.*)
Replace: \10\2\3

First run:

Name	New Name
fdgfd-10-5-2014-gfd.txt	fdgfd-10-05-2014-gfd.txt
fdgfd-1-5-2014-gfd.txt	fdgfd-1-05-2014-gfd.txt

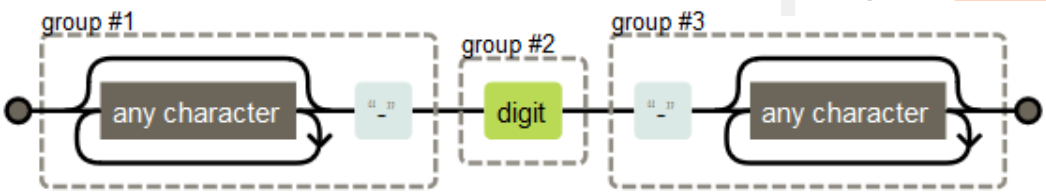
Second run:

Name	New Name
fdgfd-10-05-2014-gfd.txt	fdgfd-10-05-2014-gfd.txt
fdgfd-1-05-2014-gfd.txt	fdgfd-01-05-2014-gfd.txt



Sample:

```
Match 1: fdgfd-10-5-2014-gfd
Group 1: fdgfd-10-
Group 2: 5
Group 3: -2014-gfd
```



Forum Contributor: TheRegExpMaster

Add

Add 0 to Single numeric digit in string (position does not matter) cont.

Using: fdgfd-10-5-2014-gfd.txt

Match: (.*-)(\d)(-.*)

Where:

\1 = fdgfd-10- \2 = 5 \3 = -2014-gfd

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. -	Match against <hyphen>	Capture Group 1 = fdgfd-10-5-2014-
4. \d	Match against Numeric Digit	Capture Group 2 = 2 Changes Capture Group 1= fdgfd-10-5-
5. -	Match against <hyphen> This is the <hyphen> after '5'	Capture Group 3 = - Changes Capture Group 2 = 5 Changes Capture Group 1 = fdgfd-10-
6. .*	Match against remaining string	Capture Group 3 = -2014-gfd

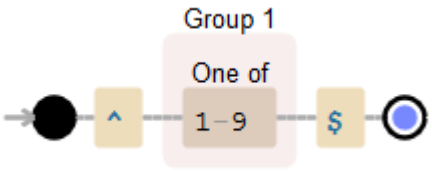
Add

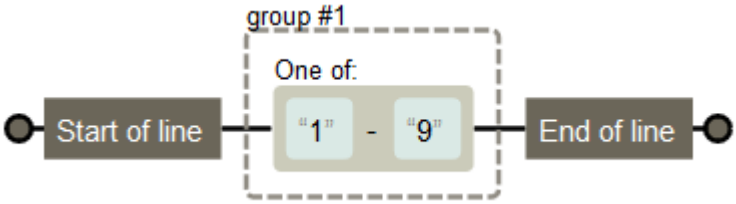
Prefix 0 if the Number is Between 1-9

1.jpg to 01.jpg
2.jpg to 02.jpg
33.jpg to Ignore

Match: ^([1-9])$
Replace: 0\1

Name	New Name
1.jpg	01.jpg
2.jpg	02.jpg
33.jpg	33.jpg





Sample:

```
............
Match 1: 1
Group 1: 1
```

Forum Contributor: truth

Using: 1

Where:

\1 = 1

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [1-9]	Match against a class consisting of a numeric digit '1' through '9'	Capture Group 1 = 1
3. $	Must be at EOL (End of Line)	

Add

Add Quotation signs to String

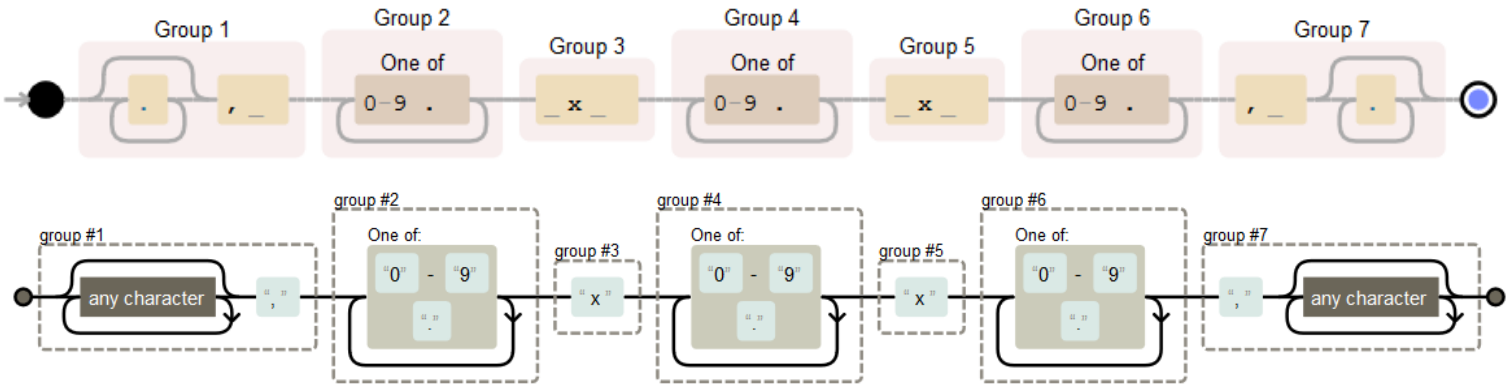
Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5 x 1.5 x 11, JU00500.png
to
Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5" x 1.5" x 11", JU00500.png

Observations:

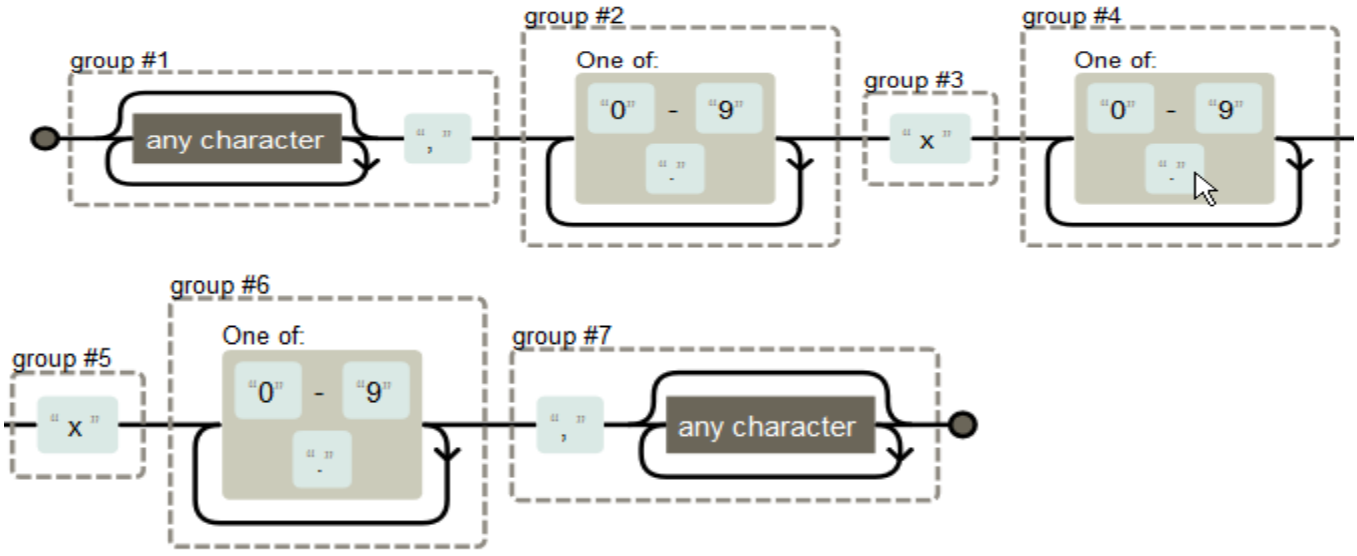
1. Add a " after numbers that are separated by an x to indicate dimensions.
2. Changed to single apostrophes in the Replace String, ‘ because “ (double quotation marks) are an illegal character for filenames.

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)
Replace: \1\2"\3\4"\5\6"\7

Name	New Name
Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0....	jungle toys, blocks, mini squares, 100 pieces, drilled, 0.5" x 1.5" x 11", ju00500.png



Expanded:



Add

Add Quotation signs to String cont.

Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5 x 1.5 x 11, JU00500.png

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)

Sample:

```
.....................................................................................
Match 1: Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5 x 1.5 x 11, JU00500
Group 1: Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled,
Group 2: 0.5
Group 3:  x
Group 4: 1.5
Group 5:  x
Group 6: 11
Group 7: , JU00500
```

Tim Notes:

1. I wouldn't include a post like this normally because it is too specific to one task, however it does demonstrate how to use single quotation marks (the apostrophes) instead of the double quotation marks not allowed by Windows.

2. The Replace string originally used " - I changed this to two single quotation marks to mimic the look and it displayed just fine --- ' ' together = "

Forum Contributor: truth, CharlesFromVA

Add

Add Quotation signs to String cont.

Using: Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5 x 1.5 x 11, JU00500.png

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)

Breaks down:

Capture Group 1

(.*,)

Capture Group 2 Capture Group 3 Capture Group 4 Capture Group 5 Capture Group 6

([0-9\.]+) (x) ([0-9\.]+) (x) ([0-9\.]+)

Capture Group 7

(, .*)

As usual – mostly repetition. Nothing to be intimidated.

Break down further:

1
(.*,)
↓
Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled,<space>

2 3 4 5 6 7
([0-9\.]+) (x) ([0-9\.]+) (x) ([0-9\.]+) (, .*)
↓ ↓ ↓ ↓ ↓ ↓
0.5 <space> x <space> 1.5 <space> x <space> 11 , JU00500

Tim Notes:

1. One of the things I like about Regex Buddy is that it clearly shows <spaces> and other characters that might otherwise be missed in just looking at the RegEx. Doesn't help with missed characters from the Replacement String, though.

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)

2. If you ever have a problem with an example from one of my books not working even though I have it verified that it does, this could be one of the reasons. I encourage you to copy and paste any examples into BRU directly but if you want to see the RegEx clearly, then use a program like Regex Buddy.

Add

Add Quotation signs to String cont.

Using: Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5 x 1.5 x 11, JU00500.png

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)

Where:

\1 = Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled,<space>

\2 = 0.5 \3 = <space> x <space> \4 = 1.5 \5 = <space> x <space>
\6 = 11 \7 = , JU00500

Tim Analysis:

1..	Match against any character	Capture Group 1 = J
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ,	Match against <comma> followed by… Backtracks.	Capture Group 1 = Jungle Toys, Blocks, Mini Squares, 100
4. <space>	.. Match against a <space> character	<space> Pieces, Drilled, 0.5 x 1.5 x 11, Capture Group 1 = Jungle Toys, Blocks, Mini Squares, 100 <space> Pieces, Drilled, 0.5 x 1.5 x 11, <space>

Begin Repetition –

5. [0-9\.]	Match against a class consisting of a. a numeric digit '0' through '9' b. a literal <dot> Matches the '0' of '0.5'	Capture Group 2 = 0 Changes Capture Group 1 =0 Jungle Toys, Blocks, Mini Squares, 100 <space> Pieces, Drilled, <space>
6. +	Make it Greedy	Capture Group 2 = 0.5
7. <space>	Match against a <space> character followed by… This is the <space> after '0.5'	Capture Group 3 = <space>
8. x	.. Match against a string 'x' followed by … This is the 'x' after '0.5<space>'	Capture Group 3 = <space> x
9. <space>	.. Match against <space> character, followed by … This is the last <space> of '0.5 <space> x <space>'	Capture Group 3 = <space> x <space>

Add

Add Quotation signs to String cont.

Using: Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled, 0.5 x 1.5 x 11, JU00500.png

Match: (.*,)([0-9\.]+)(x)([0-9\.]+)(x)([0-9\.]+)(, .*)

Where:

\1 = Jungle Toys, Blocks, Mini Squares, 100 Pieces, Drilled,<space>

\2 = 0.5 \3 = <space> x <space> \4 = 1.5 \5 = <space> x <space>
\6 = 11 \7 = , JU00500

Tim Analysis: cont.

Begin Repetition – cont.

10. [0-9\.]	Match against a class consisting of a. a numeric digit '0' through '9' b. a literal <dot> Matches against the '1' of '1.5'	Capture Group 4 = 1
11. +	Make it Greedy	Capture Group 4 = 1.5
12. <space>	Match against a <space> character followed by… This is the <space> after '1.5'	Capture Group 5 = <space>
13. x	.. Match against a string 'x' followed by … This is the 'x' after '1.5 <space>'	Capture Group 5 = <space> x
14. <space>	.. Match against <space> character, followed by … This is the last <space> of '1.5<space> x <space>'	Capture Group 5 = <space> x <space>
15. [0-9\.]	Match against a class consisting of a. a numeric digit '0' through '9' b. a literal <dot> This is the '1' of '11'	Capture Group 6 = 1
16. +	Make it Greedy	Capture Group 6 = 11

End Repetition.

17. ,	Match against <comma> This is the <comma> after '11'	Capture Group 7 = ,
18. .*	Match against remaining string	Capture Group 7 = , JU00500

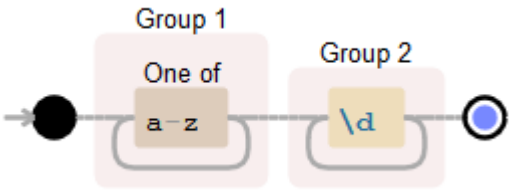
Add

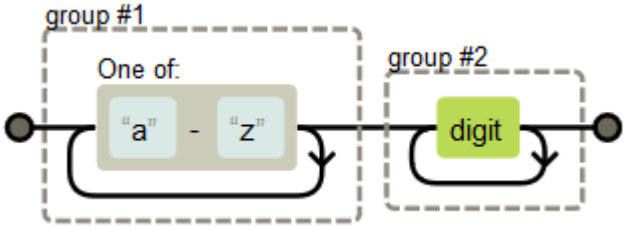
How to add '-' between letters and digits

a1.ext to a-1.ext
abc555.ext to abc-555.ext
ab55555.ext to abc-55555.ext

Match: ([a-z]+)(\d+)
Replace: \1-\2

Name	New Name
abc555.ext	abc-555.ext
a1.ext	a-1.ext
ab55555.ext	ab-55555.ext





Sample:

```
.................
Match 1: abc555
Group 1: abc
Group 2: 555
```

Contributor Analysis:

1. Match against first alpha character
2. Make it Greedy so it captures up until numeric digits
3. Capture first numeric digit
4. Make it Greedy so it captures until EOL (End of Line)

Forum Contributor: Stefan

Add

How to add '-' between letters and digits cont.

Using: abc555.ext

Match: ([a-z]+)(\d+)

Where:

\1 = abc \2 = 555

Tim Analysis:

1. [a-z]	Match against a class consisting of lowercase letter	Capture Group 1 = a
2. +	Make it Greedy	Capture Group 1 = abc
3. \d	Match against Numeric Digit	Capture Group 2 = 5
4. +	Make it Greedy	Capture Group 2 = 555

Add

Pad Zeros to Single Numeric Digit MM, DD in Dates

1-30-2014 to 2014-01-30
1-10-2014 to 2014-01-10
1-2-2014 to 2014-01-02
11-3-2014 to 2014-11-03

Observations:

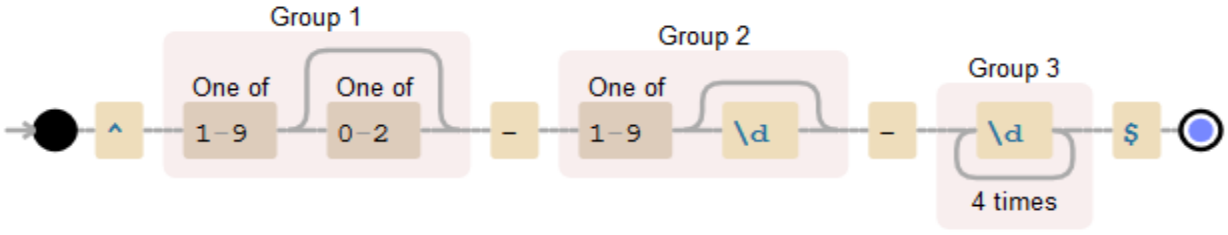
1. The date appears as MONTH-DAY-YEAR
2. Pattern Established.
3. Requires 2 runs.

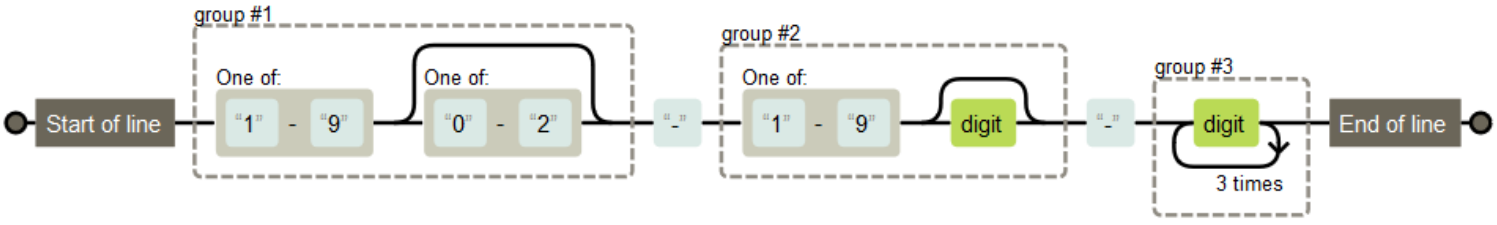
First run:

Match: ^([1-9][0-2]{0,1})-([1-9]\d{0,1})-(\d{4})$
Replace: \3-0\1-0\2

Name	New Name
11-3-2014	2014-011-03
1-2-2014	2014-01-02
1-30-2014	2014-01-030
1-10-2014	2014-01-010

Tim Notes:
Since the maximum length of the pattern is always two (because the pattern is 0-9 single numeric digits with added padded zero = 2 places), prefix one zero to each numeric section, other than the year YYYY and rearrange as specified in the request.





Sample:

```
.....................
Match 1: 1-30-2014
Group 1: 1
Group 2: 30
Group 3: 2014
```

Forum Contributor: truth

Add

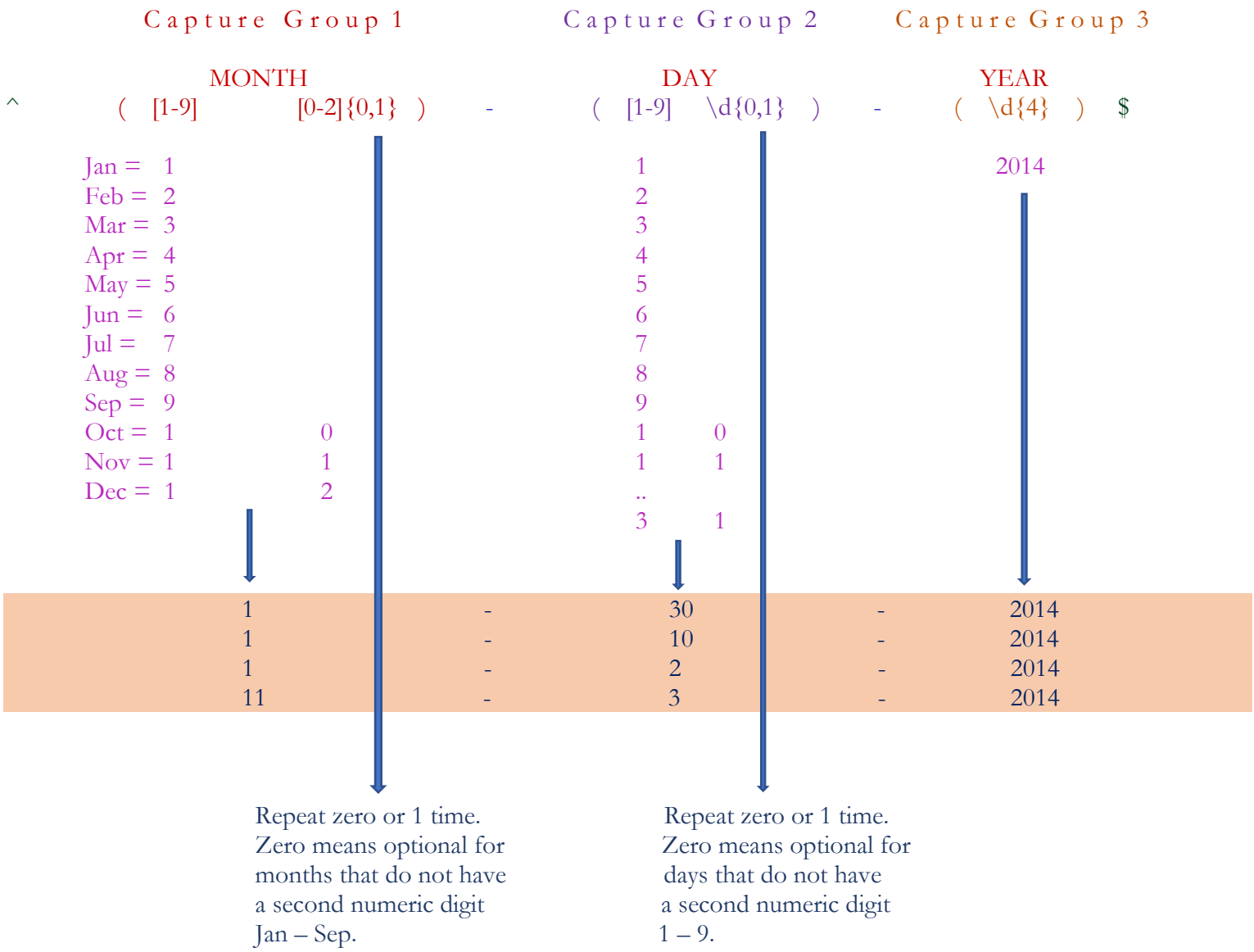
Pad Zeros to Single Numeric Digit MM, DD in Dates cont.

1-30-2014	to	2014-01-30
1-10-2014	to	2014-01-10
1-2-2014	to	2014-01-02
11-3-2014	to	2014-11-03

First run: cont.

Match: ^([1-9][0-2]{0,1})-([1-9]\d{0,1})-(\d{4})$

Breaks down:



Tim Notes:

1. ‘ {0,1} ’ is the equivalent of ‘ ? ’

Add

Pad Zeros to Single Numeric Digit MM, DD in Dates cont.

First run: cont.

Match: ^([1-9][0-2]{0,1})-([1-9]\d{0,1})-(\d{4})$

Tim Analysis:

1. ^ Anchor. Start at beginning of filename (Beginning of Line or BOL)

Capture Group 1

2. [1-9] match against the first single numeric digit of Month, 1-9
   a. covers months beginning with 1 – 9 or 'January' through 'September'
3. [0-2] followed by either a '0', '1' or '2'
   a. for months having second numeric digit – '10', '11', '12'

   Won't match, i.e., '14' because (there is no month 14) of the limitation imposed by [0-2]

4. {0,1} limits the match of [0-2] in step 3 to either 0 or 1 time
   a. To be TRUE, there is or does not have to be a second numeric digit to the 'month'
5. - followed by a <hyphen> (Not Captured)

Capture Group 2

6. [1-9] match against the first single numeric digit of Day, 1-9
   a. covers days beginning with 1 – 9
7. \d Followed by a numeric
8. {0,1} limits the match of \d in step 2 to either 0 or 1 time

   a. To be TRUE, there is or does not have to be a numeric digit to the 'day'
   b. Unlike in the first Capture Group, it does not account for typos, it will match day 35:

9. - followed by a <hyphen> (Not Captured)

Capture Group 3

10. \d{4} Capture last four numeric digits = 'Year'
11. $ Last expression must have matched at EOL
    (must be at End of Line to be TRUE)

Add

Pad Zeros to Single Numeric Digit MM, DD in Dates cont.

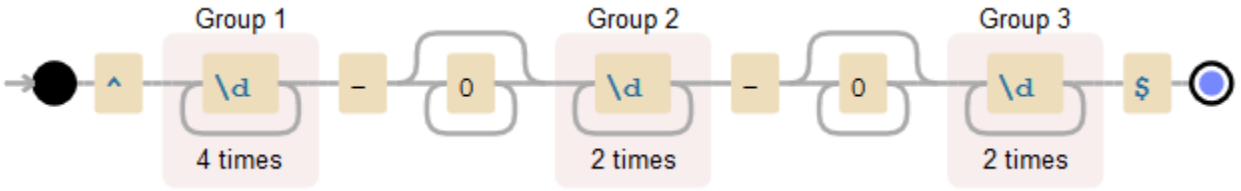
1-30-2014	to	2014-01-30
1-10-2014	to	2014-01-10
1-2-2014	to	2014-01-02
11-3-2014	to	2014-11-03

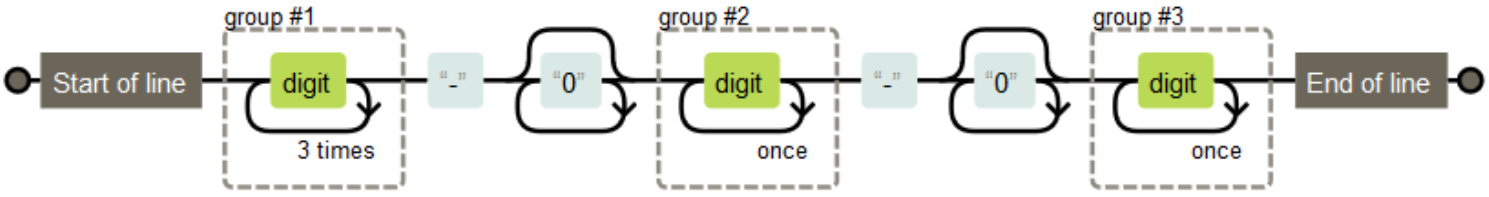
Second run:

Kill unnecessary leading-zeros:

Match: ^(\d{4})-0*(\d{2})-0*(\d{2})$
Replace: \1-\2-\3

Name	New Name
2014-011-03	2014-11-03
2014-01-02	2014-01-02
2014-01-030	2014-01-30
2014-01-010	2014-01-10





Sample:

```
.......................
Match 1: 2014-011-03
Group 1: 2014
Group 2: 11
Group 3: 03
```

Forum Contributor: truth

Add

Pad Zeros to Single Numeric Digit MM, DD in Dates cont.

Second run: cont.

Match: ^(\d{4})-0*(\d{2})-0*(\d{2})$

Tim Analysis:

1. ^ Anchor. Start at beginning of filename (Beginning of Line or BOL)

Capture Group 1

2. \d{4} Match against YYYY. Remember that after rearranging in first run, the year has been moved to the beginning of the filename followed by the month and day.

Not Captured

3. -0 Followed by the <hyphen> and a leading zero (added in the first run)
4. * Make it Greedy.

Capture Group 2

5. \d{2} Capture next two numeric digits.

Without the Greedy Quantifier in step 4, it would only match '011' in '2014-011-03' because it is the only sample that has the pattern of '0' + 2 numeric digits. The other filenames, for example, '2014-01-02', have a pattern of '0' + 1 numeric digit. By making it Greedy, it will include any numeric digits after the zero, regardless.

This retains two numeric digits of the Month while eliminating any months with three or more numeric digits (the padded zero).

Not Captured

6. -0* Same as in Step 3 and 4.

Capture Group 3

7. \d{2} Same as in step 5
This retains two numeric digits of the Day while eliminating any days with three or more numeric digits (the padded zero).

8. $ Last expression must have matched at EOL
(must be at End of Line to be TRUE)

Add

Pad Zeros to Single Numeric Digit MM, DD in Dates cont.

Tim Analysis: cont.

Why doesn't it affect '2014-01-02' ? After the first run, it does not need to be corrected in the second run but there isn't anything in the RegEx above that would preclude this.

It is affected.

If you change the Replacement string to \1\2\3

Replace: \1\2\3

Name	New Name
2014-011-03	20141103
2014-01-02	20140102
2014-01-030	20140130
2014-01-010	20140110

You can see the filename is definitely affected by the RegEx. So it is the Replacement string that holds the key to excluding the filename.

Remember, that there will be no name change if the resulting filename is the same as the original filename. By adding the <hyphens> in the replacement string, you can see this:

Replace: \1-\2\3

Name	New Name
2014-011-03	2014-1103
2014-01-02	2014-0102
2014-01-030	2014-0130
2014-01-010	2014-0110

Replace: \1-\2-\3

Name	New Name
2014-011-03	2014-11-03
2014-01-02	2014-01-02
2014-01-030	2014-01-30
2014-01-010	2014-01-10

There you have it. Adding the <hyphen> after backreference 2 results in a filename of 2014-01-02, the same name as the original filename, and thus, no new name is required – the filename is ignored.

Add

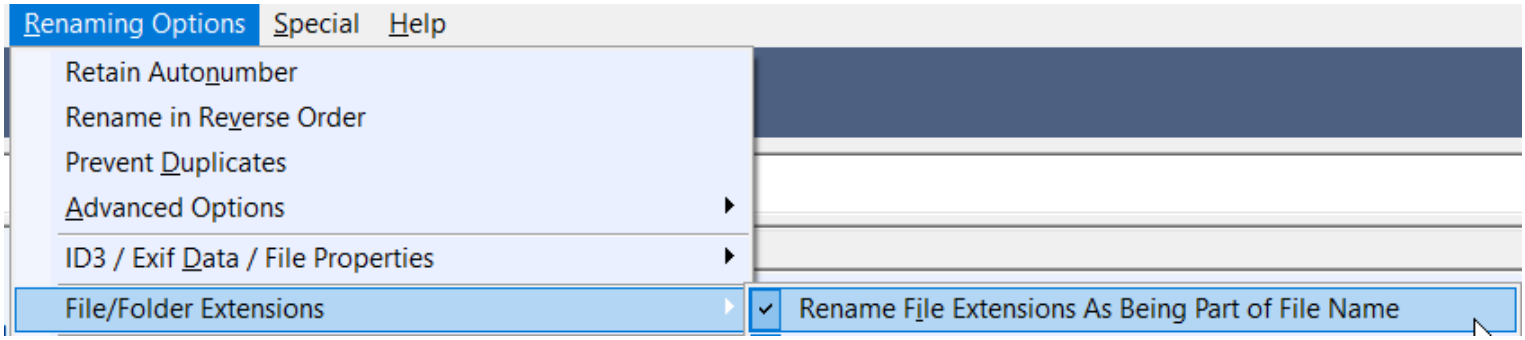
Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!)

Observations:

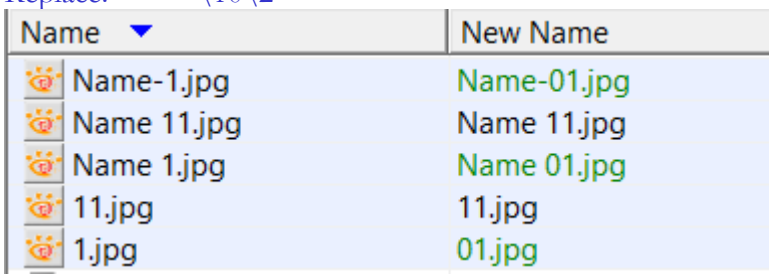
1. Pad zero to single numeric digit at end of filename

1.jpg	to	01.jpg
11.jpg	to	Ignore
Name 1.jpg	to	Name 01.jpg
Name-1.jpg	to	Name-01.jpg
Name 11.jpg	to	Ignore
Name Name 1.jpg	to	Name Name 01.jpg

Set 'Rename File Extensions As Being part of File Name' in Renaming Options Menu:



Match: ^(.*[^0-9,\w])*(\d.jpg)$
Replace: \10\2



Tim Notes:

1. It accomplishes the padding (append) operation in a single run. There are a few more examples in the book that use similar techniques.

2. This is one of those examples where the result is really an append rather than a pad.

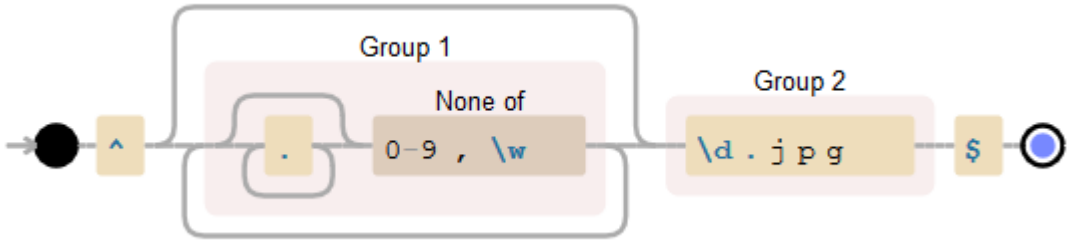
Forum Contributor: truth

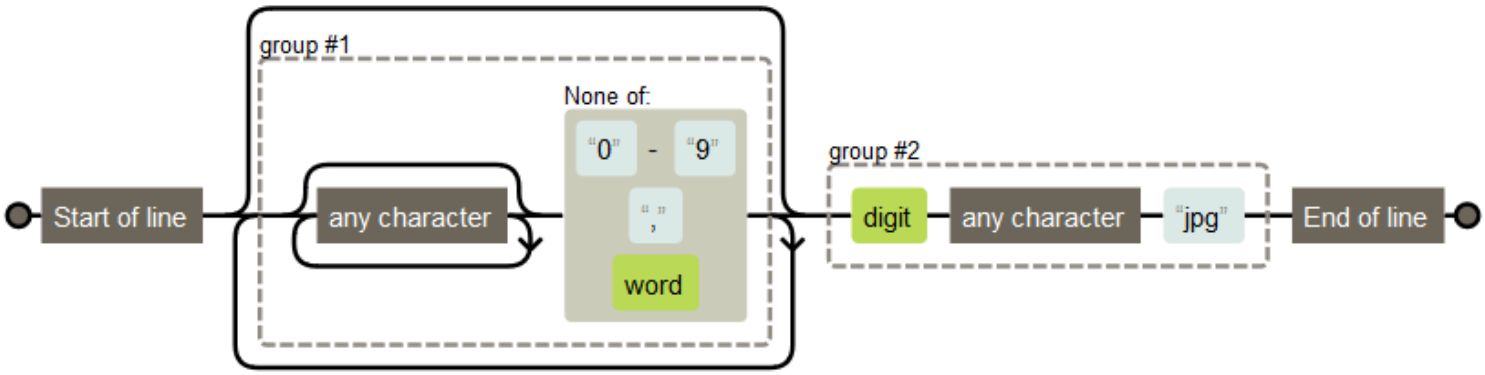
Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

1.jpg	to	01.jpg
11.jpg	to	Ignore
Name 1.jpg	to	Name 01.jpg
Name-1.jpg	to	Name-01.jpg
Name 11.jpg	to	Ignore
Name Name 1.jpg	to	Name Name 01.jpg

Match: ^(.*[^0-9,\w])*(\d.jpg)$





Sample:

```
.....................
Match 1: Name 1.jpg
Group 1: Name
Group 2: 1.jpg
```

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution –

Match: ^(.*[^0-9,\w])*(\d.jpg)$

Where:

Name 1.jpg	Name -1.jpg	1.jpg
\1 = Name	\1 = Name-	\1 = null
\2 = 1.jpg	\2 = 1.jpg	\2 = null

Tim Analysis:

1. ^ Start at beginning of filename (Beginning of Line or BOL)

Capture Group 1

2. .* Capture characters Greedy..
3. [^0-9,\w] Do not include 0-9 or a word character

4. * Make it Greedy

Capture Group 2

5. \d Match against numeric digit
6. .jpg Match against '.jpg'.
   a. Remember, that the extension is being treated as part of the filename.

7. $ Should be at EOL (End of Line) = TRUE

Here is a much broader sample.

Name	New Name
Name-1.jpg	Name-01.jpg
Name Name 1.jpg	Name Name 01.jpg
Name 4 Name Name 55.jp...	Name 4 Name Name 55.jpg
Name 11.jpg	Name 11.jpg
Name 11 Name 3.jpg	Name 11 Name 03.jpg
Name 1.jpg	Name 01.jpg
Name 1 Name 2.jpg	Name 1 Name 02.jpg
11.jpg	11.jpg
1.jpg	01.jpg
1 Name Name 5.jpg	1 Name Name 5.jpg

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

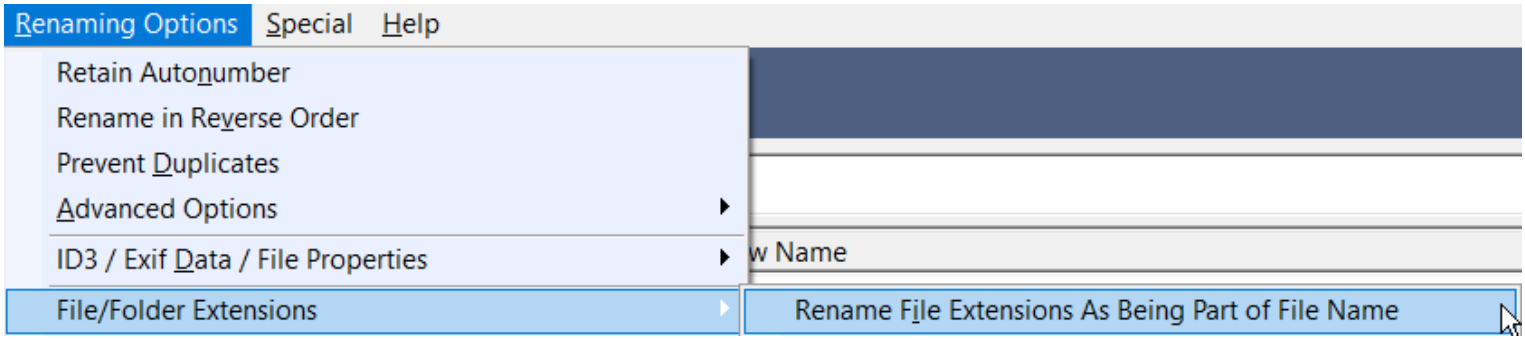
Tim Contribution – cont.

Observations:

1. It only affects those filenames that begin with text (excludes, '1 Name Name 5.jpg'). [^0-9]
2. Only changes numeric digits at end of filename not those within.(\d.jpg)$

So it got me thinking and here is the closest I can get to padding/appending multiple single numeric digits in a single run. By doing so, it crosses the line between padding and appending – so be it.

You don't need to have the extension treated as part of the filename so this can be turned off:



This is what I came up with. These RegEx are very limited in scope. They are meant for filenames that contain SINGLE numeric digits. It can affect other filenames that contain more than single numeric digits so be careful. Perhaps someone can improve upon this and restrict it only to those file patterns so it doesn't inadvertently affect others.

For example, all of these files were affected by my, 'Pad Filenames that contain 3 Single Numeric Digit', RegEx:

Name	New Name
Name 4 Name 5 Name 88.jpg	Name 04 Name 05 Name 08.jpg
Name 4 Name 5 Name 878.jpg	Name 04 Name 05 Name 08.jpg
Name 4 Name 5 Name 878 Name 6.jpg	Name 04 Name 05 Name 878 Name 06.jpg
Name 4 Name 5 Name 8.jpg	Name 04 Name 05 Name 08.jpg

What a mess if you are not careful about the selection of which files get renamed.

You only want to select those that meet the pattern of:

Text <space> # <space> Text <space> # <space> etc.

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution – cont.

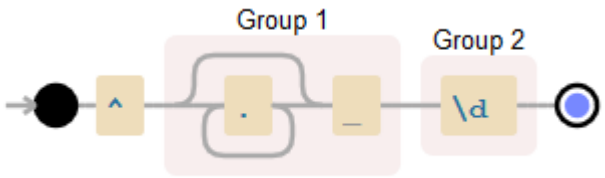
So with all the warnings out of the way, here are my Pad/Append contributions:

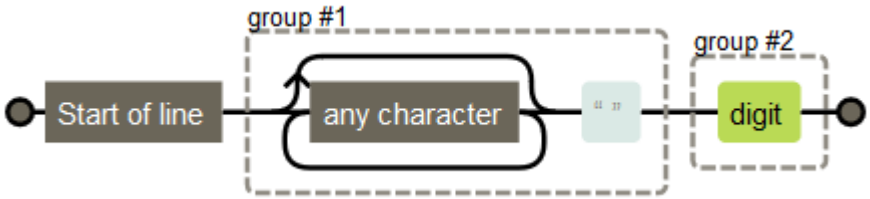
Pad Filenames that contain 1 Single Numeric Digit

Name 1.jpg to Name 01.jpg

Match: ^(.*?)(\d{1})
Replace: \10\2

Name	New Name
Name 1.jpg	Name 01.jpg





Sample:

```
Match 1: Name 1
Group 1: Name
Group 2: 1
```

Tim Notes:

1. The RegEx presented by the Contributor is better than this. This is only included to show the progression of my RegEx from singles to multiples.

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution – cont.

Pad Filenames that contain 1 Single Numeric Digit cont.

Using: Name 1.jpg

Match: ^(.*?)(\d{1})

Where:

\1 = Name <space> \2 = 1

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = N
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. <space>	Match against a <space> character	Capture Group 1 = Name<space>
6. \d	Match against Numeric Digit	Capture Group 2 = 1
7. {1}	Repeat last expression to match 1 iteration.	Capture Group 2 = Unchanged

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

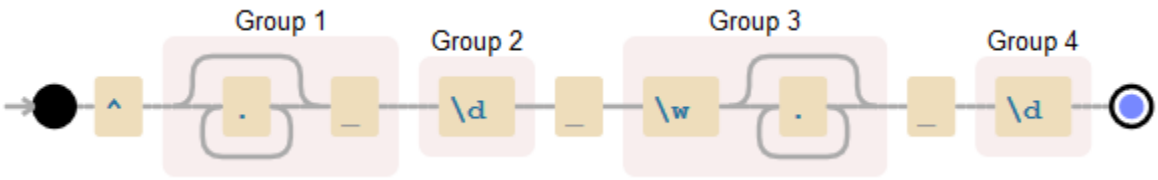
Tim Contribution – cont.

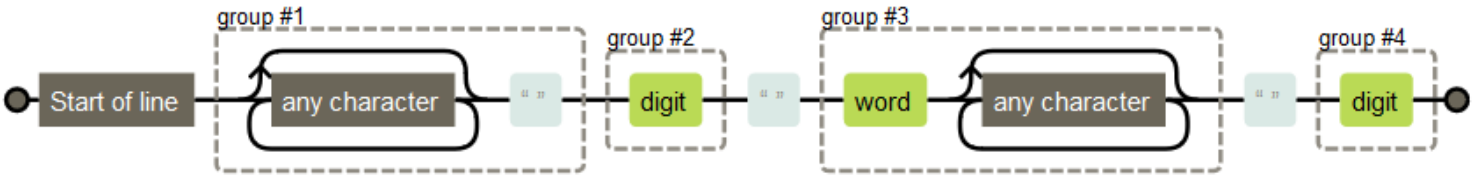
Pad Filenames that contain 2 Single Numeric Digits

Name 3 Name 1.jpg to Name 03 Name 01.jpg

Match: ^(.*?)(\d{1}) (\w.*?) (\d{1})
Replace: \10\2 \3 0\4

Name	New Name
Name 3 Name 1.jpg	Name 03 Name 01 .jpg





Sample:

```
.......................
Match 1: Name 3 Name 1
Group 1: Name
Group 2: 3
Group 3: Name
Group 4: 1
```

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution – cont.

Pad Filenames that contain 2 Single Numeric Digits cont.

Using: Name 3 Name 1.jpg

Match: ^(.*?)(\d{1}) (\w.*?) (\d{1})

Where:

\1 = Name <space> \2 = 3 \3 = Name \4 = 1

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = N
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. <space>	Match against a <space> character	Capture Group 1 = Name<space>
6. \d	Match against Numeric Digit	Capture Group 2 = 3
7. {1}	Repeat last expression to match 1 iteration.	Capture Group 2 = Unchanged
8. <space>	Match against <space> character (Not Captured) This is the <space> after the '3'	
9. \w	Match against Word Metacharacter	Capture Group 3 = N
10. .	Match against any character	Capture Group 3 = Na
11. *	Make it Greedy (This is the second 'Name')	Capture Group 3 = Name 1
12. ?	But not too Greedy	Capture Group 3 = N
13. <space>	Match against <space> character (Not Captured) This is the <space> after the second 'Name'	Changes Capture Group 3 = Name
14. \d	Match against Numeric Digit	Capture Group 4 = 1
15. {1}	Repeat last expression to match 1 iteration.	Capture Group 4 = Unchanged

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

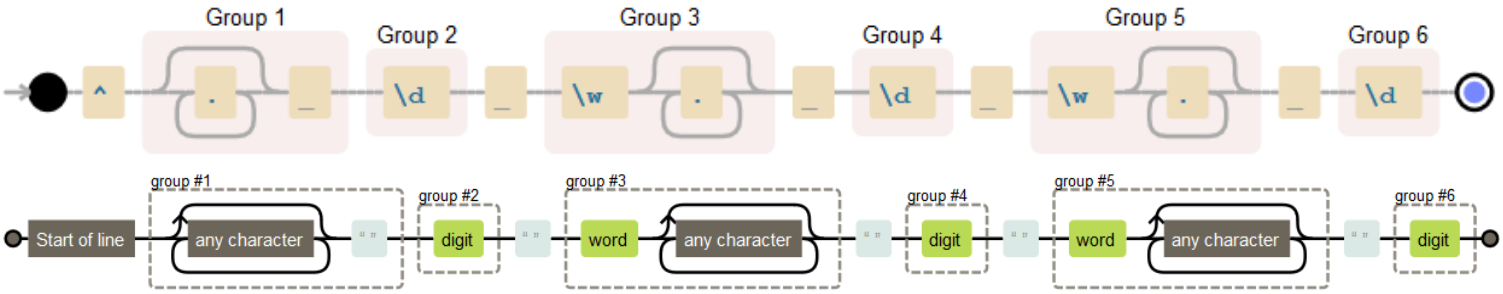
Tim Contribution – cont.

Pad Filenames that contain 3 Single Numeric Digits

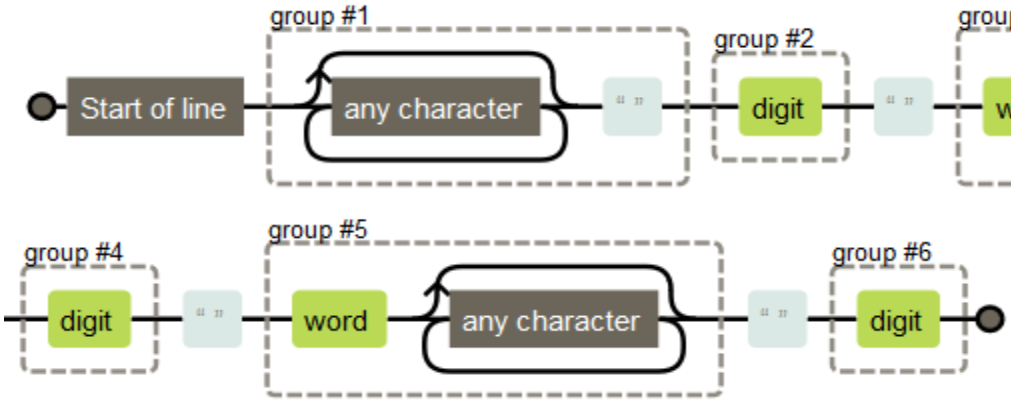
Name 4 Name 5 Name 8.jpg to Name 04 Name 05 Name 08.jpg

Match: ^(.*?)(\d{1}) (\w.*?) (\d{1}) (\w.*?) (\d{1})
Replace: \10\2 \3 0\4 \5 0\6

Name	New Name
Name 4 Name 5 Name 8.jpg	Name 04 Name 05 Name 08.jpg



Expanded:



Sample:

```
.....................................
Match 1: Name 4 Name 5 Name 8
Group 1: Name
Group 2: 4
Group 3: Name
Group 4: 5
Group 5: Name
Group 6: 8
```

For each additional numeric digit add …

(\w.*?) (\d{1})

.. to end of RegEx

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution – cont.

Pad Filenames that contain 3 Single Numeric Digits cont.

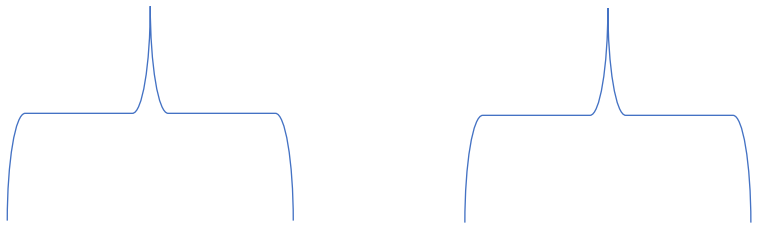
Using: Name 4 Name 5 Name 8.jpg

Match: ^(.*?)(\d{1}) (\w.*?) (\d{1}) (\w.*?) (\d{1})

Where:

\1 = Name <space> \2 = 4 \3 = Name \4 = 5 \5 = Name \6 = 8

Breaks down:



^	(.*? <space>)	(\d{1}) <space>	(\w.*?) <space>	(\d{1}) <space>	(\w.*?) <space>	(\d{1})
	1	2	3	4	5	6
	Name <space>	4	Name	5	Name	8

See the Repetition?

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution – cont.

Pad Filenames that contain 3 Single Numeric Digits cont.

Using: Name 4 Name 5 Name 8.jpg

Match: ^(.*?)(\d{1}) (\w.*?) (\d{1}) (\w.*?) (\d{1})

Where:

\1 = Name <space> \2 = 4 \3 = Name \4 = 5 \5 = Name \6 = 8

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = N
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. <space>	Match against a <space> character This is the first 'Name' in the string.	Capture Group 1 = Name<space>

Begin Repetition –

6. \d	Match against Numeric Digit	Capture Group 2 = 4
7. {1}	Repeat last expression to match 1 iteration.	Capture Group 2 = Unchanged
8. <space>	Match against <space> character (Not Captured) This is the <space> after the '4'	
9. \w	Match against Word Metacharacter	Capture Group 3 = N
10. .	Match against any character	Capture Group 3 = Na
11. *	Make it Greedy	Capture Group 3 = Name 5 Name 8
12. ?	But not too Greedy	Capture Group 3 = N
13. <space>	Match against <space> character (Not Captured) This is the <space> after the second 'Name' Capture Group 3 = second 'Name' in string	Changes Capture Group 3 = Name

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

Tim Contribution – cont.

Pad Filenames that contain 3 Single Numeric Digits cont.

Using: Name 4 Name 5 Name 8.jpg

Match: ^(.*?)(\d{1}) (\w.*?) (\d{1}) (\w.*?) (\d{1})

Where:

\1 = Name <space> \2 = 4 \3 = Name \4 = 5 \5 = Name \6 = 8

Tim Analysis:

Begin Repetition – cont.

14. \d	Match against Numeric Digit	Capture Group 4 = 5
15. {1}	Repeat last expression to match 1 iteration.	Capture Group 4 = Unchanged
13. <space>	Match against <space> character (Not Captured) This is the <space> after the '5'	
14. \w	Match against Word Metacharacter	Capture Group 5 = N
15. .	Match against any character	Capture Group 5 = Na
16. *	Make it Greedy	Capture Group 5 = Name 8
17. ?	But not too Greedy	Capture Group 5 = N
18. <space>	Match against <space> character (Not Captured) This is the <space> after third 'Name' Capture Group 5 = third 'Name' in string	Changes Capture Group 5 = Name
19. \d	Match against Numeric Digit	Capture Group 6 = 8
20. {1}	Repeat last expression to match 1 iteration.	Capture Group 6 = Unchanged

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 digits in a string (and pad/append up to 16).

This is just a sampling of the test strings used.

Name-1.jpg	Name 1.jpg
Name Name 1.jpg	Name 1 Name 2.jpg
Name 4 Name Name 55.jpg	11.jpg
Name 11.jpg	1.jpg
Name 11 Name 3.jpg	1 Name Name 5.jpg

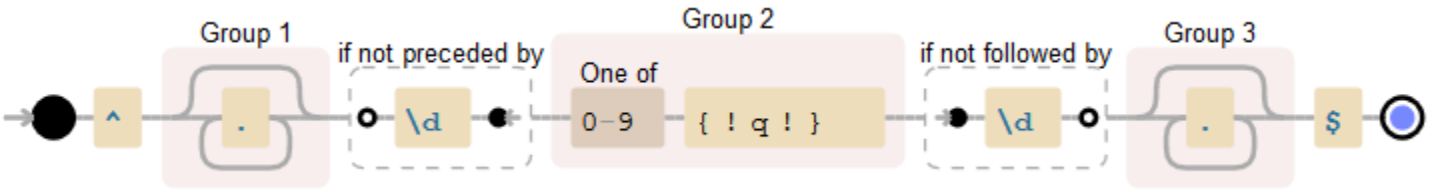
Observations:

1. Remove /Execute to preview renames, & edit line2 with a folder-path:

Batch file:

```
@echo off
:: cd/d "C:\YourDirPath\"
(Set/p pd=Specify any padding from 2-16:_) & (Set/a pd=pd) & (Set q=2)
(If %pd% lss 2 exit) & (If %pd% gtr 16 exit)
SetLocal EnableDelayedExpansion &&  For /L %%A IN (2,1,%pd%) DO (Set/a q="%%A-1")&&(Set
pad%%A=/RegExp:"^^(.*?)(?<^!\d)([0-9]{!q!})(?^!\d)(.*)$:\10\2\3")
SetLocal DisableDelayedExpansion && For /f "delims== tokens=1,2" %%A IN ('set pad') DO Set most%%A=%%B
%%B %%B %%B %%B %%B %%B %%B %%B %%B
SetLocal EnableDelayedExpansion && (For /f "delims==" %%A IN ('set mostpad^|sort /+9') DO Set ord=!ord!
%%%%%A%%) && SetLocal DisableDelayedExpansion
For /F "delims=" %%A IN ('echo %ord%') DO Set regex=%%A
brc32 /Pattern:* /NoFolders %regex%
pause>nul
```

Forum Contributor: RegexNinja



Tim Notes:

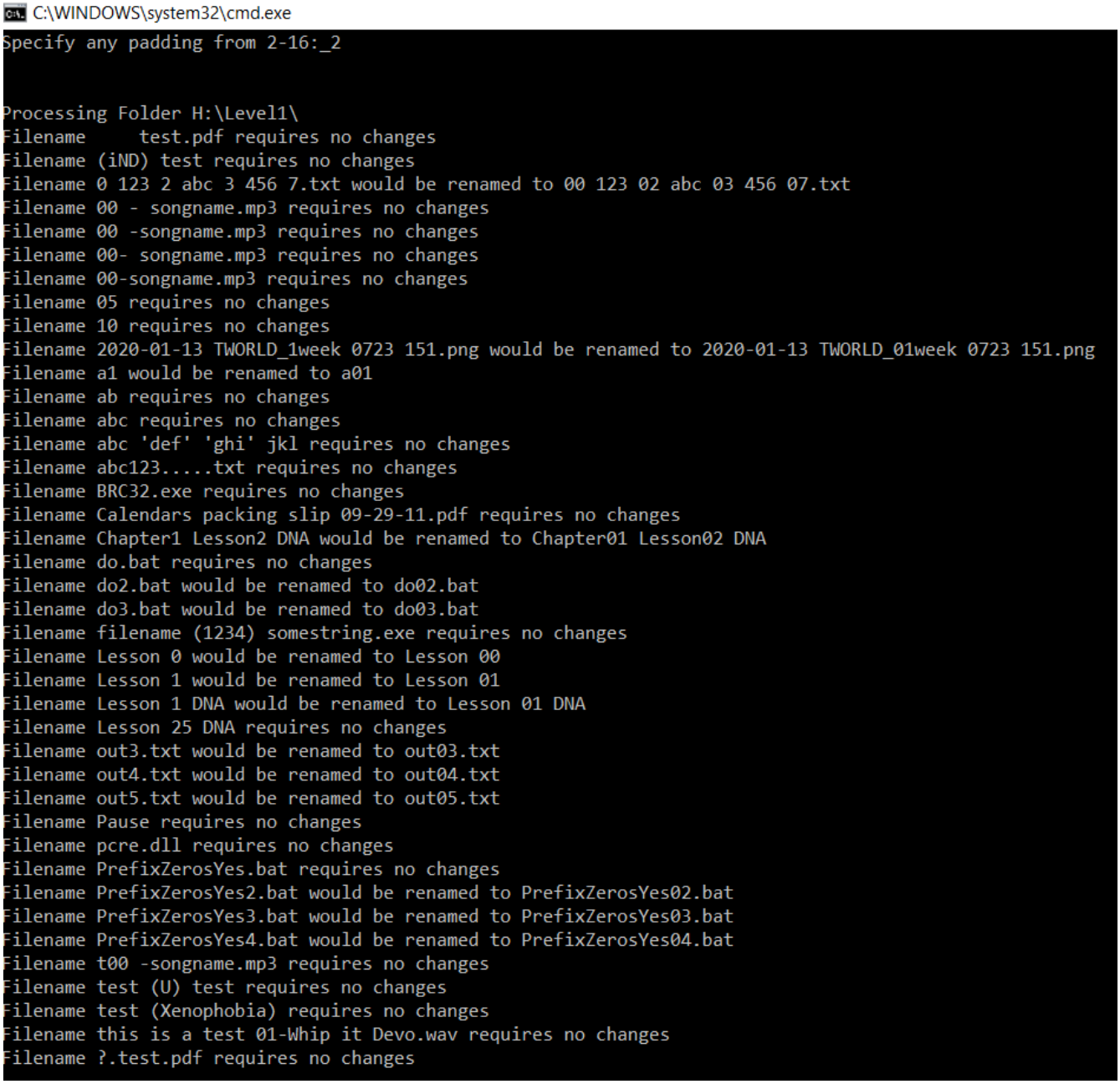
1. With DelayedExpansion, the ' ! ' becomes a special-character, since its used to expand ' ! <variables> ! '
2. To specify a literal ' ! ' <exclamation point >, you must use ' ^! ' Same thing with a literal ' ^ ' requiring ' ^^ '
   a. ' %% ' is used to specify a literal ' % ' in batch programs.
3. the ' !q! ' in Capture Group 2 refers to a variable, ' q ' used as a Quantifier.

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 digits in a string (and pad up to 16). cont.

Sample run with Preview:



Tim Notes:

1. To run, you need to append /Execute to the end of the BRC command –

    brc32 /Pattern:* /NoFolders %regex% /Execute

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 numeric digits in a string (and pad up to 16). cont.

Contributor Analysis:

```
@echo off
cd/d "C:\YourDirPath\"
```

:: Ask user for a pad value.. Set**/a** ensures its **numeric**.. Set q=2 (quantifier)

```
(Set/p pd=Specify any padding from 2-16:_) & (Set/a pd=pd) & (Set q=2)
```

:: If padding doesn't range from 2-16, then exit

```
(If %pd% lss 2 exit) & (If %pd% gtr 16 exit)

SetLocal EnableDelayedExpansion

For /L %%A IN (2,1,%pd%) DO (
                    Set/a q="%%A-1"
                    Set pad%%A=/RegExp:"^^(.*?)(?<^!\d)([0-9]{!q!})(?^!\d)(.*)$:\10\2\3"
                    )
```

:: For /L creates a math-loop. My syntax: For /L %%A IN (**start**,**increment**,**stop**) DO (whatever %%A whatever)
:: During the ForLoop, %%A starts as **2**, increments by **1** per each run, till reaching '**stop**' (the user-entered pad: **%pd%**)
:: The FOR variable %%A is *designed* to change values per each run.

:: Other %vars% *may* be changed, but their new values cant be expanded within a ForLoop, unless using
DelayedExpansion with **!**var**!**

:: DelayedExpansion also makes **!** a special-char, so if you need to spec it literally, the syntax is **^!** (like ^^ in the regex)

:: So for each run in the ForLoop, we have 2 variables: **%%A** **!q!** (quantifier in regex)

:: If user enters **4**: Run1: **2** **1** (%%A**-1**)
:: Run2: **3** **2**
:: Run3: **4** **3**

:: Using these variables, **DO** runs commands like: Set pad**2**=/Regexp:With **{1}** as its quantifier
:: Set pad**3**=/Regexp:With **{2}** as its quantifier
:: and so forth

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 numeric digits in a string (and pad up to 16). cont.

Contributor Analysis: cont.

```
SetLocal DisableDelayedExpansion && For /f "delims== tokens=1,2" %%A IN ('Set pad') DO Set most%%A=%%B %%B %%B %%B %%B %%B %%B %%B %%B %%B
```

```
:: Another ForLoop..... It runs once for every varName starting with pad
:: For every varName beginning as pad, 'Set pad' outputs a TextString like:

::                pad#=/Regexp:With{#}


:: Using = as a separator(delims) & tokens=1,2  The FOR vars:   %%A expands pad#  %%B expands /Regexp:With{#}
:: So DO will run commands like:
::                Run1:      Set mostpad2=/Regexp:With{1} /Regexp:With{1} /Regexp:With{1} etc.
::                Run2:      Set mostpad3=/Regexp:With{2} /Regexp:With{2} /Regexp:With{2} etc.
```

```
SetLocal EnableDelayedExpansion && (For /f "delims==" %%A IN ('Set mostpad^|sort /+9') DO Set ord=!ord! %%%%A%%) && SetLocal DisableDelayedExpansion
```

```
:: Another ForLoop.. The goal: Create the 'ord' variable to hold a simple TextString like:

::                %mostpad2% %mostpad3% %mostpad4% ...

:: For every varName beginning as mostpad, SET outputs a TextString like

::                mostpad#=/Regexp:With{#} /Regexp:With{#} /Regexp:With{#} etc.

:: No need to spec 'tokens=1,#'  FOR matches TextString1 by default, so %%A matches mostpad# when using
:: "delims=="

:: SET by-itself lists pad10 first! So its piped to SORT..  The var 'ord' will define brc's regex-order, so its critical.

:: So for each run, the DO-command:

::                    Set ord=!ord! %%%%A%%                    ord's RESULT (1Space precedes %mostpad2%)
::                    Run1:  Set ord=nul %mostpad2%            %mostpad2%
::                    Run2:  Set ord=%mostpad2% %mostpad3%     %mostpad2% %mostpad3% etc
```

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 numeric digits in a string (and pad up to 16). cont.

Contributor Analysis: cont.

```
For /F "delims=" %%A IN ('echo %ord%') DO Set regex=%%A

:: Not really a Loop (1-run).. We echo/output %ord%, expanding its VALUE, and save the entire-string as regex for brc
:: to process.

:: By default, FOR uses a delims of <space> Or Tab as separators of its text string.. (So %%A would only match until the
:: 1st- <space> in ord)

:: The only way to disable separators is to omit any separator(s) like: "delims="
:: NOTE: By default, the command-shell (cmd.exe) pre-expands all %vars% BEFORE running any command-line
:: (disabled with EnableDelayedExpansion).

:: So echo %ord% actually runs the command: echo %mostpad2% %mostpad3% %mostpad4% etc (outputting the
:: VALUES of each %mostpad#%)
:: Thus outputting: /Regexp:With{1} /Regexp:With{1}... /Regexp:With{2} /Regexp:With{2}... to be saved by the
:: variable regex.

brc32 /Pattern:*.* /NoFolders %regex%
pause>nul
```

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 numeric digits in a string (and pad up to 16). cont.

I am only going to analyze the RegEx.

Using: Name-1.jpg

Match: ^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$
Replace: \10\2\3

Name	New Name
Name-1.jpg	Name-01.jpg

Where:

\1 = Name- \2 = 1

Tim Notes:

1. Quantifier expressed as variable !q! is equal to the number of passes (each run). This is the first (and only) run in this test so I set the Quantifier as '1'.

2. In the batch file this increments with each successive run (in the For Loop).

3. The quantifier is set to match against the number of subsequent numeric digits encountered. The string only contains one single numeric digit so that also requires the Quantifier to be set to '1', otherwise there would be no match in this test.

Sample:

```
·····Start·Length····
Match 1: Name-1     0      6
Group 1: Name-      0      5
Group 2: 1          5      1
Group 3:            6      0
```

```
Beginning match attempt at character 0
 1 ok
 2 ok
 3 ok
 4 ok
 5 backtrack
 6 backtrack
 7 N
 8 N
 9 backtrack
10 ok
11 backtrack
12 backtrack
13 Na
14 Na
15 Nbacktrack
16 Nok
17 Nbacktrack
18 Nbacktrack
19 Nam
20 Nam
21 Nabacktrack
22 Naok
23 Nabacktrack
24 Nabacktrack
25 Name
26 Name
27 Nambacktrack
28 Namok
29 Nambacktrack
30 Nambacktrack
31 Name-
32 Name-
33 Namebacktrack
34 Nameok
35 Name1
36 Name1
37 Name1backtrack
38 Name1ok
39 Name1ok
40 Name1ok
41 Name1ok
Match found in 41 steps
```

Add

Prefix final digit with zero if preceded by word-separator (or null) (Pad In SINGLE Pass!) cont.

This is BRC batch solution that can handle up to 10 numeric digits in a string (and pad up to 16). cont.

I am only going to analyze the RegEx since the Batch file has previously been analyzed.

Using: Name-1.jpg

Match: ^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$

Where:

\1 = Name- \2 = 1 \3 = null

Breaks down:

Capture Group 1		Capture Group 2		Capture Group 3	
Anchor		Negative Lookbehind Non-Marking Group		Negative Lookahead Non-Marking Group	Anchor
^	(.*?)	(?<!\d)	([0-9]{1})	(?!\d)	(.*)	$

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = N
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1= BOL
5. (?<!\d)[0-9]		
	a. (?<!\d)[0-9] Negative Lookbehind. Look to the left that the preceding is not a numeric digit	Changes Capture Group 1 = Name-
	b. [0-9]{1} Match against a class consisting of a numeric digit '0' through '9' repeated '1' time. This determines the length of the numeric portion to match against.	Capture Group 2 = 1
6. (?!\d)	Negative Lookahead. Look to the right that the next character is not a numeric digit. If there was another numeric digit then the length would exceed that specified by the Quantifier and fail.	All Capture Groups = Unchanged
7. .*	Match against remaining string. Already at EOL = null	Capture Group 3 = null
8. $	Must be at EOL (End of Line)	

Add

Add Autonumber in Middle of String (variable placement)

Tim Notes:

1. The Autonumber function of 'Section #10: Numbering', is limited to placement at the beginning, end of the string, both the beginning and end of the string and at a specific position within the string. Here is a technique that can be used to place the autonumber at a variable position within the string.

2. For further information on using Autonumber, please refer to Volume 1 of Bulk Rename Utility Operations Manual. It can be found under 'Section #10: Numbering' that begins on page titled, 'The 14 Sections Used to Specify the Criteria'.

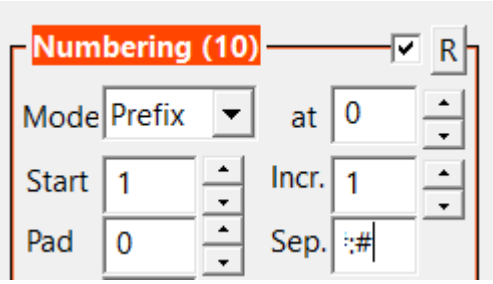
Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3
to
Within Temptation [The Best of Gothic Rock] 1 - Running Up That Hill.mp3

Nightwish [The Best of Gothic Rock] - Bless the Child.mp3
to
Nightwish [The Best of Gothic Rock] 2 - Bless the Child.mp3

This will be accomplished in two separate runs.

First run:

'Section #10: Numbering' –



Set Mode Prefix
Use a Sep of '#:#' (# : #) for clarification. No parentheses please.
Start: 1
Incr.: 1 (auto increment is at 1)

By using #:# we are sure to match the right numeric digit in the second run.

After first run –

Name	New Name
Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3	#1#Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3
Nightwish [The Best of Gothic Rock] - Bless the Child.mp3	#2#Nightwish [The Best of Gothic Rock] - Bless the Child.mp3

In case this wasn't clear in the photo, the new names are:

#1#Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3
#2#Nightwish [The Best of Gothic Rock] - Bless the Child.mp3

Add

Add Autonumber in Middle of String (variable placement) cont.

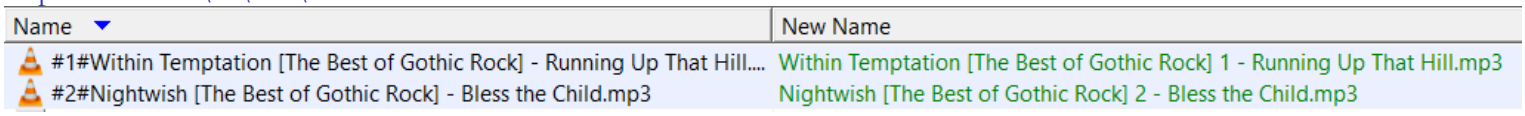
Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3
to
Within Temptation [The Best of Gothic Rock] 1 - Running Up That Hill.mp3

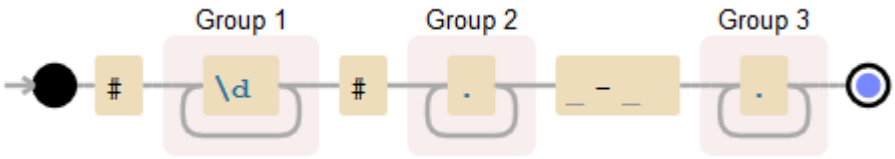
Nightwish [The Best of Gothic Rock] - Bless the Child.mp3
to
Nightwish [The Best of Gothic Rock] 2 - Bless the Child.mp3

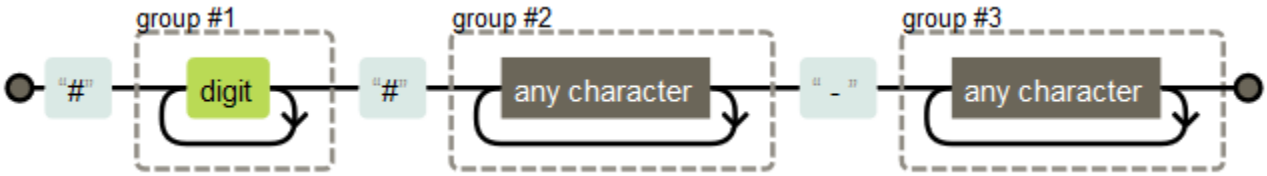
Second run:

Using: #1#Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3

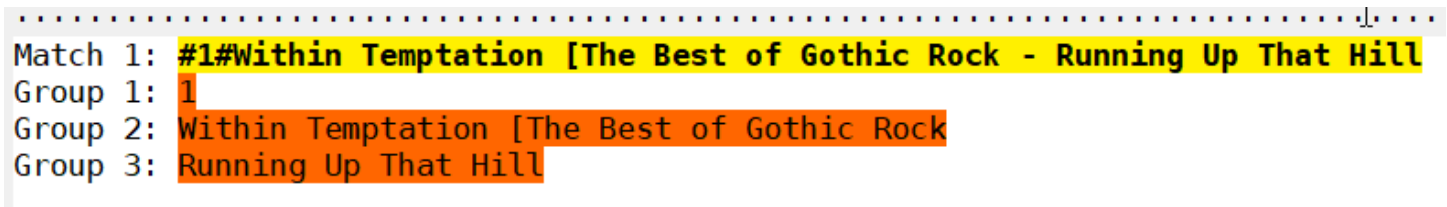
Match: #(\d+)#(.+) - (.+)
Replace: \2 \1 - \3







Sample:



In case this wasn't clear in the photo, the new names are:

Within Temptation [The Best of Gothic Rock] 1 - Running Up That Hill.mp3
Nightwish [The Best of Gothic Rock] 2 - Bless the Child.mp3

Forum Contributor: Stefan

Add

Add Autonumber in Middle of String (variable placement) cont.

Using: #1#Within Temptation [The Best of Gothic Rock] - Running Up That Hill.mp3

Match: #(\d+)#(.+) - (.+)

Where:

\1 = 1 \2 = Within Temptation [The Best of Gothic Rock] \3 = Running Up That Hill

Tim Analysis:

1. Match but don't capture the first '#'

Capture Group 1
1. (\d+) Match against the first numeric digit – the first autonumber. The + will make it Greedy and include other numbers if the autonumber exceeds the numeral 9, i.e. 10, so the + would include the '10' not just the '1'.
= 1

2. Match but don't capture the second '#'

Capture Group 2
3. (.+) Capture text until…

4. - .. Match against <space> <hyphen> <space> but don't capture
= Within Temptation [The Best of Gothic Rock]

Capture Group 3
5. (.+) Capture remainder of string
= Running Up That Hill

Using a Replacement String of: \2<space>\1<space><hyphen><space>\3 =

Within Temptation [The Best of Gothic Rock] 1 - Running Up That Hill.mp3

Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal)

SantaMonicaPier_051213_STST_001.jpg	to	SantaMonicaPier_2005-12-13_STST_001.jpg
GoldenGateBridge_100409_STST_001.jpg	to	GoldenGateBridge_2010-04-09_STST_001.jpg
BeverlyHills90210_130415_STST_001.jpg	to	BeverlyHills90210_2013-04-15_STST_001.jpg
Yosemite_000630_STST_001.jpg	to	Yosemite_2000-06-30_STST_001.jpg

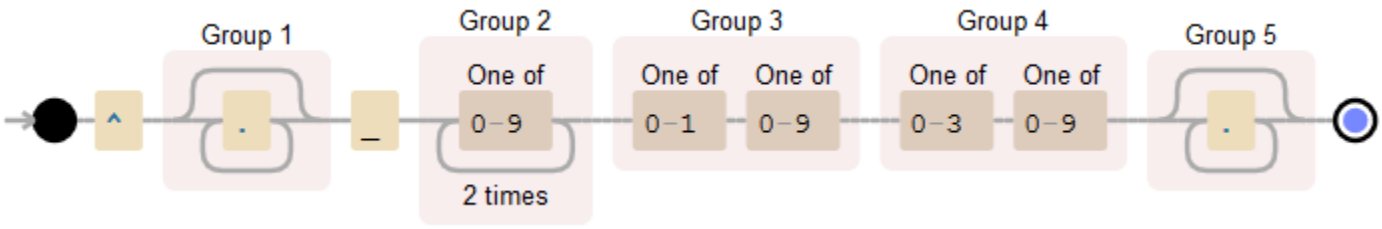
Observations:

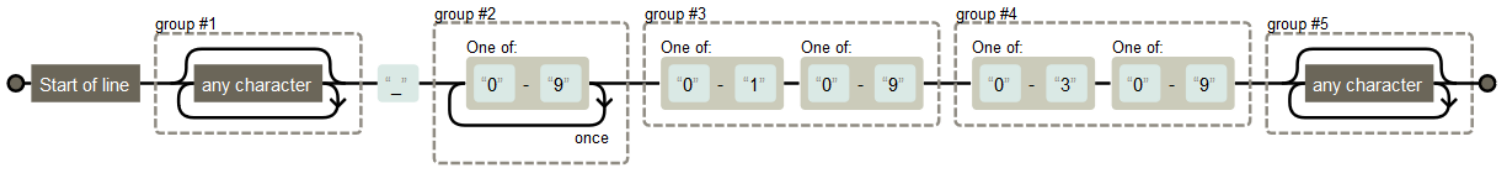
1. There are often numbers throughout.
2. The dates are ALWAYS six numeric digits with an underscore on either side, i.e., _130812_
3. The new file names should solely be based on the old ones, not pulled from Exif metadata.
4. No dates from the 1900s – only from 2000's.

Restrictive -

Match: ^(.*)_([0-9]{2})([0-1][0-9])([0-3][0-9])(.*)
Replace: 1_20\2-\3-\4\5

Name	New Name
Yosemite_000630_STST_001.jpg	1_2000-06-30_STST_001.jpg
SantaMonicaPier_051213_STST_001.jpg	1_2005-12-13_STST_001.jpg
GoldenGateBridge_100409_STST_001.jpg	1_2010-04-09_STST_001.jpg
BeverlyHills90210_130415_STST_001.jpg	1_2013-04-15_STST_001.jpg





Sample:

```
.....................................
Match 1: Yosemite_000630_STST_001
Group 1: Yosemite
Group 2: 00
Group 3: 06
Group 4: 30
Group 5: _STST_001
```

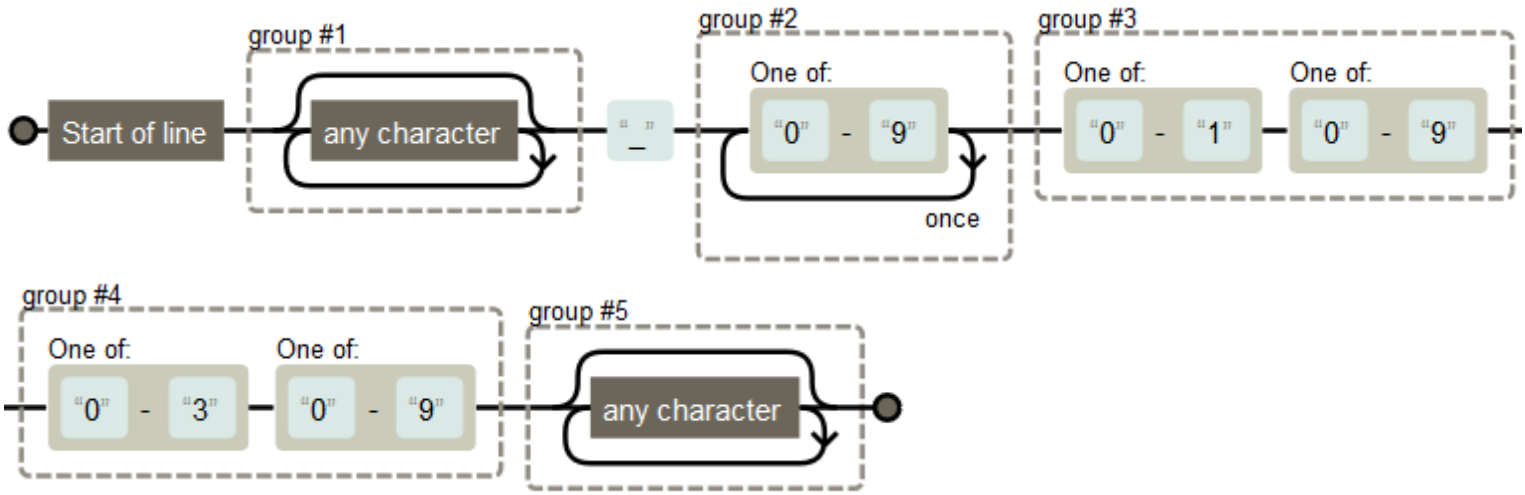
Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

Restrictive – cont.

Match: ^(.*)_([0-9]{2})([0-1][0-9])([0-3][0-9])(.*)

Expanded:



Tim Notes:

1. Extremely restrictive.
2. Takes apart the date based on the exact number of numeric digits and further restricted to the range of each section of the date.

Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

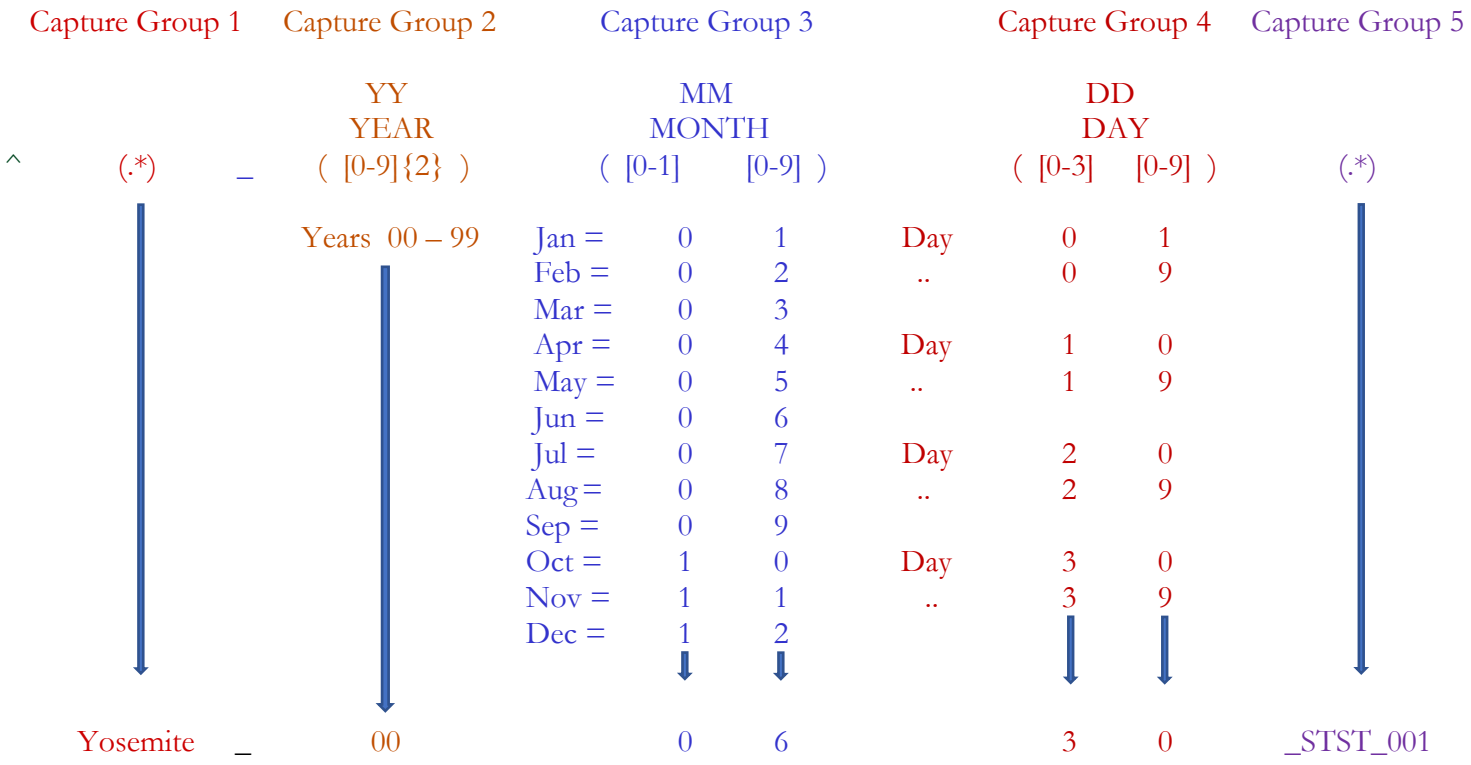
Using: Yosemite_000630_STST_001.jpg

Match: ^(.*)_([0-9]{2})([0-1][0-9])([0-3][0-9])(.*)

Where:

\1 = Yosemite \2 = 00 \3 = 06 \4 = 30 \5 = _STST_001

Breaks down:



Tim Notes:

1. Does not verify that a day cannot be 32 through 39.
2. Year is presumed to be YY in the 2000 – 2099 range as specified under the 'Observations':

0-9 {2}
00 = 2000
01 = 2001
..
99 = 2099

Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

Using: Yosemite_000630_STST_001.jpg

Match: ^(.*)_([0-9]{2})([0-1][0-9])([0-3][0-9])(.*)

Where:

\1 = Yosemite \2 = 00 \3 = 06 \4 = 30 \5 = _STST_001

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = Y
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. _	Match against underscore (Not Captured)	Changes Capture Group 1 = Yosemite_000630_STST
5. [0-9]	Match against a class consisting of a numeric digit '0' through '9' This is the '0' of '001'	Capture Group 2 = 0 Capture Group 1 = Unchanged
6. {2}	Repeat last expression to match 2 iterations. This is the '00' of '001'	Capture Group 2 = 00 Capture Group 1 = Unchanged
7. [0-1]	Match against a class consisting of a numeric digit '0' through '1' This is the '1' of '001'	Capture Group 3 = 1
8. [0-9]	Match against a class consisting of a numeric digit '0' through '9' Backtracks to match against '06'. Changes Capture Group 3 value to '06'	Capture Group 3 = 06 Changes Capture Group 1 = Yosemite

Tim Notes:

1. After the '1' of '001' is captured, the next match is for a numeric digit '0-9'. The RegEx Engine is at EOL so it backtracks looking for a match of [0-1] and [06] – it matches a '0' followed by a '6'.

9. [0-3]	Match against a class consisting of a numeric digit '0' through '3'	Capture Group 4 = 3
10. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 4 = 30
11. .*	Match against remaining string	Capture Group 5 = _STST_001

Add

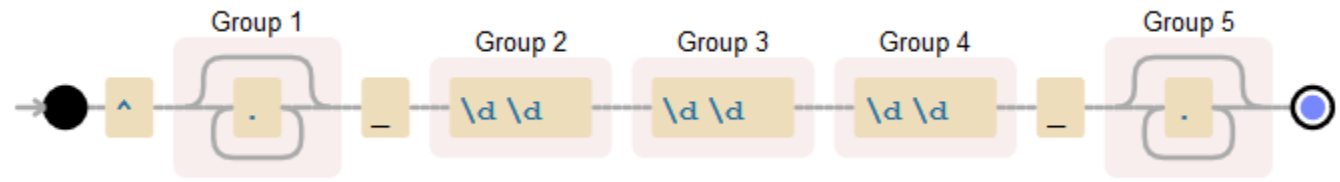
Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

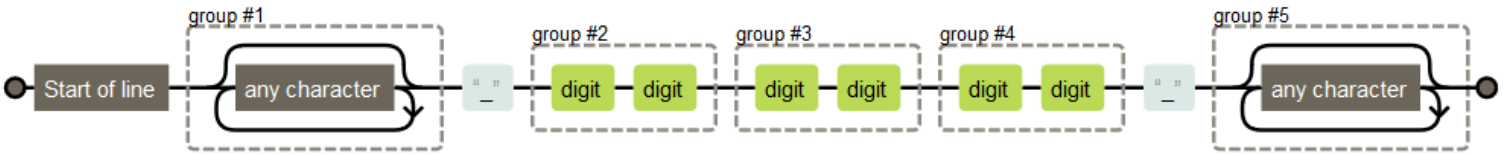
SantaMonicaPier_051213_STST_001.jpg	to	SantaMonicaPier_2005-12-13_STST_001.jpg
GoldenGateBridge_100409_STST_001.jpg	to	GoldenGateBridge_2010-04-09_STST_001.jpg
BeverlyHills90210_130415_STST_001.jpg	to	BeverlyHills90210_2013-04-15_STST_001.jpg
Yosemite_000630_STST_001.jpg	to	Yosemite_2000-06-30_STST_001.jpg

More Universal –

Match: ^(.*)_(\d\d)(\d\d)(\d\d)_(.*)
Replace: 1_20\2-\3-\4\5

Name	New Name
Yosemite_000630_STST_001.jpg	1_2000-06-30STST_001.jpg
SantaMonicaPier_051213_STST_001.jpg	1_2005-12-13STST_001.jpg
GoldenGateBridge_100409_STST_001.jpg	1_2010-04-09STST_001.jpg
BeverlyHills90210_130415_STST_001.jpg	1_2013-04-15STST_001.jpg





Sample:

```
.................................
Match 1: Yosemite_000630_STST_001
Group 1: Yosemite
Group 2: 00
Group 3: 06
Group 4: 30
Group 5: STST_001
```

Tim Notes:

1. The \d matches a numeric digit. The date is taken apart similar to the first RegEx but the range is not specified.

Forum Contributor: Stefan

Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

Using: Yosemite_000630_STST_001.jpg

More Universal – cont.

Match: ^(.*)_(\d\d)(\d\d)(\d\d)_(.*)

can be rewritten as:

Match: ^(.*)_(\d{2})(\d{2})(\d{2})_(.*)

Where:

\1 = Yosemite \2 = 00 \3 = 06 \4 = 30 \5 = STST_001

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = Y
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. _	Match against underscore (Not Captured)	Changes Capture Group 1 = Yosemite_000630_STST
5. \d{2}	Match against 2 numeric digits This is the '00' of '001'	Capture Group 2 = 00 Capture Group 1 = Unchanged
6. \d{2}	Match against 2 numeric digits Capture Group 2 changes to the '00' after 'Yosemite_'	Capture Group 3 = 06 Changes Capture Group 2 = 00 Changes Capture Group 1 = Yosemite
7. \d{2}	Match against 2 numeric digits	Capture Group 4 = 30
8. _	Match against underscore (Not Captured)	
9. .*	Match against remaining string	Capture Group 5 = STST_001

Add

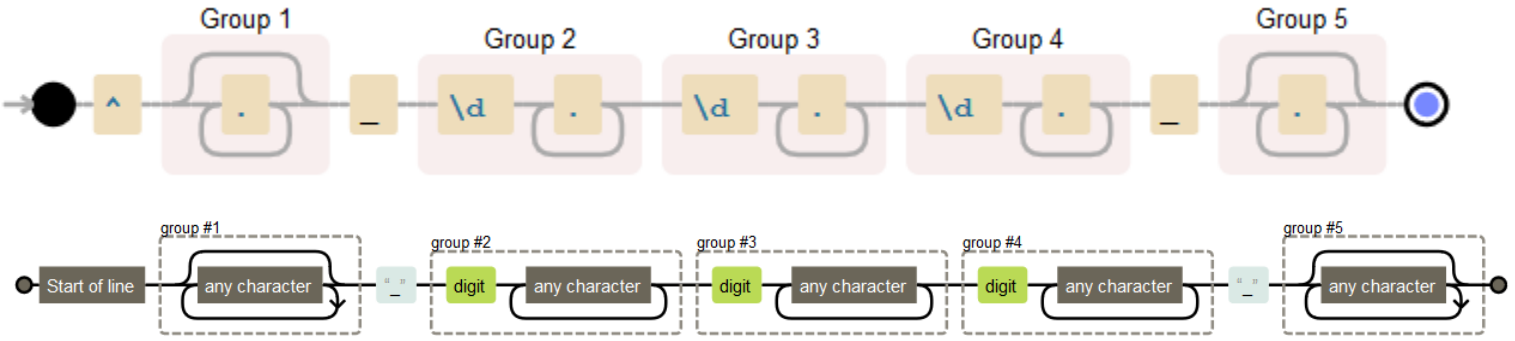
Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

Tim Contribution –

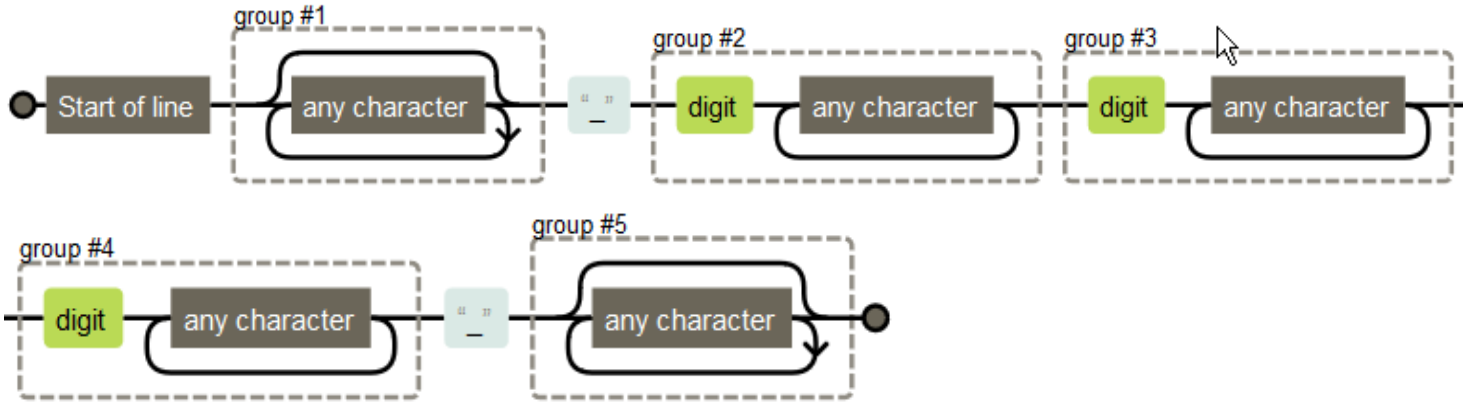
Even More Universal -

Match: ^(.*)_(\d.+?)(\d.+?)(\d.+?)_(.*)
Replace: 1_20\2-\3-\4\5

Name	New Name
Yosemite_000630_STST_001.jpg	1_2000-06-30STST_001.jpg
SantaMonicaPier_051213_STST_001.jpg	1_2005-12-13STST_001.jpg
GoldenGateBridge_100409_STST_001.jpg	1_2010-04-09STST_001.jpg
BeverlyHills90210_130415_STST_001.jpg	1_2013-04-15STST_001.jpg



Expanded:



Sample:

```
Match 1: Yosemite_000630_STST_001
Group 1: Yosemite
Group 2: 00
Group 3: 06
Group 4: 30
Group 5: STST_001
```

Tim Notes:

1. There is no specification on the number of numeric digits in each section and no range.

Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

Tim Contribution – cont.

Using: Yosemite_000630_STST_001.jpg

Even More Universal – cont.

Match: ^(.*)_(\d.+?)(\d.+?)(\d.+?)_(.*)

Where:

\1 = Yosemite \2 = 00 \3 = 06 \4 = 30 \5 = STST_001

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = Y
3. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. _	Match against underscore (Not Captured) Backtracks.	Changes Capture Group 1 = Yosemite_000630_STST
5. \d	Match against a numeric digit This is the '0' of '001'	Capture Group 2 = 0
6. .	Match against any character This is the '00' of '001'	Capture Group 2 = 00
7. +	Make it Greedy	Capture Group 2 = 001
8. ?	But not too Greedy This is still the '00' of '001'	Capture Group 2 = 00
9. \d	Match against a numeric digit This is the '1' of '001'	Capture Group 3 = 1
10. .	Match against any character This is the third zero of '000630' Capture Group 2 changes to the '00' of '000630'	Capture Group 3 = 0 Changes Capture Group 2 =00 Changes Capture Group 1 = Yosemite

Tim Notes:

1. It has reached the EOL at the end of '001'. It backtracks back to the beginning matching against the start of the RegEx, (.*)_
2. Because it is moving forward, the underscore matched is the one after 'Yosemite' not the underscore after 'STST' as it did in step 4 because it was backtracking.
3. The (\d.+?) of Capture Group 2 reevaluates to the '00' of '000630' and Capture Group 3 captures the match of the third '0' of '000630'.
4. This changes Capture Group 1 = Yosemite

11.. +	Make it Greedy	Capture Group 3 = 000630_STST_001
12. ?	But not too Greedy	Capture Group 3 = 06

Add

Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal) cont.

Tim Contribution – cont.

Using: Yosemite_000630_STST_001.jpg

Even More Universal – cont.

Match: ^(.*)_(\d.+?)(\d.+?)(\d.+?)_(.*)

Where:

\1 = Yosemite \2 = 00 \3 = 06 \4 = 30 \5 = STST_001

Tim Analysis: cont.

13. \d	Match against a numeric digit	Capture Group 4 = 3
14. .	Match against any character	Capture Group 4 = 30
15. +	Make it Greedy	Capture Group 4 = 30_STST_001
16. ?	But not too Greedy	Capture Group 4 = 30
17. _	Match against underscore (Not Captured) This is the underscore after '000630'	
18. .*	Match against remaining string	Capture Group 5 = STST_001

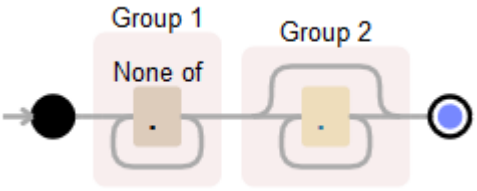
Add

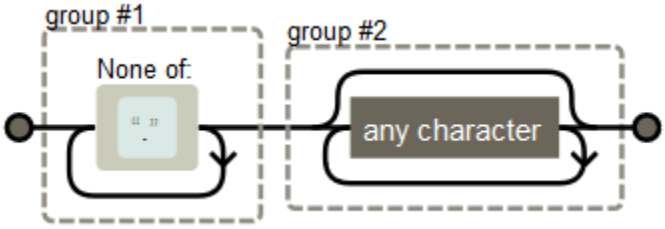
Add the Word DVD to Files (Demonstrate Negated Character Class)

Animal House.avi	to	Animal House DVD.avi
Animal House.fanart.jpg	to	Animal House DVD.fanart.jpg
Animal House.nfo	to	Animal House DVD.nfo

Match: ([^.]+)(.*)
Replace: \1 DVD\2

Name	New Name
Animal House.nfo	Animal House DVD.nfo
Animal House.fanart.jpg	Animal House DVD.fanart.jpg
Animal House.avi	Animal House DVD.avi





Sample:

```
.....................Start·Length···
Match 1: Animal House      0      12
Group 1: Animal House      0      12
Group 2:                  12       0
```

Sample:

```
.............................
Match 1: Animal House.fanart
Group 1: Animal House
Group 2: .fanart
```

Contributor Notes:

1. [^.] means any character **except** a dot. This is a negating Character class.
2. Multiple dots in the filename won't affect it because it stops looking for the dot when it hits the first one.

Contributor Analysis:

1. [^.]+	Captures the original filename up to the first dot.	Capture Group 1
2. (.*)	Captures the remainder of the string.	Capture Group 2

Tim Notes:

1. Simple example of parsing string and inserting text using Replacement String.
2. If no text follows, then a Zero Occurrence Match results in a null value for Capture Group 2.
3. This is an example of A Zero Occurrence Match caused by (.*) rather then (.*?) which is what I see typically.

Forum Contributor: Jane

Add

Add the Word DVD to Files (Demonstrate Negated Character Class) cont.

Using: Animal House.avi

Match: ([^.]+)(.*)

Where:

\1 = Animal House \2 = null

Tim Analysis:

1. [^.]	Match against a Negated Class consisting of a literal <dot>. Note that you don't have to use an Escape metacharacter inside a class for literal characters. Matches against any character except the <dot>.	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = Animal House
3. .*	Match against remaining string	Capture Group 2 = null

Using: Animal House.fanart.jpg

Match: ([^.]+)(.*)

Where:

\1 = Animal House \2 = .fanart

Tim Analysis:

1. [^.]	Match against a Negated Class consisting of a literal <dot>. Note that you don't have to use an Escape metacharacter inside a class for literal characters. Matches against any character except the <dot>.	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = Animal House
3. .*	Match against remaining string	Capture Group 2 = .fanart

Add

Add the text in Parentheses (not move it) to beginning of filename

other_artist_-_something_else-(LBL002)-WEB-2012-grp
to
LBL002_other_artist_-_something_else-(LBL002)-WEB-2012-grp

rawr_rwar_-_justin_bieber_isagirl-(LBL008)-WEB-2012-grp
to
LBL008_rawr_rwar_-_justin_bieber_isagirl-(LBL008)-WEB-2012-grp

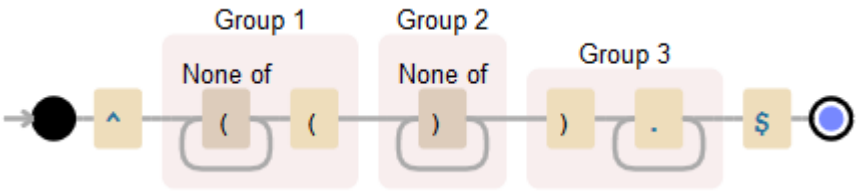
artist01_-_release-(LBL005)-WEB-2012-grp	to	LBL005_artist01_-_release-(LBL005)-WEB-2012-grp
thirdsample_-_blabla-(LBL013)-WEB-2012-grp	to	LBL013_thirdsample_-_blabla-(LBL013)-WEB-2012-grp

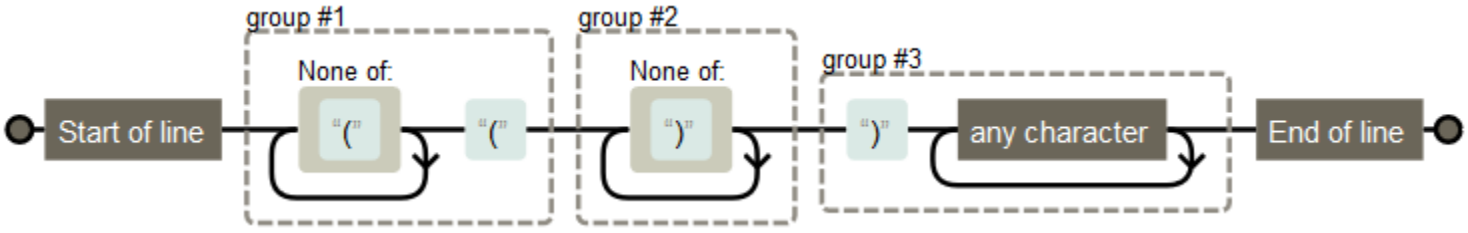
Observations:

1. "LBLxxx" is catalogue number of the music release.
2. This would provide a sorting by catalogue number.
3. Only one set of parentheses.

Match: ^([^(]+\()([^)]+)(\).+)$
Replace: \2_\1\2\3

Name	New Name
thirdsample_-_blabla-(LBL013)-WEB-2012-grp	LBL013_thirdsample_-_blabla-(LBL013)-WEB-2012-grp
rawr_rwar_-_justin_bieber_isagirl-(LBL008)-WEB-2012-grp	LBL008_rawr_rwar_-_justin_bieber_isagirl-(LBL008)-WEB-2012-grp
artist01_-_release-(LBL005)-WEB-2012-grp	LBL005_artist01_-_release-(LBL005)-WEB-2012-grp





Add

Add the text in Parentheses (not move it) to beginning of filename cont.

thirdsample_-_blabla-(LBL013)-WEB-2012-grp

Match: ^([^(]+\()([^)]+)(\).+)$

Sample:

```
.....................................................
Match 1: thirdsample_-_blabla-(LBL013)-WEB-2012-grp
Group 1: thirdsample_-_blabla-(
Group 2: LBL013
Group 3: )-WEB-2012-grp
```

Contributor Analysis:

1. ^([^(]+\() Match everything from the start (Beginning of Line or BOL) of the title that is NOT an opening parenthesis along with the opening parenthesis, and capture it as group 1.

2. ([^)]+) Matches the contents after the opening parenthesis up to the closing parenthesis, and capture it as group 2.

3. (\).+)$ Matches the closing parenthesis and everything thereafter to the end of the title, and captures it as group 3.

Forum Contributor: Jane

Add

Add the text in Parentheses (not move it) to beginning of filename cont.

Using: thirdsample_-_blabla-(LBL013)-WEB-2012-grp

Match: ^([^(]+\()([^)]+)(\).+)$

Where:

\1 = thirdsample_-_blabla-(\2 = LBL013 \3 =)-WEB-2012-grp

Breaks down:

	Capture Group 1			Capture Group 2			Capture Group 3		
Anchor		Negated Class	Literal		Negated Class	Literal			Anchor
^	(	[^ ( ] +	\(	)	(| [^)] +	)	(\) .+	)	$

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^(]	Match against a Negated Class consisting of a literal left parenthesis. Match against anything but a left parenthesis.	Capture Group 1 = t
3. +	Make it Greedy	Capture Group 1 = thirdsample_-_blabla-
4. \(	Match against a literal left parenthesis	Capture Group 1 = thirdsample_-_blabla-(
5. [^)]	Match against a Negated Class consisting of a literal right parenthesis. Match against anything but a right parenthesis.	Capture Group 2 = L
6. +	Make it Greedy	Capture Group 2= LBL013
7. \)	Match against a literal right parenthesis	Capture Group 3 =)
8. .+	Match against remaining string	Capture Group 3 =)-WEB-2012-grp
9. $	Must be at EOL (End of Line)	

Add

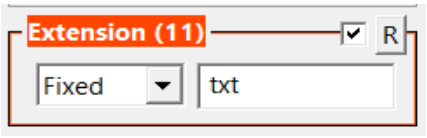
Renaming all files with no extension to .txt

Tim Notes:

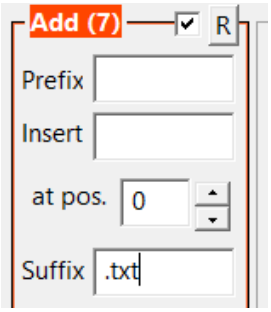
1. Yes, I know you can do this in a myriad of ways:

   a. Command prompt: rename *. *.txt

   b. BRU 'Section #11: Extension' –



   c. BRU 'Section #7: Add' –



But did you know it could be done with a RegEx?

File:

Animal House	to	Animal House.txt

Folders:

1863_22_10_2019	to	1863_22_10_2019.txt
junk	to	junk.txt

Match: ^([^.]+)$
Replace: \1.txt

Name	New Name
Animal House	Animal House.txt
junk	junk.txt
1863_22_10_2019	1863_22_10_2019.txt

Add

Renaming all files with no extension to .txt cont.

File:

Animal House	to	Animal House.txt

Folders:

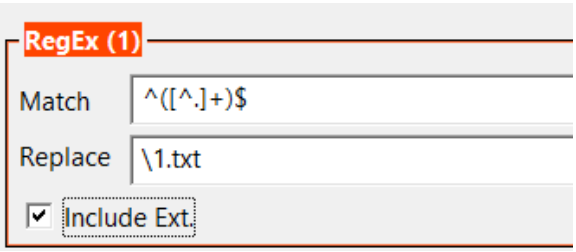
1863_22_10_2019	to	1863_22_10_2019.txt
junk	to	junk.txt

Match: ^([^.]+)$

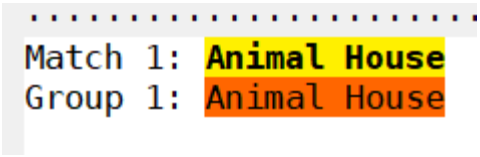
Where:

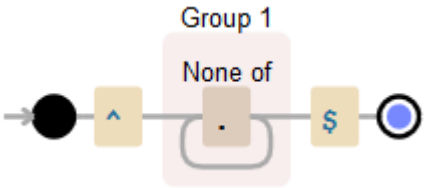
\1 = Animal House

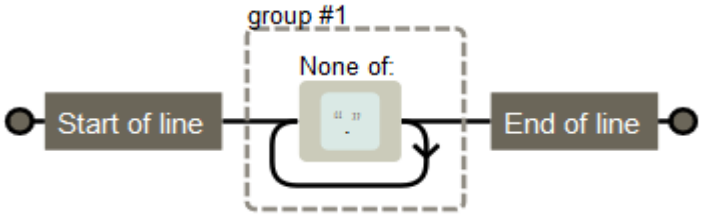
For those filenames that have extensions, enable, 'Include Ext'.



Sample:







Tim Analysis:

1. [^.]	Match only filenames which don't contain an extension dot (Negated Character set)	
2. +	Make it Greedy so it captures the entire string to…	Capture Group 1
3. $	.. EOL (End of Line)	

Forum Contributor: bitmonger

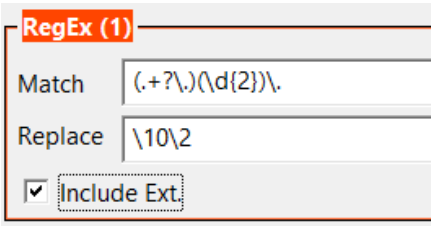
Add

Pad Double digit at end of string (in one pass!)

Tim Notes:

1. Another one I found that can do it in one pass albeit only at end of filename
2. Interesting in that it uses the extension file dot to exclude three or more numeric digits
3. I decided to REALLY elaborate on this technique in 'My Contribution' section.
4. Pad/ Append – Tomatoes, Tomatos, ☺. The line will get very blurry as you will see.

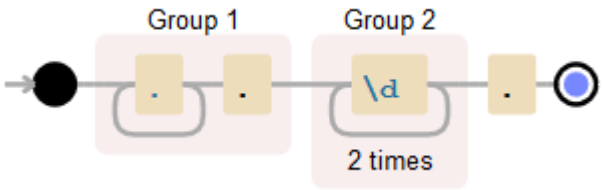
file_1_name.01.tif	to	file_1_name.001.tif
file_1_name.02.tif	to	file_1_name.002.tif
file_1_name.101.tif	to	Ignore

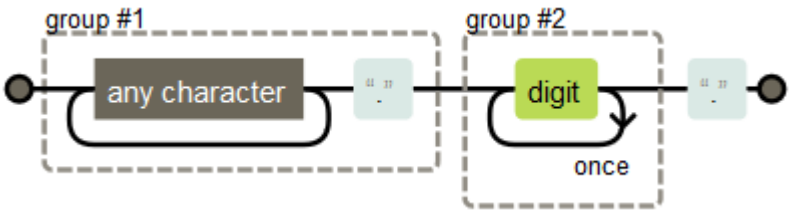


Include Ext: Enable
.. because there are dot characters in string

Match: (.+?\.)(\d{2})\.
Replace: \10\2

Name	New Name
file_1_name.01.tif	file_1_name.001.tif
file_1_name.02.tif	file_1_name.002.tif
file_1_name.101.tif	file_1_name.101.tif





Sample:

```
.............................
Match 1: file_1_name.01.
Group 1: file_1_name.
Group 2: 01
```

Forum Contributor: olwynn, Stefan

Add

Pad Double digit at end of string (in one pass!) cont.

file_1_name.01.tif	to	file_1_name.001.tif
file_1_name.02.tif	to	file_1_name.002.tif
file_1_name.101.tif	to	Ignore

Match: (.+?\.)(\d{2})\.

Where:

\1 = file_1_name \2 = 01

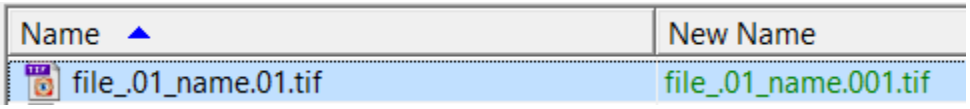
The following is a general analysis. I provide a more complete analysis later.

Tim Analysis:

1. .+?	Match up until …	Capture Group 1
2. \.	.. first dot	Capture Group 1
3. \d{2}	Match the two numeric digits	Capture Group 2
4. \.	Match against final dot (the extension). For \d{2} not to include '101' above, the final dot must immediately follow the 2 numeric digits. If you were to eliminate this, the '101' would be included and result in '101' to '010'.	

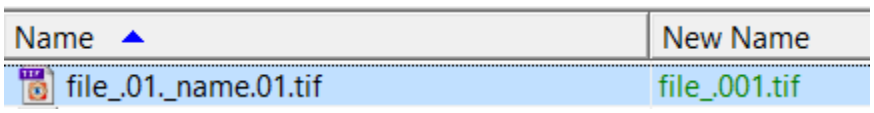
Tim Notes: Limitations to this technique

1. Limited – will only work with <dot> characters in string.
2. <dot> character must appear directly after the numeric portion to be padded.



Even though a <dot> character appears before the first '01' in the string, it is the second '01' that gets padded – why? Because the <dot> character appears after the second numeric group.

3. <dot> character determines which numeric group gets padded and how much of the string is retained:



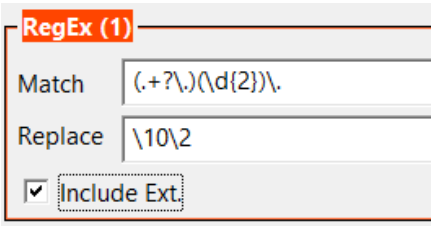
4. Be careful that it doesn't inadvertently renumber undesired files.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution –

Once again I blur the line between Pad/Append. My examples are more of an append.



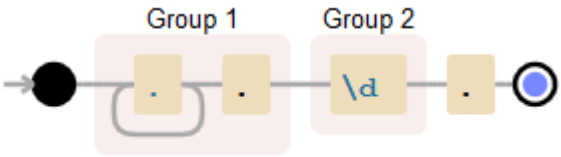
Include Ext: Enable
.. because there are dot characters in string

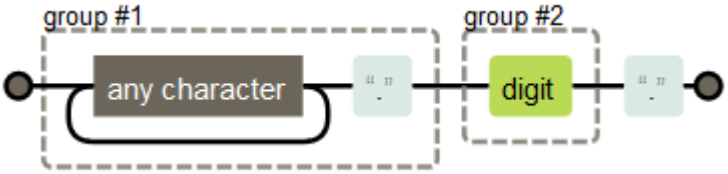
1. This technique can easily be adapted for other quantity of numeric digits at the end of the filename.

For single numeric digits:

Match: (.+?\.)(\d{1})\.
Replace: \10\2

Name	New Name
file_1_name.101.tif	file_1_name.101.tif
file_1_name.1.tif	file_1_name.01.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif





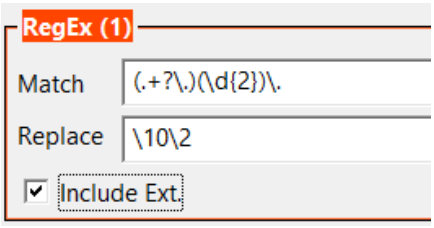
Sample:

```
.........................
Match 1: file_1_name.1.
Group 1: file_1_name.
Group 2: 1
```

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



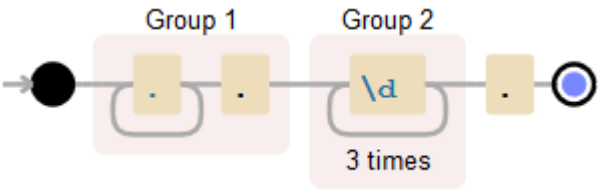
Include Ext: Enable
.. because there are dot characters in string

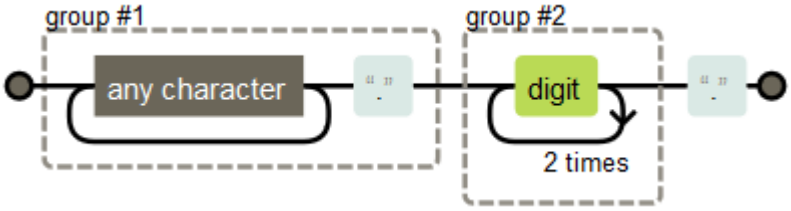
1. This technique can easily be adapted for other quantity of numeric digits at the end of the filename. cont.

For triple numeric digits:

Match: (.+?\.)(\d{3})\.
Replace: \10\2

Name	New Name
file_1_name.101.tif	file_1_name.0101.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif





Sample:

```
.............................
Match 1: file_1_name.101.
Group 1: file_1_name.
Group 2: 101
```

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Because the RegEx for these are so similar – only changing the value of the Quantifier – the analysis is presented together.

For single numeric digits:

Using: file_1_name.1.tif

Match: (.+?\.)(\d{1})\.

Where:

\1 = file_1_name \2 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. \.	Match against literal <dot>	Capture Group 1 = file_1_name.
5. \d{1}	Match against 1 numeric digit Repeat 1 time.	Capture Group 2 = 1
6. \.	Match against literal <dot> (Not Captured) This is the <dot> of the extension. Remember the RegEx is set to include the Extension in the match.	

Tim Notes:

1. See 'Tim Notes: Limitations to this technique' on previous pages
2. Just change the Quantifier to the length you want to match.
3. If you don't want to include the <dot> character after 'name' just change it to this:

Match: (.+?)\.(\d{1})\.

For double numeric digits (the original posting):

Using: file_1_name.01.tif

Match: (.+?\.)(\d{2})\.

Where:

\1 = file_1_name \2 = 01

For triple numeric digits:

Using: file_1_name.101.tif

Match: (.+?\.)(\d{3})\.

Where:

\1 = file_1_name \2 = 101

The analysis is the same for each.

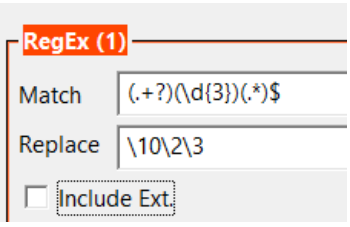
Add

Pad Double digit at end of string (in one pass!) cont.

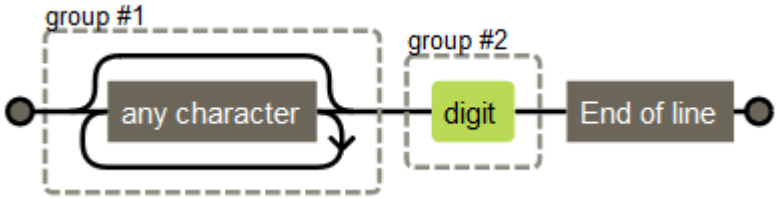
Tim Contribution – cont.

file_1_name1.tif	to	file_1_name01.tif
file_1_name.1.tif	to	file_1_name.01.tif
file_1_name.01.tif	to	file_1_name.001.tif
file_1_name01.tif	to	file_1_name001.tif
file_1_name.101.tif	to	file_1_name.0101.tif
file_1_name101.tif	to	file_1_name0101.tif

This will match against the last match it finds regardless of dot in string:

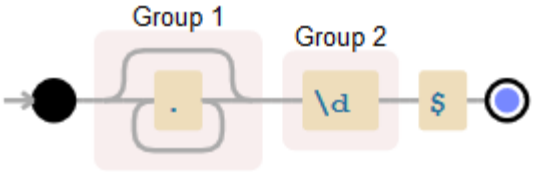


Include Ext.: Disable
.. even though there are dot characters in string



Match: (.*)(\d{1})$
Replace: \10\2

Name	New Name
file_1_name1.tif	file_1_name01.tif
file_1_name.1.tif	file_1_name.01.tif



Match: (.*)(\d{2})$
Replace: \10\2

Name	New Name
file_1_name.01.tif	file_1_name.001.tif
file_1_name01.tif	file_1_name001.tif

Sample:

```
.....................
Match 1: file_1_name1
Group 1: file_1_name
Group 2: 1
```

Match: (.*)(\d{3})$
Replace: \10\2

Name	New Name
file_1_name101.tif	file_1_name0101.tif
file_1_name.101.tif	file_1_name.0101.tif

Tim Notes:

1. You must be careful with selection of files to be renamed. These RegEx will inadvertently affect other files.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Using: file_1_name1.tif

This will match against the last match it finds regardless of dot in string: cont.

Match: (.*)(\d{1})$

Where:

\1 = file_1_name \2 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. \d{1}	Match against 1 numeric digit Repeat 1 time. Backtracks. This is the '1; after 'name.'	Capture Group 2 = 1 Changes Capture Group 1 = file_1_name.
4. $	Must be at EOL (End of Line)	

Tim Notes:

1. Limited.
2. Be careful that it doesn't inadvertently renumber undesired files.

So how to capture the first part of the numeric group instead of the last? Using (.*), will capture the entire string and backtrack but move forward from there, so that won't work. But adding the Lazy Quantifier, ?, will move to the BOL 'setting up' capture Group 1, as I like to refer to it – in actuality it results in a null value, but as soon as I specify the next expression, the RegEx Engine will move forward adding characters to Capture Group 1 depending on the character position of that evaluation. I could have also used the (.+?). This would move to the first character that was captured by Capture Group 1, the 'f'.

See Next Page.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Expanded Sample Base:

w f- 00088 jerry028	file_1_name01	file_01_name.1	file_.01._name.01	file_01_name.01
file_123_test101_name201	file_.01_name.01	file_1_name.01	file_1_name.101	file_1_name1
file_1_name101	file_1_name.1	file_1_test1_name01	file_1_test1_name1	

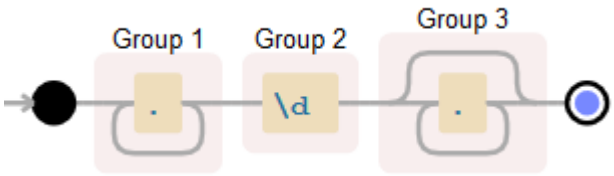
This will match against the first match it finds regardless of dot in string:

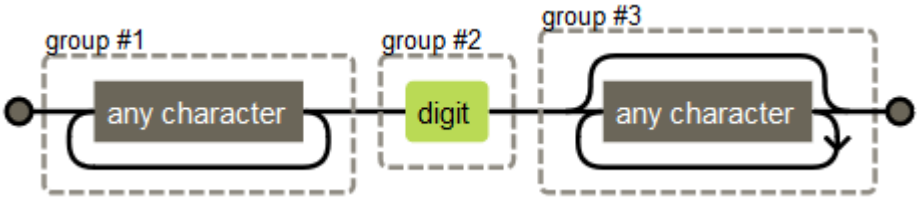
Match: (.+?)(\d{1})(.*)
Replace: \10\2\3

Name	New Name
w f- 00088 jerry028.jpg	w f- 000088 jerry028.jpg
file_.01._name.01.tif	file_.001._name.01.tif
file_.01_name.01.tif	file_.001_name.01.tif
file_01_name.01.tif	file_001_name.01.tif
file_01_name.1.tif	file_001_name.1.tif
file_123_test101_name201.tif	file_0123_test101_name201.tif
file_1_name.01.tif	file_01_name.01.tif
file_1_name.1.tif	file_01_name.1.tif
file_1_name.101.tif	file_01_name.101.tif
file_1_name01.tif	file_01_name01.tif
file_1_name1.tif	file_01_name1.tif
file_1_name101.tif	file_01_name101.tif
file_1_test1_name01.tif	file_01_test1_name01.tif
file_1_test1_name1.tif	file_01_test1_name1.tif

Sample:

```
.....................
Match 1: file_1_name1
Group 1: file_
Group 2: 1
Group 3: _name1
```





Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Using: file_1_name101

This will match against the first match it finds regardless of dot in string: cont.

Match: (.+?)(\d{1})(.*)

Where:

\1 = file_ \2 =1 \3 = _name101

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. \d{1}	Match against 1 numeric digit Repeat 1 time. Moves Forward. This is the '1; after 'file_'	Capture Group 2 = 1 Changes Capture Group 1 = file_
5. .*	Match against remaining string	Capture Group 3 = _name101

Tim Notes:

1. Limited to first match it finds.
2. Be careful that it doesn't inadvertently renumber undesired files.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

At this point, I think we have gone past merely 'padding' and have embarked upon "appending' Zeros.

Let's continue.

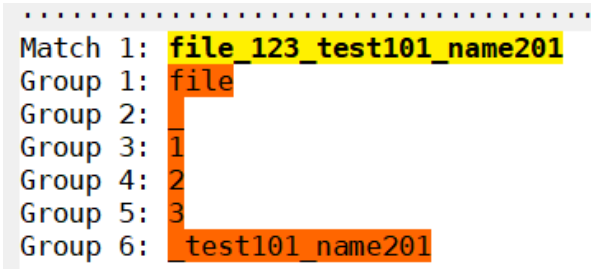
Match against first two numeric groups regardless of <dot>

Match: (.+?)([_-])\.?(\d{1})(.+?)(\d{1})(.*)$
Replace: \1\20\3\40\5\6

Name	New Name
file_.01._name.01.tif	file_001._name.001.tif
file_.01_name.01.tif	file_001_name.001.tif
file_01_name.01.tif	file_001_name.001.tif
file_01_name.1.tif	file_001_name.01.tif
file_1 456name.101.tif	file_01 0456name.101.tif
file_123_test101_name201.tif	file_01203_test101_name201.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.101.tif
file_1_ 456 name101.tif	file_01_ 0456 name101.tif
file_1_name.01.tif	file_01_name.001.tif
file_1_name.1.tif	file_01_name.01.tif
file_1_name.101.tif	file_01_name.0101.tif
file_1_name01.tif	file_01_name001.tif
file_1_name1.tif	file_01_name01.tif
file_1_name101.tif	file_01_name0101.tif
file_1_test1_name01.tif	file_01_test01_name01.tif
file_1_test1_name1.tif	file_01_test01_name1.tif

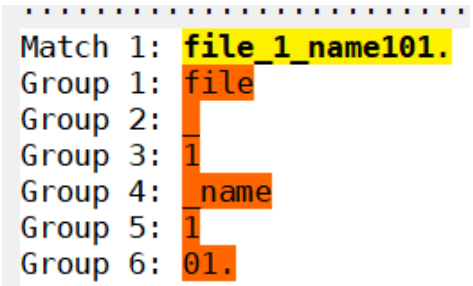
Matches correctly all except file_123_test101_name201.tif. It is impossible to match all but especially this file because it has a different pattern that requires a different RegEx. Can't parse the string out – Group 6 indicates this:

text_#_text###_text###



whereas the others are:

text _#_text#
text_#_text##
text_#_text###
text_#_text#_text#
text_#_text#_text##
text_##_text#
text_##_text##
text_##_text<dot>##
text_<dot>##<dot>_text<dot>##
text_#_text <space>###text <dot>###
text_#_text<space>###text###



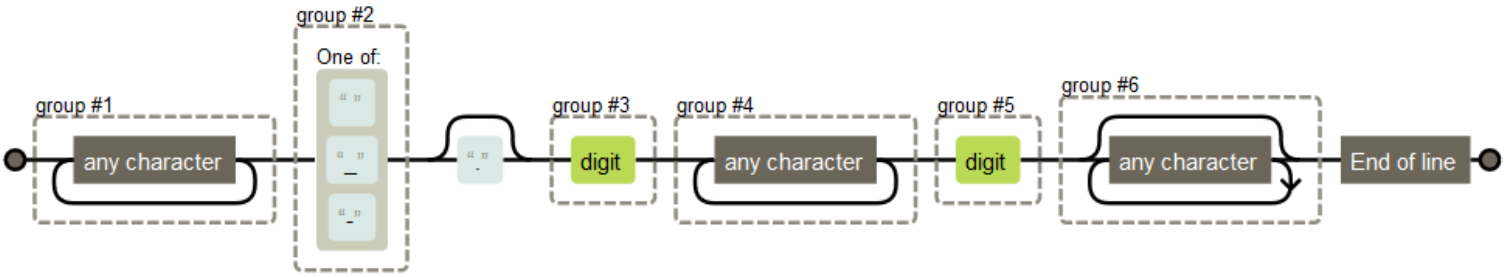
So you see it does match quite a number of patterns already.

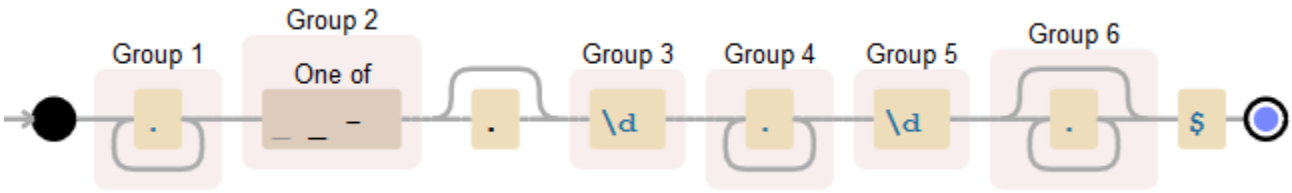
Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Match against first two numeric groups regardless of <dot> cont.





Breaks down:

1	2		3	4	5	6	
(.+?)	([_-])	\. ?	(\d{1})	(.+?)	(\d{1})	(.*)	$
Text	_	(optional <dot>	#	last portion of Numeric Group + Text	#	last portion of Numeric Group + Text	EOL

What do I mean about 'last portion of Numeric Group'? I mean that the \d only matches against a single numeric digit. If, in fact, a numeric group consists of more than one numeric digit, this is carried over and captured within the (.+?) or the (.*) Groups. It looks like this:

file_01_name01.tif

```
.......................
Match 1: file_01_name01
Group 1: file
Group 2: _
Group 3: 0
Group 4: 1_name
Group 5: 0
Group 6: 1
```

It still works to my satisfaction because in order to pad or append, I only need to isolate the first numeric digit of the numeric group. Remember the appending and padding is done in the Replacement String. The RegEx only serves to parse the string. In the above example, the isolated first numeric digits are the '0' in Capture Group 3 and the '0' of Capture Group 5.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Match against first two numeric groups regardless of <dot> cont.

Using: file_01_name01.tif

Match: (.+?)([_-])\.?(\d{1})(.+?)(\d{1})(.*)$

Where:

\1 = file \2 = _ \3 = 0 \4 = 1_name \5 = 0 \6 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. [_-]	Match against a class consisting of a <space>, underscore or <hyphen>	Capture Group 2 = _ Changes Capture Group 1 = file
5. \.	Match against literal <dot> This string does not contain a <dot> character.	= No Match
6. ?	Make it optional. For strings that do not have a <dot> Character.	Restores Capture Group 1 = file Restores Capture Group 2 = _
7. \d{1}	Match against 1 numeric digit Repeat 1 time. This is the '0' after 'file_'	Capture Group 3 = 0
8..	Match against any character This is the '1' of 'file_0'	Capture Group 4 = 1
9 +	Make it Greedy	Capture Group 4 = 1_name01
10. ?	But not too Greedy	Capture Group 4 = 1
11. \d{1}	Match against 1 numeric digit Repeat 1 time. This is the '0' of 'name01'	Capture Group 5 = 0 Changes Capture Group 4 = 1_name
12. .*	Match against remaining string	Capture Group 6 = 1
13. $	Must be at EOL (End of Line)	

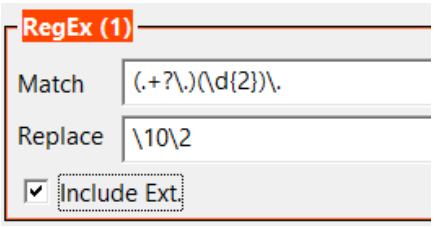
Tim Notes:

1. Limited.
2. Be careful that it doesn't inadvertently renumber undesired files.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



Include Ext: Enable
.. because there are dot characters in string

Let's take it a step further. I have improved upon the original RegEx. How about multiple digits within the same string!

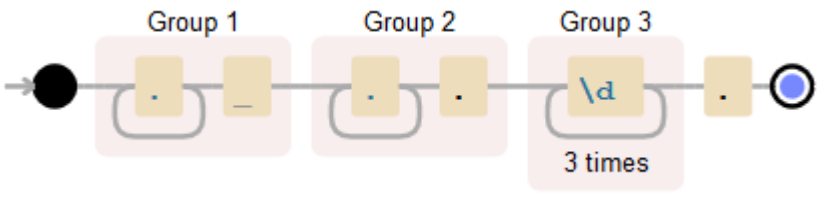
Here is a three numeric digit sample with <dot> characters:

file_1_ 456 name.101.tif to file_1_ 0456 name.0101.tif
file_1 456name.101.tif to file_1 0456name.0101.tif

Match: (.+?)(.+?\.)(\d{3})\.
Replace: \10\2 0\3

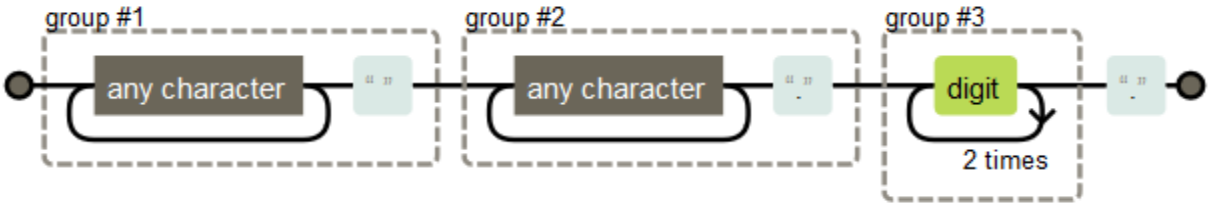
Name	New Name
file_1_name.101.tif	file_1_name.101.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 5-name.1.tif	file_1_ 5-name.1.tif
file_1_ 456 name.101.tif	file_1_ 0456 name. 0101.tif

All you need to do is prefix .+? in Capture Group 1 - and this is important - after the ? you need to add the character that distinguishes the numeric digits from the previous part of the string. In the original post it was a dot (.+?\.) Here I am using a <space>. So it becomes: (.+?<space>) and matches <space>456



Sample:

```
Match 1: file_1_ 456 name.101.
Group 1: file_1_
Group 2: 456 name.
Group 3: 101
```



Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Here is a three numeric digit sample with <dot> characters: cont.

Using: file_1 456name.101.tif

Match: (.+?)(.+?\.)(\d{3})\.

Where:

\1 = file_1 <space> \2 = 456name. \3 = 101

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. <space>	Match against a <space> character	Capture Group 1 = file_1 <space>
5..	Match against any character	Capture Group 2 = 4
6. +	Make it Greedy	Capture Group 2 = <entire string>
7. ?	But not too Greedy	Capture Group 2 = 4
8. \.	Match against literal <dot>	Capture Group 2 = 456name.
9. \d{3}	Match against 1 numeric digit Repeat 3 iterations. Moves Forward.	Capture Group 3 = 101
10. \.	Match against literal <dot> (Not Captured) This is the <dot> of the extension.	

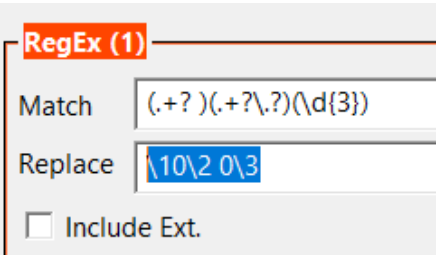
Tim Notes:

1. Limited.
2. Be careful that it doesn't inadvertently renumber undesired files.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



Include Ext: Disable
.. even though there are dot characters in string

Here is a three numeric digit sample With or Without <dot> characters:

file_1_ 456 name.101.tif to file_1_ 0456 name.0101.tif

Match: (.+?)(.+?\.?)(\d{3})
Replace: \10\2 0\3

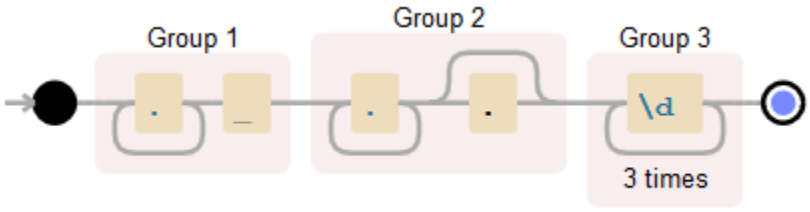
Name	New Name
file_1_ 456 name101.tif	file_1_ 0456 name 0101.tif
file_1_ 456 name.101.tif	file_1_ 0456 name. 0101.tif

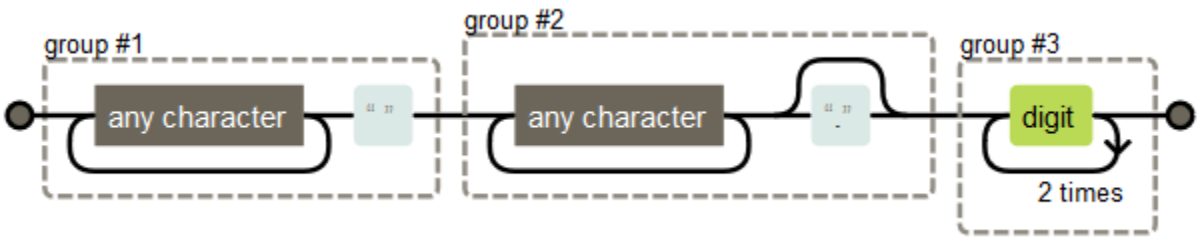
Sample: with <dot>

```
.....................................
Match 1: file_1_ 456 name.101
Group 1: file_1_
Group 2: 456 name.
Group 3: 101
```

Sample: without <dot>

```
.....................................
Match 1: file_1_ 456 name101
Group 1: file_1_
Group 2: 456 name
Group 3: 101
```





Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Here is a three numeric digit sample With or Without <dot> characters: cont.

Using: file_1_ 456 name.101.tif

Match: (.+?)(.+?\.?)(\d{3})

Where:

\1 = file_1_ <space> \2 = 456 name. \3 = 101

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. <space>	Match against a <space> character	Capture Group 1 = file_1 <space>
5..	Match against any character	Capture Group 2 = 4
6. +	Make it Greedy	Capture Group 2 = 456name.101
7. ?	But not too Greedy	Capture Group 2 = 4
8. \.	Match against literal <dot>	Capture Group 2 = 456name.
9. ?	Make it optional. For strings that do not have a <dot> Character.	Capture Group 2 = 4
10. \d{3}	Match against 1 numeric digit Repeat 3 iterations. Moves Forward.	Capture Group 3 = 101 Changes Capture Group 2 = 456name.

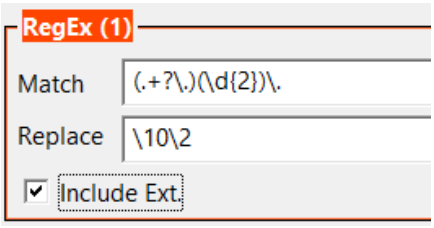
Tim Notes:

1. Limited.
2. Be careful that it doesn't inadvertently renumber undesired files.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



Include Ext: Enable
.. because there are dot characters in string

Multiple Digits within the same string - Single numeric digits:

file_1_ 5-name.1.tif to file_1_ 05-name.01.tif

Match: (.+?)(.+?\.)(\d{1})\.
Replace: \10\2 0\3

Name	New Name
file_1_name.101.tif	file_1_name.101.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 5-name.1.tif	file_1_ 05-name. 01.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif

Tim Notes:

1. Same RegEx as the one I used for Three Numeric Digits – all I had to do was change the Quantifier for the numeric digit to {1}. Additional Visualizations to represent this change would be redundant. Also refer to the 'Three Numeric Digits' for the analysis.

This same RegEx will work with Mixed!

file_1_ 25 name.1.tif to file_1_025 name.01.tif

Match: (.+?)(.+?\.)(\d{1})\.
Replace: \10\2 0\3

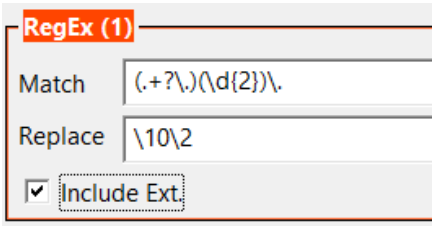
Name	New Name
file_1_name.101.tif	file_1_name.101.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 5-name.1.tif	file_1_ 05-name. 01.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 25 name.1.tif	file_1_ 025 name. 01.tif

It matches both the Mixed and the Single numeric digit samples!

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



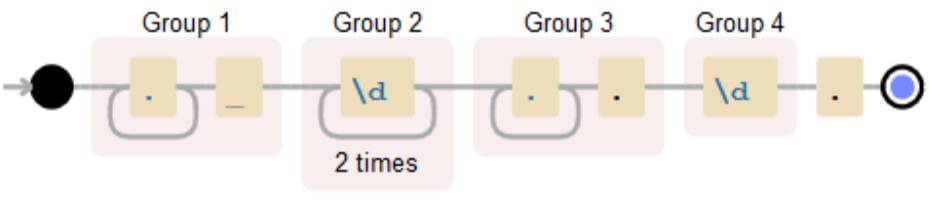
Include Ext: Enable
.. because there are dot characters in string

To Further restrict the sample use this:

Match: (.+?)(\d{2})(.+?\.)(\d{1})\.
Replace: \10\2\3 0\4

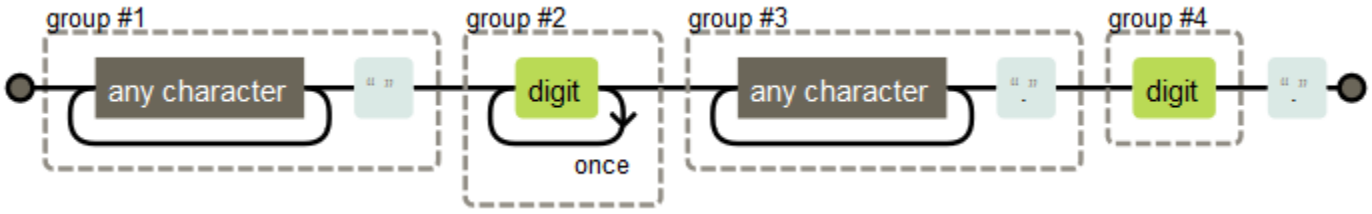
Name	New Name
file_1_name.101.tif	file_1_name.101.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 5-name.1.tif	file_1_ 5-name.1.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 25 name.1.tif	file_1_ 025 name. 01.tif

Now it will only match those files that have a mix of 2 numeric digits and 1 numeric digit. Change the \d{2} and \d{1} in the string to determine the mix you want to restrict.



Sample:

```
Match 1: file_1_ 25 name.1.
Group 1: file_1_
Group 2: 25
Group 3:  name.
Group 4: 1
```



Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

To Further restrict the sample use this: cont.

Using: file_1_ 25 name.1.tif

Match: (.+?)(\d{2})(.+?\.)(\d{1})\.

Where:

\1 = file_1 <space> \2 = 25 \3 = <space>name. \4 = 1

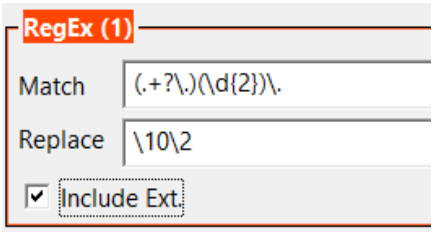
Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. <space>	Match against a <space> character	Capture Group 1 = file_1<space>
5. \d{2}	Match against 1 numeric digit Repeat 2 iterations. Moves Forward.	Capture Group 2 = 25
6..	Match against any character This is the <space> after '25'	Capture Group 3 = <space>
7. +	Make it Greedy	Capture Group 3 = <space> name.1
8. ?	But not too Greedy	Capture Group 3 = <space>
9. \.	Match against literal <dot>	Capture Group 3 = <space> name.
10. \d{1}	Match against 1 numeric digit Repeat 1 iteration. Moves Forward.	Capture Group 4 = 1
11. \.	Match against literal <dot> (Not Captured) This is the <dot> of the extension.	

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



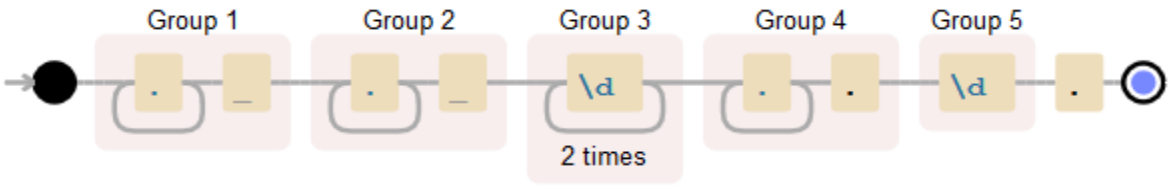
Include Ext: Enable
.. because there are dot characters in string

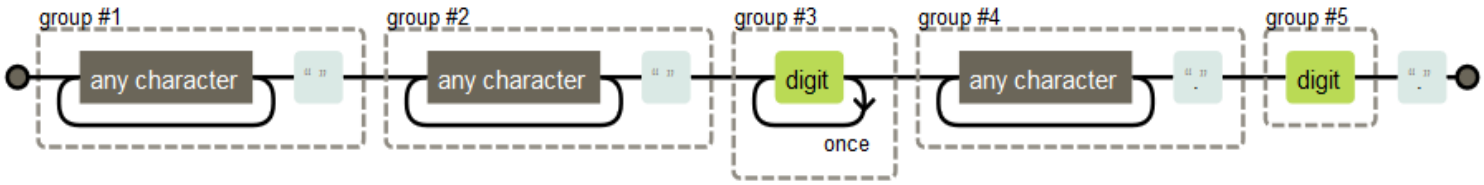
How about a mix of three??

file_1_ 7 25 name.1.tif to file_1_ 007 025 name.001.tif

Match: (.+?)(.+?)(\d{2})(.+?\.)(\d{1})\.
Replace: \10\20\3\40\5

Name	New Name
file_1_name.101.tif	file_1_name.101.tif
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 7 25 name.1.tif	file_1_ 007 025 name.001.tif
file_1_ 5-name.1.tif	file_1_ 5-name.1.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 25 name.1.tif	file_1_ 25 name.1.tif





Sample:

```
.................................
Match 1: file_1_ 7 25 name.1.
Group 1: file_1_
Group 2: 7
Group 3: 25
Group 4:  name.
Group 5: 1
```

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

How about a mix of three?? cont.

Using: file_1_ 7 25 name.1.tif

Match: (.+?)(.+?)(\d{2})(.+?\.)(\d{1})\.

Where:

\1 = file_1_ <space> \2 = 7 <space> \3 = 25 \4 = <space> name. \5 = 1

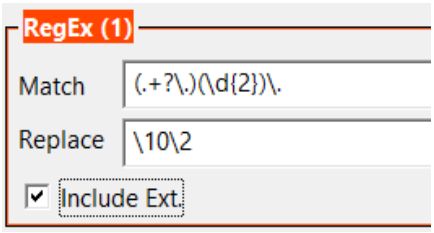
Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. <space>	Match against a <space> character	Capture Group 1 = file_1<space>
5..	Match against any character	Capture Group 2 = 7
6. +	Make it Greedy	Capture Group 2 = 7 25 name.1
7. ?	But not too Greedy	Capture Group 2 = 7
8. <space>	Match against a <space> character	Capture Group 2 = 7 <space>
9. \d{2}	Match against 1 numeric digit Repeat 2 iterations. Moves Forward.	Capture Group 3 = 25
10..	Match against any character This is the <space> after '25'	Capture Group 4 = <space>
11. +	Make it Greedy	Capture Group 4 = <space> name.1
12. ?	But not too Greedy	Capture Group 4 = <space>
13. \.	Match against literal <dot>	Capture Group 4 = <space> name.
14. \d{1}	Match against 1 numeric digit Repeat 1 iteration. Moves Forward.	Capture Group 5 = 1
15. \.	Match against literal <dot> (Not Captured) This is the <dot> of the extension.	

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



Include Ext: Enable
.. because there are dot characters in string

Appending more than one zero:

file_1_ 7 25 name.1.tif to file_1_ 007 25 name.001.tif

Of course this can be done selectively using the Replace string:

Replace: \100\20\3\400\5

Easy peasy!

You have complete control now and it can all be done in one pass. You are selecting exactly which part of the numeric digits in the string get appended and with what.

Appending more than just the end of the filename is hard to do in one pass. My technique is more restrictive to the number of numeric digits. I don't believe it can be made more universal without requiring two passes.

In the Mix of 4 (see next page), I substituted \d? which helped, but that won't work with some of these other RegEx because it matched against other files I didn't want to include.

Append only single numeric digits in mixed:

file_1_ 7 25 name.1.tif to file_1_ 07 25 name.01.tif

Replace: \10\2\3\4 0\5

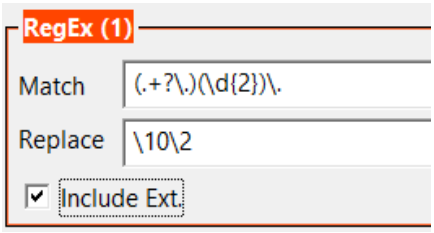
Tim Notes:

1. If the file names do not have dot characters, just remove the references to ' \. ' in the RegEx, i.e., use:
   `(.+?)(.+?)(\d{2})(.+?)(\d{1})` instead of `(.+?)(.+?)(\d{2})(.+?\.)(\d{1})\.`

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

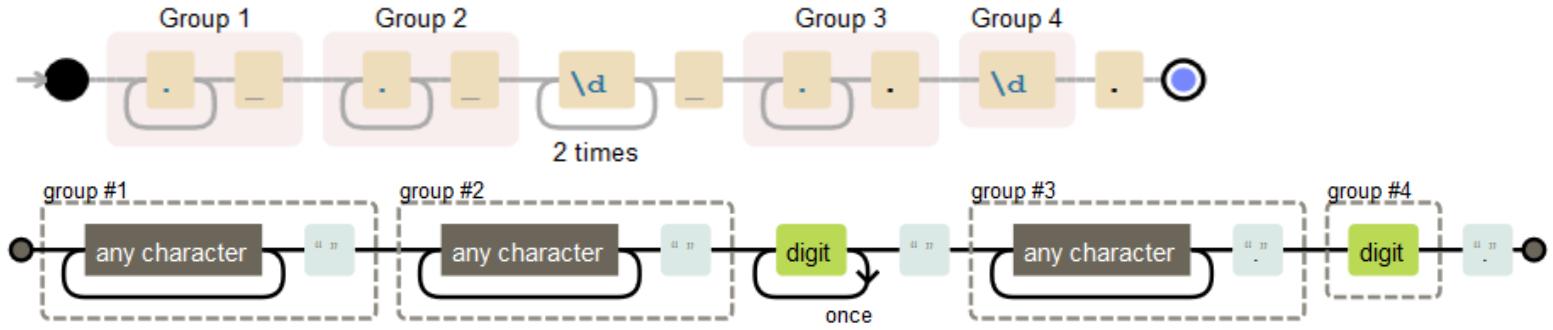


Include Ext: Enable
.. because there are dot characters in string

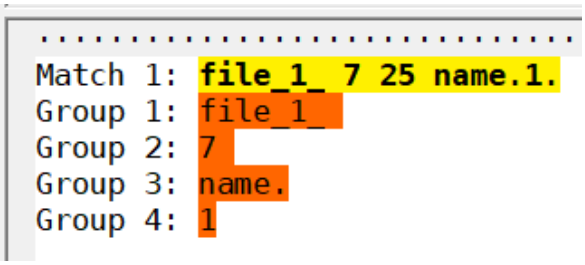
Remove the two numeric digit numbers from string:

file_1_ 7 25 name.1.tif to file_1 07 name.01.tif

Match: (.+?)(.+?)\d{2} (.+?\.)(\d{1})\.
Replace: \10\2\30\4



Sample:



Tim Notes:

1. Removed \d{2} from Capture Group and added a following <space> character after the {2}

original: Match: (.+?)(.+?) (\d{2}) (.+?\.)(\d{1})\.
New: Match: (.+?)(.+?) \d{2} (.+?\.)(\d{1})\.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Remove the two numeric digit numbers from string: cont.

Using: file_1_ 7 25 name.1.tif

Match: (.+?)(.+?)\d{2} (.+?\.)(\d{1})\.

Where:

\1 = file_1_ <space> \2 = 7 <space> \3 = name. \4 = 1

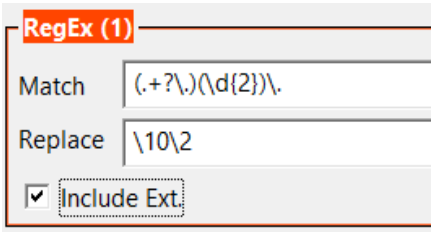
Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. <space>	Match against a <space> character	Capture Group 1 = file_1<space>
5..	Match against any character	Capture Group 2 = 7
6. +	Make it Greedy	Capture Group 2 = 7 25 name.1
7. ?	But not too Greedy	Capture Group 2 = 7
8. <space>	Match against a <space> character	Capture Group 2 = 7 <space>
9. \d{2}	Match against 1 numeric digit (Not Captured) Repeat 2 iterations. Moves Forward. = '25'	
10. <space>	Match against a <space> character (Not Captured) = '25 <space>'	
11..	Match against any character This is the <space> after '25'	Capture Group 3 = n
12. +	Make it Greedy	Capture Group 3 = name.1
13. ?	But not too Greedy	Capture Group 3 = n
14. \.	Match against literal <dot>	Capture Group 3 = name.
15. \d{1}	Match against 1 numeric digit Repeat 1 iteration. Moves Forward.	Capture Group 4 = 1
16. \.	Match against literal <dot> (Not Captured) This is the <dot> of the extension.	

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.



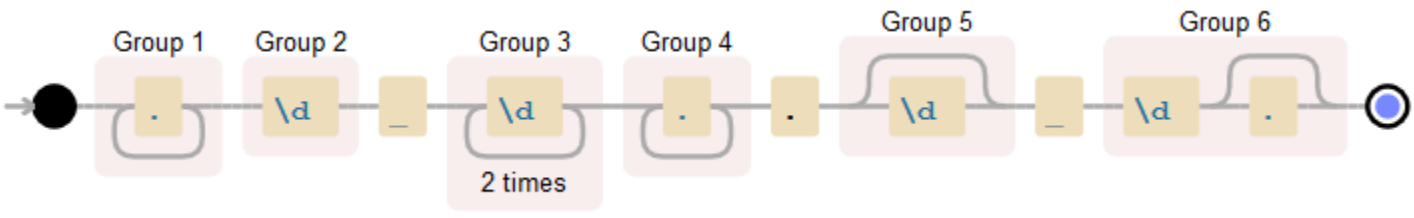
Include Ext: Enable
.. because there are dot characters in string

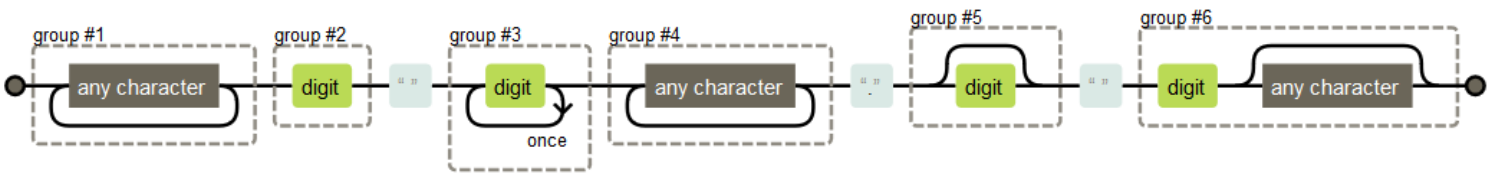
Finally here is a mixture of 4:

file_1_ 7 25 name.1 18.tif to file_1_ 07 025 name 01 018.tif

Match: (.+?)(\d{1}) (\d{2})(.+?)\.(\d?) (\d.?)
Replace: \10\2 0\3 \4 0\5 0\6

Name	New Name
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 7 25 name.1.tif	file_1_ 7 25 name.1.tif
file_1_ 7 25 name.1 18.tif	file_1_ 07 025 name 01 018.tif
file_1_ 5-name.1.tif	file_1_ 5-name.1.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 25 name.1.tif	file_1_ 25 name.1.tif





Tim Notes:

1. Extra <space> character between '025' and 'name' can be removed – it is in the Replace String as, '\3 <space>\4', above.

Add

Pad Double digit at end of string (in one pass!) cont.

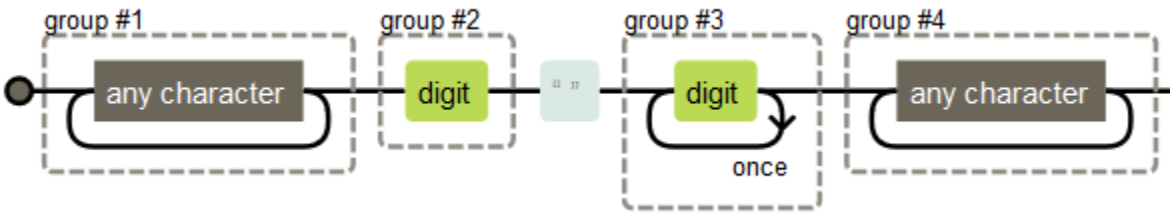
Tim Contribution – cont.

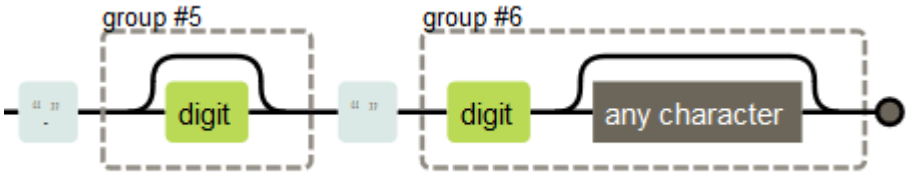
Finally here is a mixture of 4: cont.

file_1_ 7 25 name.1 18.tif to file_1_ 07 025 name 01 018.tif

Match: (.+?)(\d{1}) (\d{2})(.+?)\.(\d?) (\d.?)

Expanded:





Sample:

```
.................................
Match 1: file_1_ 7 25 name.1 18
Group 1: file_1_
Group 2: 7
Group 3: 25
Group 4:  name
Group 5: 1
Group 6: 18
```

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Finally here is a mixture of 4: cont.

Using: file_1_ 7 25 name.1 18.tif

Match: (.+?)(\d{1}) (\d{2})(.+?)\.(\d?) (\d.?)

Where:

\1 = file_1_<space> \2 = 7 \3 = 25 \4 = <space> name \5 = 1 \6 = 18

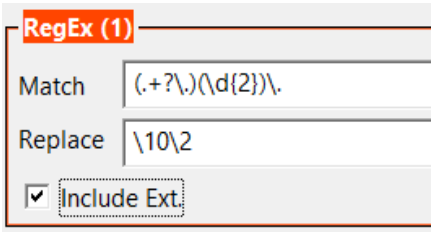
Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. \d{1}	Match against 1 numeric digit Repeat 1 iterations. Moves Forward.	Capture Group 2 = 1 Changes Capture Group 1 = file_
5. <space>	Match against a <space> character (Not Captured)	Changes Capture Group 2 = 7 Changes Capture Group 1 = file_1_<space>
6. \d{2}	Match against 1 numeric digit Repeat 2 iterations. Moves Forward	Capture Group 3 = 25
7..	Match against any character	Capture Group 4 = <space>
8. +	Make it Greedy	Capture Group 4 = <space> name.1 18
9. ?	But not too Greedy	Capture Group 4 = <space>
10. \.	Match against literal <dot> (Not Captured)	Changes Capture Group 4 = <space> name
11. \d	Match against a numeric digit	Capture Group 5 = 1
12. ?	Make it Greedy optional	Capture Group 5 = Unchanged
13. <space>	Match against a <space> character (Not Captured) This is the <space> after 'name.1'	
14. \d	Match against a numeric digit This is the '1' of '18'	Capture Group 6 = 1
15..	Match against any character	Capture Group 6 = 18
16. ?	Make it Greedy optional	Capture Group 6 = Unchanged

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

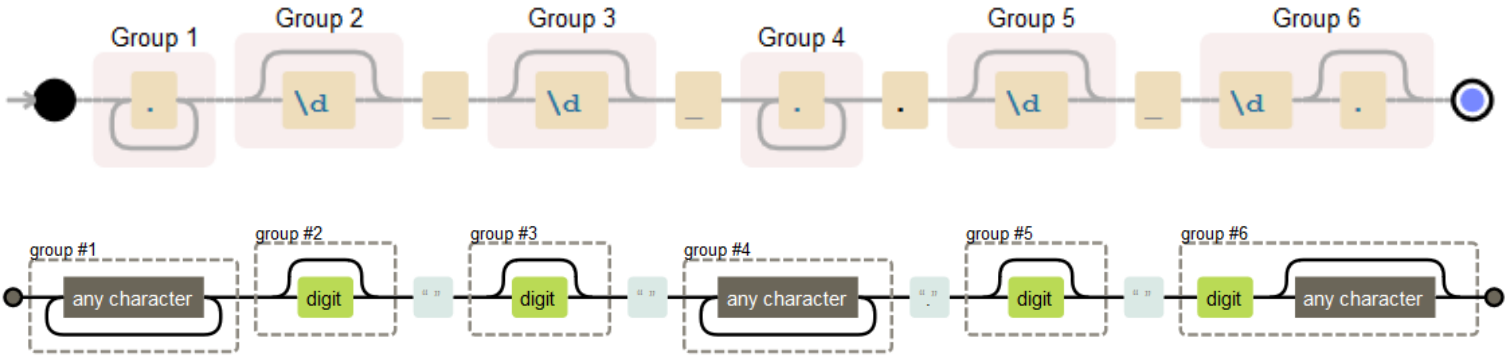


Include Ext: Enable
.. because there are dot characters in string

Make it more universal:

Match: (.+?)(\d?) (\d?) (.+?)\.(\d?) (\d.?)
Replace: \1\2 0\3 0\4 0\5 0\6

Name	New Name
file_1_name.02.tif	file_1_name.02.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_ 7 25 name.1.tif	file_1_ 7 25 name.1.tif
file_1_ 7 25 name.1 18.tif	file_1_ 07 025 name 01 018.tif
file_1_ 5-name.1.tif	file_1_ 5-name.1.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 25 name.1.tif	file_1_ 25 name.1.tif



Sample:

```
·····································Start·Length···
Match 1: file_1_ 7 25 name.1 18          0      22
Group 1: file_1_                         0       7
Group 2:                                 7       0
Group 3: 7                               8       1
Group 4: 25 name                        10       7
Group 5: 1                              18       1
Group 6: 18                             20       2
```

Add

Pad Double digit at end of string (in one pass!) cont.

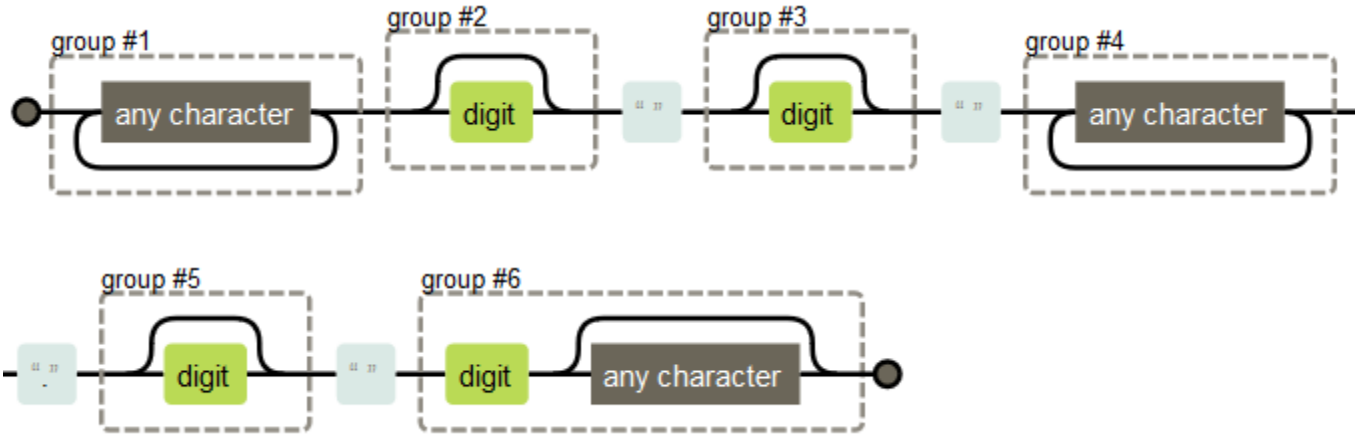
Tim Contribution – cont.

Make it more universal: cont.

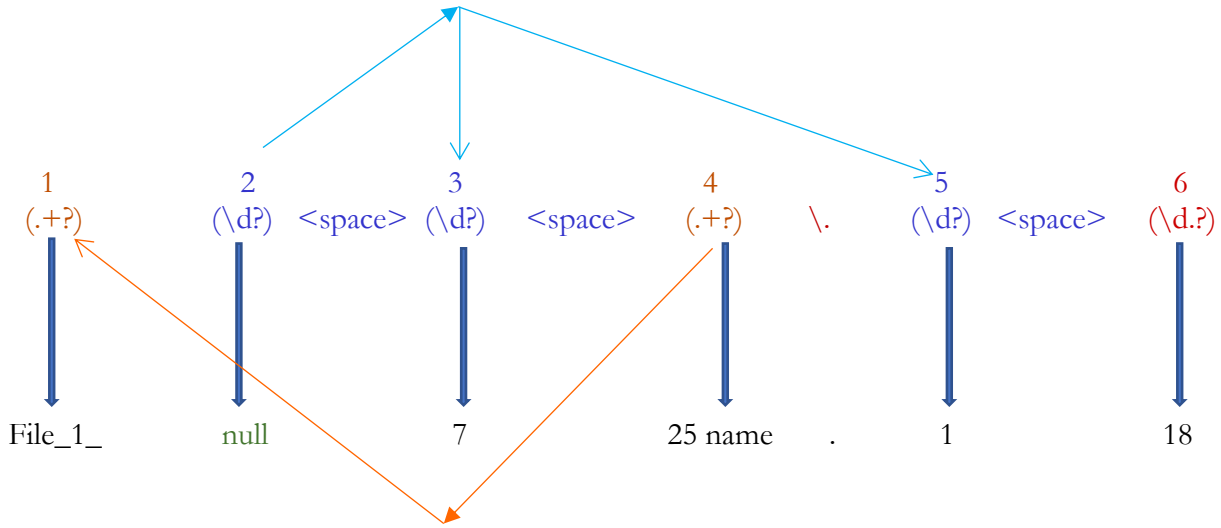
file_1_ 7 25 name.1 18.tif

Match: (.+?)(\d?) (\d?) (.+?)\.(\d?) (\d.?)

Expanded:



Breaks down:



Tim Notes:

1. When you break it down, you can see the repetition.

Add

Pad Double digit at end of string (in one pass!) cont.

Tim Contribution – cont.

Make it more universal: cont.

Using: file_1_ 7 25 name.1 18.tif

Match: (.+?)(\d?) (\d?) (.+?)\.(\d?) (\d.?)

Where:

\1 = file_1_ \2 = null \3 = 7 \4 = 25 name \5 = 1 \6 = 18

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. \d	Match against a numeric digit	Capture Group 2 = 1 Changes Capture Group 1 = file_
5. ?	Make it Greedy optional Backtracks just after the 'e' of 'file'	Capture Group 2 = null Changes Capture Group = f
6. <space>	Match against a <space> character (Not Captured)	Changes Capture Group 1 = file_1_
7. \d	Match against a numeric digit	Capture Group 3 = 7
8. ?	Make it Greedy optional	Capture Group 3 = Unchanged
9. <space>	Match against a <space> character (Not Captured)	All Capture Groups = Unchanged
10..	Match against any character	Capture Group 4 = 2
11. +	Make it Greedy	Capture Group 4 = 25 name.1 18
12. ?	But not too Greedy	Capture Group 4 = 2
13. \.	Match against literal <dot> (Not Captured)	Changes Capture Group 4 = 25 name
14. \d	Match against a numeric digit This is the '1' of 'name.1'	Capture Group 5 = 1
15. ?	Make it Greedy optional	Capture Group 5 = Unchanged
16. <space>	Match against a <space> character (Not Captured)	Changes Capture Group 2 = 7
17. \d	Match against a numeric digit This is the '1' of '18'	Capture Group 6 = 1
18..	Match against any character	Capture Group 6 = 18
19. ?	Make it Greedy optional	Capture Group 6 = Unchanged

Add

Pad Zeros to Single Numeric (in One Pass!)

Tim Notes:

1. This is yet again another example of adding Zeros to a single numeric digit at end of string in one pass.
2. Demonstrates the use of \D which matches against any single non-numeric digit (anything but a numeral)
3. Unlike a previous RegEx that used a technique of including the 'extension as part of the string', this one does not require that option.

file1.ext	to	file01.ext
file 1.ext	to	file 01.ext
file11.ext	to	Ignore
file003.ext	to	Ignore

Match: (.+\D\s*)(\d)$
Replace: \10\2

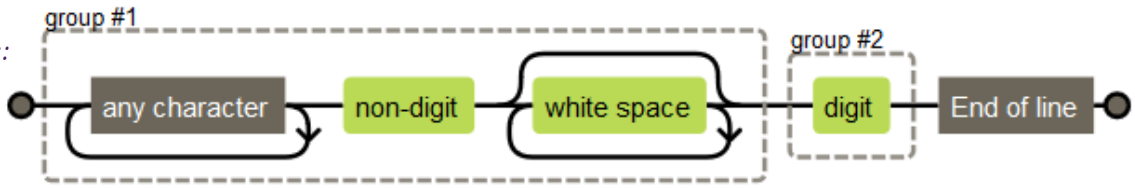
Name	New Name
file 1.ext	file 01.ext
file1.ext	file01.ext
file11.ext	file11.ext
file003.ext	file003.ext

Sample:

```
Match 1: file 1
Group 1: file
Group 2: 1
```

Forum Contributor: Stefan

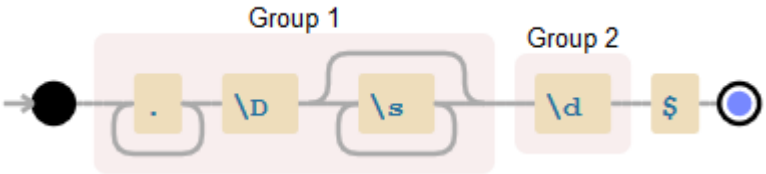
Contributor and *Tim Analysis:*



Group 1

1. .+ Capture all until…
2. \D … a non numeric digit
3. \s* followed by a <space> character(s)

Group 2



4. \d Matches against a single numeric digit
5. $ Last expression must have matched at EOL

Where:

file1.ext	file 1.ext
\1 = file	\1 = file <space>
\2 = 1	\2 = 1

Tim Notes:

1. See my previous contribution under, 'Pad Double digit at end of string (in one pass!)' in this section to see how those ideas can be adopted to Appending using this RegEx as well.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

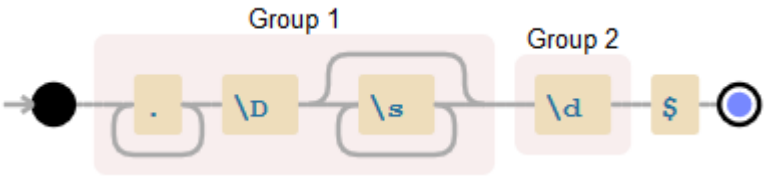
Tim Contribution –

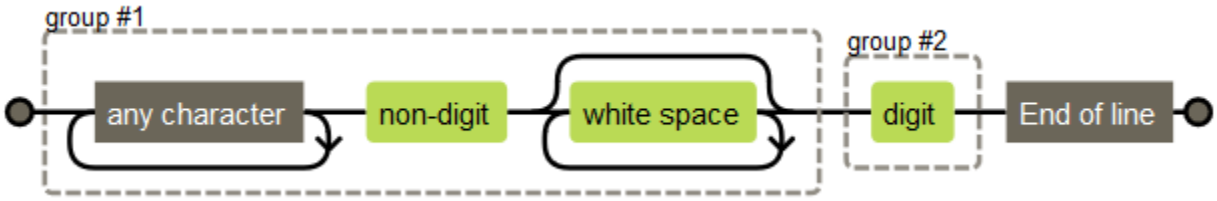
Single Numeric Digits (slight modification to Contributor's to show the progressions that follow)

Using: file1.ext

Match: (.+\D\s*)(\d{1})$
Replace: \10\2

Name	New Name
file_1_ 7 25 name.1 18.tif	file_1_ 7 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_1_ 7 25 name.01.tif
file_1_ 25 name.1.tif	file_1_ 25 name.01.tif
file11.ext	file11.ext
file1.ext	file01.ext
file 1.ext	file 01.ext
file003.ext	file003.ext
2file.ext	2file.ext





Sample: file 1.ext

```
.................
Match 1: file 1
Group 1: file
Group 2: 1
```

Sample: file1.ext

```
.................
Match 1: file1
Group 1: file
Group 2: 1
```

Tim Analysis:

1. 1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \D	Match against any non-digit numeric	Capture Group 1 = file <space>
4. \s	Match against a <space> character	Capture Group 1 = No Value
5. *	Make it Greedy	Capture Group 1 = file
6. \d	Match against Numeric Digit	Capture Group 2 = 1
7. $	Must be at EOL (End of Line)	

Add

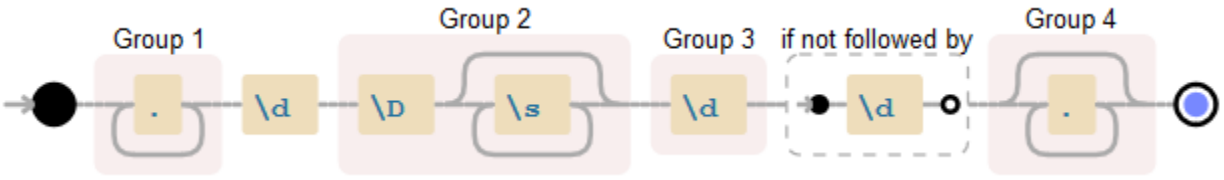
Pad Zeros to Single Numeric (in One Pass!) cont.

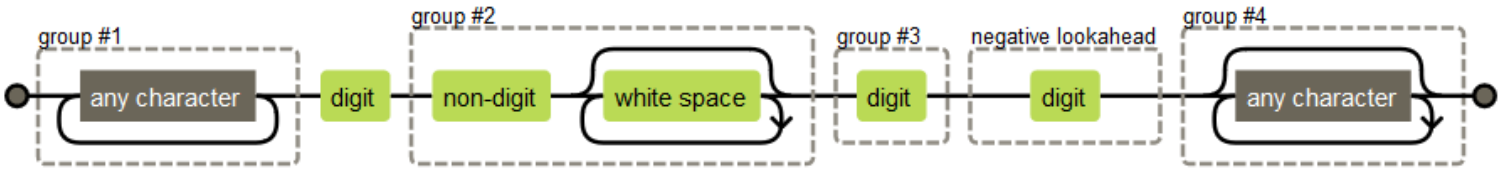
Tim Contribution – cont.

Single Numeric Digit as first numeric group in string:

Match: (.+?)\d(\D\s*)(\d{1})(?!\d{1})(.*)
Replace: \1\20\3\4

Name	New Name
file_1_ 7 25 name.1 18.tif	file__ 07 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file__ 07 25 name.1.tif
file_1_ 25 name.1.tif	file_1_ 25 name.1.tif
file11.ext	file11.ext
file1.ext	file1.ext
file 1.ext	file 1.ext
file003.ext	file003.ext
2file.ext	2file.ext





Sample:

		Start	Length
Match 1:	file_1_7 25 name.1 18	0	21
Group 1:	file_	0	5
Group 2:	_	6	1
Group 3:	7	7	1
Group 4:	25 name.1 18	8	13

Tim Notes:

1. The file, 'file_1', I did not consider to be a numeric, but text. Therefore I did not try to match against it.
2. For any examples that have dot characters in the filename will require, 'Rename File Extensions As Being Part of File Name', enabled from the Renaming Options Menu.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Single Numeric Digit as first numeral in string: cont.

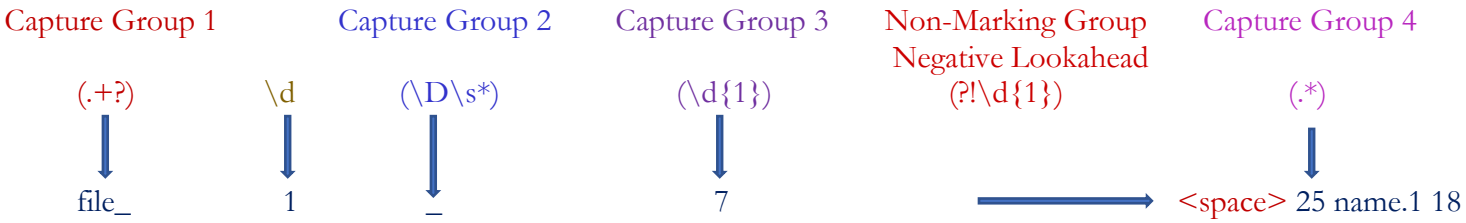
Using: file_1_7 25 name.1 18.tif

Match: (.+?)\d(\D\s*)(\d{1})(?!\d{1})(.*)

Where:

\1 = file_ \2 = _ \3 = 7 \4 = <space> 25 name.1 18

Breaks down:



Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = f
4. \d	Match against Numeric Digit (Not Captured)= '1' This is the '1' after 'file_'	Changes Capture Group 1 = file_
5. \D	Match against any non-digit numeric	Capture Group 2 = _
6. \s	Match against a <space> character	Capture Group 2 = No Value
7. *	Make it Greedy	Capture Group 2 = _
8. \d{1}	Match against 1 numeric digit Repeat 1 iteration.	Capture Group 3 = 7
9. (?!\d{1})	Lookahead. Look right that there is *not* a numeric digit. Then expression = True and evaluation Continues = <space> = TRUE	
10. .*	Match against remaining string	Capture Group 4 = <space> 25 name.1 18

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

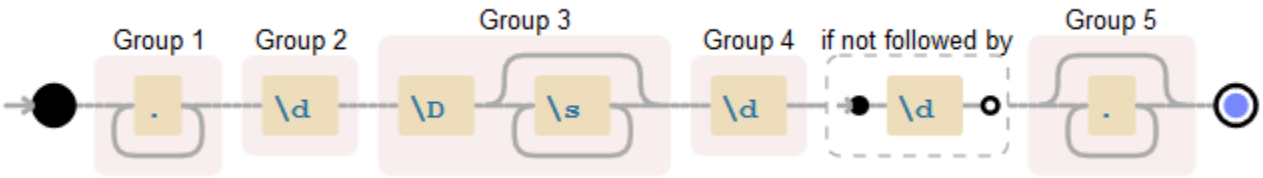
Tim Contribution – cont.

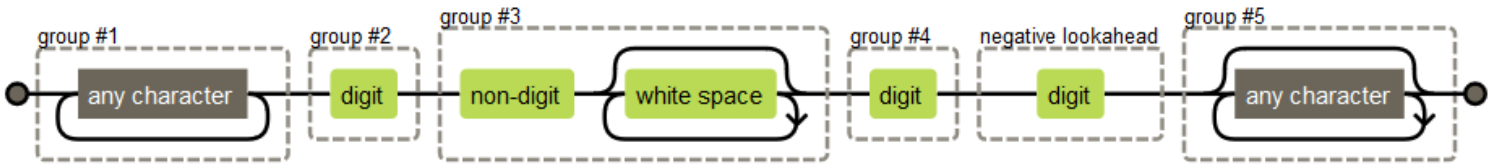
If you wanted to add the '1' of 'file_', just change it to this:

Using: file_1_7 25 name.1 18.tif

Match: (.+?)(\d)(\D\s*)(\d{1})(?!\d{1})(.*)
Replace: \10\2\30\4\5

Name	New Name
file_1_ 7 25 name.1 18.tif	file_01_ 07 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 25 name.1.tif
file_1_ 25 name.1.tif	file_1_ 25 name.1.tif
file11.ext	file11.ext
file1.ext	file1.ext
file 1.ext	file 1.ext
file003.ext	file003.ext
2file.ext	2file.ext





Sample:

	Start	Length
Match 1:	0	21
Group 1:	0	5
Group 2:	5	1
Group 3:	6	1
Group 4:	7	1
Group 5:	8	13

Tim Notes:

1. No additional analysis required. The only change was taking the \d not captured and capture it (\d). Replacement String was also changed to reflect the added Capture Group.

Add

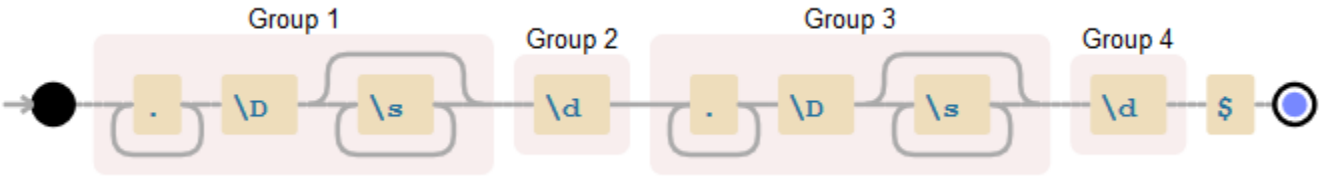
Pad Zeros to Single Numeric (in One Pass!) cont.

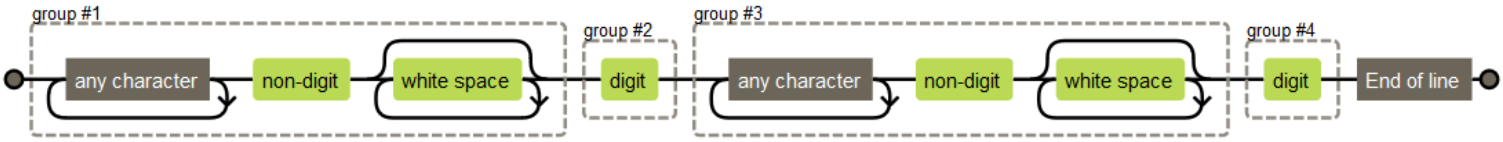
Tim Contribution – cont.

Double Numeric Digits:

Match: (.+\D\s*)(\d{1})(.+\D\s*)(\d{1})$
Replace: \10\2\3

Name	New Name
file_1_ 7 25 name.1 18.tif	file_1_ 7 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_1_ 7 025 name..tif
file_1_ 25 name.1.tif	file_1_ 025 name..tif
file11.ext	file11.ext
file1.ext	file1.ext
file 1.ext	file 1.ext
file003.ext	file003.ext
2file.ext	2file.ext





Tim Notes:

1. There are extra dot characters at the end of the Renamed string. If you don't want this, just change the RegEx to include matching the dot character without capturing it:

Current: Match (.+\D\s*)(\d{1})(.+\D\s*)(\d{1})(.*)
Modified: Match: (.+\D\s*)(\d{1})(.+\D\s*)\.(\d{1})(.*)

2. Make sure that you do not have RegEx set to include Extension. It won't work.

Add

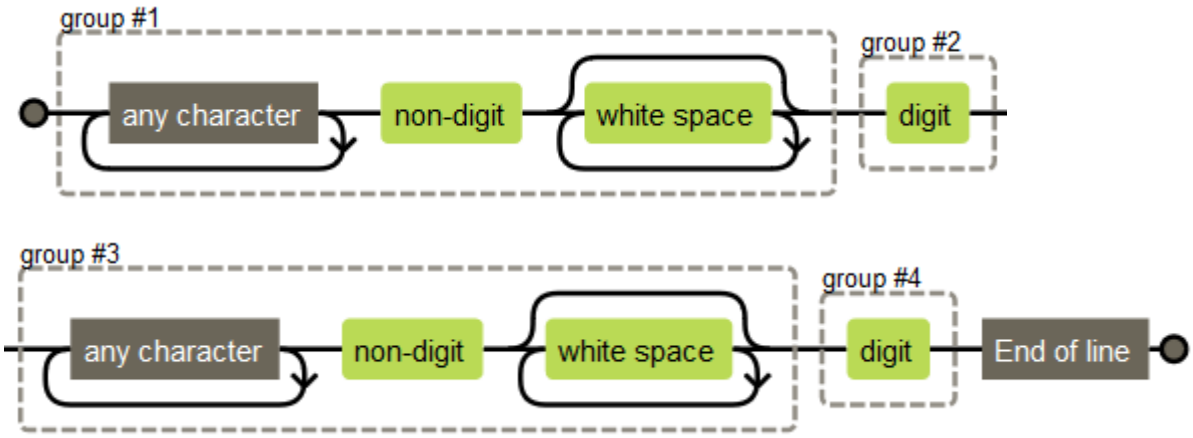
Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Double Numeric Digits: cont.

Match: (.+\D\s*)(\d{1})(.+\D\s*)(\d{1})$

Expanded:



Sample: Current

```
.............................
Match 1: file_1_ 7 25 name.1
Group 1: file_1_ 7
Group 2: 2
Group 3: 5 name.
Group 4: 1
```

Sample: Modified

```
.............................
Match 1: file_1_ 7 25 name.1
Group 1: file_1_ 7
Group 2: 2
Group 3: 5 name
Group 4: 1
```

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Double Numeric Digits: cont.

Using: file_1_7 25 name.1 18.tif

Match: (.+\D\s*)(\d{1})(.+\D\s*)(\d{1})$

Where:

\1 = file_1_ 7 <space> \2 = 2 \3 = 5 name. \4 = 1

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. \D	Match against any non-digit numeric Backtracks to the <space> after 'name.1'	Capture Group 1 = file_1_7 25 name.1<space>
4. \s	Match against a <space> character	Capture Group 1 = No Value
5. *	Make it Greedy	Capture Group 1 = file_1_7 25 name.1<space>
6. \d{1}	Match against 1 numeric digit Repeat 1 iteration. Backtracks to Match against the '1' of '18'	Capture Group 2 = 1
7..	Match against any character	Capture Group 3 = 8
8. +	Make it Greedy	Capture Group 3 = Unchanged
9. \D	Match against any non-digit numeric	Capture Group 3 = 5 name.1 <space> Changes Capture Group 2 = 2 Changes Capture Group 1 = file_1_7<space>
10. \s	Match against a <space> character	Capture Group 3 = No Value
11. *	Make it Greedy	Capture Group 3 = 5 name.1<space>
12. \d{1}	Match against 1 numeric digit Repeat 1 iteration.	Capture Group 4 = 1 Changes Capture Group 3 = 5 name.
13. $	Must be at EOL (End of Line)	

Tim Notes:

1. No Value usually refers to a non-match, at least for the moment in the evaluation.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #1:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5			

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$
Replace: \10\2\30\4\5

	New Name
file_.01._name.01.tif	file_.001._name.001.tif
file_.01_name.01.tif	file_.001_name.001.tif
file_01_name.01.tif	file_001_name.001.tif
file_01_name.1.tif	file_001_name.01.tif
file_01_name01.tif	file_001_name001.tif
file_1 456name.101.tif	file_01 0456name.101.tif
file_123_test101_name201.tif	file_0123_test0101_name201.tif
file_1_ 4 name101 347 text.123.tif	file_01_ 04 name101 347 text.123.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.101.tif
file_1_ 456 name101.tif	file_01_ 0456 name101.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 25 name.1.tif
file_1_7 25 name.1 18.tif	file_01_07 25 name.1 18.tif
file_1_name.01.tif	file_01_name.001.tif
file_1_name.1.tif	file_01_name.01.tif
file_1_name.101.tif	file_01_name.0101.tif
file_1_name01.tif	file_01_name001.tif
file_1_name1.tif	file_01_name01.tif
file_1_name101.tif	file_01_name0101.tif
file_1_test1_name01.tif	file_01_test01_name01.tif
file_1_test1_name1.tif	file_01_test01_name1.tif
text 1 2 3 4 5.tif	text 01 02 3 4 5.tif

Contributor Notes:

1. Add 0's in replacement to increase prefix.

Add

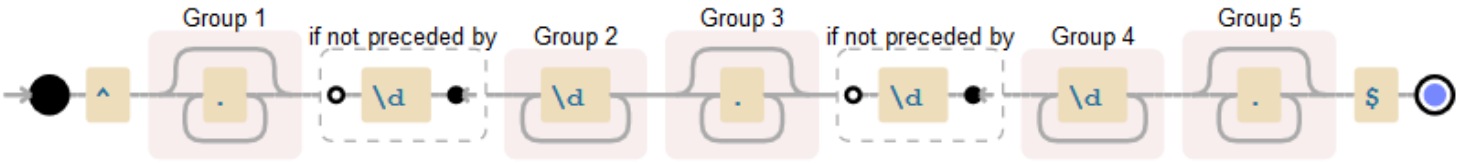
Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #1: cont.

Samples:

file_.01._name.01
file_01_name01
file_1_ 456 name101
file_1_name.01
file_1_name1
text 1 2 3 4 5
file_.01_name.01
file_1 456name.101
file_1_ 7 25 name.1
file_1_name.1
file_1_name101
file_01_name.01
file_123_test101_name201
file_1_ 7 25 name.1 18
file_1_name.101
file_1_test1_name01
file_01_name.1
file_1_ 456 name.101
file_1_7 25 name.1 18
file_1_name01
file_1_test1_name1

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$



Sample:

		Start	Length
Match 1:	file_01_name01	0	14
Group 1:	file_	0	5
Group 2:	01	5	2
Group 3:	_name	7	5
Group 4:	01	12	2
Group 5:		14	0

Sample:

		Start	Length
Match 1:	file_.01._name.01	0	17
Group 1:	file_.	0	6
Group 2:	01	6	2
Group 3:	._name.	8	7
Group 4:	01	15	2
Group 5:		17	0

Sample:

		Start	Length
Match 1:	file_1_7 25 name.1 18	0	21
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_	6	1
Group 4:	7	7	1
Group 5:	25 name.1 18	8	13

Tim Notes:

1. Be careful that it doesn't inadvertently renumber undesired files.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #1: cont.

Samples:

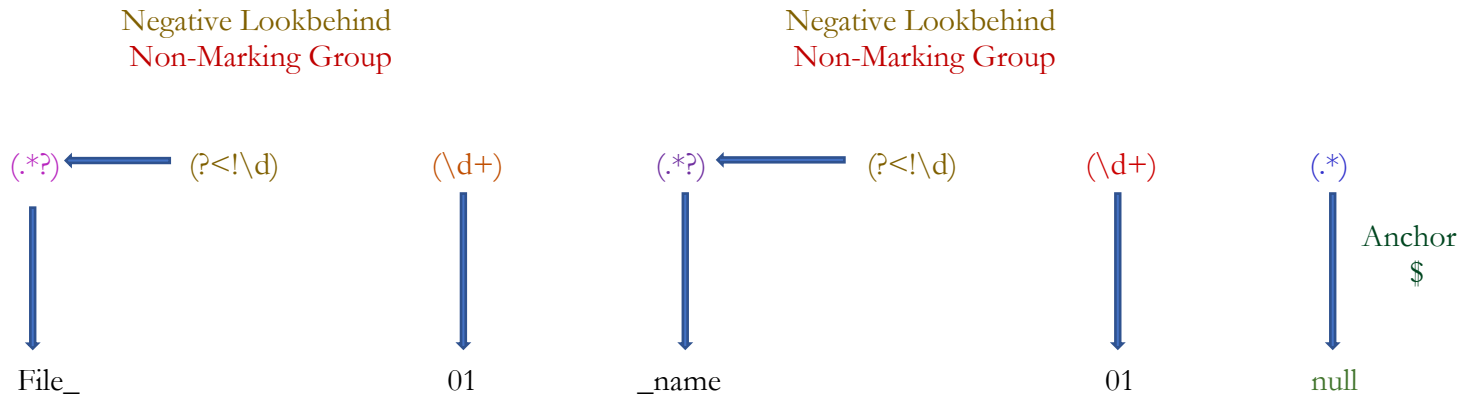
file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5			

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Breaks down:

Anchor
^

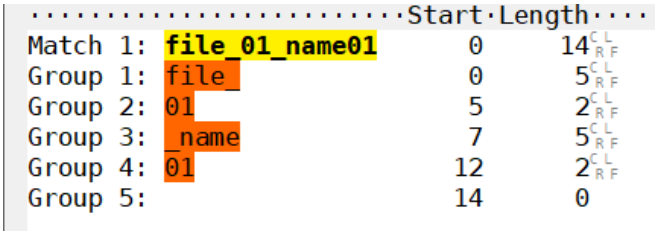
Capture Group 1 Capture Group 2 Capture Group 3 Capture Group 4 Capture Group 5



Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #1: cont.



Using file_01_name01

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Where:

\1 = file_ \2 = 01 \3 = _name \4 = 01 \5 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 =
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. (?<!\d)	Negative Lookbehind. Look to the left that the last character is 'not' a numeric digit. = BOL = True	
6. \d	Match against Numeric Digit	Capture Group 2 = 0 Changes Capture Group 1= file_.
7. +	Make it Greedy	Capture Group 2 = 01
8..	Match against any character This is the underscore after 'file_.01'	Capture Group 3 = _
9. *	Make it Greedy	Capture Group 3 = _name.01
10. ?	But not too Greedy Moves to the position just before the second underscore of 'file_.01_'	Capture Group 3 = null
11. (?<!\d)	Negative Lookbehind. Look to the left that the last Character is not a numeric digit. = underscore = True	Restores Capture Group 3 = _
12. \d	Match against Numeric Digit This is the '0' of _name.01'	Capture Group 4 = 0 Changes Capture Group 3 = _name.
13. +	Make it Greedy Current position = EOL	Capture Group 4 = 01
14. .*	Match against remaining string There are no more characters.	Capture Group 5 = null
15. $	Must be at EOL (End of Line)	

Tim Notes:

1. The first Lookbehind is not needed because it will always be BOL so there is no need to check for a numeric digit.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix numeric digits in strings that have only 2 numeric groups:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt		

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$
Replace: \10\2\30\4\5

	New Name
45 52.txt	045 052.txt
file_.01._name.01.tif	file_.001._name.001.tif
file_.01_name.01.tif	file_.001_name.001.tif
file_01_name.01.tif	file_001_name.001.tif
file_01_name.1.tif	file_001_name.01.tif
file_01_name01.tif	file_001_name001.tif
file_1 456name.101.tif	file_1 456name.101.tif
file_123_test101_name201.tif	file_123_test101_name201.tif
file_1_ 4 name101 347 text.123~{}.tif	file_1_ 4 name101 347 text.123~{}.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 456 name101.tif	file_1_ 456 name101.tif
file_1_ 7 25 name.1 18.tif	file_1_ 7 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_1_ 7 25 name.1.tif
file_1_7 25 name.1 18.tif	file_1_7 25 name.1 18.tif
file_1_name.01.tif	file_01_name.001.tif
file_1_name.1.tif	file_01_name.01.tif
file_1_name.101.tif	file_01_name.0101.tif
file_1_name01.tif	file_01_name001.tif
file_1_name1.tif	file_01_name01.tif
file_1_name101.tif	file_01_name0101.tif
file_1_test1_name01.tif	file_1_test1_name01.tif
file_1_test1_name1.tif	file_1_test1_name1.tif
text 1 2 3 4 5.tif	text 1 2 3 4 5.tif

Contributor Notes:

1. Add 0's in replacement to increase prefix.
2. The Atomic Groups keep numeric digit-strings from being split (as if there's only 1-string, but you're processing 2-string files).
3. By holding onto their match, they prevent [^\d]* from matching any null-occurrences in between numeric digits.

Add

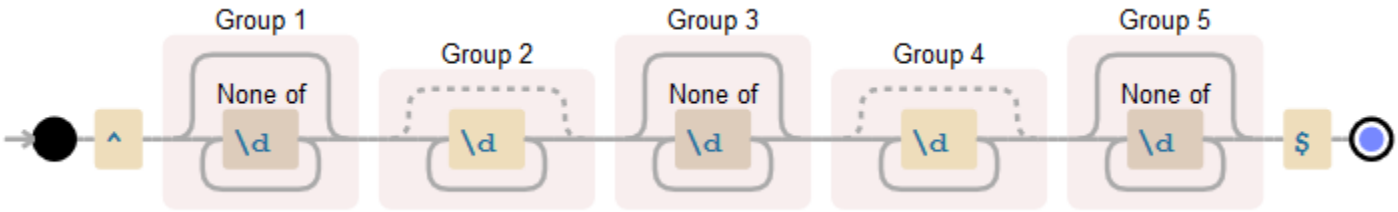
Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix numeric digits in strings that have only 2 numeric groups: cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$



Sample:

```
.........................Start·Length··
Match 1: file_01_name.01        0     15
Group 1: file_                  0      5
Group 2: 01                     5      2
Group 3: _name.                 7      6
Group 4: 01                    13      2
Group 5:                       15      0
```

Sample:

```
.............................Start·Length···
Match 1: file_01_name01 text       0     19
Group 1: file_                     0      5
Group 2: 01                        5      2
Group 3: _name                     7      5
Group 4: 01                       12      2
Group 5:  text                    14      5
```

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix numeric digits in strings that have only 2 numeric groups: cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$

Breaks down:

Capture Groups

^

([^\d]*) 1

((?>\d+)) Repetition 2

([^\d]*) 3

((?>\d+)) Repetition 4

([^\d]*) 5

$

Anchor
^

Capture Group 1

Capture Group 2
Atomic Non-Marking Group

Negated Class
([^\d.] *)

((?>\d) +))

File_

01

Capture Group 3

Negated Class
([^\d] *)

_name.

Capture Group 4
Atomic Non-Marking Group

((?>\d) +))

01

Capture Group 5

Negated Class
([^\d] *)

null

Anchor
$

Tim Notes:

1. Atomic Groups either match or they don't. There is no backtracking. Therefore it restricts the pattern match, avoiding unintentional and inadvertent matches.
2. They are Non-Marking Groups normally, but the value can be retained by placing them within Capture Groups.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix numeric digits in strings that have only 2 numeric groups: cont.

Using: file_01_name.01

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$

Where:

\1 = file_ \2 = 01 \3 = _name. \4 = 01 \5 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = file_
4. \d	Match against Numeric Digit This is the '0' of 'file_01'	Capture Group 2 = 0
5. +	Make it Greedy	Capture Group 2 = 01
6. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. This is the underscore of 'file_01_'	Capture Group 3 = _
7. *	Make it Greedy	Capture Group 3 = _name.
8. \d	Match against Numeric Digit This is the '0' of 'name.01'	Capture Group 4 = 0
9. +	Make it Greedy Current position = EOL	Capture Group 4 = 01
10. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. Already at EOL	Capture Group 5 = No Value
11. *	Make it Greedy	Capture Group 5 = null

Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #2

Samples:

file_.01._name.01
file_01_name01
file_1_ 456 name101
file_1_name.01
file_1_name1
text 1 2 3 4 5
file_.01_name.01
file_1 456name.101
file_1_ 7 25 name.1
file_1_name.1
file_1_name101
45 52.txt
file_01_name.01
file_123_test101_name201
file_1_ 7 25 name.1 18
file_1_name.101
file_1_test1_name01
file_01_name01 text.tif
file_01_name.1
file_1_ 456 name.101
file_1_7 25 name.1 18
file_1_name01
file_1_test1_name1

Making a slight change will allow Matching against the first two numeric groups for all samples:

Orig Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+)) ([^\d]*) $

Modified Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+)) (.*) $

Replace: \10\2\30\4\5

	New Name
45 52.txt	045 052.txt
file_.01._name.01.tif	file_.001._name.001.tif
file_.01_name.01.tif	file_.001_name.001.tif
file_01_name.01.tif	file_001_name.001.tif
file_01_name.1.tif	file_001_name.01.tif
file_01_name01.tif	file_001_name001.tif
file_1 456name.101.tif	file_01 0456name.101.tif
file_123_test101_name201.tif	file_0123_test0101_name201.tif
file_1_ 4 name101 347 text.123~{}.tif	file_01_ 04 name101 347 text.123~{}.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.101.tif
file_1_ 456 name101.tif	file_01_ 0456 name101.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 25 name.1.tif
file_1_7 25 name.1 18.tif	file_01_07 25 name.1 18.tif
file_1_name.01.tif	file_01_name.001.tif
file_1_name.1.tif	file_01_name.01.tif
file_1_name.101.tif	file_01_name.0101.tif
file_1_name01.tif	file_01_name001.tif
file_1_name1.tif	file_01_name01.tif
file_1_name101.tif	file_01_name0101.tif
file_1_test1_name01.tif	file_01_test01_name01.tif
file_1_test1_name1.tif	file_01_test01_name1.tif
text 1 2 3 4 5.tif	text 01 02 3 4 5.tif

Contributor Notes:

1. Add 0's in replacement to increase prefix.

2. The Atomic Groups keep numeric digit-strings from being split (as if there's only 1-string, but you're processing 2-string files)

3. By holding onto their match, they prevent [^\d]* from matching any null-occurrences in between numeric digits.

Add

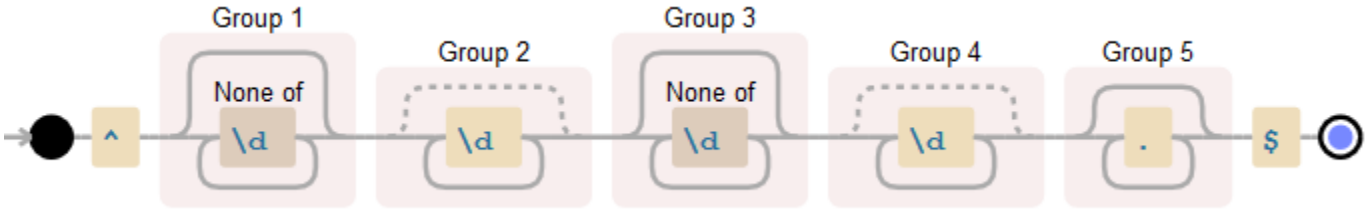
Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #2 cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))(.*)$



Samples

		Start	Length
Match 1:	text 1 2 3 4 5	0	14
Group 1:	text	0	5
Group 2:	1	5	1
Group 3:		6	1
Group 4:	2	7	1
Group 5:	3 4 5	8	6

		Start	Length
Match 1:	45 52	0	5
Group 1:		0	0
Group 2:	45	0	2
Group 3:		2	1
Group 4:	52	3	2
Group 5:		5	0

		Start	Length
Match 1:	file_1_ 7 25 name.1 18	0	22
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_	6	2
Group 4:	7	8	1
Group 5:	25 name.1 18	9	13

		Start	Length
Match 1:	file_1_ 456 name.101	0	20
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_	6	2
Group 4:	456	8	3
Group 5:	name.101	11	9

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #2 cont.

Using: file_1_ 456 name.101

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))(.*)$

Breaks down:

		Capture Groups	
^			Anchor
([^\d]*) ((?>\d+))	Repetition	1 2	Negated Class → File_ Atomic Non-marking Group → 1
([^\d]*) ((?>\d+))	Repetition	3 4	Negated Class → _<space> Atomic Non-marking Group → 456
(.*)		5	→ <space> name.101
$			Anchor

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First two numeric groups in ANY file Example #2 cont.

Using: file_1_ 456 name.101

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))(.*)$

Where:

\1 = file_ \2 = 1 \3 = _ <space> \4 = 456 \5 = <space> name.101

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = file_
4. \d	Match against numeric digit This is the '1' of 'file_1'	Capture Group 2 = 1
5. +	Make it Greedy	Capture Group 2 = Unchanged
6. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. This is the underscore <space> after 'file_1'	Capture Group 3 = _ <space>
7. \d	Match against numeric digit	Capture Group 4 = 4
8. +	Make it Greedy	Capture Group 4 = 456
9. .*	Match against remaining string	Capture Group 5 = <space> name.101

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First three numeric groups in ANY file Example #1

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt		

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$
Replace: \10\2\30\4\50\6\7

	New Name
45 52.txt	45 52.txt
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_01 0456name.0101.tif
file_123_test101_name201.tif	file_0123_test0101_name0201.tif
file_1_ 4 name101 347 text.123~{}.tif	file_1_ 4 name101 347 text.123~{}.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.0101.tif
file_1_ 456 name101.tif	file_01_ 0456 name0101.tif
file_1_ 7 25 name.1 18.tif	file_1_ 7 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_1_ 7 25 name.1.tif
file_1_7 25 name.1 18.tif	file_1_7 25 name.1 18.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.101.tif	file_1_name.101.tif
file_1_name01.tif	file_1_name01.tif
file_1_name1.tif	file_1_name1.tif
file_1_name101.tif	file_1_name101.tif
file_1_test1_name01.tif	file_01_test01_name001.tif
file_1_test1_name1.tif	file_01_test01_name01.tif
text 1 2 3 4 5.tif	text 1 2 3 4 5.tif

Contributor Notes:

1. Add 0's in replacement to increase prefix.
2. The Atomic Groups keep numeric digit-strings from being split (as if there's only 1-string, but you're processing 2-string files).
3. By holding onto their match, they prevent [^\d]* from matching any null-occurrences in between numeric digits.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

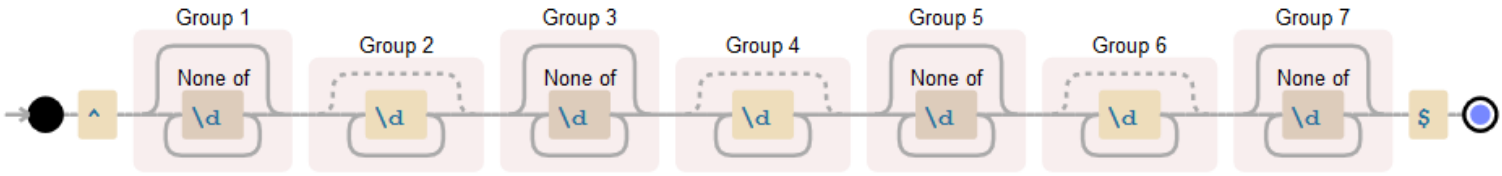
Prefix First three numeric groups in ANY file Example #1 cont.

Samples:

file_.01._name.01
file_01_name01
file_1_ 456 name101
file_1_name.01
file_1_name1
text 1 2 3 4 5
file_.01_name.01
file_1 456name.101
file_1_ 7 25 name.1
file_1_name.1
file_1_name101
45 52.txt
file_01_name.01
file_123_test101_name201
file_1_ 7 25 name.1 18
file_1_name.101
file_1_test1_name01
file_01_name01 text.tif
file_01_name.1
file_1_ 456 name.101
file_1_7 25 name.1 18
file_1_name01
file_1_test1_name1

file_123_test101_name201 text.tif

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$



Sample:

		Start	Length
Match 1:	file_1_ 456 name101	0	19
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_	6	2
Group 4:	456	8	3
Group 5:	_name	11	5
Group 6:	101	16	3
Group 7:		19	0

Sample:

		Start	Length
Match 1:	file_1_test1_name01	0	19
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_test	6	5
Group 4:	1	11	1
Group 5:	_name	12	5
Group 6:	01	17	2
Group 7:		19	0

Forum Contributor: RegexNinja

Sample:

Match 1:	file_123_test101_name201 text
Group 1:	file_
Group 2:	123
Group 3:	_test
Group 4:	101
Group 5:	_name
Group 6:	201
Group 7:	_text

Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.
2. Atomic Groups either match or they don't. There is no backtracking. Therefore it restricts the pattern match, avoiding unintentional and inadvertent matches.
3. They are Non-Marking Groups normally, but the value can be retained by placing them within Capture Groups.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First three numeric groups in ANY file Example #1 cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$

Breaks down:

First let's compare against the RegEx from the same Contributor that matched 2 numeric groups:

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+)) ([^\d]*)$

You can see in order to add a third numeric group, all that was needed was another repetition of:

([^\d]*)((?>\d+))

This is commonly done. When you have a formula that works, all that is required to expand upon it is sometimes just adding a repetition of what successfully captured previously – sometimes, not always. In this case it does. The only limitation is running out of capture Groups, thus limiting how many numeric groups can be captured using this method without having to resort to starting over and finding another method. With three numeric groups, we are now up to using seven Capture Groups. One more group of four will be nine Capture Groups and that will be the limit.

		Capture Groups	
^			Anchor
([^\d]*) ((?>\d+))	Repetition	1 2	Negated Class → File_ Atomic Non-marking Group → 1
([^\d]*) ((?>\d+))	Repetition	3 4	Negated Class → _test Atomic Non-marking Group → 1
([^\d]*) ((?>\d+))	Repetition	5 6	Negated Class → _name Atomic Non-marking Group → 01
([^\d]*)		7	Negated Class → null
$			Anchor

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Prefix First three numeric groups in ANY file Example #1 cont.

Using: file_1_test1_name01.tif

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)$

Where:

\1 = file_ \2 = 1 \3 = _test \4 = 1 \5 = _name \6 = 01 \7 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^\d]	Match against a Negated Class consisting of a numeric digit Match against anything but a numeric digit.	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = file_
4. \d	Match against Numeric Digit This is the '0' of 'file_01'	Capture Group 2 = 1
5. +	Make it Greedy	Capture Group 2 = Unchanged
6. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. This is the underscore of 'file_01_'	Capture Group 3 = _
7. *	Make it Greedy	Capture Group 3 = _test
8. \d	Match against Numeric Digit This is the '0' of 'name.01'	Capture Group 4 = 1
9. +	Make it Greedy	Capture Group 4 = Unchanged
10. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 5 = _
11. *	Make it Greedy	Capture Group 5 = _name
12. \d	Match against Numeric Digit This is the '0' of 'file_01'	Capture Group 6 = 0
13. +	Make it Greedy Current position = EOL	Capture Group 6 = 01
14. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. Already at EOL	Capture Group 7 = No Value
15. *	Make it Greedy	Capture Group 7 = null

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution –

Prefix first three numeric groups in ANY file Example #2:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif text 1 text 2 text 3 4 5 text text 1 text 2 text 3 4 5 6 7 89

Observations:

1. Basing it on the method introduced in Prefix first two numeric groups in Any File by the Contributor, it is easy to see the pattern of the RegEx:

My Match:	^(.*?)	(?<!\d)	(\d+)	(.*?)	(?<!\d)	(\d+)	(.*?)	(?<!\d)	(\d+)	(.*)$
Contributor Match Example #1:	^(.*?)	(?<!\d)	(\d+)	(.*?)	(?<!\d)	(\d+)				(.*)$

Each group of, (.*?)(?<!\d)(\d+), accounts for one more numeric group. Thereby adding one additional group adds a third numeric group.

2. A lot of seemingly complex or confusing RegEx when broken down are merely repetitious patterns.

Tim Notes:

1. Uses Negative Lookbehinds. Please refer to the RegEx Manuals found in the Appendix of Volume I and in the Appendix of this volume as well for further information.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first three numeric groups in ANY file Example #2: cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif text 1 text 2 text 3 4 5 text text 1 text 2 text 3 4 5 6 7 89

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$
Replace: \10\2\30\4\50\6\7

	New Name
45 52.txt	45 52.txt
file1.ext	file1.ext
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01 text.tif	file_01_name01 text.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_01 0456name.0101.tif
file_123_test101_name201 text.tif	file_0123_test0101_name0201 text.tif
file_123_test101_name201.tif	file_0123_test0101_name0201.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.0101.tif
file_1_ 456 name101.tif	file_01_ 0456 name0101.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 025 name.1 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 025 name.1.tif
file_1_7 25 name.1 18.tif	file_01_07 025 name.1 18.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.101.tif	file_1_name.101.tif
file_1_name01.tif	file_1_name01.tif
file_1_name1.tif	file_1_name1.tif
file_1_name101.tif	file_1_name101.tif
file_1_test1_name01.tif	file_01_test01_name001.tif
file_1_test1_name1.tif	file_01_test01_name01.tif
text 1 2 3 4 5.tif	text 01 02 03 4 5.tif
text 1 2 3 4 text 5.tif	text 01 02 03 4 text 5.tif
text 1 text 2 text 3 4 5	text 01 text 02 text 03 4 5
text 1 text 2 text 3 4 5 6 7 89	text 01 text 02 text 03 4 5 6 7 89
text 1 text 2 text 3 4 5 text	text 01 text 02 text 03 4 5 text

Tim Notes:

1. Add 0's in replacement to increase prefix.

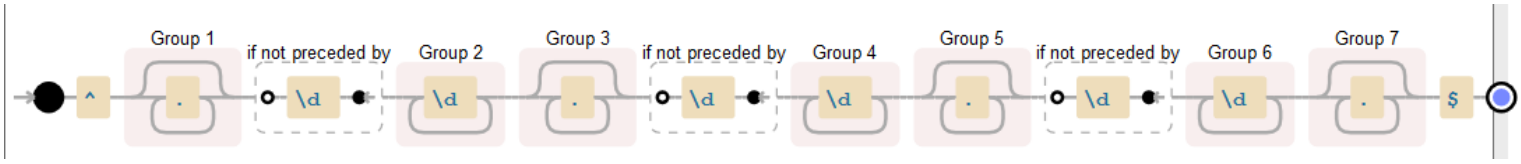
Add

Pad Zeros to Single Numeric (in One Pass!) cont.

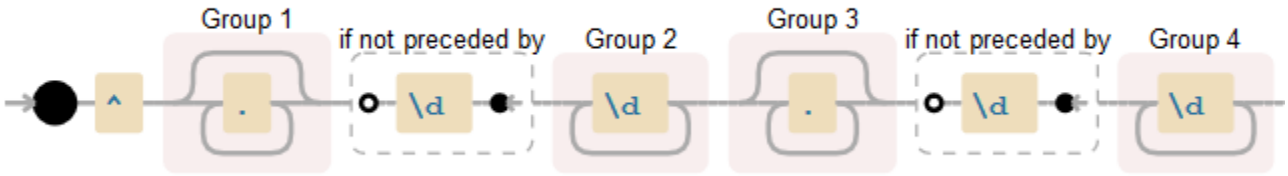
Tim Contribution – cont.

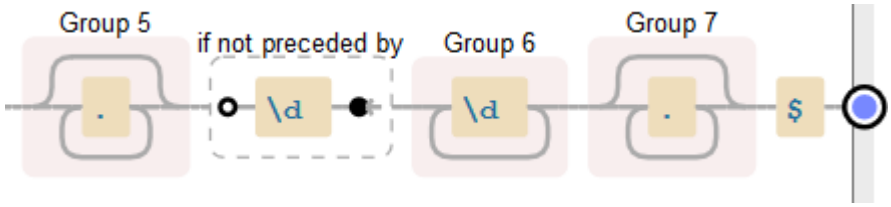
Prefix first three numeric groups in ANY file Example #2: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$



Expanded:





Sample:

	Start	Length
Match 1: **file_1_ 7 25 name.1 18**	0	22
Group 1: file_	0	5
Group 2: 1	5	1
Group 3: _	6	2
Group 4: 7	8	1
Group 5:	9	1
Group 6: 25	10	2
Group 7: name.1 18	12	10

Sample:

	Start	Length
Match 1: **file_123_test101_name201**	0	24
Group 1: file_	0	5
Group 2: 123	5	3
Group 3: _test	8	5
Group 4: 101	13	3
Group 5: _name	16	5
Group 6: 201	21	3
Group 7:	24	0

Sample:

	Start	Length
Match 1: **text 1 text 2 text 3 4 5 text**	0	29
Group 1: text	0	5
Group 2: 1	5	1
Group 3: text	6	6
Group 4: 2	12	1
Group 5: text	13	6
Group 6: 3	19	1
Group 7: 4 5 text	20	9

Add

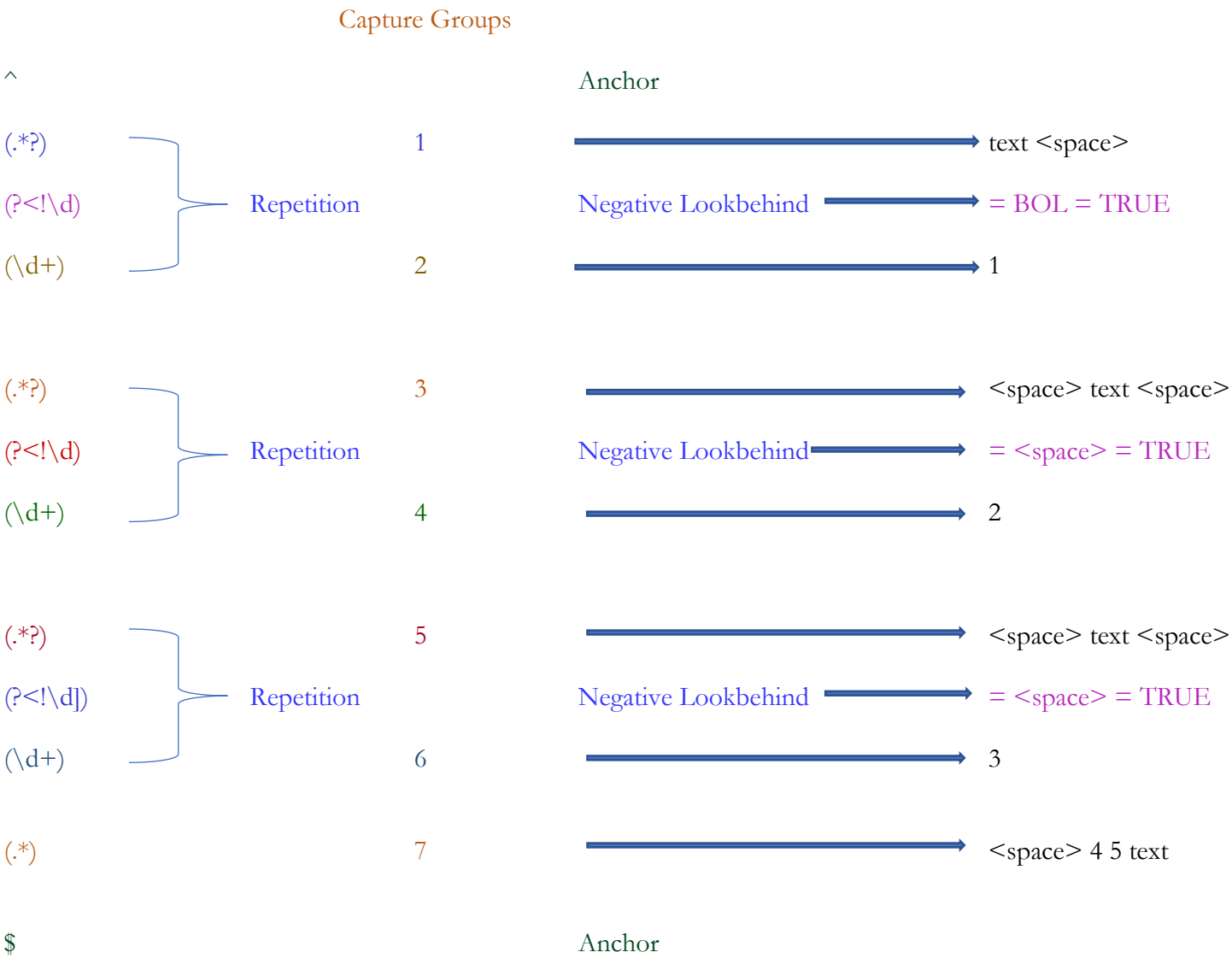
Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first three numeric groups in ANY file Example #2: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$



Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.
4. For more information on Negated Class and Negative Lookbehinds refer to RegEx Manual Volume 1 located in Bulk Rename Utility Operations Manual Volume I and RegEx Manual Volume II found in the Appendix of this book.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first three numeric groups in ANY file Example #2: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Where:

\1 = text <space> \2 = 1 \3 = <space> text <space> \4 = 2
\5 = <space text <space> \6 = 3 \7 = <space> 4 5 text

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = t
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. (?<!\d)	Negative Lookbehind. Look to the left that the last character is 'not' a numeric digit. = BOL = True	
6. \d	Match against Numeric Digit This is the first 'text' in the string.	Capture Group 2 = 1 Changes Capture Group 1 = text <space>
7. +	Make it Greedy The purpose would be to capture Additional numeric digits of this Numeric group.	Capture Group 2 = Unchanged
8..	Match against any character This is the <space> after the '1'	Capture Group 3 = <space>
9. *	Make it Greedy	Capture Group 3 = <space> text 2 text 3 4 5 text
10. ?	But not too Greedy Moves to the position just before the <space> after 'Text 1'	Capture Group 3 = null
11. (?<!\d)	Negative Lookbehind. Look to the left that the last Character is not a numeric digit. = <space> = True	Restores Capture Group 3 = <space>
12. \d	Match against Numeric Digit This is the <space> text > of 'text 2'	Capture Group 4 = 2 Changes Capture Group 3 = <space> text <space>
13. +	Make it Greedy Current position = EOL	Capture Group 4 = Unchanged

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first three numeric groups in ANY file Example #2: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Where:

\1 = text <space>	\2 = 1	\3 = <space> text <space>	\4 = 2
\5 = <space text <space>	\6 = 3	\7 = <space> 4 5 text	

Tim Analysis: cont.

14..	Match against any character This is the <space> after text 2'	Capture Group 5 = <space>
15. *	Make it Greedy	Capture Group 5 = <space> text 3 4 5 text
16. ?	But not too Greedy Moves to the position just before the <space> after 'Text 2'	Capture Group 5 = null
17. (?<!\d)	Negative Lookbehind. Look to the left that the last Character is not a numeric digit. = <space> = True	Restores Capture Group 5 = <space>
18. \d	Match against Numeric Digit This is the '0' of _name.01'	Capture Group 6 = 3 Changes Capture Group 5 = <space> text <space>
19. +	Make it Greedy Current position = EOL	Capture Group 6 = Unchanged
20. .*	Match against remaining string There are no more characters.	Capture Group 7 = <space> 4 5 text
21. $	Must be at EOL (End of Line)	

Tim Notes:

1. The first Lookbehind is not needed because it will always be BOL so there is no need to check for a numeric digit.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first four numeric groups in ANY file:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif text 1 text 2 text 3 4 5 text text 1 text 2 text 3 4 5 6 7 89

Observations:

1. Basing it on the method introduced in Prefix first two numeric groups in Any File by the Contributor, it is easy to see the pattern of the RegEx:

Four Numeric Groups:	^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)	(.*?)(?<!\d)(\d+)	(.*?)(?<!\d)(\d+)	(.*)$
Three Numeric Groups:	^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)	(.*?)(?<!\d)(\d+)		(.*)$
Two Numeric Groups:	^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)			(.*)$

Each sub-expression of, (.*?)(?<!\d)(\d+), accounts for one more numeric group. Thereby adding one additional sub-expression adds a fourth numeric group.

2. A lot of seemingly complex or confusing RegEx when broken down are merely repetitious patterns.

Tim Notes:

1. Uses Negative Lookbehinds. Please refer to the RegEx Manuals found in the Appendix of Volume I and in the Appendix of this volume as well for further information.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first four numeric groups in ANY file: cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif text 1 text 2 text 3 4 5 text text 1 text 2 text 3 4 5 6 7 89

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$
Replace: \10\2\30\4\50\6\70\8\9

	New Name
45 52.txt	45 52.txt
file1.ext	file1.ext
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01 text.tif	file_01_name01 text.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_1 456name.101.tif
file_123_test101_name201 text.tif	file_123_test101_name201 text.tif
file_123_test101_name201.tif	file_123_test101_name201.tif
file_1_ 456 name.101.tif	file_1_ 456 name.101.tif
file_1_ 456 name101.tif	file_1_ 456 name101.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 025 name.01 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 025 name.01.tif
file_1_7 25 name.1 18.tif	file_01_07 025 name.01 18.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.101.tif	file_1_name.101.tif
file_1_name01.tif	file_1_name01.tif
file_1_name1.tif	file_1_name1.tif
file_1_name101.tif	file_1_name101.tif
file_1_test1_name01.tif	file_1_test1_name01.tif
file_1_test1_name1.tif	file_1_test1_name1.tif
text 1 2 3 4 5.tif	text 01 02 03 04 5.tif
text 1 2 3 4 text 5.tif	text 01 02 03 04 text 5.tif
text 1 text 2 text 3 4 5	text 01 text 02 text 03 04 5
text 1 text 2 text 3 4 5 6 7 89	text 01 text 02 text 03 04 5 6 7 89
text 1 text 2 text 3 4 5 text	text 01 text 02 text 03 04 5 text

Tim Notes:

1. Add 0's in replacement to increase prefix.

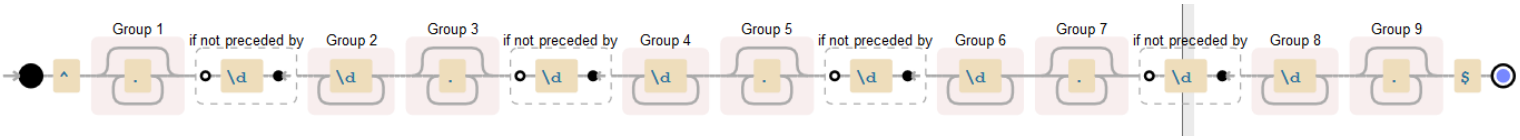
Add

Pad Zeros to Single Numeric (in One Pass!) cont.

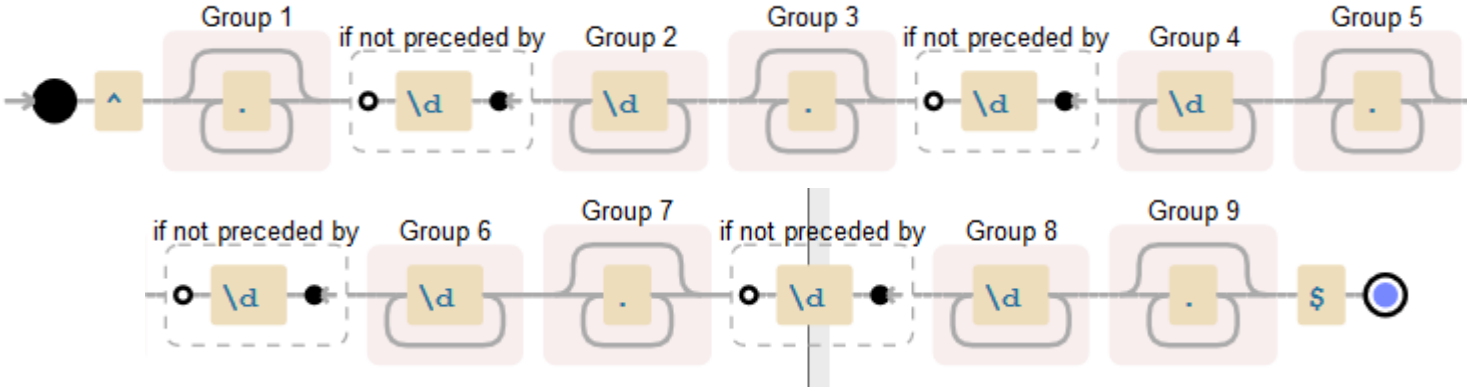
Tim Contribution – cont.

Prefix first four numeric groups in ANY file: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$



Expanded:



Sample:

		Start	Length
Match 1:	file_1_ 7 25 name.1 18	0	22
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_	6	2
Group 4:	7	8	1
Group 5:		9	1
Group 6:	25	10	2
Group 7:	name.	12	6
Group 8:	1	18	1
Group 9:	18	19	3

Sample:

		Start	Length
Match 1:	text 1 2 3 4 5	0	14
Group 1:	text	0	5
Group 2:	1	5	1
Group 3:		6	1
Group 4:	2	7	1
Group 5:		8	1
Group 6:	3	9	1
Group 7:		10	1
Group 8:	4	11	1
Group 9:	5	12	2

Sample:

		Start	Length
Match 1:	text 1 text 2 text 3 4 5 text	0	29
Group 1:	text	0	5
Group 2:	1	5	1
Group 3:	text	6	6
Group 4:	2	12	1
Group 5:	text	13	6
Group 6:	3	19	1
Group 7:		20	1
Group 8:	4	21	1
Group 9:	5 text	22	7

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first four numeric groups in ANY file: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Capture Groups

^		Anchor	
(.*?)	1	→	text <space>
(?<!\d) Repetition		Negative Lookbehind →	= BOL = TRUE
(\d+)	2	→	1
(.*?)	3	→	<space> text <space>
(?<!\d) Repetition		Negative Lookbehind →	= <space> = TRUE
(\d+)	4	→	2
(.*?)	5	→	<space> text <space>
(?<!\d]) Repetition		Negative Lookbehind →	= <space> = TRUE
(\d+)	6	→	3
(.*?)	7	→	<space>
(?<!\d]) Repetition		Negative Lookbehind →	= <space> = TRUE
(\d+)	8	→	4
(.*)	9	→	<space> 5 text
$		Anchor	

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first four numeric groups in ANY file: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Where:

\1 = text <space>	\2 = 1	\3 = <space> text <space>	\4 = 2	
\5 = <space text <space>	\6 = 3	\7 = <space>	\8 = 4	\9 = <space> 5 text

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = t
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. (?<!\d)	Negative Lookbehind. Look to the left that the last character is 'not' a numeric digit. = BOL = True	
6. \d	Match against Numeric Digit This is the first 'text' in the string.	Capture Group 2 = 1 Changes Capture Group 1 = text <space>
7. +	Make it Greedy The purpose would be to capture Additional numeric digits of this Numeric group.	Capture Group 2 = Unchanged
8..	Match against any character This is the <space> after the '1'	Capture Group 3 = <space>
9. *	Make it Greedy	Capture Group 3 = <space> text 2 text 3 4 5 text
10. ?	But not too Greedy Moves to the position just before the <space> after 'Text 1'	Capture Group 3 = null
11. (?<!\d)	Negative Lookbehind. Look to the left that the last Character is not a numeric digit. = <space> = True	Restores Capture Group 3 = <space>
12. \d	Match against Numeric Digit This is the <space> text > of 'text 2'	Capture Group 4 = 2 Changes Capture Group 3 = <space> text <space>
13. +	Make it Greedy Current position = EOL	Capture Group 4 = Unchanged

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first four numeric groups in ANY file: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Where:

\1 = text <space>	\2 = 1	\3 = <space> text <space>	\4 = 2	
\5 = <space text <space>	\6 = 3	\7 = <space>	\8 = 4	\9 = <space> 5 text

Tim Analysis: cont.

14..	Match against any character This is the <space> after text 2'	Capture Group 5 = <space>
15. *	Make it Greedy	Capture Group 5 = <space> text 3 4 5 text
16. ?	But not too Greedy Moves to the position just before the <space> after 'Text 2'	Capture Group 5 = null
17. (?<!\d)	Negative Lookbehind. Look to the left that the last Character is not a numeric digit. = <space> = True	Restores Capture Group 5 = <space>
18. \d	Match against Numeric Digit This is <space> text <space> of 'text 3'	Capture Group 6 = 3 Changes Capture Group 5 = <space> text <space>
19. +	Make it Greedy Current position = EOL	Capture Group 6 = Unchanged

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: text 1 text 2 text 3 4 5 text

Prefix first four numeric groups in ANY file: cont.

Match: ^(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*?)(?<!\d)(\d+)(.*)$

Where:

\1 = text <space>	\2 = 1	\3 = <space> text <space>	\4 = 2	
\5 = <space text <space>	\6 = 3	\7 = <space>	\8 = 4	\9 = <space> 5 text

Tim Analysis: cont.

20..	Match against any character This is the <space> after text 3'	Capture Group 7 = <space>
21. *	Make it Greedy	Capture Group 7 = <space> 4 5 text
22. ?	But not too Greedy Moves to the position just before the <space> after 'Text 3'	Capture Group 7 = null
23. (?<!\d)	Negative Lookbehind. Look to the left that the last Character is not a numeric digit. = <space> = True	Restores Capture Group 7 = <space>
24. \d	Match against Numeric Digit	Capture Group 8 = 4 Capture Group 7 = Unchanged
25. +	Make it Greedy Current position = EOL	All Capture Groups = Unchanged
26. .*	Match against remaining string There are no more characters.	Capture Group 9 = <space> 5 text
27. $	Must be at EOL (End of Line)	

Tim Notes:

1. The first Lookbehind is not needed because it will always be BOL so there is no need to check for a numeric digit.
2. This demonstrates using a Negated Class as an alternate means of capturing text.
3. For more information on Negated Class and Negative Lookbehinds refer to RegEx Manual Volume 1 located in Bulk Rename Utility Operations Manual Volume I and RegEx Manual Volume II found in the Appendix of this book.
4. Cannot do five numeric groups because there are no more available Capture Groups using this method.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first three numeric groups in ANY file Example #3:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)(.+)$
Replace: \10\2\30\4\50\6

	New Name
45 52.txt	45 52.txt
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_01 0456name.0101.tif
file_123_test101_name201.tif	file_0123_test0101_name0201.tif
file_1_ 4 name101 347 text.123~{}.tif	file_01_ 04 name0101 347 text.123~{}.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.0101.tif
file_1_ 456 name101.tif	file_01_ 0456 name0101.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 025 name.1 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 025 name.1.tif
file_1_7 25 name.1 18.tif	file_01_07 025 name.1 18.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.101.tif	file_1_name.101.tif
file_1_name01.tif	file_1_name01.tif
file_1_name1.tif	file_1_name1.tif
file_1_name101.tif	file_1_name101.tif
file_1_test1_name01.tif	file_01_test01_name001.tif
file_1_test1_name1.tif	file_01_test01_name01.tif
text 1 2 3 4 5.tif	text 01 02 03 4 5.tif

Tim Notes:

1. Add 0's in replacement to increase prefix.

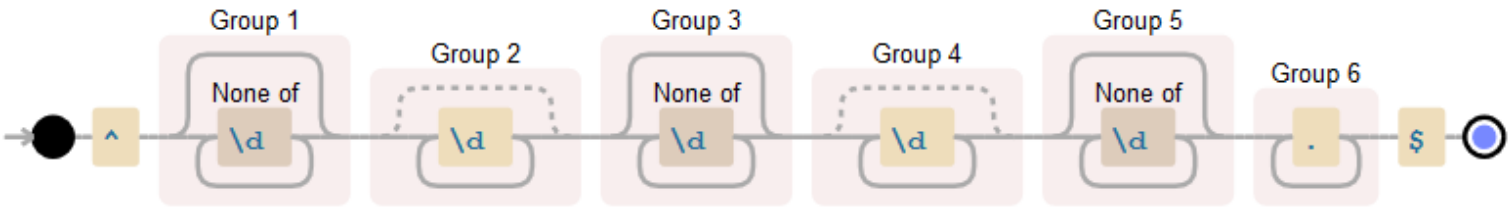
Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first three numeric groups in ANY file Example #3: cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)(.+)$



Sample:

```
·····································Start·Length···
Match 1: file_123_test101_name201 text     0      29
Group 1: file_                             0       5
Group 2: 123                               5       3
Group 3: _test                             8       5
Group 4: 101                              13       3
Group 5: _name                            16       5
Group 6: 201 text                         21       8
```

Sample:

```
····························Start·Length··
Match 1: file_1_ 456 name.101      0      20
Group 1: file_                     0       5
Group 2: 1                         5       1
Group 3: _                         6       2
Group 4: 456                       8       3
Group 5:  name.                   11       6
Group 6: 101                      17       3
```

Sample:

```
·····················Start·Length···
Match 1: text 1 2 3 4 5      0      14
Group 1: text                0       5
Group 2: 1                   5       1
Group 3:                     6       1
Group 4: 2                   7       1
Group 5:                     8       1
Group 6: 3 4 5               9       5
```

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first three numeric groups in ANY file Example #3: cont.

Using: file_1_ 456 name.101

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)(.+)$

Breaks down:

		Capture Groups	
^			Anchor
([^\d]*)	Repetition	1	Negated Class → File_
((?>\d+))	Repetition	2	Atomic Non-marking Group → 1
([^\d]*)	Repetition	3	Negated Class → _<space>
((?>\d+))	Repetition	4	Atomic Non-marking Group → 456
([^\d]*)		5	Negated Class → <space> name.
(.+)		6	→ 101
$			Anchor

Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.
2. Atomic Groups either match or they don't. There is no backtracking. Therefore it restricts the pattern match avoiding unintentional and inadvertent matches.
3. They are Non-Marking Groups normally, but the value can be retained by placing them within Capture Groups.
4. For more information on Negated Class and Atomic Groups refer to RegEx Manual Volume 1 located in Bulk Rename Utility Operations Manual Volume I and RegEx Manual Volume II found in the Appendix of this book.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first three numeric groups in ANY file Example #3: cont.

Using: file_1_ 456 name.101

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)(.+)$

Where:

\1 = file_ \2 = 1 \3 = _<space> \4 = 456 \5 = <space> name. \6 = 101

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = file_
4. \d	Match against Numeric Digit This is the '0' of 'file_01'	Capture Group 2 = 1
5. +	Make it Greedy	Capture Group 2 = Unchanged
6. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. This is the underscore of 'file_01_'	Capture Group 3 = _
7. *	Make it Greedy	Capture Group 3 = _ <space>
8. \d	Match against Numeric Digit This is the '0' of 'name.01'	Capture Group 4 = 4
9. +	Make it Greedy	Capture Group 4 = 456
10. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 5 = <space>
11. *	Make it Greedy	Capture Group 5 = <space> name.
12. .+	Match against remaining string	Capture Group 6 = 101
13. $	Must be at EOL (End of Line)	

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first two numeric groups in String with Three or more:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*?)(.+)$
Replace: \10\2\30\4\5\6

Tim Notes:

1. This is the same RegEx that I used previously in 'Prefix first three numeric groups in ANY File Example #3' with one minor modification and using a different Replacement String so it matches against two not three numeric groups.

Orig Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+)) ([^\d]*) (.+)$
Modified Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+)) ([^\d]* ? (.+)$

Observations:

1. Will not match in ANY string – so this is not the same as the example proposed previously by a forum contributor.
2. Pattern is flexible but must have text after 2nd numeric group in string to match.

i.e.

45 52.txt	to	Ignore
45 52 <space>	to	045 052

Tim Notes:

1. The ' ? ' of ([^\d]*?)(.+)$ makes the text part optional so it doesn't interfere with strings that don't have text at the end of the third numeric group.

i.e. Matches both:

file_123_test101_name201 text.tif	file_0123_test0101_name201 text.tif
file_123_test101_name201.tif	file_0123_test0101_name201.tif

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first two numeric groups in String with Three or more: cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*?)(.+)$
Replace: \10\2\30\4\5\6

	New Name
45 52.txt	45 52.txt
file1.ext	file1.ext
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01 text.tif	file_001_name001 text.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_01 0456name.101.tif
file_123_test101_name201 text.tif	file_0123_test0101_name201 text.tif
file_123_test101_name201.tif	file_0123_test0101_name201.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.101.tif
file_1_ 456 name101.tif	file_01_ 0456 name101.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_01_ 07 25 name.1.tif
file_1_7 25 name.1 18.tif	file_01_07 25 name.1 18.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.101.tif	file_1_name.101.tif
file_1_name01.tif	file_1_name01.tif
file_1_name1.tif	file_1_name1.tif
file_1_name101.tif	file_1_name101.tif
file_1_test1_name01.tif	file_01_test01_name01.tif
file_1_test1_name1.tif	file_01_test01_name1.tif
text 1 2 3 4 5.tif	text 01 02 3 4 5.tif
text 1 2 3 4 text 5.tif	text 01 02 3 4 text 5.tif
text 1 text 2 text 3 4 5	text 01 text 02 text 3 4 5

Tim Notes:

1. Add 0's in replacement to increase prefix.
2. For analysis and Visualizations see 'Prefix first three numeric groups in ANY File Example #3'

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first two numeric groups in ANY file Example #3:

Match #1: ^([^\d]*)((?>\d+) ?) ([^\d]*)((?>\d+) ?) (.+)$
Match #2: ^([^\d]*)((?>\d+)) ([^\d]*)((?>\d+)) ([^\d]*?) (.+)$

Where:

Match #1 – this Example
Match #2 – Prefix first three numeric groups in String:

See a pattern? Same RegEx, except made more text portions optional using the ' ? ' and removed the third match against text.

Replace: \10\2\30\4\5

	New Name
text 1 2 3 4 text 5.tif	text 01 02 3 4 text 5.tif
text 1 2 3 4 5.tif	text 01 02 3 4 5.tif
file_1_7 25 name.1 18.tif	file_01_07 25 name.1 18.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 25 name.1 18.tif
45 52.txt	045 052.txt
file_.01._name.01.tif	file_.001._name.001.tif
file_.01_name.01.tif	file_.001_name.001.tif
file_01_name.01.tif	file_001_name.001.tif
file_01_name.1.tif	file_001_name.01.tif
file_01_name01 text.tif	file_001_name001 text.tif
file_01_name01.tif	file_001_name001.tif
file_1 456name.101.tif	file_01 0456name.101.tif
file_123_test101_name201 text.tif	file_0123_test0101_name201 text.tif
file_123_test101_name201.tif	file_0123_test0101_name201.tif
file_1_ 456 name.101.tif	file_01_ 0456 name.101.tif
file_1_ 456 name101.tif	file_01_ 0456 name101.tif
file_1_ 7 25 name.1.tif	file_01_ 07 25 name.1.tif
file_1_name.01.tif	file_01_name.001.tif
file_1_name.1.tif	file_01_name.01.tif
file_1_name.101.tif	file_01_name.0101.tif
file_1_name01.tif	file_01_name001.tif
file_1_name1.tif	file_01_name01.tif
file_1_name101.tif	file_01_name0101.tif
file_1_test1_name01.tif	file_01_test01_name01.tif
file_1_test1_name1.tif	file_01_test01_name1.tif

Tim Notes:

1. Add 0's in replacement to increase prefix.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

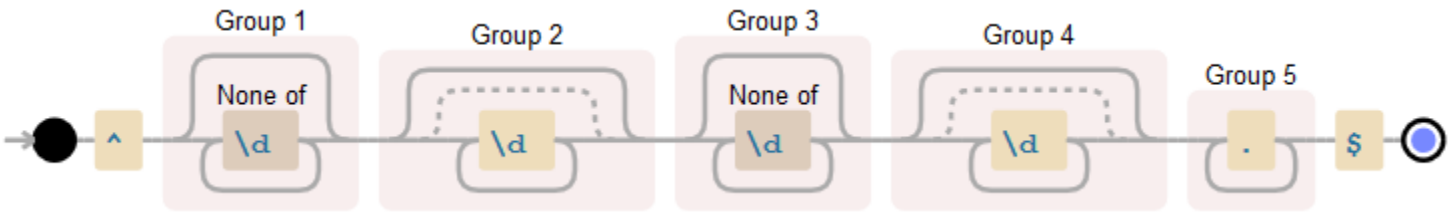
Prefix first two numeric groups in ANY file Example #3 cont.:

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif

Match: ^([^\d]*)((?>\d+)?)([^\d]*)((?>\d+)?)(.+)$



Sample:

```
·····Start·Length···
Match 1: 45 52      0      5
Group 1:            0      0
Group 2: 45         0      2
Group 3:            2      1
Group 4:            3      0
Group 5: 52         3      2
```

Sample:

```
·····························
Match 1: file_1_ 7 25 name.1
Group 1: file_
Group 2: 1
Group 3: _
Group 4: 7
Group 5:  25 name.1
```

Sample:

```
·····Start·Length···
Match 1: text 1 2 3 4 text 5      0      19
Group 1: text                     0      5
Group 2: 1                        5      1
Group 3:                          6      1
Group 4: 2                        7      1
Group 5:  3 4 text 5              8      11
```

Sample:

```
·····························
Match 1: file_1_ 456 name.101
Group 1: file_
Group 2: 1
Group 3: _
Group 4: 456
Group 5:  name.101
```

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first two numeric groups in ANY file Example #3 cont.:

Using: file_1_ 456 name.101

Match: ^([^\d]*)((?>\d+)?)([^\d]*)((?>\d+)?)(.+)$

Breaks down:

		Capture Groups		
^			Anchor	
([^\d]*)	Repetition	1	Negated Class →	File_
((?>\d+))		2	Atomic Non-marking Group →	1
([^\d]*)	Repetition	3	Negated Class →	_<space>
((?>\d+))		4	Atomic Non-marking Group →	456
(.+)		5	→	<space> name.101
$			Anchor	

Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.
2. Atomic Groups either match or they don't. There is no backtracking. Therefore it restricts the pattern match, avoiding unintentional and inadvertent matches.
3. They are Non-Marking Groups normally, but the value can be retained by placing them within Capture Groups.
4. For more information on Negated Class and Atomic Groups refer to RegEx Manual Volume 1 located in Bulk Rename Utility Operations Manual Volume I and RegEx Manual Volume II found in the Appendix of this book.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix first two numeric groups in ANY file Example #3 cont.:

Using: file_1_ 456 name.101

Match: ^([^\d]*)((?>\d+)?)([^\d]*)((?>\d+)?)(.+)$

Where:

\1 = file_ \2 = 1 \3 = _<space> \4 = 456 \5 = <space> name.101

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = file_
4. \d	Match against Numeric Digit This is the '0' of 'file_01'	Capture Group 2 = 1
5. +	Make it Greedy	Capture Group 2 = Unchanged
6. ?	Make it optional	Capture Group 2 = Unchanged

Tim Notes:

1. This is also what allows it to match against the sample string, 45 52.txt. If not for the ' ? ', it would *require* text after the second numeric in the string in order to Match, i.e., 45 52 <space>.txt with the added space after '52'.

7. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. This is the underscore of 'file_01_'	Capture Group 3 = _
8. *	Make it Greedy	Capture Group 3 = _ <space>
9. \d	Match against Numeric Digit This is the '0' of 'name.01'	Capture Group 4 = 4
10. +	Make it Greedy	Capture Group 4 = 456
11. ?	Make it optional	
12. .+	Match against remaining string	Capture Group 6 = <space> name.101
13. $	Must be at EOL (End of Line)	

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif

Tim Contribution – cont.

Prefix last two numeric groups in a string

Observations:

1. String must end with a numeric not text.
2. Minimum 3 numeric groups to unlimited in string.

Match: ^(.+)([^\d]*)((?>\d+))([^\d]*)((?>\d+))((?>\d*))([^\d]*)(.+)$
Replace: \1\2\3\40\5\6\70\8

	New Name
45 52.txt	45 52.txt
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_1 0456name.0101.tif
file_123_test101_name201.tif	file_123_test0101_name0201.tif
file_1_ 456 name.101.tif	file_1_ 0456 name.0101.tif
file_1_ 456 name101.tif	file_1_ 0456 name0101.tif
file_1_ 7 25 name.1 18.tif	file_1_ 7 25 name.01 018.tif
file_1_ 7 25 name.1.tif	file_1_ 7 025 name.01.tif
file_1_7 25 name.1 18.tif	file_1_7 25 name.01 018.tif
file_1_name.01.tif	file_1_name.01.tif
file_1_name.1.tif	file_1_name.1.tif
file_1_name.101.tif	file_1_name.101.tif
file_1_name01.tif	file_1_name01.tif
file_1_name1.tif	file_1_name1.tif
file_1_name101.tif	file_1_name101.tif
file_1_test1_name01.tif	file_1_test01_name001.tif
file_1_test1_name1.tif	file_1_test01_name01.tif
text 1 2 3 4 5.tif	text 1 2 3 04 05.tif

Tim Notes:

1. Add 0's in replacement to increase prefix.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

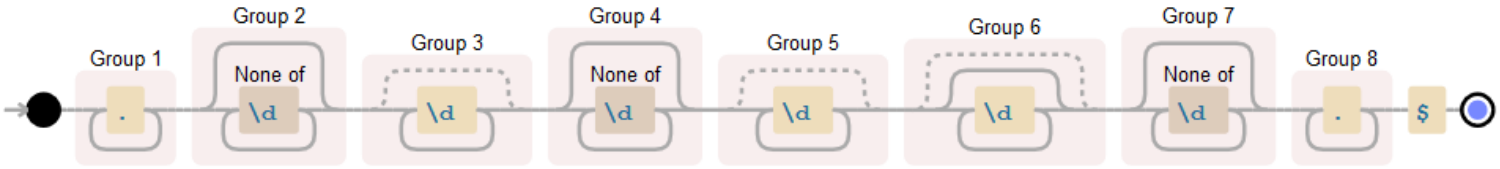
Prefix last two numeric groups in a string cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

file_123_test101_name201 text.tif text 1 text 2 text 3 4 5 6 7 89

Match: ^(.+)([^\d]*)((?>\d+))([^\d]*)((?>\d+))((?>\d*))([^\d]*)(.+)$



Sample:

		Start	Length
Match 1:	text 1 text 2 text 3 4 5 6 7 89	0	31
Group 1:	text 1 text 2 text 3 4 5	0	25
Group 2:		25	0
Group 3:	6	25	1
Group 4:		26	1
Group 5:	7	27	1
Group 6:		28	0
Group 7:		28	1
Group 8:	89	29	2

Sample:

		Start	Length
Match 1:	text 1 2 3 4 text 5	0	19
Group 1:	text 1 2 3	0	11
Group 2:		11	0
Group 3:		11	0
Group 4:		11	0
Group 5:	4	11	1
Group 6:		12	0
Group 7:	text	12	6
Group 8:	5	18	1

Sample:

		Start	Length
Match 1:	file_1_ 7 25 name.1 18	0	22
Group 1:	file_1_ 7 25 name.	0	18
Group 2:		18	0
Group 3:		18	0
Group 4:		18	0
Group 5:	1	18	1
Group 6:		19	0
Group 7:		19	1
Group 8:	18	20	2

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix last two numeric groups in a string cont.

Using: text 1 text 2 text 3 4 5 6 7 89

Match: ^(.+)([^\d]*)((?>\d+))([^\d]*)((?>\d+))((?>\d*))([^\d]*)(.+)$

Breaks down:

		Capture Groups	
^			Anchor
(.+)		1	→ text 1 text 2 text 3 4 5
([^\d]*)	Repetition	2	Negated Class → null
((?>\d+))		3	Atomic Non-marking Group → 6
([^\d]*)	Repetition	4	Negated Class → <space>
((?>\d+))		5	Atomic Non-marking Group → 7
([^\d]*)	Repetition	6	Negated Class → null
((?>\d+))		7	Atomic Non-marking Group → <space>
(.+)		8	→ 89
$			Anchor

Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.
2. Atomic Groups either match or they don't. There is no backtracking. Therefore it restricts the pattern match, avoiding unintentional and inadvertent matches.
3. They are Non-Marking Groups normally, but the value can be retained by placing them within Capture Groups.
4. Only the last two numeric groups are isolated. The previous are captured by Capture Group 1.
5. For more information on Negated Class and Atomic Groups refer to RegEx Manual Volume 1 located in Bulk Rename Utility Operations Manual Volume I and RegEx Manual Volume II found in the Appendix of this book.

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix last two numeric groups in a string cont.

Using: text 1 text 2 text 3 4 5 6 7 89

Match: ^(.+)([^\d]*)((?>\d+))([^\d]*)((?>\d+))((?>\d*))([^\d]*)(.+)$

Where:

\1 = text 1 text 2 text 3 4 5 \2 = null \3 = 6 \4 = <space> \5 = 7 \6 = null \7 = <space> \8 = 89

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = t
3. +	Make it Greedy	Capture Group 1 = <entire string>
4. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. Backtracks. Matches <space> after '7'	Capture Group 2 = <space> Changes Capture Group 1 = text 1 text 2 text 3 4 5 6 7
5. *	Make it Greedy Current position = EOL	All Capture Groups = No Value
6. \d	Match against Numeric Digit Backtracks.	Capture Group 3 = 9 Changes Capture Group 2 = null Changes Capture Group 1 = text 1 text 2 text 3 4 5 6 7 8
7. +	Make it Greedy	All Capture Groups = Unchanged
8. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 4 = <space> Changes Capture Group 3 = 7 Changes Capture Group 1 = text 1 text 2 text 3 4 5 6
9. *	Make it Greedy	Capture Group 4 = <entire string> Changes Capture Group 3 = 9 Changes Capture Group 1 = text 1 text 2 text 3 4 5 6 7 8
10. \d	Match against Numeric Digit	Capture Group 5 = 8 Capture Group 4 = <space> Changes Capture Group 3 = 7 Changes Capture Group 1 = text 1 text 2 text 3 4 5 6
11. +	Make it Greedy	Capture Group 5 = 89 All other Capture Groups = Unchanged

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix last two numeric groups in a string cont.

Using: text 1 text 2 text 3 4 5 6 7 89

Match: ^(.+)([^\d]*)((?>\d+))([^\d]*)((?>\d+))((?>\d*))([^\d]*)(.+)$

Where:

\1 = text 1 text 2 text 3 4 5 \2 = null \3 = 6 \4 = <space> \5 = 7 \6 = null \7 = <space> \8 = 89

Tim Analysis: cont.

12. \d 13. *	Match against Numeric Digit Make it Greedy	All Capture Groups = No Value Capture Group 6 = null Restores Capture Group 5 = 89 Restores Capture Group 4 = <space> Restores Capture Group 3 = 7 Restores Capture Group 1 = text 1 text 2 text 3 4 5 6
14. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 7 = <space> Changes Capture Group 6 = null Changes Capture Group 5 = 7 Capture Group 4 = Unchanged Changes Capture Group 3 = 6 Changes Capture Group 1 = text 1 text 2 text 3 4 5
15. *	Make it Greedy	Capture Group 7 = null Changes Capture Group 6 = null Changes Capture Group 5 = 89 Capture Group 4 = Unchanged Changes Capture Group 3 = 7 Changes Capture Group 1 = text 1 text 2 text 3 4 5 6
16. .+	Match against remaining string	Capture Group 8 = 89 Changes Capture Group 7 = <space> Changes Capture Group 6 = null Changes Capture Group 5 = 7 Changes Capture Group 4 = <space> Changes Capture Group 3 = 6 Capture Group 2 = null = Unchanged Changes Capture Group 1 = text 1 text 2 text 3 4 5
17. $	Must be at EOL (End of Line)	

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Prefix five numeric groups in a string

Samples:

file_.01._name.01
file_01_name01
file_1_ 456 name101
file_1_name.01
file_1_name1
text 1 2 3 4 5
file_.01_name.01
file_1 456name.101
file_1_ 7 25 name.1
file_1_name.1
file_1_name101
45 52.txt
file_01_name.01
file_123_test101_name201
file_1_ 7 25 name.1 18
file_1_name.101
file_1_test1_name01
file_01_name01 text.tif
file_01_name.1
file_1_ 456 name.101
file_1_7 25 name.1 18
file_1_name01
file_1_test1_name1

file_123_test101_name201 text.tif text 1 text 2 text 3 4 5 6 7 89

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))[^\d]*(.+)$
Replace: \10\2\30\4\50\6\70\8 0\9

	New Name
text 1 text 2 text 3 4 5 text	text 01 text 02 text 03 04 05 text
text 1 text 2 text 3 4 5 6 7 89	text 01 text 02 text 03 04 05 6 7 89
text 1 2 3 4 text 5.tif	text 01 02 03 04 05.tif
text 1 text 2 text 3 4 5	text 01 text 02 text 03 04 05
text 1 2 3 4 5.tif	text 01 02 03 04 05.tif
file_1_7 25 name.1 18.tif	file_01_07 025 name.01 018.tif
file_1_ 7 25 name.1 18.tif	file_01_ 07 025 name.01 018.tif
45 52.txt	45 52.txt
file1.ext	file1.ext
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.1.tif
file_01_name01 text.tif	file_01_name01 text.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_1 456name.101.tif
file_123_test101_name201 text.tif	file_123_test101_name201 text.tif
file_123_test101_name201.tif	file_123_test101_name201.tif

Tim Notes:

1. Unable to capture the 'text' after the 4th numeric in the string – 'text 1 2 3 4 text 5.tif' because there are no more Capture Groups available.

2. Since the <space> is removed at the fifth numeric group, it is restored in the Replacement String.

Because there are no more Capture Groups, will only display correctly using the Replacement String if it follows this pattern:

<optional text incl <space>► # ⟶ <optional text inc <space> ⟶ # ⟶ <optional text incl <space>► # ⟶ ► <optional text incl <space>► # ⟶ <optional text incl <space> ⟶ # ⟶ <optional text incl <space>

The RegEx is unable to account for the text or characters that precedes the fifth numeric group – Not Captured. So a string that contains text before the fifth numeric group will not be retained.

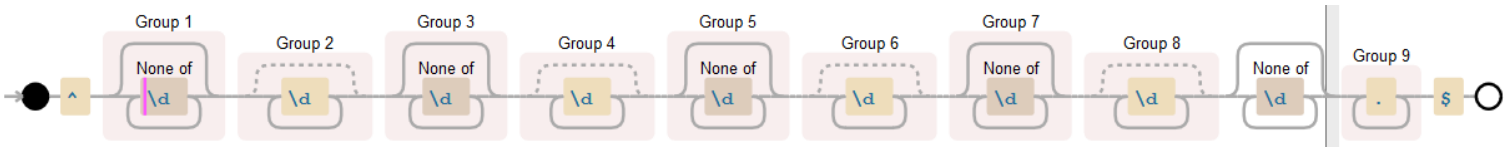
Add

Pad Zeros to Single Numeric (in One Pass!) cont.

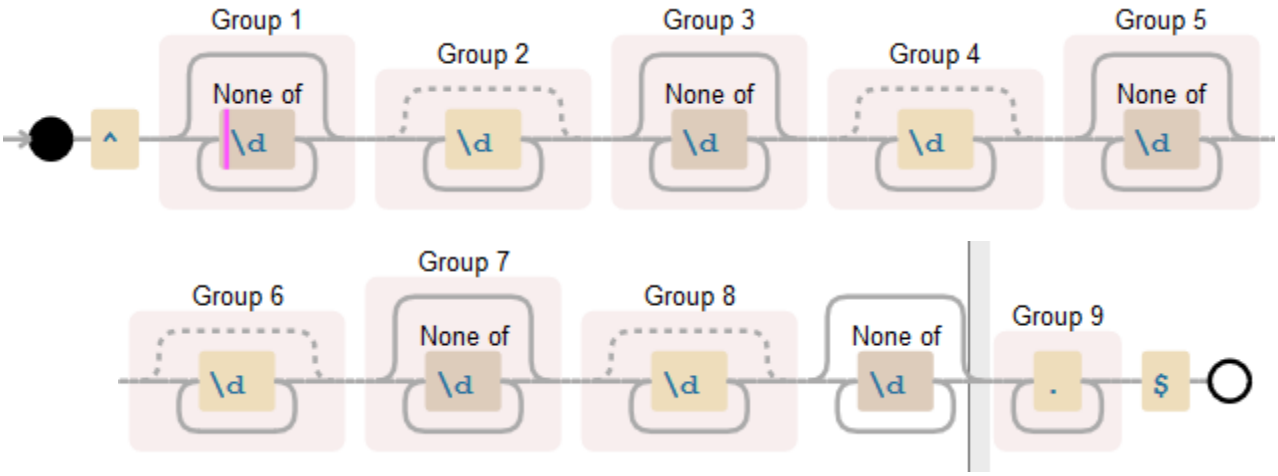
Tim Contribution – cont.

Prefix five numeric groups in a string cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))[^\d]*(.+)$



Expanded:



Samples

		Start	Length
Match 1:	text 1 text 2 text 3 4 5 6 7 89	0	31
Group 1:	text	0	5
Group 2:	1	5	1
Group 3:	text	6	6
Group 4:	2	12	1
Group 5:	text	13	6
Group 6:	3	19	1
Group 7:		20	1
Group 8:	4	21	1
Group 9:	5 6 7 89	23	8

		Start	Length
Match 1:	text 1 2 3 4 text 5	0	19
Group 1:	text	0	5
Group 2:	1	5	1
Group 3:		6	1
Group 4:	2	7	1
Group 5:		8	1
Group 6:	3	9	1
Group 7:		10	1
Group 8:	4	11	1
Group 9:	5	18	1

		Start	Length
Match 1:	text 1 text 2 text 3 4 5 text	0	29
Group 1:	text	0	5
Group 2:	1	5	1
Group 3:	text	6	6
Group 4:	2	12	1
Group 5:	text	13	6
Group 6:	3	19	1
Group 7:		20	1
Group 8:	4	21	1
Group 9:	5 text	23	6

		Start	Length
Match 1:	file_1_ 7 25 name.1 18	0	22
Group 1:	file_	0	5
Group 2:	1	5	1
Group 3:	_	6	2
Group 4:	7	8	1
Group 5:		9	1
Group 6:	25	10	2
Group 7:	name.	12	6
Group 8:	1	18	1
Group 9:	18	20	2

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: file_1_ 7 25 name.1 18.tif

Prefix five numeric groups in a string cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+))[^\d]*(.+)$

Breaks down:

		Capture Groups	
^			Anchor
([^\d]*) ((?>\d+))	Repetition	1 2	Negated Class → File_ Atomic Non-marking Group → 1
([^\d]*) ((?>\d+))	Repetition	3 4	Negated Class → _<space> Atomic Non-marking Group → 7
([^\d]*) ((?>\d+))	Repetition	5 6	Negated Class → <space> Atomic Non-marking Group → 25
([^\d]*) ((?>\d+))	Repetition	7 8	Negated Class → <space> name. Atomic Non-marking Group → 1
([^\d]*)	N o t C a p t u r e d		Negated Class →
(.+)		9	→ 18
$			Anchor

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: file_1_ 7 25 name.1 18.tif

Prefix five numeric groups in a string cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+)) ([^\d]*)((?>\d+)) [^\d]*(.+)$

Where:

\1 = file_ \2 = 1 \3 = _ <space> \4 = 7 \5 = <space> \6 = 25
\7 = <space> name. \8 = 1 \9 = 18

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = file_
4. \d	Match against Numeric Digit	Capture Group 2 = 1
5. +	Make it Greedy	Capture Group 2 = Unchanged
6. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 3 = _
7. *	Make it Greedy	Capture Group 3 = _ <space>
8. \d	Match against Numeric Digit	Capture Group 4 = 7
9. +	Make it Greedy	Capture Group 4 = Unchanged
10. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 5 = <space>
11. *	Make it Greedy	Capture Group 5 = Unchanged
12. \d	Match against Numeric Digit	Capture Group 6 = 2
13. +	Make it Greedy	Capture Group 6 = 25

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Tim Contribution – cont.

Using: file_1_ 7 25 name.1 18.tif

Prefix five numeric groups in a string cont.

Match: ^([^\d]*)((?>\d+))([^\d]*)((?>\d+))([^\d]*)((?>\d+)) ([^\d]*)((?>\d+)) [^\d]*(.+)$

Where:

\1 = file_	\2 = 1	\3 = _ <space>	\4 = 7	\5 = <space>	\6 = 25
\7 = <space> name.	\8 = 1	\9 = 18			

Tim Analysis: cont.

14. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 7 = <space>
15. *	Make it Greedy	Capture Group 7 = <space> name.
16. \d	Match against Numeric Digit	Capture Group 8 = 1
17. +	Make it Greedy	Capture Group 8 = Unchanged
18. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. (Not Captured) = <space>	
19. *	Make it Greedy. Not Captured = Unchanged	
20. .+	Match against remaining string	Capture Group 9 = 18
21. $	Must be at EOL (End of Line)	

Tim Notes:

1. This demonstrates using a Negated Class as an alternate means of capturing text.
2. Atomic Groups either match or they don't. There is no backtracking. Therefore it restricts the pattern match, avoiding unintentional and inadvertent matches.
3. They are Non-Marking Groups normally, but the value can be retained by placing them within Capture Groups.
4. For more information on Negated Class and Atomic Groups refer to RegEx Manual Volume 1 located in Bulk Rename Utility Operations Manual Volume I and RegEx Manual Volume II found in the Appendix of this book.

Conclusion

This is just to give you an idea of what you can do when you start analyzing and making your own modifications to the examples presented in this document and how you can apply them to your own needs.

Remember, these RegEx work with the samples presented. It is up to you to tweak them to work with your own. The other purpose of this book is, through your own self analysis and reading through the notes, etc. as well as using it in combination with Volume 1, you will learn RegEx – I did.

Add

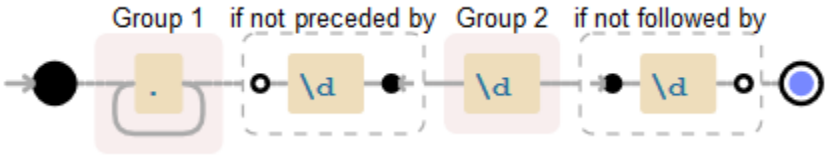
Pad Zeros to Single Numeric (in One Pass!) cont.

Variation:

Using a Negative Lookahead and a Negative Lookbehind to determine a Single numeric digit at end of string

Match: (.+)(?<!\d)(\d)(?!\d)
Replace: \10\2

Name	New Name
file 1.ext	file 01.ext
file1.ext	file01.ext
file11.ext	file11.ext
file003.ext	file003.ext



Sample:

```
.................
Match 1: file 1
Group 1: file
Group 2: 1
```

Tim Notes:

1. (?!\d) Lookahead Negative – Match only if no characters in the string are followed by a numeric digit.
2. (?>!\d) Lookbehind Negative – Match only if a numeric digit did not precede the current character.

Forum Contributor: bobkoure

Add

Pad Zeros to Single Numeric (in One Pass!) cont.

Variation: cont.

Using a Negative Lookahead and a Negative Lookbehind to determine a Single numeric digit at end of string cont.

Using: file 1.ext
Using: file003.ext
Using: file1.ext

Match: (.+)(?<!\d)(\d)(?!\d)

Where:

\1 = File \2 = 1

Tim Analysis:

1. (.+)	Capture until…	Capture Group 1 = file <space>
2. (?>!\d)	Lookbehind Negative. Current position is at the 1 of either filename. This looks back and if the last character was not a numeric, it will match but not capture. Capture Group 1 is changed from 'file 1' to 'file' or 'file <space>' depending on whether the filename is 'file1.ext' or 'file 1.ext'.	
3. \d	Capture the following numeric digit However, if at 'file003.ext', Capture Group 2 = first occurrence 0, current position = first occurrence 0	Capture Group 2 = 1
4. (?!\d)	Lookahead Negative. If file 1.ext or file1.ext, current position at EOL but if file is 'file003.ext', current position is first occurrence 0. The lookahead will match and not capture if the following character is not a numeric digit.= False, the following character is a numeric, a second occurrence of '0', so expression Fails on i.e., 'file003.ext'.	

Tim Notes:

1. For more information on Lookarounds see, 'Regular Expressions Manual' found in the Appendix of 'Volume 1 Bulk Rename Utility Operations Manual'.
2. Sample used in analysis is file .ext.

Add

Changing YYMMDD to YYYYMMDD and rearrange (demonstrate \s whitespace for <space> character)

IME John Smith 01-01-10.tiff	to	20100101 John Smith IME.tiff
UR Jane Doe 01-03-05.tiff	to	20050301 Jane Doe UR.tiff
IME John Doe 04-04-05.tiff	to	20050404 John Doe IME.tiff

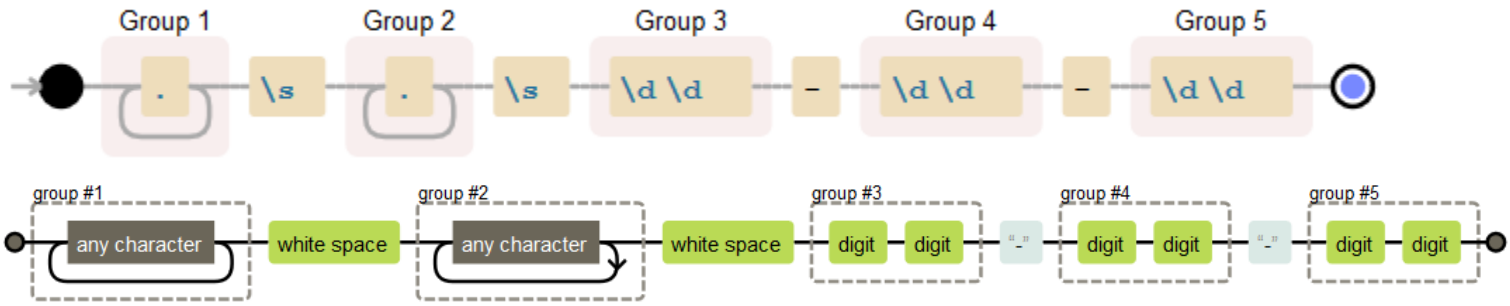
Observations:

1. The first part, the type of file, can be IME, UR, ADD UR, ADD IME, 2ND IME, or 2ND UR.
2. The name is usually first and last, but some files have up to five names listed.
3. The date is currently in mm-dd-yy format and I want it to be converted to yyyymmdd.

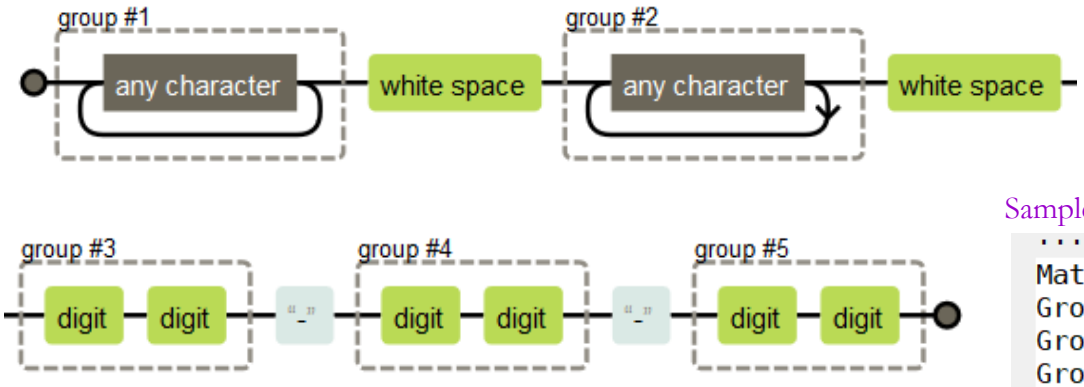
Match: (.+?)\s(.+)\s(\d\d)-(\d\d)-(\d\d)
Replace: 20\5\4\3 \2 \1

Name	New Name
UR Jane Doe 01-03-05.tiff	20050301 Jane Doe UR.tiff
IME John Smith 01-01-10.tiff	20100101 John Smith IME.tiff
IME John Doe 04-04-05.tiff	20050404 John Doe IME.tiff

Forum Contributor: Stefan



Expanded:



Sample:

```
Match 1: UR Jane Doe 01-03-05
Group 1: UR
Group 2: Jane Doe
Group 3: 01
Group 4: 03
Group 5: 05
```

Add

Changing YYMMDD to YYYYMMDD and rearrange (demonstrate \s whitespace for <space> character) cont.

Using: UR Jane Doe 01-03-05.tiff

Match: (.+?)\s(.+)\s(\d\d)-(\d\d)-(\d\d)

Where:

\1 = UR \2 = Jane Doe \3 =01 \4 =03 \5 =05

Tim Analysis:

1. .+	Match till…	Capture Group 1 = UR
2. \s	.. a <space> but don't capture	
3. .+	Match till…	Capture Group 2 = Jane Doe
4. \s	.. a <space> but don't capture	
5. \d\d	followed by 2 numeric digits (Year)…	Capture Group 3 = 01
6. -	.. and a <hyphen> but don't capture	
7. \d\d	followed by two more numeric digits (Month)	Capture Group 4 = 03
8. -	.. and a <hyphen> but don't capture	
9. \d\d	followed by 2 numeric digits (Day)	Capture Group 5 = 05

Replace string includes "20" for century 2000

Add

Add spaces between first hyphen

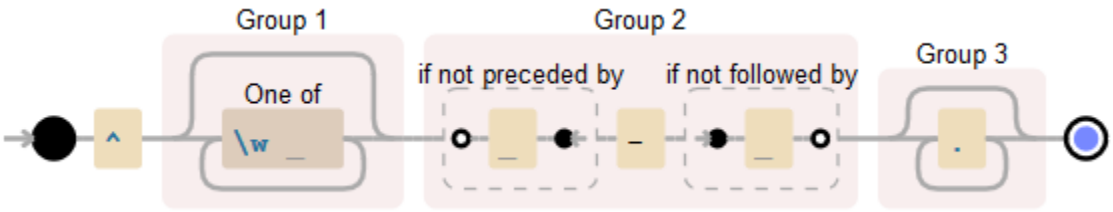
ABCDEF-GHIJKL-MN-OP	to	ABCDEF – GHIJKL-MN-OP
ABC-DEFGHIJKL-MN-OP	to	ABC – DEFGHIJKL-MN-OP
ABC_DEFG-HIJKL-MNO	to	ABC_DEFG – HIJKL-MNO
ABC_DEF_GHI-123-45	to	ABC_DEF_GHI – 123-45
ABC-123-45	to	ABC – 123-45
ABCD – EFGHI_JKL-P	to	Ignore

Observations:

1. Add a <space> on either side of the FIRST <hyphen> "-".
2. If a name already has a <space> on either side of the first "-" then it should ignore it.
3. Keep everything else intact.

Match: ^([\w]*)((?<!)-(?!))(.*)
Replace: \1 \2 \3

Name	New Name
ABCDEF-GHIJKL-MN-OP	ABCDEF - GHIJKL-MN-OP
ABCD - EFGHI_JKL-P	ABCD - EFGHI_JKL-P
ABC_DEFG-HIJKL-MNO	ABC_DEFG - HIJKL-MNO
ABC_DEF_GHI-123-45	ABC_DEF_GHI - 123-45
ABC-DEFGHIJKL-MN-OP	ABC - DEFGHIJKL-MN-OP
ABC-123-45	ABC - 123-45



Sample:

```
..............................
Match 1: ABCDEF-GHIJKL-MN-OP
Group 1: ABCDEF
Group 2: -
Group 3: GHIJKL-MN-OP
```

Add

Add spaces between first hyphen cont.

ABCDEF-GHIJKL-MN-OP	to	ABCDEF – GHIJKL-MN-OP
ABC-DEFGHIJKL-MN-OP	to	ABC – DEFGHIJKL-MN-OP
ABC_DEFG-HIJKL-MNO	to	ABC_DEFG – HIJKL-MNO
ABC_DEF_GHI-123-45	to	ABC_DEF_GHI – 123-45
ABC-123-45	to	ABC – 123-45
ABCD – EFGHI_JKL-P	to	Ignore

Match: ^([\w]*)((?<!)-(?!))(.*)

Contributor Notes:

1. Using the class \w (Word character) instead of [A-Za-z], you include what [A-Za-z] does but also any numbers and the underscore_. (This is mentioned because an initial attempt used a class of [A-Za-z] which matches against an upper and lowercase letter – not shown here).

2. \w doesn't include the <space> character, so it wouldn't match <hyphens> with a <space> around them.

3. To make sure it only worked with the first occurrence of the <hyphen> I used ^ at the start.

   So the initial RegEx would be:

   ^([\w]*)(-)(.*)

4. The only problem now is that it won't match something like:

   xxx xxx-xxx.txt

   because it has a <space> in the name.

5. This can be fixed by adding a <space> to make Capture Group 1 as:

   ([\w])

6. To make it ignore the filename if it has a <space> before or after the initial <hyphen> (or both), you can use negative lookbehind\lookahead around group 2, which was initially just a <hyphen> (-)

7. Now you have this for group 2: ((?<!)-(?!))

Forum Contributor: Glenn

Add

Add spaces between first hyphen cont.

Using; ABCDEF-GHIJKL-MN-OP

Match: ^([\w]*)((?<!)-(?!))(.*)

Where:

\1 = ABCDEF \2 = - \3 = GHIJKL-MN-OP

Breaks down:

	Capture Group 1			Capture Group 2				Capture Group 3
Anchor	Class			Negative Lookbehind		Negative Lookahead		
^	(	[\w]	*)	(	(?<!)	-	(?!))	(.*)

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [\w]	Match against a class consisting of a: a. Word Metacharacter. Match against any single letter, number or underscore. b. <space>	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = ABCDEF
4. (?>!)	Negative Lookbehind. Look to the left that the last Character is 'not' a <space> = 'F' = True	Capture Group 2= No Value Capture Group 1= Unchanged
5. -	Match against a <hyphen> character This is the <hyphen> after 'ABCDEF'	Capture Group 2 = -
6. (?!)	Negative Lookahead. Look to the right that the next Character is 'not' a <space> = 'G' = True	Capture Group 2 = Unchanged
7. .*	Match against remaining string	Capture Group 3= GHIJKL-MN-OP

Add

Simple Pad of (Zero) to Suffix

2006-04-24 01_03.wav	to	2006-04-24 01_03(0).wav
2006-04-24 01_03(1).wav	to	Ignore
2006-04-24 01_03(2).wav	to	Ignore

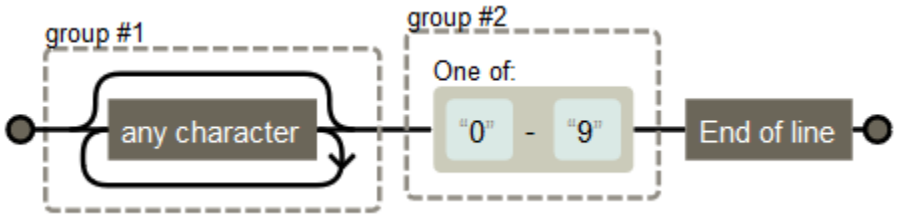
Observations:

1. Only add (0) to those filenames that do not already have a similar suffix.

Match: (.*)([0-9])$
Replace: \1\2(0)

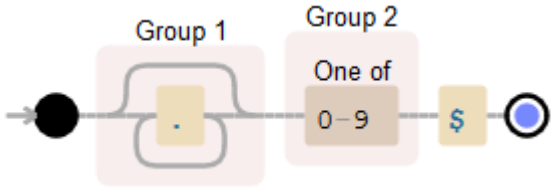
Name	New Name
2006-04-24 01_03.wav	2006-04-24 01_03(0).wav
2006-04-24 01_03(1).wav	2006-04-24 01_03(1).wav
2006-04-24 01_03(2).wav	2006-04-24 01_03(2).wav

Forum Contributor: jimwillsher (Original Publisher BRU)



Sample:

```
.........................
Match 1: 2006-04-24 01_03
Group 1: 2006-04-24 01_0
Group 2: 3
```



Where:

\1 = 2006-04024 01_0 \2 = 3

Tim Analysis:

1..	Match against any character	Capture Group 1 = 2
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [0-9]	Match against a class consisting of a numeric digit, '0' - '9' Backtracks.	Capture Group 2 = 3 Changes Capture Group 1 = 2006-04024 01_0
4. $	Must be at EOL (End of Line)	

Add

Renaming based on the contents of the original file name

title_isbn_Cover_5x8.pdf to title_isbn_Cover_5x8_10ptC1S.pdf
title_isbn_Text_5x8.pdf to title_isbn_Text_5x8 _50White.pdf

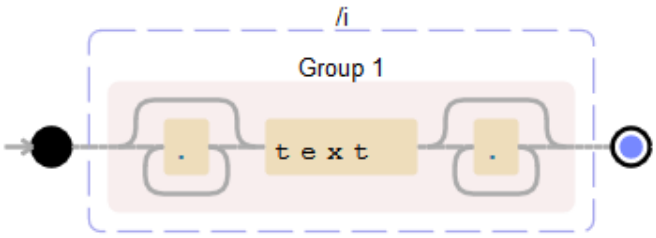
Observations:

1. Content identified by 'Text' or 'Cover'
2. Needs to append the paper type, i.e., 10ptC1S
3. Will need to be done in to two separate runs, one for each paper type and content:

First Run -

Match: (?i)(.*text.*)
Replace: \1_50White

Name	New Name
title_isbn_Text_5x8.pdf	title_isbn_Text_5x8_50White.pdf



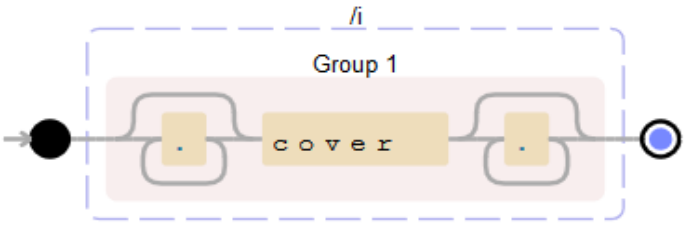
Sample:

```
Match 1: title_isbn_Text_5x8
Group 1: title_isbn_Text_5x8
```

Second Run -

Match: (?i)(.*cover.*)
Replace: \1'_10ptC1S

Name	New Name
title_isbn_Cover_5x8.pdf	title_isbn_Cover_5x8'_10ptC1S.pdf



Sample:

```
Match 1: title_isbn_Cover_5x8
Group 1: title_isbn_Cover_5x8
```

Tim Notes:

1. The (?i) makes the Case insensitive. It an option that is fully explained in the RegEx Manual found in the Appendix of 'Volume I Bulk Rename Utility Operations Manual'.

Forum Contributor: KenP

Add

Renaming based on the contents of the original file name cont.

Using: title_isbn_Text_5x8.pdf

Match: (?i)(.*text.*)

Where:

\1 = title_isbn_Text_5x8

Tim Notes:

1. Since the RegEx used is basically the same for both runs, only one analysis is required.
2. This example is only to showcase the work-around for using Case Insensitivity in BRU's RegEx.
3. All the work is done in the Replacement String.

Tim Analysis:

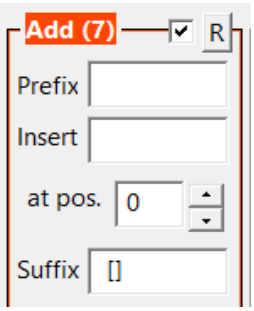
1. (?i)	Turn on Case Insensitivity	
1..	Match against any character	Capture Group 1 = t
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. text	Match against string 'text'. Because the RegEx Is Case Insensitive, it matches the string 'Text'	Capture Group 1 = title_isbn_Text
4. .*	Match against remaining string	Capture Group 1 = title_isbn_Text_5x8

Add

Autonumber and Add Brackets to Suffix Number

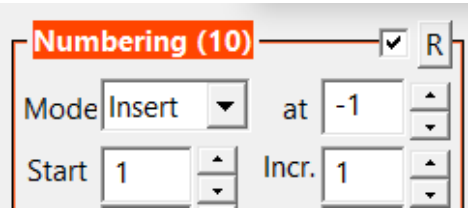
filename a.any	to	filename a [1].any
filename b.any	to	filename b [2].any
filename c.any	to	filename c [3].any
filename d.any	to	filename d [4].any

1. 'Section #7: Add' -

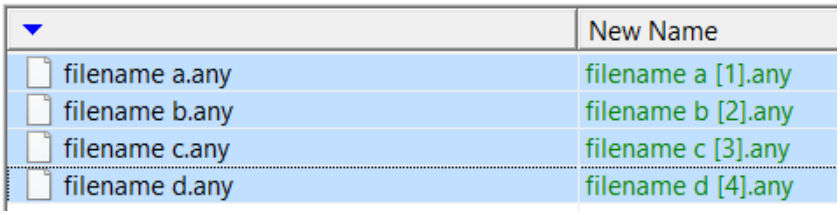


Set Suffix = (two characters with no <spaces> between them) i.e. brackets – '[ ]'

2. 'Section #10: Numbering' -



Set:
Mode: Insert
At: -1 (the Second to last position. This will position the close bracket)

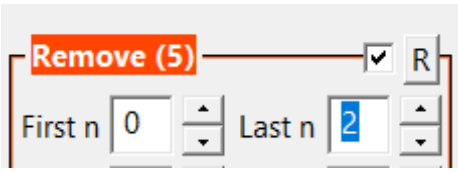


Tim Notes:

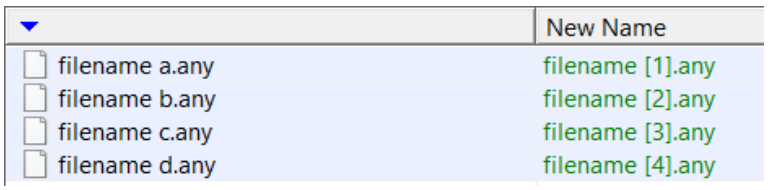
1. Instead of Brackets you could also use parentheses, braces, <hyphens> or any other character that Windows allows in filename naming convention.

2. To Replace the alpha just with the numeric add this to the above:

'Section #5: Remove' -



Set:
Last n: 2
(Removes last 2 characters)



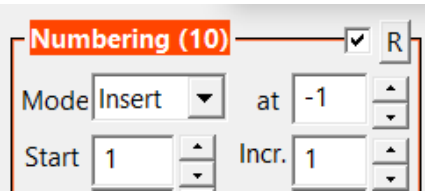
Forum Contributor: (based on an idea) Stefan

Add

Autonumber and Add Brackets to Suffix Number cont.

Tim Contribution –

1. 'Section #10: Numbering' –

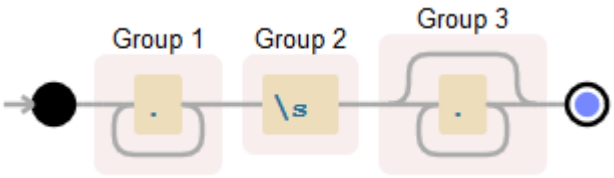


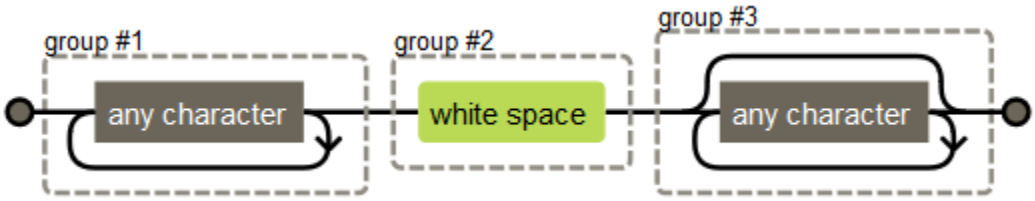
Set:
Mode: Insert
At: -1

2. RegEx –

Match: (.+)(\s)(.*)
Replace: \1\2\3 []

	New Name
filename a.any	filename a [1].any
filename b.any	filename b [2].any
filename c.any	filename c [3].any
filename d.any	filename d [4].any





Sample:

```
·····················Start·Length··
Match 1: filename a       0      10
Group 1: filename         0       8
Group 2:                  8       1
Group 3: a                9       1
```

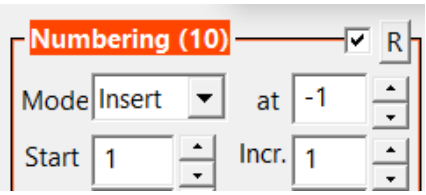
Add

Autonumber and Add Brackets to Suffix Number cont.

Tim Contribution – cont.

Or To Replace the alpha just with the numeric:

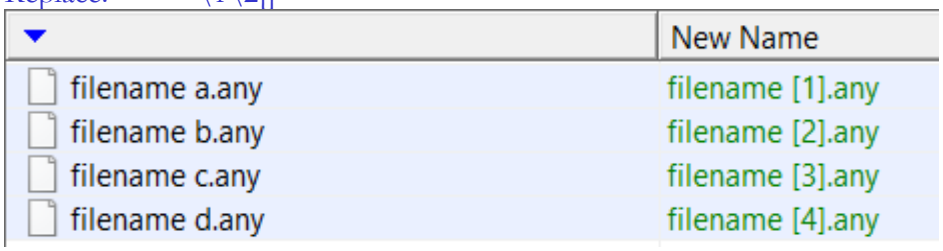
1. 'Section #10: Numbering' –



Set:
Mode: Insert
At: -1

2. RegEx –

Match: (.+)(\s)(.*)
Replace: \1\2[]



Tim Notes:

1. Uses same RegEx, just the Replacment String is changed.

Where:

\1 = filename \2 = <space> \3 = a

Tim Analysis:

1..	Match against any character	Capture Group 1 = f
2. +	Make it Greedy	Capture Group 1 = <entire string>
	Current position = EOL	
3. \s	Match against <space> character	Capture Group 2 = <space>
	Backtracks. This is the <space> after 'filename'	Changes Capture Group 1 = filename
4. .*	Match against remaining string	Capture Group 3 = a

Add

A Word on the 'Section #10: Numbering' - Insert At - Option

This was briefly discussed in Volume 1, but here is a better explanation of how it works.

'At' specifies the *position where in the filename the numbering sequence should appear*. The 'At' refers to the character position on the string. In the following example, the numeric '1' will be repositioned on the string.

Using example from '**Autonumber and Add Brackets to Suffix Number**',

	f	i	l	e	n	a	m	e		a
Position	1	2	3	4	5	6	7	8	9	10

A perfect example of, "What Goes around, Comes around":

'At' Position 0 = filename a [].any

'At' Position +1 = f1ilename a [].any

'At' Position +2 = fi1lename a [].any

'At' Position +3 = fil1ename a [].any

'At' Position +4 = file1name a [].any

'At' Position +5 = filen1ame a [].any

'At' Position +6 = filena1me a [].any

'At' Position +7 = filenam1e a [].any

'At' Position +8 = filename1 a [].any

'At' Position +9 = filename 1a [].any

'At' Position +10 = filename a1 [].any

'At' Position +11 = filename a 1[].any

'At' Position +12 = filename a [1].any

'At' Position -12 = f1ilename a [].any

'At' Position – 11 = fi1lename a [].any

'At' Position – 10 = fil1ename a [].any

'At' Position – 9 = file1name a [].any

'At' Position – 8 = filen1ame a [].any

'At' Position – 7 = filena1me a [].any

'At' Position – 6 = filenam1e a [].any

'At' Position – 5 = filename1 a [].any

'At' Position – 4 = filename 1a [].any

'At' Position – 3 = filename a1 [].any

'At' Position – 2 = filename a 1[].any

'At' Position – 1 = filename a [1].any

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String

I present an item that was a joint collaboration, Tim (idea, concept and execution) with RegexNinja (BRC).

Observations:

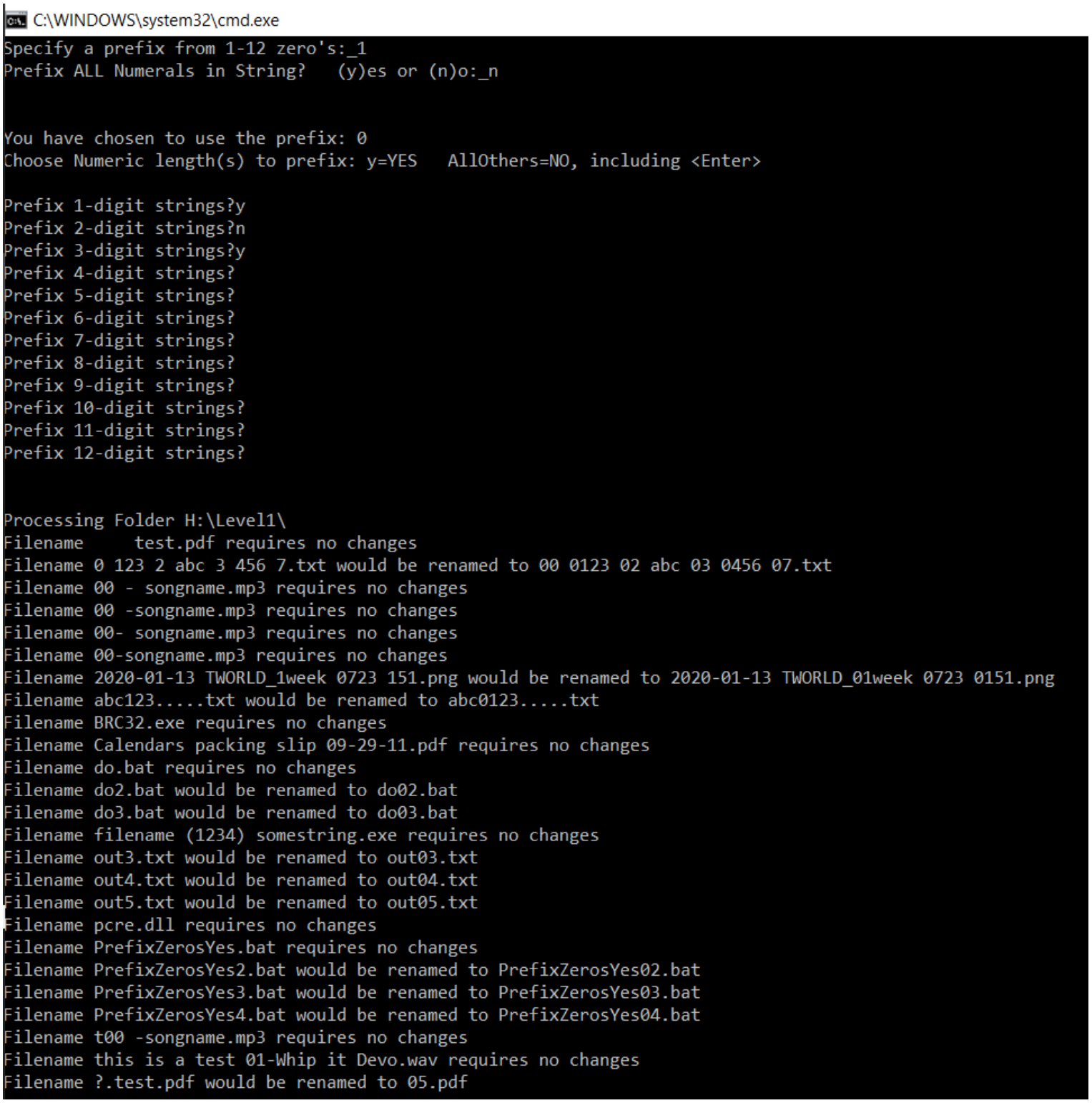
1. Append Zeros to Prefix of numeric numbers in string.
2. Numbers can be of varying lengths of up to 12
3. Prefixed Zeros can be up to 12

```
@echo off
:: cd/d "C:\YourFolderPath"

(Set/p zz=Specify a prefix from 1-12 zero's:_) & (Set/a zz=zz)
(If %zz% lss 1 exit) & (If %zz% gtr 12 exit)
(SetLocal EnableDelayedExpansion) & (For /L %%A IN (1,1,%zz%) DO Set pr=!pr!0)

:ask
Set/p all=Prefix ALL Numerals in String?  ^(y^)es or ^(n^)o:_
If /i "%all%"=="y" ((SetLocal DisableDelayedExpansion) & (GoTo all))
If /i NOT "%all%"=="n" ((echo.                    Please answer:   y for yes   n for no) & (echo.) & (Goto ask))
(echo.)&(echo.)&echo.You have chosen to use the prefix: %pr%
echo.Choose Numeric length^(s^) to prefix: y=YES   AllOthers=NO, including ^<Enter^>
echo.

For /L %%A IN (1,1,12) DO Set/p incl%%A= Prefix %%A-digit strings?
For /L %%A IN (1,1,12) DO If /i not "!incl%%A!"=="y" Set incl%%A=
Set incl 2>nul 1>nul||((echo.) & (echo.NO selections made^^!  Press any key to exit) & (pause>nul) & (EXIT))
For /f "delims=l= tokens=1,2" %%A IN ('Set incl') DO Set reg%%B=/RegExp:"^^(.*?)(?<^!\d)([0-
9]{%%B})(?^!\d)(.*)$:\1%pr%\2\3"
(SetLocal DisableDelayedExpansion) & GoTo multiply

:all
For /L %%A IN (1,1,12) DO Set reg%%A=/RegExp:"^(.*?)(?<!\d)([0-9]{%%A})(?!\d)(.*)$:\1%pr%\2\3"

:multiply
For /f "delims== tokens=1,2" %%A IN ('Set reg') DO Set %%A=%%B %%B %%B %%B %%B %%B %%B %%B
%%B %%B
SetLocal EnableDelayedExpansion
For /f "delims==" %%A IN ('Set reg^|Sort /r /+5') DO Set ord=!ord! %%%%A%%
SetLocal DisableDelayedExpansion
For /F "delims=" %%A IN ('echo %ord%') DO Set regex=%%A

: Rename
brc32 /Pattern:*.* /NoFolders %regex%
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Sample run:



Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Tim Notes:

1. Does not distinguish between numbers of same lengths in string

What I mean by this is if you have a string:

String = This 1 is a 2nd test of 345 _1.txt

And the program is instructed to append two Zeros to single numeric numbers -

New Name = This 001 is a 002nd test of 345 _001.txt

So be careful when making your selections.

If you don't want, i.e., ' 2nd ' included, then you are better off using a different RegEx.

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Contributor Analysis:

```
@echo off
cd/d "C:\YourDirPath\"
```

:: **Ask** user **how many zeros** to prefix.. Set**/a** ensures its numeric.(echo.This Does not Pad Zeros.) & (echo.It Prefixes Zeros based on the LENGTH of numeric digit-strings) & (echo.)

```
(Set/p zz= Specify a prefix from 1-12 zero's:_) & (Set/a zz=zz)
```

:: If **number-of-zeros** doesn't range from **1-12**, then exit

```
(If %zz% lss 1 exit) & (If %zz% gtr 12 exit)
```

:: Create a simple **string-of-zeros** to be used as %**pr**% (prefix)

```
(SetLocal EnableDelayedExpansion) & (For /L %%A IN (1,1,%zz%) DO Set pr=!pr!0)
```

:: Within ForLoops: %%**A** changes every run.. Typically, other %***vars***% may be changed, but their *new values* cant be expanded within a ForLoop.

: EnableDelayedExpansion allows such expansions with the format **!*var*!** (**%*var*%** is still allowed, but expands the pre-ForLoop value).

:: ForLoop runs **zz** times.. Each run adds **0** to **pr** (Note: *For's* variable %%**A** is not used here)

```
::  For /L %%A IN (1,1,%zz%) DO Set pr=!pr!0
::          Assuming user entered 3
::                  Run1:   Set pr=0
::                  Run2:   Set pr=00
::                  Run3:   Set pr=000  etc
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Contributor Analysis: cont.

```
:ask

::          Ask users if they want ALL-strings-processed (to avoid further prompts).

Set/p all=Prefix ALL lengths of digit-strings?   ^(y^)es or ^(n^)o:_

If /i "%all%"=="y" ((SetLocal DisableDelayedExpansion) & (GoTo all))

If /i NOT "%all%"=="n" ((echo.                    Please answer:   y for yes   n for no) & (echo.) & (Goto ask))

::          If answer is y, GoTo all... If answer is not n, GoTo ask and keep asking.

:: Show user the prefix, and how to answer upcoming prompts

(echo.)&(echo.)&echo.You have chosen to use the prefix: %pr%
echo.Choose below length^(s^) to process: y=YES   AllOthers=NO, including ^<Enter^>
echo.

:: Prompt user to create some variable names

For /L %%A IN (1,1,12) DO Set/p incl%%A= Prefix %%A-digit strings?

:: For's variable: %%A begins as 1, increments by 1 each run, and stops on 12
::          Run1:   Set/p incl1= Prefix 1-digit strings?
::          Run2:   Set/p incl2= Prefix 2-digit strings?  etc until...
::          Run12:  Set/p incl12= Prefix 12-digit strings?

:: Delete any variable-names (incl1 thru incl12) that do not have the value y or Y
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Contributor Analysis: cont.

```
For /L %%A IN (1,1,12) DO If /i not "!incl%%A!"=="y" Set incl%%A=

::          Run1:   If /i not "incl1"=="y" Set incl1=
::          Run2:   If /i not "incl2"=="y" Set incl2=  etc until...
::          Run12:  If /i not "incl12"=="y" Set incl12=

::          Note:
::          IF's /i ignores case-sensitivity, testing for not-equal to y or Y
::          SET deletes variables when its value is omitted after the = symbol.

:: If user makes no choices: Inform, then exit

Set incl 2>nul 1>nul || ((echo.) & (echo.NO selections made^^!  Press any key to exit) & (pause>nul) & (EXIT))

::          SET attempts to list variable-names beginning with incl, sending output to nul (not displayed).
::          If SET fails, its following (grouped commands) will process (|| being opposite of &&)

:: Create regex-variables based on user-selection

For /f "delims=l= tokens=1,2" %%A IN ('Set incl') DO Set reg%%B=/RegExp:"^^(.*?)(?<^!\d)([0-
9]{%%B})(?^!\d)(.*)$:\1%pr%\2\3"

::  ForLoop runs once for every varName beginning with incl

::          IF a user had selected: 1, 4, & 11 digit-strings..  Set incl outputs text-lines like: incl4=y

::          We use delims=l= to specify both l and = as separators to determine the FOR variables %%A and
::          %%B

::          With DO –
::          Set reg%%B=/RegExp:"^^(.*?)(?<^!\d)([0-9]{%%B})(?^!\d)(.*)$:\1%pr%\2\3"
::                  Run1:       Set reg1=/RegExp:"^^(.*?)(?<^!\d)([0-9]{1})(?^!\d)(.*)$:\1%pr%\2\3"
::                  Run2:       Set reg4=/RegExp:"^^(.*?)(?<^!\d)([0-9]{4})(?^!\d)(.*)$:\1%pr%\2\3"
::                  Run3:       Set reg11=/RegExp:"^^(.*?)(?<^!\d)([0-9]{11})(?^!\d)(.*)$:\1%pr%\2\3"
::                                          (where %pr% = prefixed-zeros)

::  Notes:
::    1. DelayedExpansion makes ! a special-character like ^, hence all the ^! and ^^'s
::    2. Since %pr%'s value never changed, there's no need for !pr!, while %%B is designed to change each run.
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Contributor Analysis: cont.

```
:: Regex-variables complete.. Skip past all-section

(SetLocal DisableDelayedExpansion) & GoTo multiply

:all

::      Create regex-variables for all lengths(1-12) of digit-strings

For /L %%A IN (1,1,12) DO Set reg%%A=/RegExp:"^(.*?)(?<!\d)([0-9]{%%A})(?!\d)(.*)$:\1%pr%\2\3"

:: ForLoop runs 12-times
::      Run1:   Set reg1=/RegExp:"^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$:\1%pr%\2\3"
::      Run2:   Set reg2=/RegExp:"^(.*?)(?<!\d)([0-9]{2})(?!\d)(.*)$:\1%pr%\2\3"  etc until 12

:multiply

::      For every regex-variable:  Set its new-value to its original-value repeated 10-times

For /f "delims== tokens=1,2" %%A IN ('Set reg') DO Set %%A=%%B %%B %%B %%B %%B %%B %%B %%B %%B %%B

:: ForLoop runs once for every varName starting with reg

::      'Set reg' will output TextLines in the format:
::              reg#=/RegExp:"^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$:\1%pr%\2\3"

::      With = as the separator, & tokens=1,2:
::              %%A=reg#  %%B=/Regexp:..

::      DO runs commands like:
::              Set reg#=/Regexp:.. /Regexp:.. /Regexp:.. /Regexp:.. (6 more times)
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Contributor Analysis: cont.

```
SetLocal EnableDelayedExpansion

:: Reverse-order all regex varNames (reg12,reg11...reg1) and save the text-string as ord

:: Crucial:
::      With prefix=0:
::      Processing 1/2/3/4 digit-lengths would yield: 2->02->002->0002->00002

For /f "delims==" %%A IN ('Set reg^|Sort /r /+5') DO Set ord=!ord! %%%%A%%

:: ForLoop runs once for every varName starting with reg

::      Set reg|Sort /r /+5 outputs TextLines like:  reg#=/RegExp:"...  (outputting the highest reg# 1st)

::      Using = as the separator, %%A will expand reg#

::      IF a user selected 1/2/6 digit-lengths:

::              DO Set ord=!ord! %%%%A%% runs commands like:
::                      Run1:   Set ord= %reg6%
::                      Run2:   Set ord= %reg6% %reg2%
::                      Run3:   Set ord= %reg6% %reg2% %reg1%

SetLocal DisableDelayedExpansion

For /F "delims=" %%A IN ('echo %ord%') DO Set regex=%%A

::      Not really a Loop (1-run)..  If a user is only processing 1/2/6 digit-lengths:

::      echo %ord% would run the command echo %reg6% %reg2% %reg1%, outputting the values of
::      each %reg#%

::      Outputting a string like: /Regexp:..{6}..(9more-times) /Regexp:..{2}..(9more-times) ::  /Regexp:..{1}..(9more-times)

::      With no separator specified by delims, %%A represents the entire-string, to be saved by the
::      varName regex

: Rename

brc32 /Pattern:*.* /NoFolders %regex%
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Tim Contribution:

This first RegEx will match the first occurrence of a numeric group that meets the requirements set by pr and %B%

Where pr = number of zeroes to prefix onto numeric groups
%B% = the length of the numeric group to prefix.

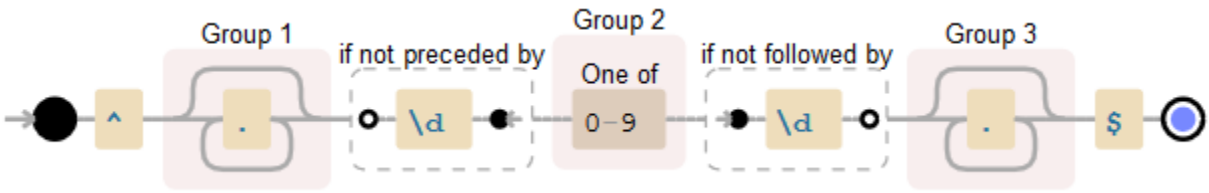
i.e.

5 = length = %B% = 1
24 = length = %B% = 2
345 = length = %B% = 3

For this example:

Pr - the number of zeroes to prefix = 3 – displayed in the Replacement String as prefixed zeroes
%B% - the length = 1

Match: ^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$



Samples

```
.................................
Match 1: file_1_ 7 25 name.1 18
Group 1: file_
Group 2: 1
Group 3: _ 7 25 name.1 18
```

```
.................Start.Length...
Match 1: file1       0      5
Group 1: file        0      4
Group 2: 1           4      1
Group 3:             5      0
```

```
.............................
Match 1: text 1 2 3 4 5
Group 1: text
Group 2: 1
Group 3:  2 3 4 5
```

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Tim Contribution: cont.

Samples:

file_.01._name.01	file_.01_name.01	file_01_name.01	file_01_name.1
file_01_name01	file_1 456name.101	file_123_test101_name201	file_1_ 456 name.101
file_1_ 456 name101	file_1_ 7 25 name.1	file_1_ 7 25 name.1 18	file_1_7 25 name.1 18
file_1_name.01	file_1_name.1	file_1_name.101	file_1_name01
file_1_name1	file_1_name101	file_1_test1_name01	file_1_test1_name1
text 1 2 3 4 5	45 52.txt	file_01_name01 text.tif	

Match: ^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$
Replace: \1000\2\3

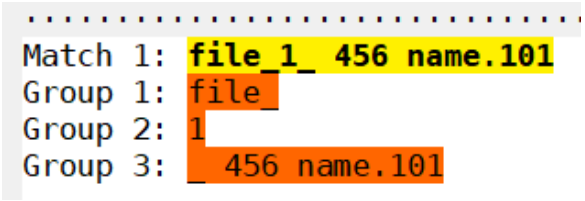
	New Name
45 52.txt	45 52.txt
file1.ext	file0001.ext
file_.01._name.01.tif	file_.01._name.01.tif
file_.01_name.01.tif	file_.01_name.01.tif
file_01_name.01.tif	file_01_name.01.tif
file_01_name.1.tif	file_01_name.0001.tif
file_01_name01 text.tif	file_01_name01 text.tif
file_01_name01.tif	file_01_name01.tif
file_1 456name.101.tif	file_0001 456name.101.tif
file_123_test101_name201 text.tif	file_123_test101_name201 text.tif
file_123_test101_name201.tif	file_123_test101_name201.tif
file_1_ 456 name.101.tif	file_0001_ 456 name.101.tif
file_1_ 456 name101.tif	file_0001_ 456 name101.tif
file_1_ 7 25 name.1 18.tif	file_0001_ 7 25 name.1 18.tif
file_1_ 7 25 name.1.tif	file_0001_ 7 25 name.1.tif
file_1_7 25 name.1 18.tif	file_0001_7 25 name.1 18.tif
file_1_name.01.tif	file_0001_name.01.tif
file_1_name.1.tif	file_0001_name.1.tif
file_1_name.101.tif	file_0001_name.101.tif
file_1_name01.tif	file_0001_name01.tif
file_1_name1.tif	file_0001_name1.tif
file_1_name101.tif	file_0001_name101.tif
file_1_test1_name01.tif	file_0001_test1_name01.tif
file_1_test1_name1.tif	file_0001_test1_name1.tif
text 1 2 3 4 5.tif	text 0001 2 3 4 5.tif
text 1 2 3 4 text 5.tif	text 0001 2 3 4 text 5.tif
text 1 text 2 text 3 4 5	text 0001 text 2 text 3 4 5
text 1 text 2 text 3 4 5 6 7 89	text 0001 text 2 text 3 4 5 6 7 89
text 1 text 2 text 3 4 5 text	text 0001 text 2 text 3 4 5 text

Add

Append Zeros to Prefix of Numeric Numbers of varying Lengths in String cont.

Tim Contribution: cont.

Using: file_1_ 456 name.101.tif



Match: ^(.*?)(?<!\d)([0-9]{1})(?!\d)(.*)$

Where:

\1 = file_ \2 = 1 \3 = _ 456 name.101

Breaks down:

Capture Group 1		Capture Group 2		Capture Group 3		
	Negative Lookbehind		Negative Lookahead			
Anchor		Class				Anchor
^	(.*?)	(?<!\d)	([0-9]{1})	(?!\d)	(.*)	$

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 =
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. (?>!\d)	Negative Lookbehind. (Not Captured) Look to the left that the last Character is 'not' a numeric digit. = BOL = True	Capture Group = Unchanged
6. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 2 = 1 Changes Capture Group 1 = file_
7. {1}	Repeat last expression to match 1 iteration.	
8. (?!\d)	Negative Lookahead. (Not Captured) Look to the right that the next Character is 'not' a numeric digit. = underscore = True	
9. .*	Match against remaining string	Capture Group 3 = _ 456 name.101
10. $	Must be at EOL (End of Line)	

Add

Insert Spaces between First Occurrence of Hyphen in String

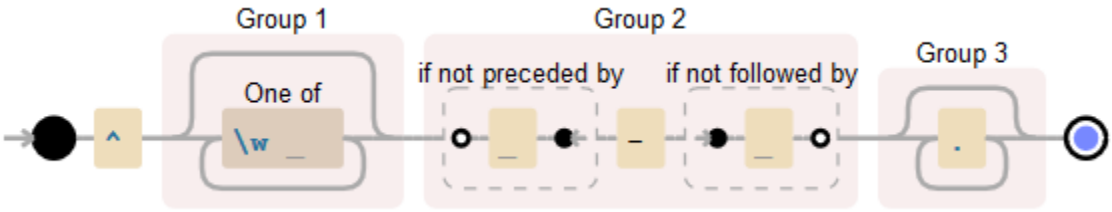
xxx_xxxx-xxxxx-xxx	to	xxx_xxxx - xxxxx-xxx
ABC_DEF_GHI-123-45	to	ABC_DEF_GHI - 123-45
ABC-123-45	to	ABC - 123-45
xxx xxx-xxx.txt	to	xxx xxx - xxx.txt
1234 678 - abcs	to	Ignore

Observations:

1. Place a <space> on either side of the FIRST <hyphen> "-" of the filename.
2. Ignore those files that already have this format.
3. Retain everything before the <hyphen>.
4. Retain everything after last added <space>
5. Pattern established: <text>-<text>

Match: ^([\w]*)((?<!)-(?!))(.*)
Replace: \1 \2 \3

Name	New Name
1234 678 - abcs	1234 678 - abcs
xxx xxx-xxx.txt	xxx xxx - xxx.txt
ABC_DEF_GHI-123-45	ABC_DEF_GHI - 123-45
ABC-123-45	ABC - 123-45
xxx_xxxx-xxxxx-xxx	xxx_xxxx - xxxxx-xxx



Sample:

```
.............................
Match 1: ABC_DEF_GHI-123-45
Group 1: ABC_DEF_GHI
Group 2: -
Group 3: 123-45
```

Add

Insert Spaces between First Occurrence of Hyphen in String cont.

xxx_xxxx-xxxxx-xxx	to	xxx_xxxx - xxxxx-xxx
ABC_DEF_GHI-123-45	to	ABC_DEF_GHI - 123-45
ABC-123-45	to	ABC - 123-45
xxx xxx-xxx.txt	to	xxx xxx - xxx.txt
1234 678 - abcs	to	Ignore

Match: ^([\w]*)((?<!)-(?!))(.*)

Contributor Analysis:

1. The Word Metacharacter \w matches against uppercase and lowercase letters, numeric digits and the underscore. In addition this does not include <space> characters, so it wouldn't match <hyphens> with <spaces> bordering them.
   a. This would account for not matching against filenames that already have the format of:

<space> <hyphen> <space>

2. To make sure it only works with the first occurrence of the <hyphen> I used ^ (BOL Anchor) at the start.
3. So the initial regex would be:

^([\w]*)(-)(.*)

4. The only problem now is that it won't match something like:

xxx xxx-xxx.txt

because it has a <space> in the name. We can fix this by adding a <space> to make Capture Group 1.

([\w])

5. To make it ignore the filename if it has a <space> before or after the initial <hyphen> (or both), you can use negative lookbehind\lookahead around Capture Group 2, which was initially just a <hyphen> (-).

Now you have this for group 2:

((?<!)-(?!))

Forum Contributor: Glenn

Tim Notes:

1. This is the same RegEx as in, 'Add spaces between first hyphen'.

Add

Insert Spaces between First Occurrence of Hyphen in String cont.

Using: ABC_DEF_GHI-123-45

Match: ^([\w]*)((?<!)-(?!))(.*)

Where:

\1 = ABC_DEF_GHI \2 = - \3 = 123-45

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. [\w]	Match against a class consisting of a. Word Metacharacter. Match against any single letter, number or underscore. b. <space> character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1= ABC_DEF_GHI
4. (?<!)	Negative Lookbehind. Look to the left that the last character is 'not' a <space> = 'H' = True	Capture Group 2 = No Value
5. -	Match against <hyphen> This is the <hyphen>	Capture Group 2 = -
6. (?!)	Negative Lookahead. Look to the right that the next Character is 'not' a <space> = '1' = True	Capture Group 2 = Unchanged
7. .*	Match against remaining string	Capture Group 3 = 123-45

Add

Add prefix Based on name

Begin_SAIL	to	S_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	R_Begin_RAIL_End
Bahá'í Muñoz	to	Ignore
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	Ignore

Observations:

1. Add a prefix based on a keyword in the filename.
2. keyword is in all uppercase
3. Prefix is the first initial of the keyword.
4. Keyword can appear anywhere in string.

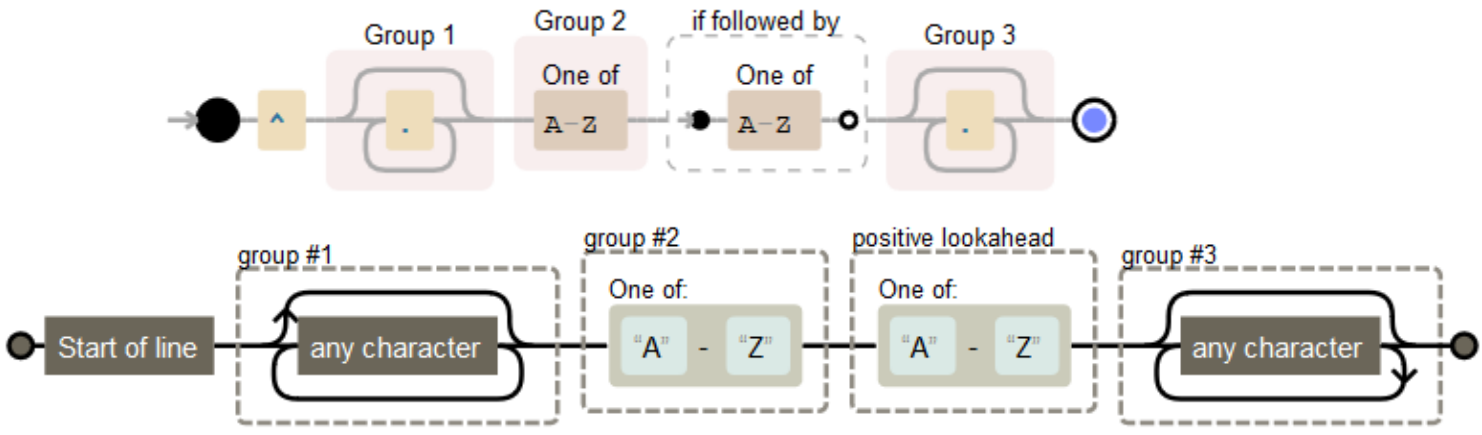
Tim Contribution –

Take uppercase letter of the first uppercase word and prefix it to front

Method #1

Match: ^(.*?)([A-Z])(?=[A-Z])(.*)
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	A_ABC_DEF_GHI-123-45
Begin_RAIL_End	R_Begin_RAIL_End
TAIL_end	T_TAIL_end
Begin_SAIL	S_Begin_SAIL
8259438 some text	8259438 some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	Bahá'í Muñoz



Add

Add prefix Based on name cont.

Begin_SAIL	to	S_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	R_Begin_RAIL_End
Bahá'í Muñoz	to	Ignore
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	Ignore

Tim Contribution – cont.

Take uppercase letter of the first uppercase word and prefix it to front cont.

Method #1

Match: ^(.*?)([A-Z])(?=[A-Z])(.*)

Sample:

```
.........................
Match 1: Begin_RAIL_End
Group 1: Begin_
Group 2: R
Group 3: AIL_End
```

Sample:

```
.....................
Match 1: Begin_SAIL
Group 1: Begin_
Group 2: S
Group 3: AIL
```

Sample:

```
.............................Start·Length···
Match 1: ABC_DEF_GHI-123-45      0      18
Group 1:                         0       0
Group 2: A                       0       1
Group 3: BC_DEF_GHI-123-45       1      17
```

Breaks down:

	Capture Group 1	Capture Group 2	Lookahead Non-marking Group	Capture Group 3
Anchor		Class	Class	
^	(.*?)	([A-Z])	(?=[A-Z])	(.*)

Add

Add prefix Based on name cont.

Using: ABC_DEF_GHI-123-45

Tim Contribution – cont.

Take uppercase letter of the first uppercase word and prefix it to front cont.

Method #1

Match: ^(.*?)([A-Z])(?=[A-Z])(.*)

Where:

\1 = null (BOL) \2 = A \3 = BC_DEF_GHI-123-45

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = A Capture Group 1 = Unchanged
6. (?=[A-Z])	Lookahead. Look to the right. If next character is a single Uppercase letter then expression = True and evaluation continues.= 'B' so TRUE	
7. .*	Match against remaining string	Capture Group 3 = BC_DEF_GHI-123-45

Add

Add prefix Based on name cont.

Begin_SAIL	to	S_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	R_Begin_RAIL_End
Bahá'í Muñoz	to	Ignore
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	Ignore

Tim Contribution – cont.

Take uppercase letter of the first uppercase word and prefix it to front cont.

Method #2:

Match: ^(([A-Z])(?=[A-Z])|(.+?)([A-Z])(?=[A-Z]))(.*)
Replace: \2\4_\1\5

Name	New Name
ABC_DEF_GHI-123-45	A_ABC_DEF_GHI-123-45
Begin_RAIL_End	R_Begin_RAIL_End
TAIL_end	T_TAIL_end
Begin_SAIL	S_Begin_SAIL
8259438 some text	8259438 some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	Bahá'í Muñoz
My File v.1.2.345	My File v.1.2.345

Sample:

```
·····················Start·Length·····
Match 1: ABC-123-45       0      10
Group 1: A                0       1
Group 2: A                0       1
Group 3 did not participate in the match
Group 4 did not participate in the match
Group 5: BC-123-45        1       9
```

Sample:

```
·····················Start·Length·····
Match 1: Begin_SAIL       0      10
Group 1: Begin_S          0       7
Group 2 did not participate in the match
Group 3: Begin_           0       6
Group 4: S                6       1
Group 5: AIL              7       3
```

Sample:

```
·····················Start·Length·····
Match 1: Begin_RAIL_End       0      14
Group 1: Begin_R              0       7
Group 2 did not participate in the match
Group 3: Begin_               0       6
Group 4: R                    6       1
Group 5: AIL_End              7       7
```

Add

Add prefix Based on name cont.

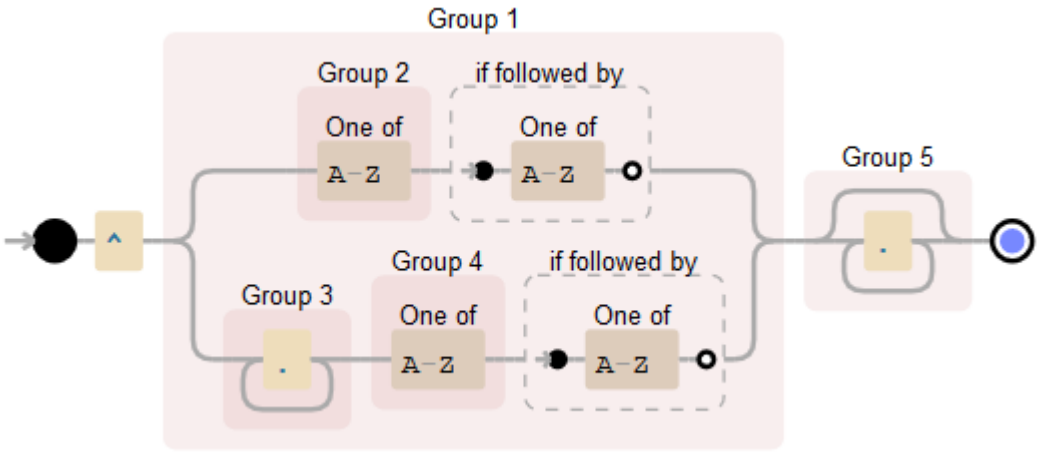
Begin_SAIL	to	S_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	R_Begin_RAIL_End
Bahá'í Muñoz	to	Ignore
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	Ignore

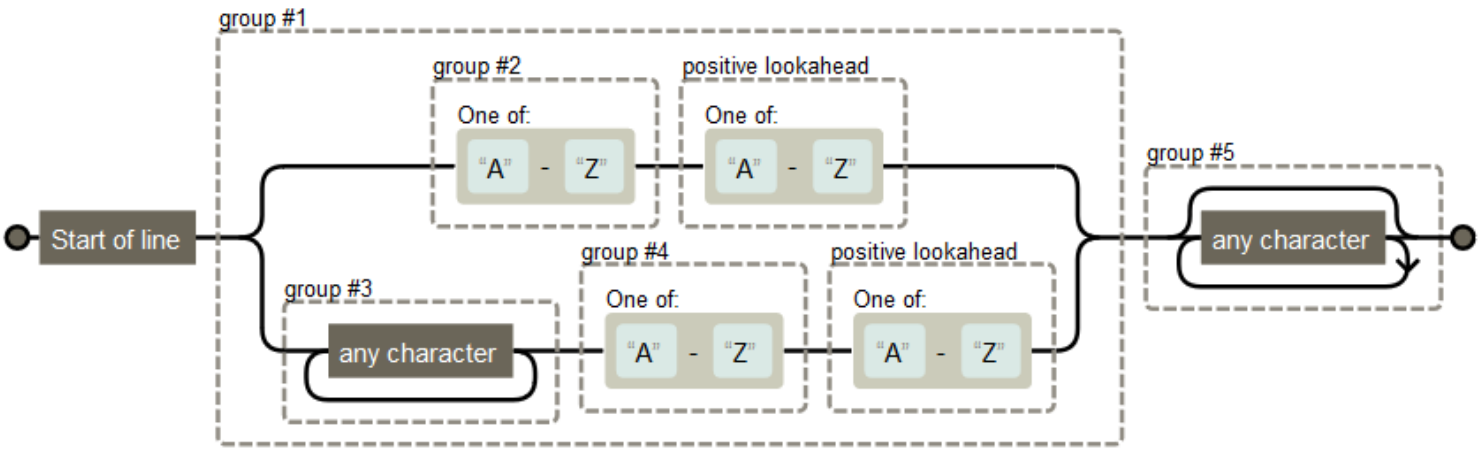
Tim Contribution – cont.

Take uppercase letter of the first uppercase word and prefix it to front cont.

Method #2: cont.

Match: ^(([A-Z])(?=[A-Z])|(.+?)([A-Z])(?=[A-Z]))(.*)





Add

Add prefix Based on name cont.

Using: Begin_RAIL_End

Tim Contribution – cont.

Take uppercase letter of the first uppercase word and prefix it to front cont.

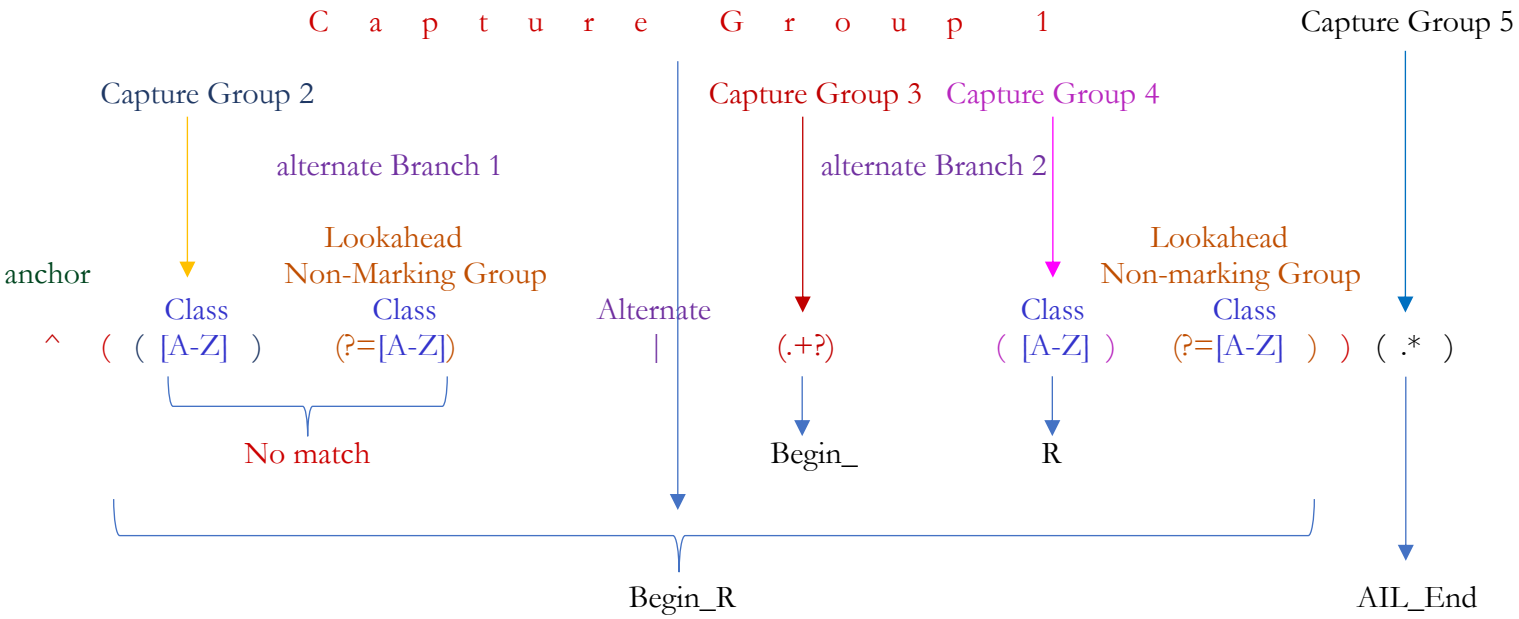
Method #2: cont.

Match: ^(([A-Z])(?=[A-Z])|(.+?)([A-Z])(?=[A-Z]))(.*)

Where:

\1 = Begin_R \2 = No value \3 = Begin_ \4 = R \5 = AIL_End

Breaks down:



This uses an Alternate for Capture Group 1, 2, 3 & 4

This OR This

([A-Z])(?=[A-Z]) | (.+?)([A-Z])(?=[A-Z])

Tim Notes:

1. The alternates are used to determine the location of the keyword in the string. The keyword is in all uppercase.
2. The first alternate tests for an uppercase letter followed by a second uppercase letter. This first alternate is to test if the keyword is at the beginning of the string.
3. The second alternate tests for the keyword in other parts of the string.

Add

Add prefix Based on name cont.

Using: Begin_RAIL_End

Tim Contribution – cont.

Take uppercase letter of the first uppercase word and prefix it to front cont.

Method #2: cont.

Match: ^(([A-Z])(?=[A-Z])|(.+?)([A-Z])(?=[A-Z]))(.*)

Where:

\1 = Begin_R \2 = No value \3 = Begin_ \4 = R \5 = AIL_End

Tim Analysis:

1. ^ Start at beginning of string (BOL)

2. ([A-Z])(?=[A-Z])|(.+?)([A-Z])(?=[A-Z])

Alternate (see Previous Page)

a. [A-Z]	Match against a Class consisting of an uppercase letter	Capture Group 1 = B Capture Group 2 = B
b. (?=[A-Z])	Lookahead. Look to the right. If next character is a single uppercase letter then expression = True and evaluation continues. = 'e' so False No Match	Capture Group 1 = No Value Capture Group 2 = No Value

or

a. ..	Match against any character	Capture Group 3, 1 = B
b. +	Make it Greedy	Capture Group 3, 1 = <entire string>
c. ?	But not too Greedy	Capture Group 3, 1 = B Capture Group 2 = Unchanged
d. [A-Z]	Match against a Class consisting of an uppercase letter	Capture Group 4 = R Capture Group 3 = Begin_ Capture Group 1 = Begin_R
e. (?=[A-Z])	Lookahead. Look to the right. If next character is a single uppercase letter then expression = True and evaluation continues. = 'A' so True	All Capture Groups = Unchanged

3. .* Match against remaining string Capture Group 5 = AIL_End

Add

Add prefix Based on name cont.

Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
Bahá'í Muñoz	to	z_Bahá'í Muñoz
ABC-123-45	to	C_ABC-123-45
ABC_DEF_GHI-123-45	to	I_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text

Observations:

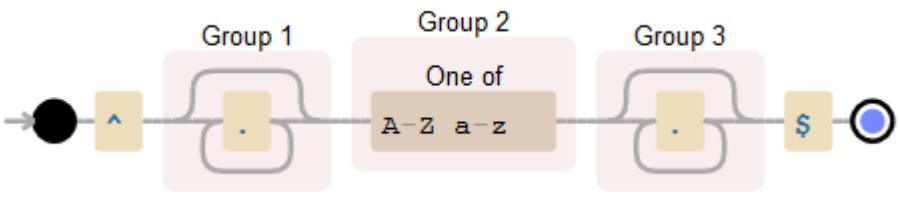
1. Add a prefix based on a character in the filename.
2. Character can be uppercase or lowercase
3. Prefix is the last alpha character in the string.

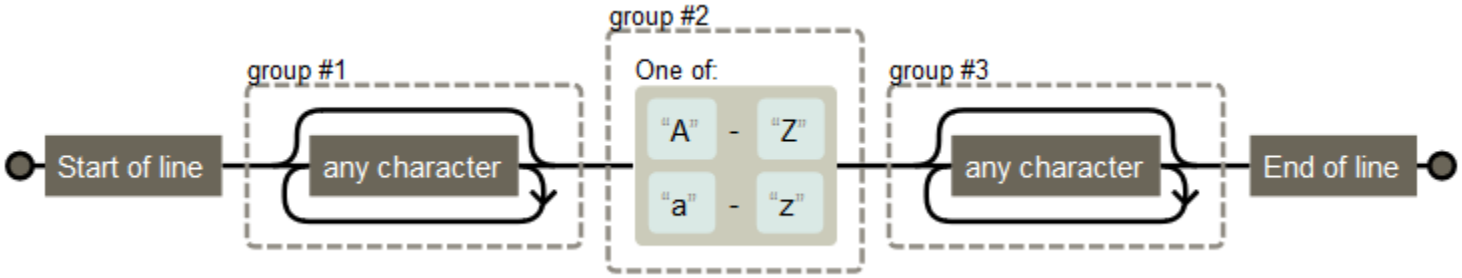
Tim Contribution –

Take the last lowercase or uppercase letter of string and prefix it to front:

Match: ^(.*)([A-Za-z])(.*)$
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	I_ABC_DEF_GHI-123-45
Begin_RAIL_End	d_Begin_RAIL_End
TAIL_end	d_TAIL_end
Begin_SAIL	L_Begin_SAIL
8259438 some text	t_8259438 some text
ABC-123-45	C_ABC-123-45
Bahá'í Muñoz	z_Bahá'í Muñoz
My File v.1.2.345	v_My File v.1.2.345





Add

Add prefix Based on name cont.

Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
Bahá'í Muñoz	to	z_Bahá'í Muñoz
ABC-123-45	to	C_ABC-123-45
ABC_DEF_GHI-123-45	to	I_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text

Tim Contribution – cont.

Take the last lowercase or uppercase letter of string and prefix it to front: cont.

Match: ^(.*)([A-Za-z])(.*)$

Breaks down:

Capture Group 1	Capture Group 2	Capture Group 3
	Class	
Anchor		Anchor
^ (.*)	([A-Za-z])	(.*) $

Samples

	Start	Length
Match 1:	Begin_SAIL	0	10
Group 1:	Begin_SAI	0	9
Group 2:	L	9	1
Group 3:		10	0

	Start	Length
Match 1:	Begin_RAIL_End	0	14
Group 1:	Begin_RAIL_En	0	13
Group 2:	d	13	1
Group 3:		14	0

Match 1:	my.File2.v1.2.345
Group 1:	my.File2.
Group 2:	v
Group 3:	1.2.345

	Start	Length
Match 1:	8259438 some text	0	17
Group 1:	8259438 some tex	0	16
Group 2:	t	16	1
Group 3:		17	0

Add

Add prefix Based on name cont.

Using: Begin_RAIL_End

Tim Contribution – cont.

Take the last lowercase or uppercase letter of string and prefix it to front: cont.

Match: ^(.*)([A-Za-z])(.*)$

Where:

\1 = Begin_Rail_En \2 = d \3 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = B
3. *	Make it Greedy	Capture Group 1 = <entire string>
	Current position = EOL	
4. [A-Za-z]	Match against a class consisting of an uppercase or lowercase letter	Capture Group 2 = d
	Backtracks to match against the 'd'	Changes Capture Group 1 = Begin_Rail_En
	Current position = EOL	
5. .*	Match against remaining string	Capture Group 3 = null
	Already at EOL so it results in a Zero Occurrence Match	
6. $	Must be at EOL (End of Line)	

Add

Add prefix Based on name cont.

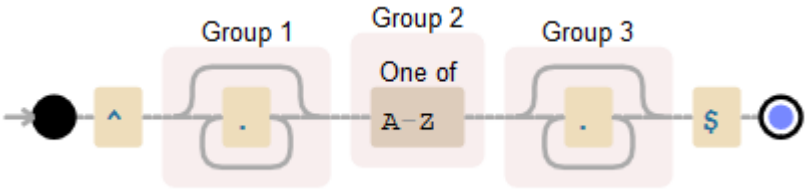
Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
Bahá'í Muñoz	to	z_Bahá'í Muñoz
ABC-123-45	to	C_ABC-123-45
ABC_DEF_GHI-123-45	to	I_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text

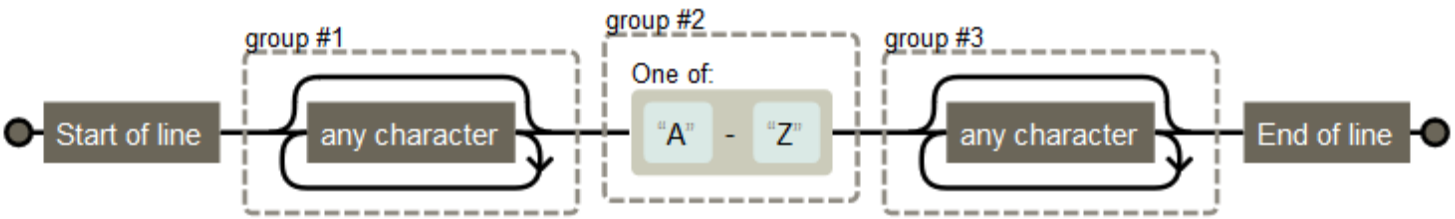
Tim Contribution – cont.

Take the last Uppercase character of string and prefix it to front:

Match: ^(.*)([A-Z])(.*)$
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	I_ABC_DEF_GHI-123-45
Begin_RAIL_End	E_Begin_RAIL_End
TAIL_end	L_TAIL_end
Begin_SAIL	L_Begin_SAIL
8259438 some text	8259438 some text
ABC-123-45	C_ABC-123-45
Bahá'í Muñoz	M_Bahá'í Muñoz
My File v.1.2.345	F_My File v.1.2.345





Add

Add prefix Based on name cont.

Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
Bahá'í Muñoz	to	z_Bahá'í Muñoz
ABC-123-45	to	C_ABC-123-45
ABC_DEF_GHI-123-45	to	I_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text

Tim Contribution – cont.

Take the last Uppercase character of string and prefix it to front: cont.

Match: ^(.*)([A-Z])(.*)$

Breaks down:

	Capture Group 1		Capture Group 2		Capture Group 3	
			Class			
Anchor						Anchor
^	(.*)	(	[A-Z]	)	(.*)	$

Samples

```
.....................Start·Length··
Match 1: Begin_SAIL      0     10
Group 1: Begin_SAI       0      9
Group 2: L               9      1
Group 3:                10      0
```

```
.............................
Match 1: ABC_DEF_GHI-123-45
Group 1: ABC_DEF_GH
Group 2: I
Group 3: -123-45
```

```
.............................
Match 1: My File v.1.2.345
Group 1: My
Group 2: F
Group 3: ile v.1.2.345
```

```
.........................
Match 1: Begin_RAIL_End
Group 1: Begin_RAIL_
Group 2: E
Group 3: nd
```

Add

Add prefix Based on name cont.

Using: Begin_RAIL_End

Tim Contribution – cont.

Take the last Uppercase character of string and prefix it to front: cont.

Match: ^(.*)([A-Z])(.*)$

Where:

\1 = Begin_Rail_ \2 = E \3 = nd

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = B
3. *	Make it Greedy	Capture Group 1 = <entire string>
	Current position = EOL	
4. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = E
	Backtracks to match against the 'd'	Changes Capture Group 1 = Begin_Rail_
	Current position = EOL	
5. .*	Match against remaining string	Capture Group 3 = nd
	Already at EOL so it results in a Zero Occurrence Match	
6. $	Must be at EOL (End of Line)	

Tim Notes:

1. Same RegEx as 'Take the last lowercase or uppercase letter of string and prefix it to front',..only removed lowercase from class.

Add

Add prefix Based on name cont.

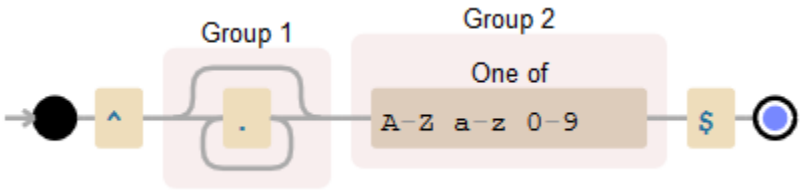
Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
ABC_DEF_GHI-123-45	to	5_ABC_DEF_GHI-123-45
nc_017733	to	3_nc_017733
1	to	1_1

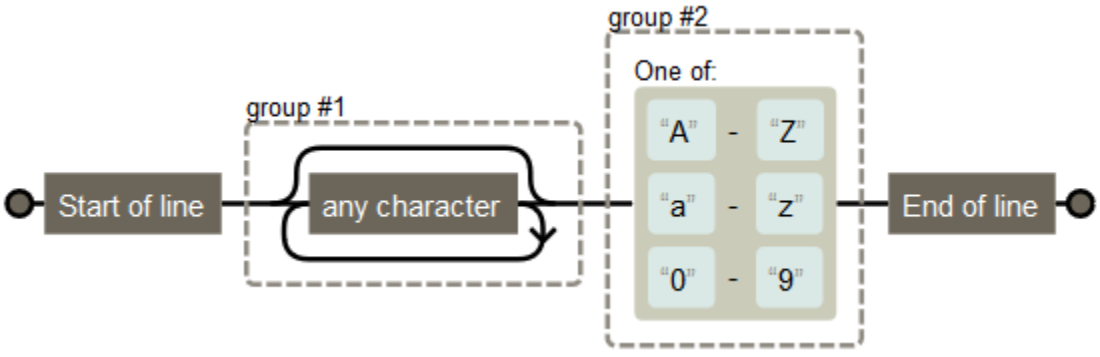
Tim Contribution – cont.

Take the last alphanumeric character of string and prefix it to front:

Match: ^(.*)([A-Za-z0-9])$
Replace: \2_\1\2

Name	New Name
nc_017733	3_nc_017733
ABC_DEF_GHI-123-45	5_ABC_DEF_GHI-123-45
Begin_RAIL_End	d_Begin_RAIL_End
1	1_1
TAIL_end	d_TAIL_end
Begin_SAIL	L_Begin_SAIL





Add

Add prefix Based on name cont.

Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
ABC_DEF_GHI-123-45	to	5_ABC_DEF_GHI-123-45
nc_017733	to	3_nc_017733
1	to	1_1

Tim Contribution – cont.

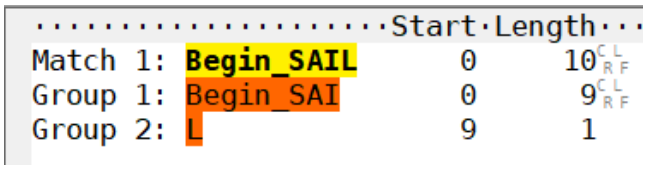
Take the last alphanumeric character of string and prefix it to front: cont.

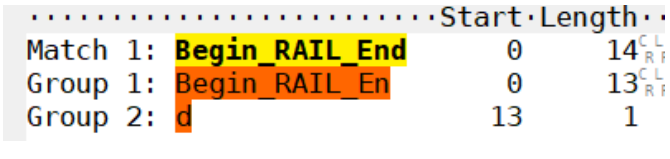
Match: ^(.*)([A-Za-z0-9])$

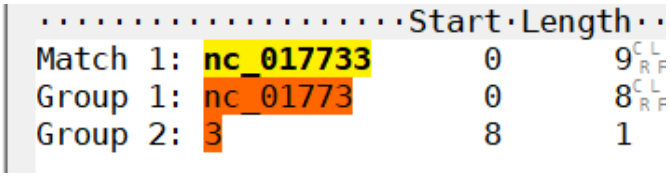
Breaks down:

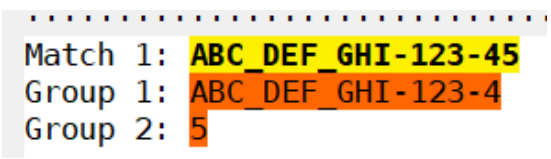
Capture Group 1		Capture Group 2		
		Class		
Anchor ^	(.*)	(	[A-Za-z0-9]	)	Anchor $

Samples









Add

Add prefix Based on name cont.

Using: ABC_DEF_GHI-123-45

Tim Contribution – cont.

Take the last alphanumeric character of string and prefix it to front: cont.

Match: ^(.*)([A-Za-z0-9])$

Where:

\1 = ABC_DEF_GHI-123-4 \2 = 5

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = <entire string>
	Current position = EOL	
4. [A-Za-z0-9]	Match against a class consisting of an uppercase or lowercase letter or a numeric digit.	Capture Group 2 = 5 Changes Capture Group 1 = ABC_DEF_GHI-123-4
	Backtracks to match against the 'd'	
	Current position = EOL	
5. $	Must be at EOL (End of Line)	

Add

Add prefix Based on name cont.

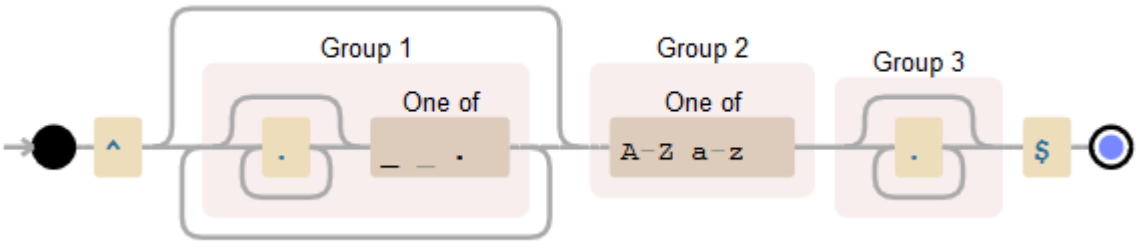
Begin_SAIL	to	S_Begin_SAIL
TAIL_end	to	e_TAIL_end
Begin_RAIL_End	to	E_Begin_RAIL_End
Bahá'í Muñoz	to	M_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	G_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text
My File v.1.2.345	to	v_My File v.1.2.345

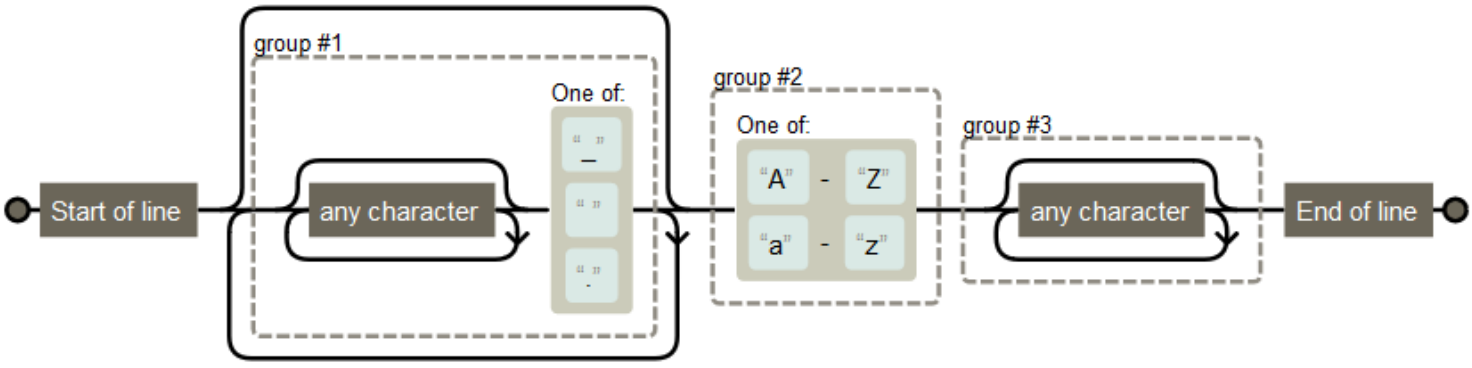
Tim Contribution – cont.

Take the initial of the last word (upper or lowercase) and prefix it to the front:

Match: ^(.*[_ \.-])*([A-Za-z])(.*)$
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	G_ABC_DEF_GHI-123-45
Begin_RAIL_End	E_Begin_RAIL_End
TAIL_end	e_TAIL_end
Begin_SAIL	S_Begin_SAIL
8259438 some text	t_8259438 some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	M_Bahá'í Muñoz
My File v.1.2.345	v_My File v.1.2.345





With thanks to RegexNinja

Add

Add prefix Based on name cont.

Begin_SAIL	to	S_Begin_SAIL
TAIL_end	to	e_TAIL_end
Begin_RAIL_End	to	E_Begin_RAIL_End
Bahá'í Muñoz	to	M_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	G_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text
My File v.1.2.345	to	v_My File v.1.2.345

Tim Contribution – cont.

Take the initial of the last word (upper or lowercase) and prefix it to the front cont.

Match: ^(.*[_ \.-])*([A-Za-z])(.*)$

Breaks down:

	Capture Group 1				Capture Group 2			Capture Group 3	
Anchor		Class			Class				Anchor
^	(	.*	[_ \.-]	)	*	(	[A-Za-z]	)	(.*)	$

Where:

Class of Capture Group 1 is made up of: [underscore <space> <dot> <hyphen>]

Samples

```
.............................
Match 1: ABC_DEF_GHI-123-45
Group 1: ABC_DEF_
Group 2: G
Group 3: HI-123-45
```

```
.....................
Match 1: Begin_SAIL
Group 1: Begin_
Group 2: S
Group 3: AIL
```

```
.........................
Match 1: My File v.1.2.345
Group 1: My File
Group 2: v
Group 3: .1.2.345
```

```
.........................
Match 1: Begin_RAIL_End
Group 1: Begin_RAIL_
Group 2: E
Group 3: nd
```

```
.............................
Match 1: 8259438 some text
Group 1: 8259438 some
Group 2: t
Group 3: ext
```

Add

Add prefix Based on name cont.

Using: ABC_DEF_GHI-123-45

Tim Contribution – cont.

Take the initial of the last word (upper or lowercase) and prefix it to the front cont.

Match: ^(.*[_ \.-])*([A-Za-z])(.*)$

Where:

\1 = ABC_DEF_ \2 = G \3 = HI-123-45

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = A
3. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. [_ \.-]	Match against a class consisting of an a. underscore b. <space> c. literal <dot> d. <hyphen>	Capture Group 1 = ABC_DEF_GHI-123
5. *	Make it Greedy	Capture Group 1 = Unchanged
6. [A-Za-z]	Match against a class consisting of an uppercase or lowercase letter	Capture Group 2 = G Changes Capture Group 1 = ABC_DEF_
7. .*	Match against remaining string	Capture Group 3 = HI-123-45
8. $	Must be at EOL (End of Line)	

Add

Add prefix Based on name cont.

Using: ABC_DEF_GHI-123-45

Tim Contribution – cont.

Take the initial of the last word (upper or lowercase) and prefix it to the front cont.

Match: ^(.*[_ \.-])*([A-Za-z])(.*)$

Tim Notes:

1. To change it to the Last word instead of last initial change it from:

Match: ^(.*[_ \.-])*([A-Za-z])(.*)$

to:

Match: ^(.*[_ \.-])*([A-Za-z]+)(.*)$

by just adding the Greedy Quantifier. (Capture Group 2 is now the Last Word in string)

Name	New Name
ABC_DEF_GHI-123-45	GHI
Begin_RAIL_End	End
TAIL_end	end
Begin_SAIL	SAIL
8259438 some text	text
ABC-123-45	ABC
Bahá'í Muñoz	Mu
My File v.1.2.345	v.345

Won't work with the High ASCII of Bahá'í Muñoz.

Add

Add prefix Based on name cont.

Using: ABC_DEF_GHI-123-45

Tim Contribution – cont.

Take the initial of the last word (upper or lowercase) and prefix it to the front cont.

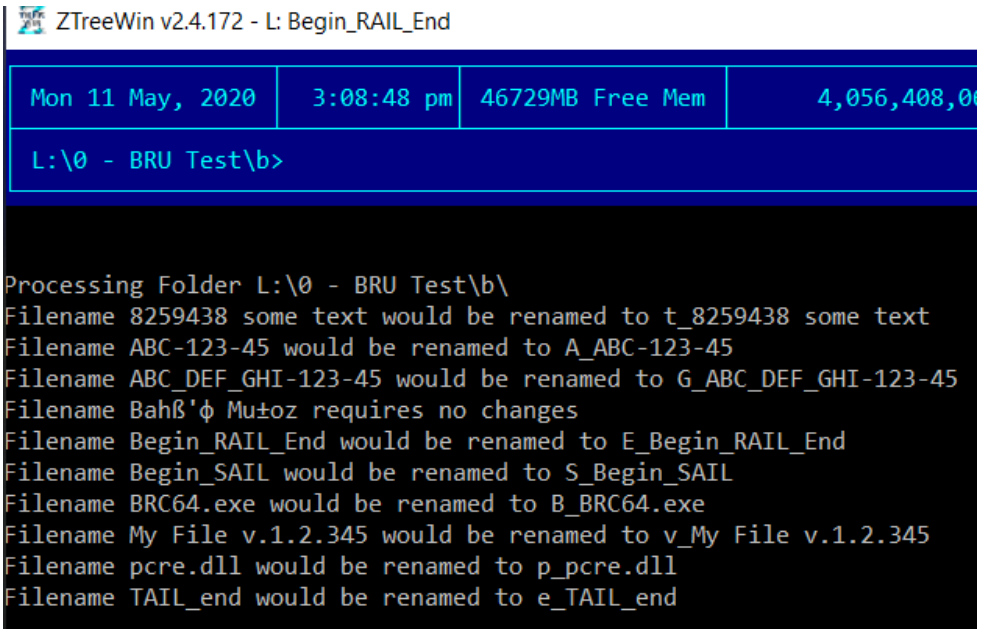
JavaScript Equivalent:

```
newName=name.replace(/^(.*[_ \.])*([A-Za-z])(.*)$/, '$2'+'_'+'$1'+'$2'+'$3')
```

Name	New Name
ABC_DEF_GHI-123-45	G_ABC_DEF_GHI-123-45
Begin_RAIL_End	E_Begin_RAIL_End
TAIL_end	e_TAIL_end
Begin_SAIL	S_Begin_SAIL
8259438 some text	t_8259438 some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	M_Bahá'í Muñoz
My File v.1.2.345	v_My File v.1.2.345

BRC Equivalent:

brc64 /Pattern: /Dir: /NoFolders /Regexp:"^(.*[_ \.])*([A-Za-z])(.*)$:\2_\1\2\3"



Tim Notes:

1. BRC does NOT support High ASCII (this is a known issue). Therefore the sample, Bahá'í Muñoz, is unchanged.

Thanks to RegexNinja

Add

Add prefix Based on name cont.

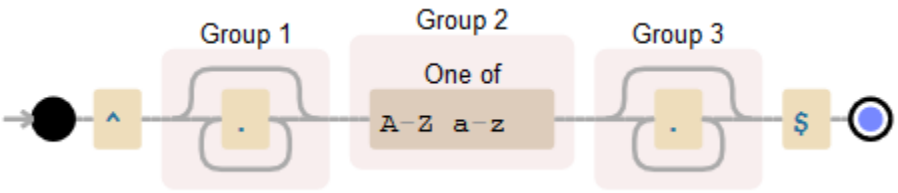
Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

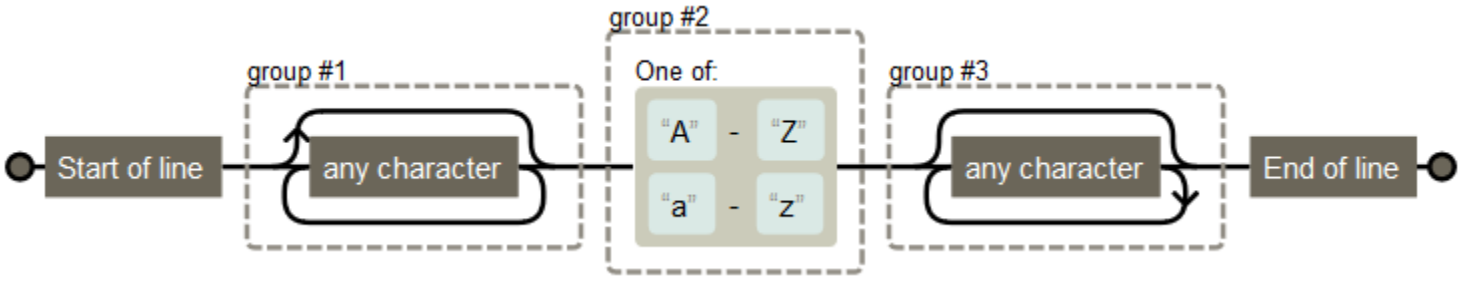
Tim Contribution – cont.

Take the first letter (upper or lowercase) and prefix it to the front:

Match: ^(.*?)([A-Za-z])(.*)$
Replace: \2_\1_\2\3

Name	New Name
ABC_DEF_GHI-123-45	A_ABC_DEF_GHI-123-45
Begin_RAIL_End	B_Begin_RAIL_End
TAIL_end	T_TAIL_end
Begin_SAIL	B_Begin_SAIL
8259438 some text	s_8259438 _some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	B_Bahá'í Muñoz
My File v.1.2.345	M_My File v.1.2.345





Add

Add prefix Based on name cont.

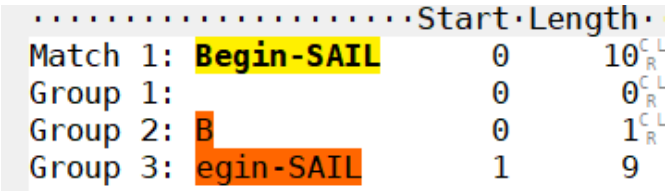
Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

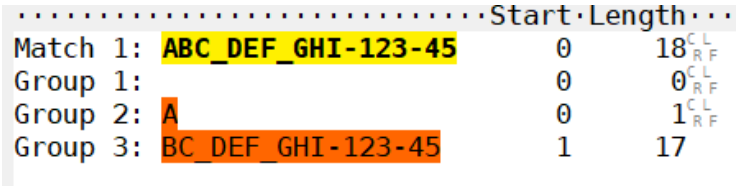
Tim Contribution – cont.

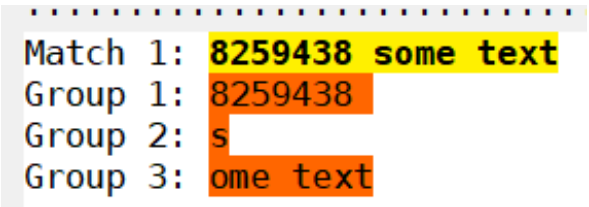
Take the first letter (upper or lowercase) and prefix it to the front: cont.

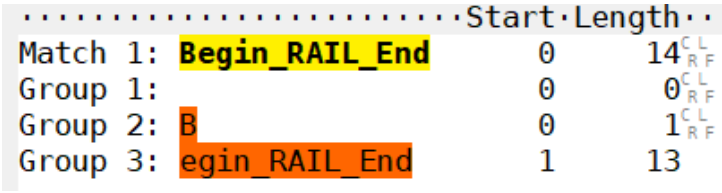
Match: ^(.*?)([A-Za-z])(.*)$

Samples









Add

Add prefix Based on name cont.

Using: 8259438 some text

Tim Contribution – cont.

Take the first letter (upper or lowercase) and prefix it to the front: cont.

Match: ^(.*?)([A-Za-z])(.*)$

Where:

\1 = 8259438 <space> \2 = s \3 = ome text

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = 8
3. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 - BOL
5. [A-Za-z]	Match against a class consisting of an uppercase or lowercase letter	Capture Group 2 = s Changes Capture Group 1 = 8259438 <space>
6. .*	Match against remaining string	Capture Group 3 = ome text
7. $	Must be at EOL (End of Line)	

Add

Add prefix Based on name cont.

Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front:

Method #1

Match: ^(([A-Za-z])(?=[A-Za-z])|(.+?)([A-Za-z])(?=[A-Za-z]))(.*)
Replace: \2\4_\3\4\2\5

Name	New Name
ABC_DEF_GHI-123-45	A_ABC_DEF_GHI-123-45
Begin_RAIL_End	B_Begin_RAIL_End
TAIL_end	T_TAIL_end
Begin_SAIL	B_Begin_SAIL
8259438 some text	s_8259438 some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	B_Bahá'í Muñoz
My File v.1.2.345	M_My File v.1.2.345

Samples:

```
·····························Start·Length·····
Match 1: Begin_RAIL_End       0      14
Group 1: B                    0       1
Group 2: B                    0       1
Group 3 did not participate in the match
Group 4 did not participate in the match
Group 5: egin_RAIL_End        1      13
```

```
·····················Start·Length·········
Match 1: Begin_SAIL       0      10
Group 1: B                0       1
Group 2: B                0       1
Group 3 did not participate in the match
Group 4 did not participate in the match
Group 5: egin_SAIL        1       9
```

```
·····························Start·Length··
Match 1: 8259438 some text       0      17
Group 1: 8259438 s               0       9
Group 2 did not participate in the match
Group 3: 8259438                 0       8
Group 4: s                       8       1
Group 5: ome text                9       8
```

Add

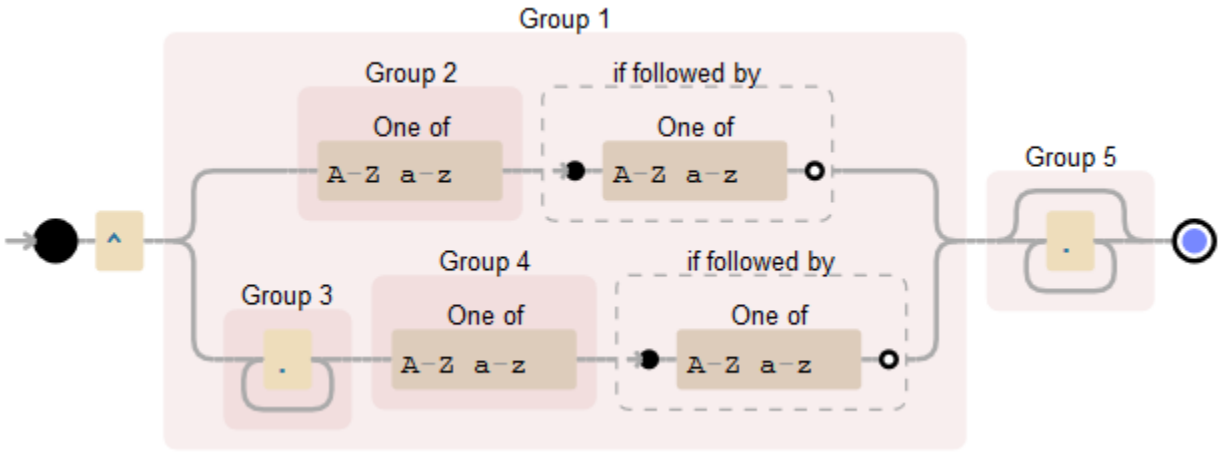
Add prefix Based on name cont.

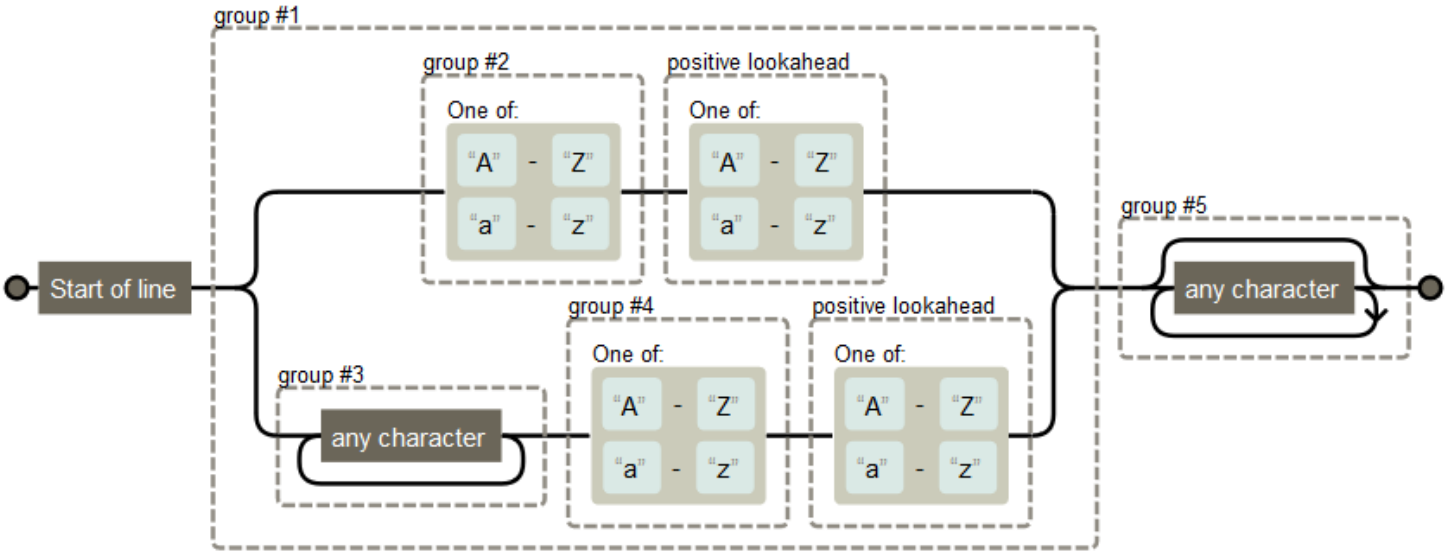
Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #1 cont.

Match: ^(([A-Za-z])(?=[A-Za-z])|(.+?)([A-Za-z])(?=[A-Za-z]))(.*)





Add

Add prefix Based on name cont.

Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #1 cont.

Here I demonstrate where just modifying the Class slightly will change the entire purpose of the RegEx. In a previous example, I took the first uppercase letter matched of an all uppercase keyword. This was the original intent of the heading. It used a class consisting of all uppercase letters. By changing the class to include lowercase letters as well, it changes the meaning to, taking the first initial of the first word matched.

Here is the comparison spaced out for clarification-

Current Match: ^(([A-Za-z]) (?=[A-Za-z]) | (.+?) ([A-Za-z]) (?=[A-Za-z])) (.*)

Keyword Match: ^(([A-Z]) (?=[A-Z]) | (.+?) ([A-Z]) (?=[A-Z])) (.*)

Not too much difference. The diagrams and analysis won't be that much different either.

Add

Add prefix Based on name cont.

Using: Begin_SAIL

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

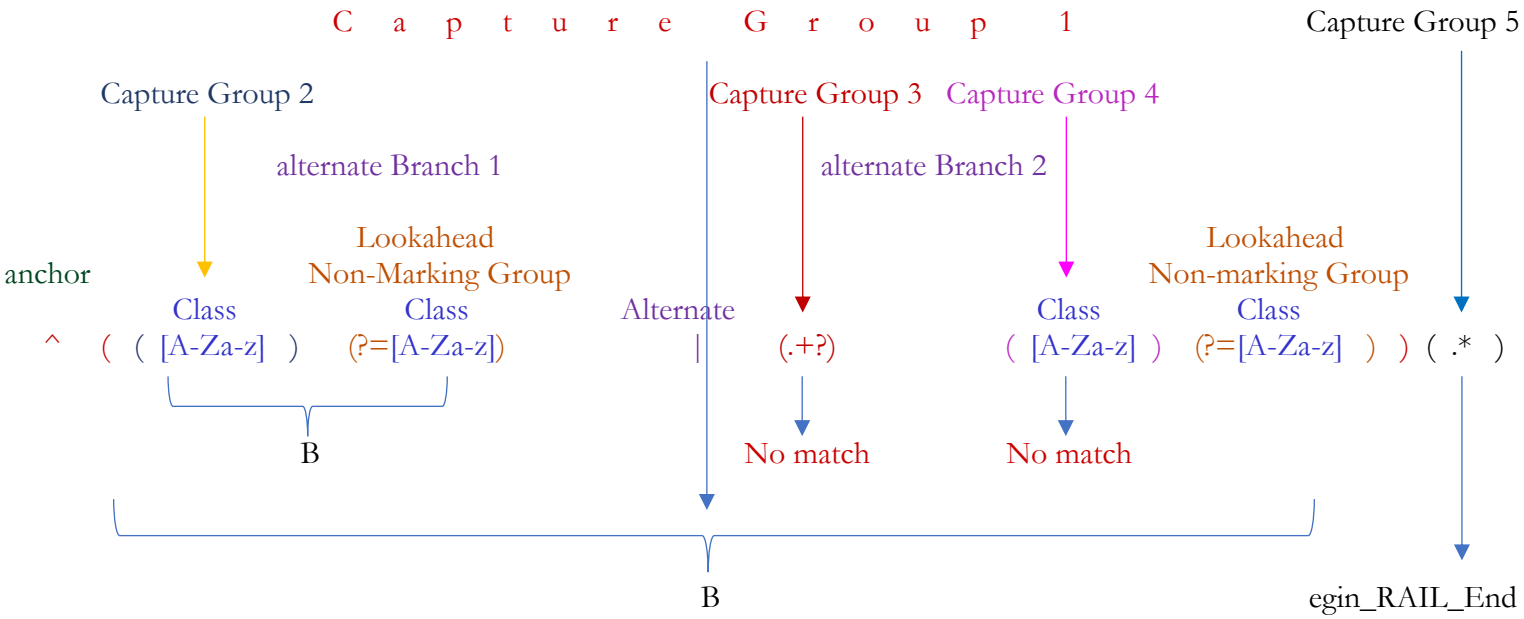
Method #1 cont.

Match: ^(([A-Za-z])(?=[A-Za-z])|(.+?)([A-Za-z])(?=[A-Za-z]))(.*)

Where:

\1 = B \2 = B \3 = No value \4 = No value \5 = egin_RAIL_End

Breaks down:



This uses an Alternate for Capture Group 1, 2, 3 & 4

This OR This

([A-Za-z])(?=[A-Za-z]) | (.+?)([A-Za-z])(?=[A-Za-z])

Tim Notes:

1. The alternates are used to determine the location of the first word in the string.
2. The first alternate tests for an uppercase or lowercase letter followed by a second uppercase or lowercase letter. This first alternate is to test if the word is at the beginning of the string.
3. The second alternate tests for the first word in other parts of the string.
4. Capture Groups 3 and 4 have No Value because that branch did not match.

Add

Add prefix Based on name cont.

Using: 8259438 some text

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

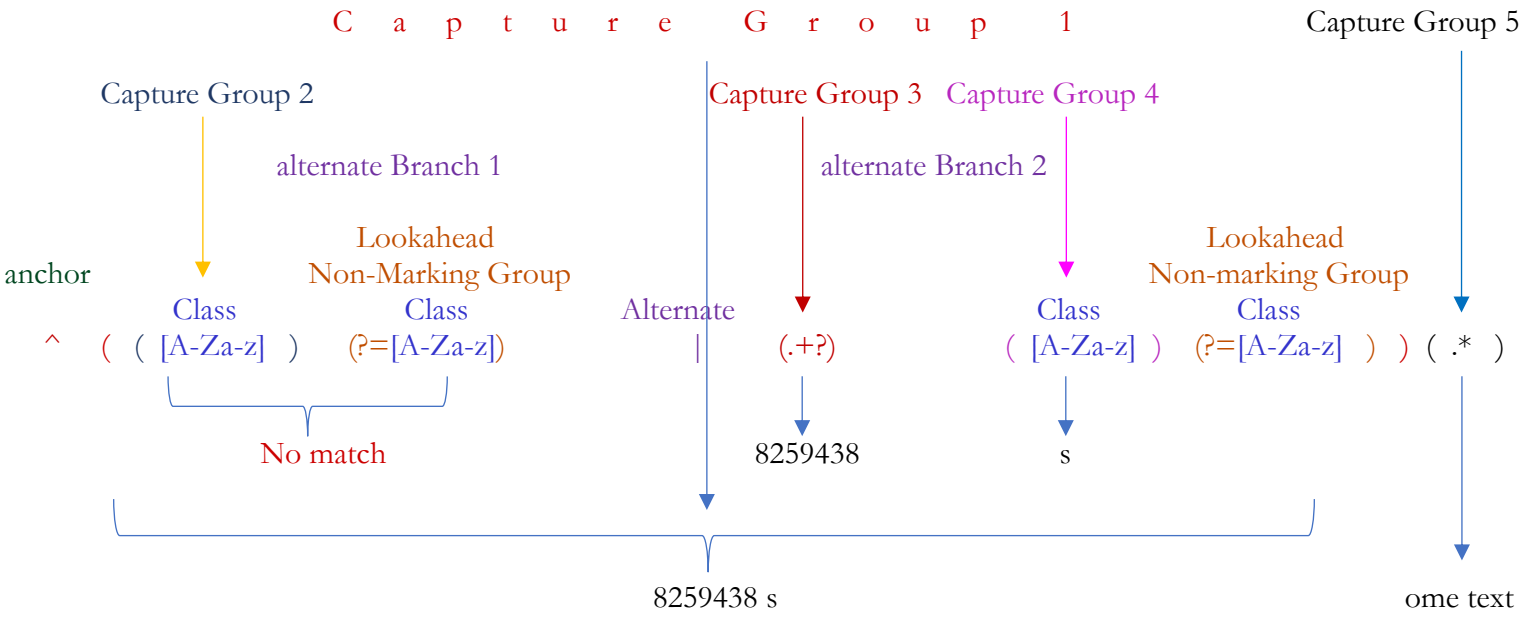
Method #1 cont.

Match: ^(([A-Za-z])(?=[A-Za-z])|(.+?)([A-Za-z])(?=[A-Za-z]))(.*)

Where:

\1 = 8259438 s \2 = No value \3 = 8259438 <space> \4 = s \5 = ome text

Breaks down:



Tim Notes:

1. The alternates are used to determine the location of the first word in the string.
2. The first alternate tests for an uppercase or lowercase letter followed by a second uppercase or lowercase letter. This first alternate is to test if the word is at the beginning of the string.
3. The second alternate tests for the first word in other parts of the string.
4. This is to show an example of where the first word is not matched at the beginning of the string.
5. An initial single letter is not considered a word in this example – that is why each alternate tests for two letters. Each one testing for an actual character, which if exists is captured, and looking ahead to see if there is a subsequent character, which if exists the value is not retained, by the Non-Marking Group
6. Capture Group 2 has No Value because that branch did not match.

Add

Add prefix Based on name cont.

Using: 8259438 some text

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #1: cont.

Match: ^(([A-Za-z])(?=[A-Za-z])|(.+?)([A-Za-z])(?=[A-Za-z]))(.*)

Where:
\1 = 8259438 s \2 = No value \3 = 8259438 <space> \4 = s \5 = ome text

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. ([A-Za-z])(?=[A-Za-z])|(.+?)([A-Za-z])(?=[A-Za-z])		
	Alternate (see Previous Page)	
	a. [A-Za-z] Match against a Class consisting of an uppercase or lowercase letter = '8' so False	Capture Group 1 = No Value Capture Group 2 = No Value
	b.(?=[A-Za-z]) Lookahead. Look to the right. If next character is a single uppercase or lowercase letter then expression = True and evaluation continues. = '2' so False No Match	Capture Group 1 = No Value Capture Group 2 = No Value
	or	
	a. .. Match against any character	Capture Group 3, 1 = 8
	b. + Make it Greedy	Capture Group 3, 1 = <entire string>
	c. ? But not too Greedy	Capture Group 3, 1 = 8 Capture Group 2 = Unchanged
	d. [A-Za-z] Match against a Class consisting of an uppercase or lowercase letter	Capture Group 4 = s Capture Group 3 = 88259438 <space> Capture Group 1 = 88259438 s
	e.(?=[A-Za-z]) Lookahead. Look to the right. If next character is a single uppercase letter then expression = True and evaluation continues. = 'o' so True	All Capture Groups = Unchanged
3. .*	Match against remaining string	Capture Group 5 = ome text

Add

Add prefix Based on name cont.

Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #2

Match: ^(.*?)([A-Za-z])(?=[A-Za-z])(.*)
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	A_ABC_DEF_GHI-123-45
Begin_RAIL_End	B_Begin_RAIL_End
TAIL_end	T_TAIL_end
Begin_SAIL	B_Begin_SAIL
8259438 some text	s_8259438 some text
ABC-123-45	A_ABC-123-45
Bahá'í Muñoz	B_Bahá'í Muñoz
My File v.1.2.345	M_My File v.1.2.345

This method is similar to the, 'Take the first letter (upper or lowercase) and prefix it to the front'

3rd Method Match: ^ (.*?) ([A-Za-z]) (?=[A-Za-z]) (.*)

First Letter Method Match: ^ (.*?) ([A-Za-z]) (.*) $

The first method was matching against the first letter, therefore the match was for any single letter, whereas this method is looking to match against two letters and again I use the Lookahead to match against a second subsistent letter.

Add

Add prefix Based on name cont.

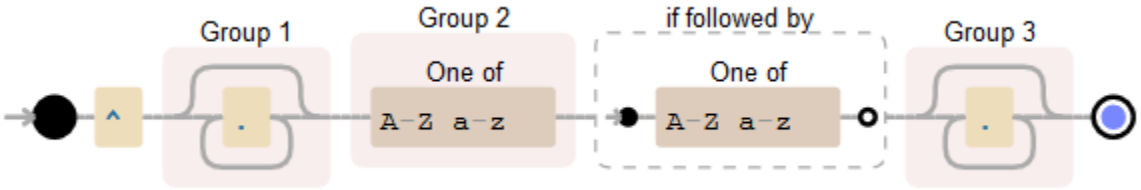
Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

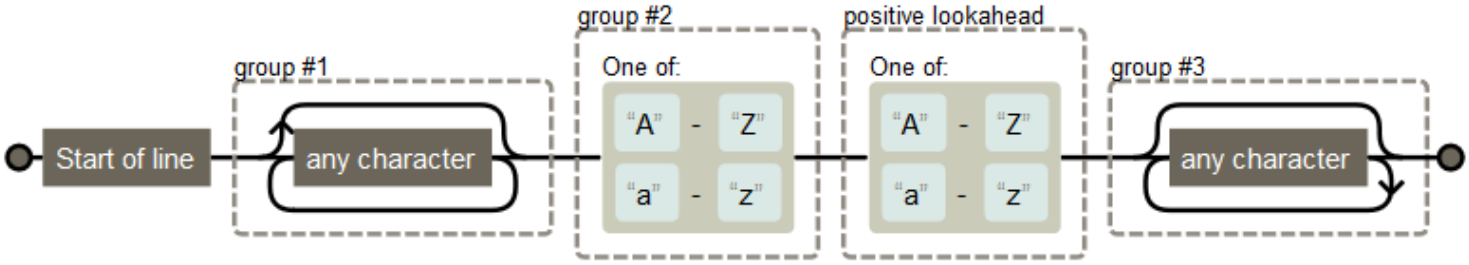
Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #2 cont.

Match: ^(.*?)([A-Za-z])(?=[A-Za-z])(.*)





Samples

```
                              Start  Length
Match 1: ABC_DEF_GHI-123-45       0      18
Group 1:                          0       0
Group 2: A                        0       1
Group 3: BC_DEF_GHI-123-45        1      17
```

```
                              Start  Length
Match 1: Begin_SAIL               0      10
Group 1:                          0       0
Group 2: B                        0       1
Group 3: egin_SAIL                1       9
```

```
                              Start  Length
Match 1: My File v.1.2.345        0      17
Group 1:                          0       0
Group 2: M                        0       1
Group 3: y File v.1.2.345         1      16
```

```
Match 1: 8259438 some text
Group 1: 8259438
Group 2: s
Group 3: ome text
```

Add

Add prefix Based on name cont.

Using: My File v.1.2.345

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #2 cont.

Match: ^(.*?)([A-Za-z])(?=[A-Za-z])(.*)

Where:

\1 = null \2 = M \3 = y File v.1.2.345

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = M
3. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 – BOL

Tim Notes:

1. Because of the <dot> in the string, the .345 is treated as the extension, which in all likelihood it probably is. So although in BRU, the value of Capture Group 1 is BOL, the extension is displayed in the New Name as '.345'. This can be ignored as the real current position of the RegEx Engine is at BOL.

5. [A-Za-z]	Match against a class consisting of an uppercase or lowercase letter	Capture Group 2 = M Changes Capture Group 1 = null

Tim Notes:

1. Because of the match of the 'M' as the first character that satisfies the match, the current position of Capture Group 1 remains at BOL which has a null value. The value did not really change but remains at null. Typically, if Capture Group 2 had captured any value after the first character, this would have changed Capture Group 1's value accordingly as it has done in numerous examples throughout this book.

For Example:

Capture Group 1 = BOL
Capture Group 2 = File
Changes Capture Group 1 = My <space>

Add

Add prefix Based on name cont.

Using: My File v.1.2.345

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #2 cont.

Match: ^(.*?)([A-Za-z])(?=[A-Za-z])(.*)

Where:

\1 = null \2 = M \3 = y File v.1.2.345

Tim Analysis: cont.

6.(?=[A-Za-z]) Lookahead.
Look to the right. If next character is a
single uppercase or lowercase letter then
expression = True and evaluation continues.
= 'y ' so = True
6. .* Match against remaining string Capture Group 3 = y File v.1.2.345
7. $ Must be at EOL (End of Line)

Add

Add prefix Based on name cont.

Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

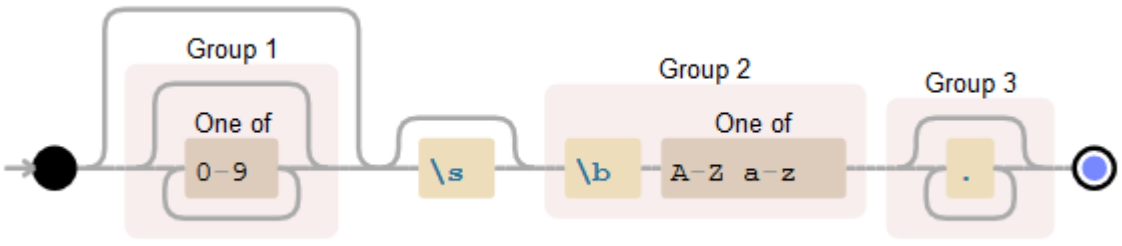
Tim Contribution – cont.

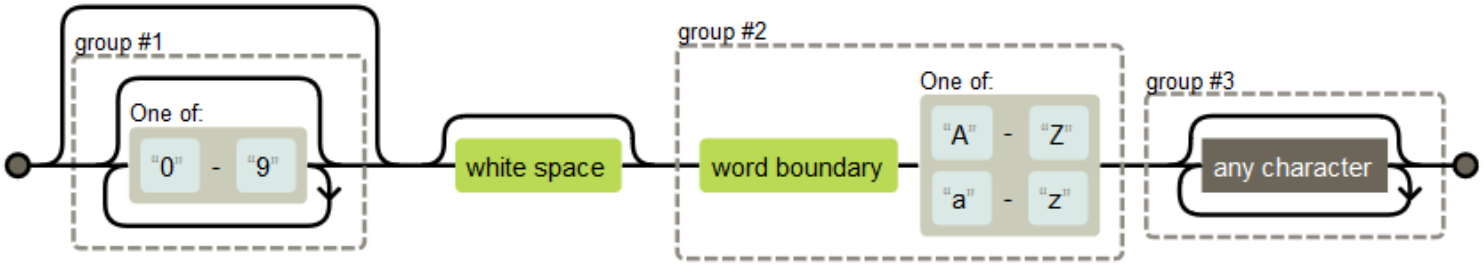
Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #3

Match: ([0-9]*)?\s?(\b[A-Za-z])(.*)
Replace: \2_\1 \2\3

Name	New Name
ABC_DEF_GHI-123-45	A_ ABC_DEF_GHI-123-45
Begin_RAIL_End	B_ Begin_RAIL_End
TAIL_end	T_ TAIL_end
Begin_SAIL	B_ Begin_SAIL
8259438 some text	s_8259438 some text
ABC-123-45	A_ ABC-123-45
Bahá'í Muñoz	B_ Bahá'í Muñoz
My File v.1.2.345	M_ My File v.1.2.345





Add

Add prefix Based on name cont.

Begin_SAIL	to	B_Begin_SAIL
TAIL_end	to	T_TAIL_end
Begin_RAIL_End	to	B_Begin_RAIL_End
Bahá'í Muñoz	to	B_Bahá'í Muñoz
ABC-123-45	to	A_ABC-123-45
ABC_DEF_GHI-123-45	to	A_ABC_DEF_GHI-123-45
8259438 some text	to	s_8259438 some text
My File v.1.2.345	to	M_My File v.1.2.345

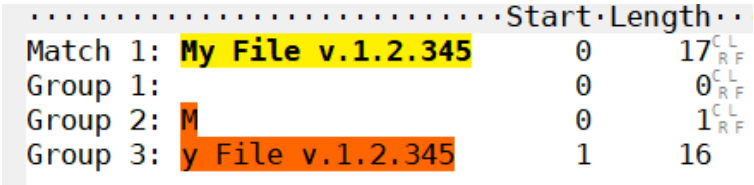
Tim Contribution – cont.

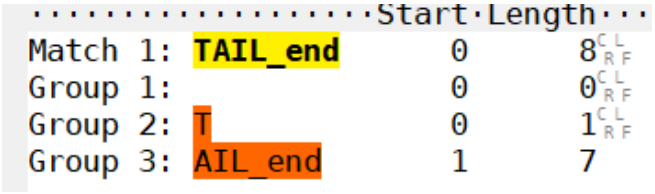
Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

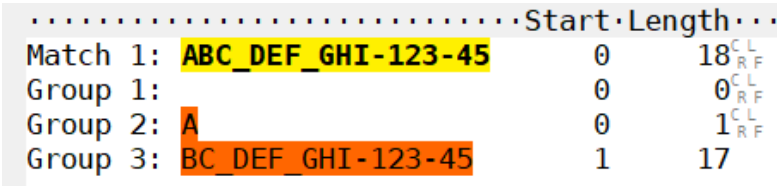
Method #3 cont.

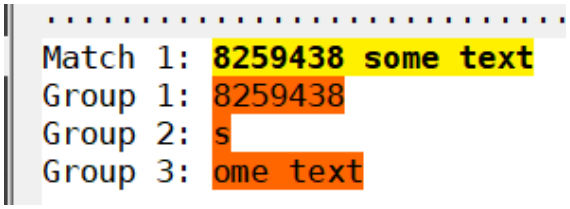
Match: ([0-9]*)?\s?(\b[A-Za-z])(.*)

Samples









Add

Add prefix Based on name cont.

Using: TAIL_end

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #3 cont.

Match: ([0-9]*)?\s?(\b[A-Za-z])(.*)

Where:

\1 = null \2 = T \3 = AIL_End

Breaks down:

Capture Group 1			Capture Group 2		Capture Group 3
Class		<space>	Word Boundary	Class	
([0-9] *) ?		\s ?	(\b	[A-Za-z])	(.*)

Tim Analysis:

1. [0-9]	Match against a class consisting of a numeric digit '0' through '9'	Capture Group 1 = No Value
2. *	Make it Greedy	Capture Group 1 = Unchanged
3. ?	Make the expression Optional Optional for strings that do not have numeric digits at the beginning of the string.	Capture Group 1 = Invalid
4. \s	Match against a <space> character	Capture Group 1 = No Value
5. ?	Make it optional Optional for strings that do not have a <space> character preceding the first word of the string.	Capture Group 1 = Invalid
6. \b	Word Boundary. Matches against a letter (Upper or lowercase), a numeric digit, or an underscore character. = BOL = False	Capture Group 2 = No Value Capture Group 1 = No Value
7. [A-Za-z]	[A-Za-z] Match against an upper or lowercase letter.	Capture Group 2 = T Capture Group 1 = null
8. .*	Match against remaining string	Capture Group 3 = AIL_End

Add

Add prefix Based on name cont.

Using: TAIL_end

Tim Contribution – cont.

Take the initial of the first word (upper or lowercase) and prefix it to the front: cont.

Method #3 cont.

Match: ([0-9]*)?\s?(\b[A-Za-z])(.*)

Tim Analysis: cont.

Tim Notes:

1. Much of the time Capture Group 1 changes from No Value, meaning No Match has taken place, to Invalid, meaning that the value never happened because the expression was made optional (in two places using the ' ? '). The value finally becomes null because of a Zero match Occurrence when Capture Group 2 matches against the first character of the string, the 'T'. Capture Group 1's position is at the BOL which is a null value.

2. In step 6, it tests for a 'Word', but the current position is BOL, so it is False, but the RegEx Engine moves forward and satisfies the match, the 'T', with the next expression.

3. To change it to the First word instead of first initial change it from:

   Match: ([0-9]*)?\s?(\b[A-Za-z])(.*)

   to:

   Match: ([0-9]*)?\s?(\b[A-Za-z]*)(.*)

   by just adding the Greedy Quantifier. (Capture Group 2 is now the First Word)

Name	New Name
ABC_DEF_GHI-123-45	ABC
Begin_RAIL_End	Begin
TAIL_end	TAIL
Begin_SAIL	Begin
8259438 some text	some
ABC-123-45	ABC
Bahá'í Muñoz	Bah
My File v.1.2.345	My.345

Won't work with the High ASCII of Bahá'í Muñoz.

Add

Add prefix Based on name cont.

Begin_SAIL	to	L_Begin_SAIL
TAIL_end	to	d_TAIL_end
Begin_RAIL_End	to	d_Begin_RAIL_End
Bahá'í Muñoz	to	z_Bahá'í Muñoz
ABC-123-45	to	5_ABC-123-45
ABC_DEF_GHI-123-45	to	5_ABC_DEF_GHI-123-45
8259438 some text	to	t_8259438 some text
My File v.1.2.345	to	2_My File v.1.2.345

Tim Contribution – cont.

Take last character in string and prefix it to front:

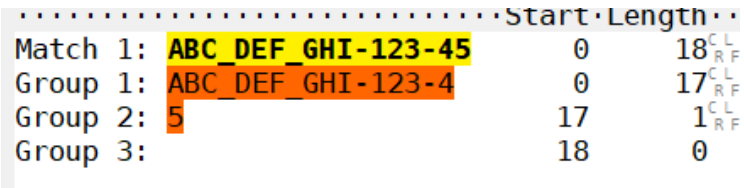
Match: (.*)([^_])(\w*)
Replace: \2_\1\2\3

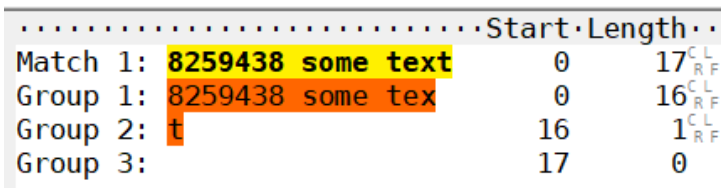
Name	New Name
ABC_DEF_GHI-123-45	5_ABC_DEF_GHI-123-45
Begin_RAIL_End	d_Begin_RAIL_End
TAIL_end	d_TAIL_end
Begin_SAIL	L_Begin_SAIL
8259438 some text	t_8259438 some text
ABC-123-45	5_ABC-123-45
Bahá'í Muñoz	z_Bahá'í Muñoz
My File v.1.2.345	2_My File v.1.2.345

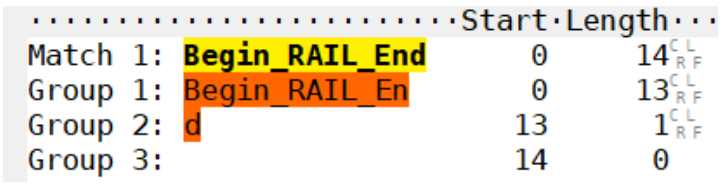
Tim Notes:

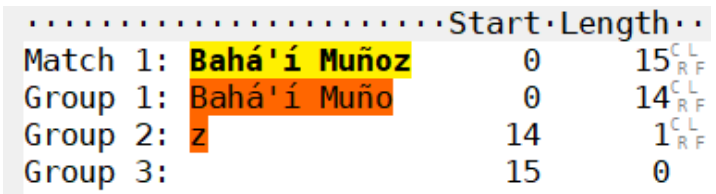
1. In the sample, 2_My File v1.2.345, the '2' is seen as the last character because BRU is interpreting the '.345' as the filename extension.

Samples









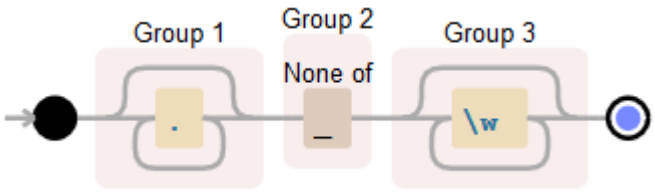
Add

Add prefix Based on name cont.

Tim Contribution – cont.

Take last character in string and prefix it to front: cont.

Match: (.*)([^_])(\w*)



Tim Notes:

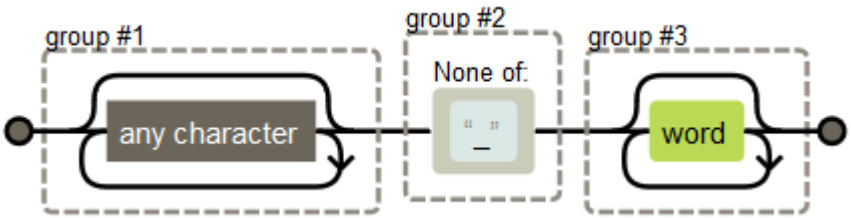
1. From these samples, it is clear that the last Capture Group, Capture Group 3, is not required to satisfy all of the possible sample matches. I could just as easily have used:

   Match: (.*)([^_])
   Replace: \2_\1\2

   ..and still have satisfied all of the samples. This is universal enough to match against an expanded Sample group:

Name	New Name
ABC_DEF_GHI-123-45	5_ABC_DEF_GHI-123-45
Begin_RAIL_End	d_Begin_RAIL_End
TAIL_end	d_TAIL_end
Begin_SAIL	L_Begin_SAIL
8259438 some text	t_8259438 some text
ABC-123-45	5_ABC-123-45
Bahá'í Muñoz	z_Bahá'í Muñoz
My File v.1.2.345	2_My File v.1.2.345
InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.txt	t_InThe4HrsISpentResearchingThisICouldHaveTypedThemOut.txt
Longer_File_name (2016_01_30 07_16_32 UTC)	)_Longer_File_name (2016_01_30 07_16_32 UTC)
my.File2.v1.2.345	2_my.File2.v1.2.345
RAIL	L_RAIL
SAIL	L_SAIL
TAIL	L_TAIL
nc_017733	3_nc_017733
ThisIsReally&Annoying$&.whatever.pdf	r_ThisIsReally&Annoying$&.whatever.pdf
xxx xxx-xxx.txt	x_xxx xxx-xxx.txt
xxx_xxxx-xxxxx-xxx	x_xxx_xxxx-xxxxx-xxx
1	1_1

2. The point is, that sometimes through Analysis, you can improve upon or weed out problems that you were not even aware existed. For now I will leave it as is, because that is the original RegEx I used, but just be aware that it could be improved.



Add

Add prefix Based on name cont.

Using: Begin_RAIL_End

Tim Contribution – cont.

Take last character in string and prefix it to front: cont.

Match: (.*)([^_])(\w*)

Where:

\1 = Begin_Rail_En \2 = d \3 = null

Tim Analysis:

1..	Match against any character	Capture Group 1 = B
2. *	Make it Greedy Position = EOL	Capture Group 1 = <entire string>
3. [^_]	Match against a Negated Class consisting of underscore. Match against anything but a underscore. Backtracks. matches the 'd' of 'End'.	Capture Group 2 = d Changes Capture Group 1 = Begin_Rail_En
4. \w	Word Metacharacter. Match against any single letter, number or underscore Current position = EOL	Capture Group 3 = d Changes Capture Group 2 = n Changes Capture Group 1 = Begin_Rail_E
5. *	Make it Greedy Already at EOL. Results in a Zero Match Occurrence.	Capture Group 3 = null Changes Capture Group 2= d Changes Capture Group 1 = Begin_Rail_En

Add

Add prefix Based on name cont.

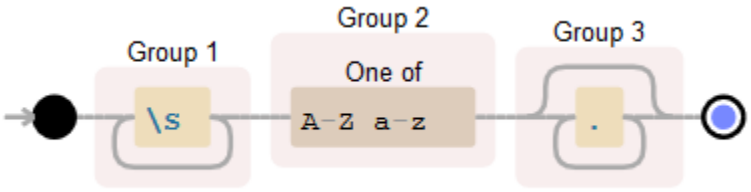
Begin_SAIL	to	e_Begin_SAIL
TAIL_end	to	A_TAIL_end
Begin_RAIL_End	to	e_Begin_RAIL_End
Bahá'í Muñoz	to	a_Bahá'í Muñoz
ABC-123-45	to	B_ABC-123-45
ABC_DEF_GHI-123-45	to	B_ABC_DEF_GHI-123-45
8259438 some text	to	o_8259438 some text
My File v.1.2.345	to	y_My File v.1.2.345

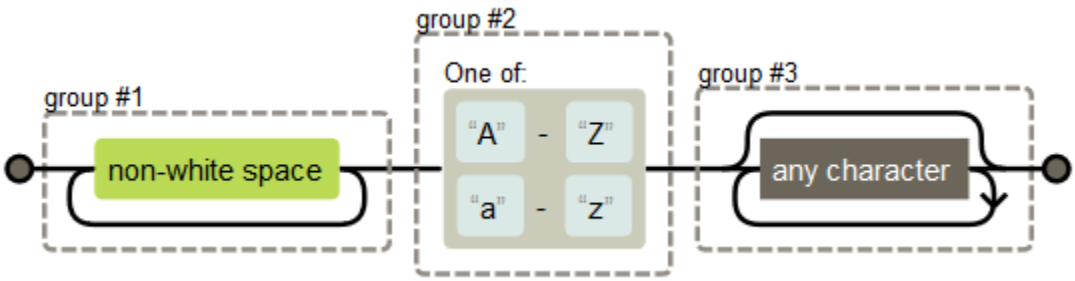
Tim Contribution – cont.

Take second character of string and prefix it to front:

Match: (\S+?)([A-Za-z])(.*)
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	B_ABC_DEF_GHI-123-45
Begin_RAIL_End	e_Begin_RAIL_End
TAIL_end	A_TAIL_end
Begin_SAIL	e_Begin_SAIL
8259438 some text	o_some text
ABC-123-45	B_ABC-123-45
Bahá'í Muñoz	a_Bahá'í Muñoz
My File v.1.2.345	y_My File v.1.2.345





Add

Add prefix Based on name cont.

Begin_SAIL	to	e_Begin_SAIL
TAIL_end	to	A_TAIL_end
Begin_RAIL_End	to	e_Begin_RAIL_End
Bahá'í Muñoz	to	a_Bahá'í Muñoz
ABC-123-45	to	B_ABC-123-45
ABC_DEF_GHI-123-45	to	B_ABC_DEF_GHI-123-45
8259438 some text	to	o_8259438 some text
My File v.1.2.345	to	y_My File v.1.2.345

Tim Contribution – cont.

Take second character of string and prefix it to front cont.

Match: (\S+?)([A-Za-z])(.*)

Sample:

```
.................................................
Match 1: Bahá'í Muñoz 1233 XC^&^^123 hello
Group 1: B
Group 2: a
Group 3: há'í Muñoz 1233 XC^&^^123 hello
```

Sample:

```
.....................
Match 1: some text
Group 1: s
Group 2: o
Group 3: me text
```

Sample:

```
.............................
Match 1: Begin_RAIL_End
Group 1: B
Group 2: e
Group 3: gin_RAIL_End
```

Breaks down:

Capture Group 1	Capture Group 2	Capture Group 3
Non-Whitespace	Class	
(\S+?)	([A-Za-z])	(.*)

Add

Add prefix Based on name cont.

Using: 8259438 some text

Tim Contribution – cont.

Take second character of string and prefix it to front cont.

Match: (\S+?)([A-Za-z])(.*)

Where:

\1 = s \2 = o \3 = me text

Tim Analysis:

1. \S	Matches any Non-Whitespace Character	Capture Group 1 = 8
2. +	Make it Greedy	Capture Group 1 = 8259438
3. ?	But not too Greedy	Capture Group 1 = 8
4. [A-Za-z]	[A-Za-z] Match against an upper or lowercase letter	Capture Group 2 = o Changes Capture Group 1 = s

Tim Notes:

1. Why didn't Capture Group 2 match against the 's'?
   Because it continues to match each character against the \S+ *and* the [A-Za-z]. It doesn't match against the numeric digits so it moves forward. When it hits the <space> after 8259438, this doesn't match against the \S, so again it moves forward and matches the \S to the 's' changing Capture Group 1 to the value of 's' and moves forward to capture the value of 'o' for Capture Group 2.

5. .*	Match against remaining string	Capture Group 3 = me text

Add

Add prefix Based on name cont.

Using: 8259438 some text

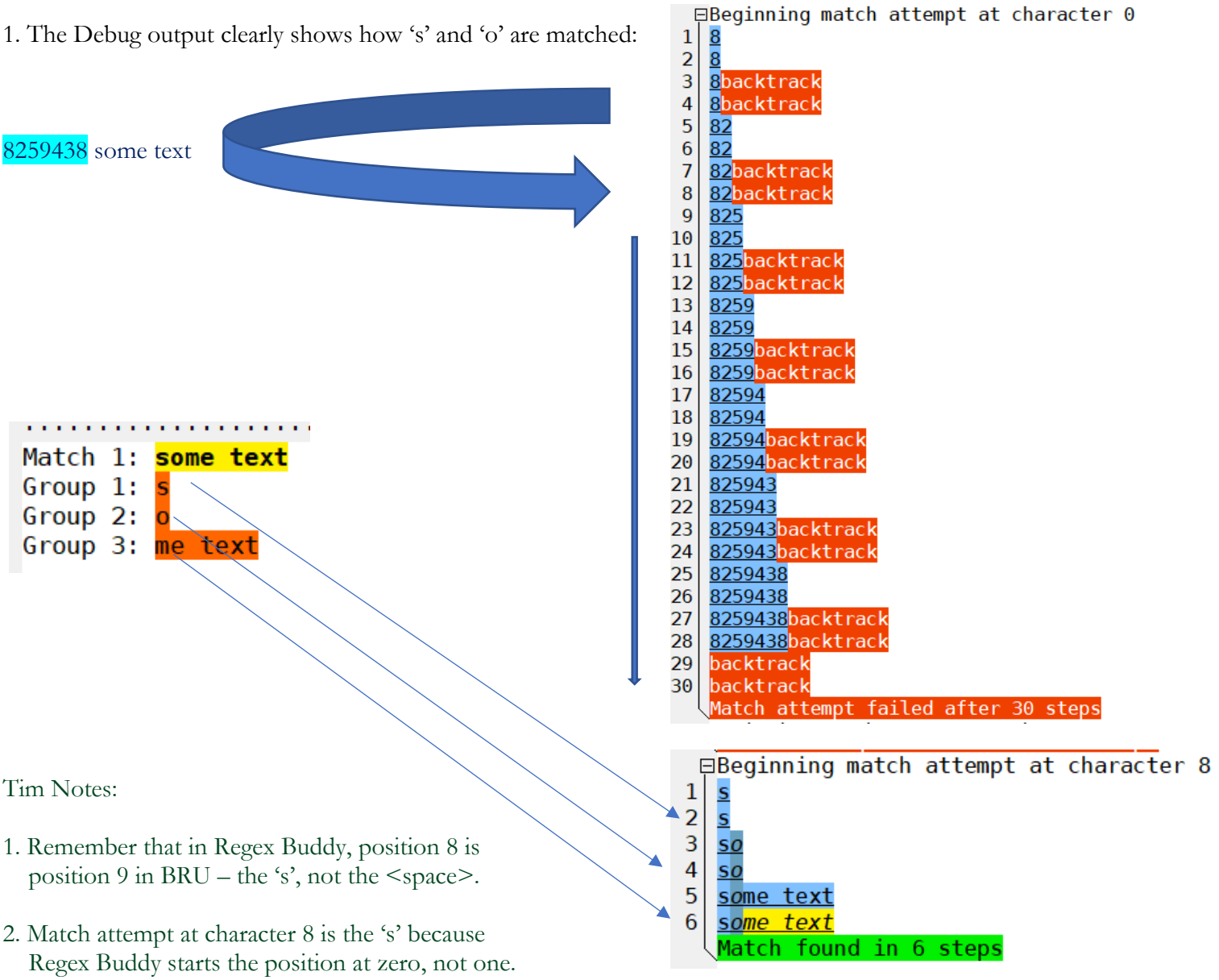
Tim Contribution – cont.

Take second character of string and prefix it to front cont.

Match: (\S+?)([A-Za-z])(.*)

Tim Notes:

1. The Debug output clearly shows how ‘s’ and ‘o’ are matched:



Tim Notes:

1. Remember that in Regex Buddy, position 8 is position 9 in BRU – the ‘s’, not the <space>.

2. Match attempt at character 8 is the ‘s’ because Regex Buddy starts the position at zero, not one.

Add

Add prefix Based on name cont.

Using: Begin_RAIL_End

Tim Contribution – cont.

Take second character of string and prefix it to front cont.

Match: (\S+?)([A-Za-z])(.*)

Where:

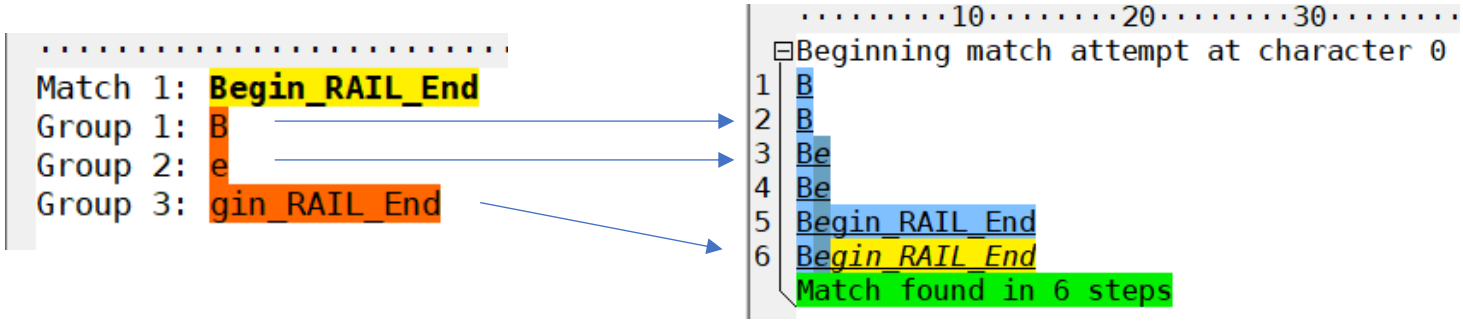
\1 = B \2 = e \3 = gin_RAIL_End

Tim Analysis:

1. \S	Matches any Non-Whitespace Character	Capture Group 1 = B
2. +	Make it Greedy	Capture Group 1 = Begin
3. ?	But not too Greedy	Capture Group 1 = B
4. [A-Za-z]	[A-Za-z] Match against an upper or lowercase letter	Capture Group 2 = e
5. .*	Match against remaining string	Capture Group 3 = gin_RAIL_End

Tim Notes:

1. Using this sample, the analysis is more straight forward. the \S immediately matches against the 'B' and the [A-Za-z] immediately matches against the 'e'



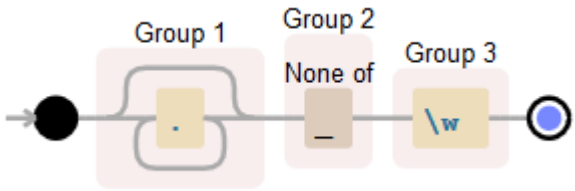
Add

Add prefix Based on name cont.

Begin_SAIL	to	I_Begin_SAIL
TAIL_end	to	n_TAIL_end
Begin_RAIL_End	to	n_Begin_RAIL_End
Bahá'í Muñoz	to	o_Bahá'í Muñoz
ABC-123-45	to	4_ABC-123-45
ABC_DEF_GHI-123-45	to	4_ABC_DEF_GHI-123-45
8259438 some text	to	x_8259438 some text
My File v.1.2.345	to	._My File v.1.2.345

Tim Contribution – cont.

Take second character from end of string and prefix to front:



Method #1

Match: (.*)([^-_])(\w)
Replace: \2_\1\2\3

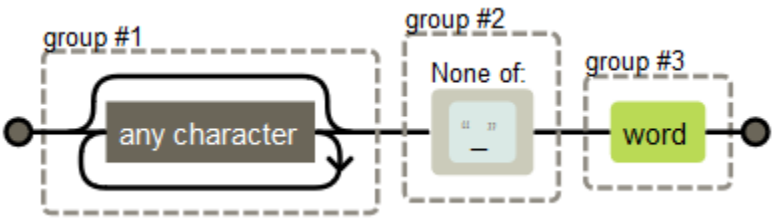
Name	New Name
ABC_DEF_GHI-123-45	4_ABC_DEF_GHI-123-45
Begin_RAIL_End	n_Begin_RAIL_End
TAIL_end	n_TAIL_end
Begin_SAIL	I_Begin_SAIL
8259438 some text	x_8259438 some text
ABC-123-45	4_ABC-123-45
Bahá'í Muñoz	o_Bahá'í Muñoz
My File v.1.2.345	._My File v.1.2.345

Sample:

```
Match 1: 8259438 some text
Group 1: 8259438 some te
Group 2: x
Group 3: t
```

Sample:

```
Match 1: ABC_DEF_GHI-123-45
Group 1: ABC_DEF_GHI-123-
Group 2: 4
Group 3: 5
```



Add

Add prefix Based on name cont.

Using: ABC_DEF_GHI-123-45

Tim Contribution – cont.

Take second character from end of string and prefix to front: cont.

Method #1 cont.

Match: (.*)([^-_])(\w)

Where:

\1 = ABC_DEF_GHI-123- \2 = 4 \3 = 5

Tim Analysis:

1..	Match against any character	Capture Group 1 = A
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [^-_]	Match against a Negated Class consisting of a. <hyphen> b. underscore Backtracks to match the '5' Current position = EOL Match against anything but a <hyphen> or underscore.	Capture Group 2 = 5 Changes Capture Group 1 = ABC_DEF_GHI-123-4
4. \w	Word Metacharacter. Match against any single letter, number or underscore. Backtracks to match against the '5'	Capture Group 3 = 5 Changes Capture Group 2 = 4 Changes Capture Group 1 = ABC_DEF_GHI-123-

Add

Add prefix Based on name cont.

Begin_SAIL	to	I_Begin_SAIL
TAIL_end	to	n_TAIL_end
Begin_RAIL_End	to	n_Begin_RAIL_End
Bahá'í Muñoz	to	o_Bahá'í Muñoz
ABC-123-45	to	4_ABC-123-45
ABC_DEF_GHI-123-45	to	4_ABC_DEF_GHI-123-45
8259438 some text	to	x_8259438 some text
My File v.1.2.345	to	._My File v.1.2.345

Tim Contribution – cont.

Take second character from end of string and prefix to front: cont.

Method #2

Match: (.*)([^_])([^_])(\S)
Replace: \3_\1\2\3\4

Name	New Name
ABC_DEF_GHI-123-45	4_ABC_DEF_GHI-123-45
Begin_RAIL_End	n_Begin_RAIL_End
TAIL_end	n_TAIL_end
Begin_SAIL	I_Begin_SAIL
8259438 some text	x_8259438 some text
ABC-123-45	4_ABC-123-45
Bahá'í Muñoz	o_Bahá'í Muñoz
My File v.1.2.345	._My File v.1.2.345

Sample:

```
..............................
Match 1: ABC_DEF_GHI-123-45
Group 1: ABC_DEF_GHI-123
Group 2: -
Group 3: 4
Group 4: 5
```

Sample:

```
......................
Match 1: Bahá'í Muñoz
Group 1: Bahá'í Mu
Group 2: ñ
Group 3: o
Group 4: z
```

Sample:

```
....................
Match 1: Begin_SAIL
Group 1: Begin_S
Group 2: A
Group 3: I
Group 4: L
```

Add

Add prefix Based on name cont.

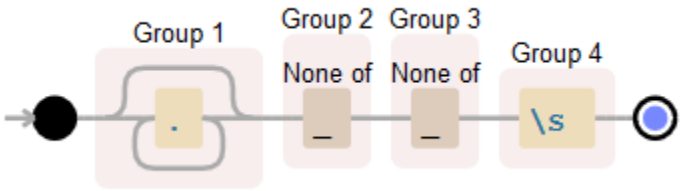
Using: Begin_SAIL

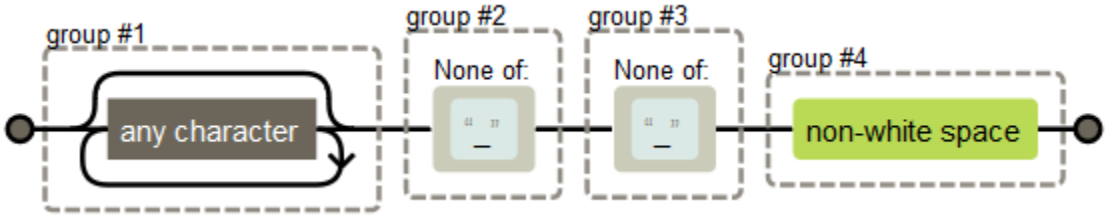
Tim Contribution – cont.

Take second character from end of string and prefix to front: cont.

Method #2 cont.

Match: (.*)([^-_])([^-_])(\S)





Where:

\1 = Begin_S \2 = A \3 = I \4 = L

Tim Analysis:

1..	Match against any character	Capture Group 1 = B
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [^_]	Match against a Negated Class consisting of a <hyphen> Backtracks to match the 'L' Current position = EOL	Capture Group 2 = L Changes Capture Group 1 = Begin_SAI
4. [^_]	Match against a Negated Class consisting of a <hyphen> Backtracks to match the 'L' Current position = EOL Match against anything but a <hyphen> or underscore.	Capture Group 3 = L Changes Capture Group 2 = I Changes Capture Group 1 = Begin_SA
5. \S	Matches any Non-Whitespace Character	Capture Group 4 = L Changes Capture Group 3 = I Changes Capture Group 2 = A Changes Capture Group 1 = Begin_S

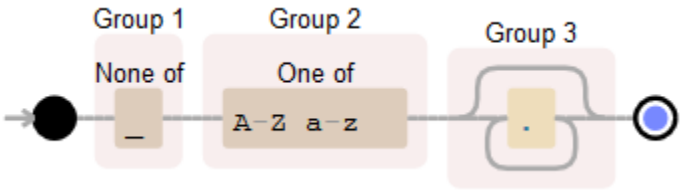
Add

Add prefix Based on name cont.

Begin_SAIL	to	e_Begin_SAIL
TAIL_end	to	A_TAIL_end
Begin_RAIL_End	to	e_Begin_RAIL_End
Bahá'í Muñoz	to	a_Bahá'í Muñoz
ABC-123-45	to	B_ABC-123-45
ABC_DEF_GHI-123-45	to	B_ABC_DEF_GHI-123-45
8259438 some text	to	o_8259438 some text
My File v.1.2.345	to	y_My File v.1.2.345

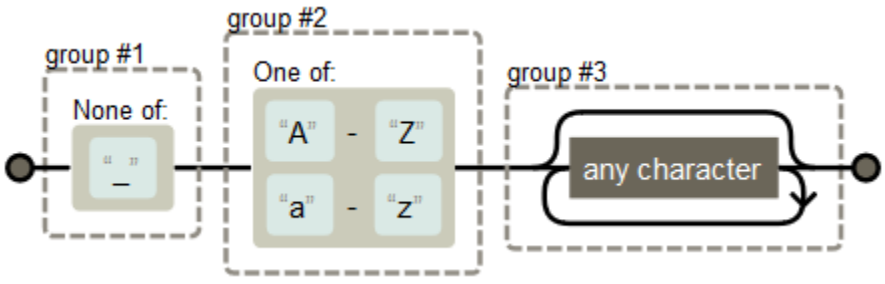
Tim Contribution – cont.

Take second letter at beginning of string and prefix to front:



Match: ([^_])([A-Za-z])(.*)
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	B_ABC_DEF_GHI-123-45
Begin_RAIL_End	e_Begin_RAIL_End
TAIL_end	A_TAIL_end
Begin_SAIL	e_Begin_SAIL
8259438 some text	o_some text
ABC-123-45	B_ABC-123-45
Bahá'í Muñoz	a_Bahá'í Muñoz
My File v.1.2.345	y_My File v.1.2.345



Samples

```
.........................
Match 1: Begin_RAIL_End
Group 1: B
Group 2: e
Group 3: gin_RAIL_End
```

```
.....................
Match 1: ABC-123-45
Group 1: A
Group 2: B
Group 3: C-123-45
```

```
.............................
Match 1: My File v.1.2.345
Group 1: M
Group 2: y
Group 3:  File v.1.2.345
```

Add

Add prefix Based on name cont.

Using: ABC-123-45

Tim Contribution – cont.

Take second letter at beginning of string and prefix to front: cont.

Match: ([^_])([A-Za-z])(.*)

Where:

\1 = A \2 = B \3 = C-123-45

Tim Analysis:

1. [^_]	Match against a Negated Class consisting of a <hyphen> Match against any character but a <hyphen>	Capture Group 1 = A
2. [A-Za-z]	[A-Za-z] Match against an upper or lowercase letter.	Capture Group 2 = B
3. .*	Match against remaining string	Capture Group 3 = C-123-45

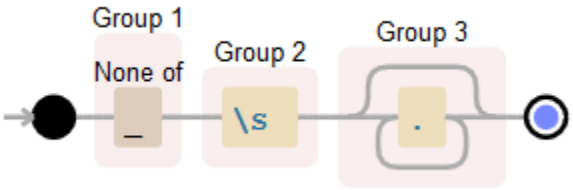
Add

Add prefix Based on name cont.

Begin_SAIL	to	e_Begin_SAIL
TAIL_end	to	A_TAIL_end
Begin_RAIL_End	to	e_Begin_RAIL_End
Bahá'í Muñoz	to	a_Bahá'í Muñoz
ABC-123-45	to	B_ABC-123-45
ABC_DEF_GHI-123-45	to	B_ABC_DEF_GHI-123-45
8259438 some text	to	2_8259438 some text
My File v.1.2.345	to	y_My File v.1.2.345

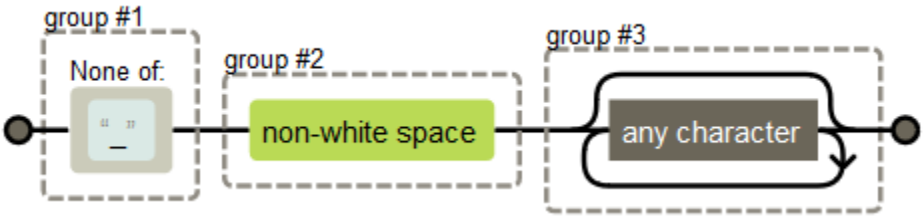
Tim Contribution – cont.

Take second character at beginning of string and prefix to front:



Match: ([^_])(\S)(.*)
Replace: \2_\1\2\3

Name	New Name
ABC_DEF_GHI-123-45	B_ABC_DEF_GHI-123-45
Begin_RAIL_End	e_Begin_RAIL_End
TAIL_end	A_TAIL_end
Begin_SAIL	e_Begin_SAIL
8259438 some text	2_8259438 some text
ABC-123-45	B_ABC-123-45
Bahá'í Muñoz	a_Bahá'í Muñoz
My File v.1.2.345	y_My File v.1.2.345



Samples:

```
.............................
Match 1: ABC_DEF_GHI-123-45
Group 1: A
Group 2: B
Group 3: C_DEF_GHI-123-45
```

```
.............................
Match 1: My File v.1.2.345
Group 1: M
Group 2: y
Group 3:  File v.1.2.345
```

```
.............................
Match 1: 8259438 some text
Group 1: 8
Group 2: 2
Group 3: 59438 some text
```

Add

Add prefix Based on name cont.

Using: 8259438 some text

Tim Contribution – cont.

Take second character at beginning of string and prefix to front: cont.

Match: ([^_])(\S)(.*)

Where:

\1 = 8 \2 = 2 \3 = 59438 some text

Tim Analysis:

1. [^_]	Match against a Negated Class consisting of a <hyphen> Match against any character but a <hyphen>	Capture Group 1 = 8
2. \S	Matches any Non-Whitespace Character	Capture Group 2 = 2
3. .*	Match against remaining string	Capture Group 3 = 59438 some text

Add

Add prefix Based on name cont.

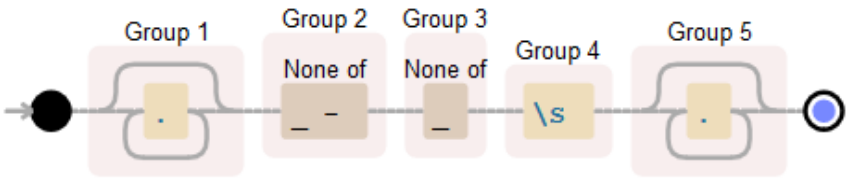
Begin_SAIL	to	e_Begin_SAIL
TAIL_end	to	A_TAIL_end
Begin_RAIL_End	to	e_Begin_RAIL_End
Bahá'í Muñoz	to	a_Bahá'í Muñoz
ABC-123-45	to	B_ABC-123-45
ABC_DEF_GHI-123-45	to	B_ABC_DEF_GHI-123-45
8259438 some text	to	2_8259438 some text
My File v.1.2.345	to	y_My File v.1.2.345

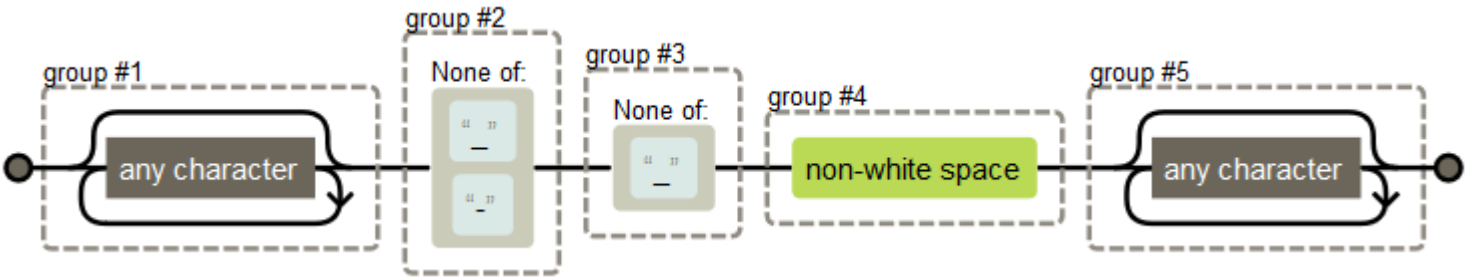
Tim Contribution – cont.

Take third character from end of string and prefix to front:

Match: (.*)([^_-])([^_])(\S)(.*)
Replace: \2_\1\2\3\4\5

Name	New Name
ABC_DEF_GHI-123-45	3_ABC_DEF_GHI-123-45
Begin_RAIL_End	E_Begin_RAIL_End
TAIL_end	e_TAIL_end
Begin_SAIL	A_Begin_SAIL
8259438 some text	e_8259438 some text
ABC-123-45	3_ABC-123-45
Bahá'í Muñoz	ñ_Bahá'í Muñoz
My File v.1.2.345	1_My File v.1.2.345





Add

Add prefix Based on name cont.

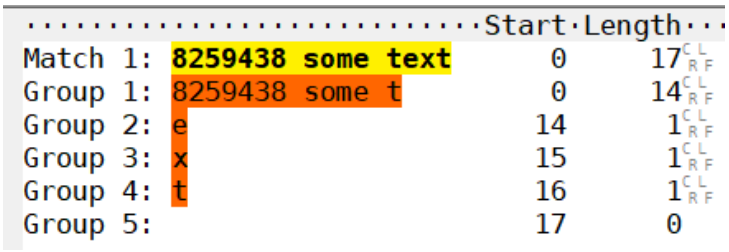
Using: 8259438 some text

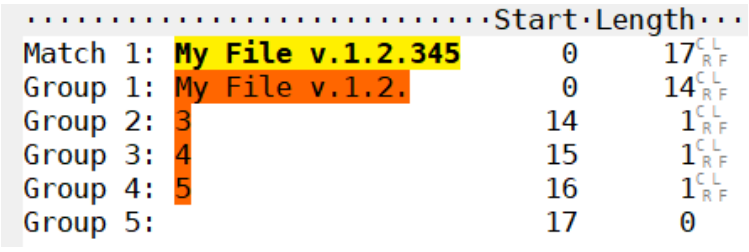
Tim Contribution – cont.

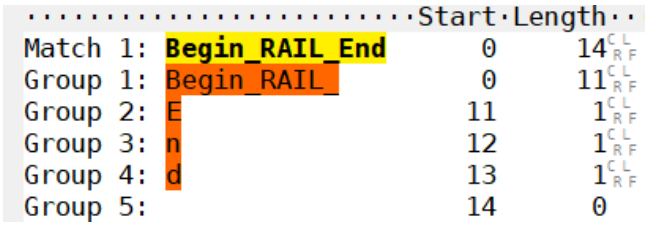
Take third character from end of string and prefix to front: cont.

Match: (.*)([^_-])([^_])(\S)(.*)

Samples:







Add

Add prefix Based on name cont.

Using: 8259438 some text

Tim Contribution – cont.

Take third character from end of string and prefix to front: cont.

Match: (.*)([^_-])([^_])(\S)(.*)

Where:

\1 = 8259438 some t \2 = e \3 = x \4 = t \5 = null

Tim Analysis:

1..	Match against any character	Capture Group 1 = 8
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [^_]	Match against a Negated Class consisting of a <hyphen> Match against any character but a <hyphen> Backtracks to match against the last 't' of 'text' Current position = EOL	Capture Group 2 = t Changes Capture Group 1 = 8259438 some tex
4. [^_]	Match against a Negated Class consisting of a <hyphen> Match against any character but a <hyphen> Backtracks to match against the last 't' of 'text' Current position = EOL	Capture Group 3 = t Changes Capture Group 2 = x Changes Capture Group 1 = 8259438 some te
5. \S	Matches any Non-Whitespace Character Backtracks to match against the last 't' of 'text' Current position = EOL	Capture Group 4 = t Changes Capture Group 3 = x Changes Capture Group 2 = e Changes Capture Group 1 = 8259438 some t
6. .*	Match against remaining string Already at EOL	Capture Group 5 = null All Other Capture Groups = Unchanged

Add

Add prefix Based on name cont.

Tim Notes:

1. Here's another one of those Examples, when analyzed, reveals a weakness in the RegEx, mainly that Capture Group 5 represented by the (.*) at the end, is unnecessary and could be removed. In all of the samples, Group 5 came up as a Zero Occurrence Match with a resulting null value. This is because when (\S) is evaluated, the EOL has already been reached. There is nothing more to capture.

2. All of these examples have prefixed to the front of the string. How do we suffix to the end of the string? Simple. It is all done in the Replacement String. The RegEx parses the string. You can put it together the way you want.

   i.e., using the last example,

   all we do is reverse the Replacement String from –

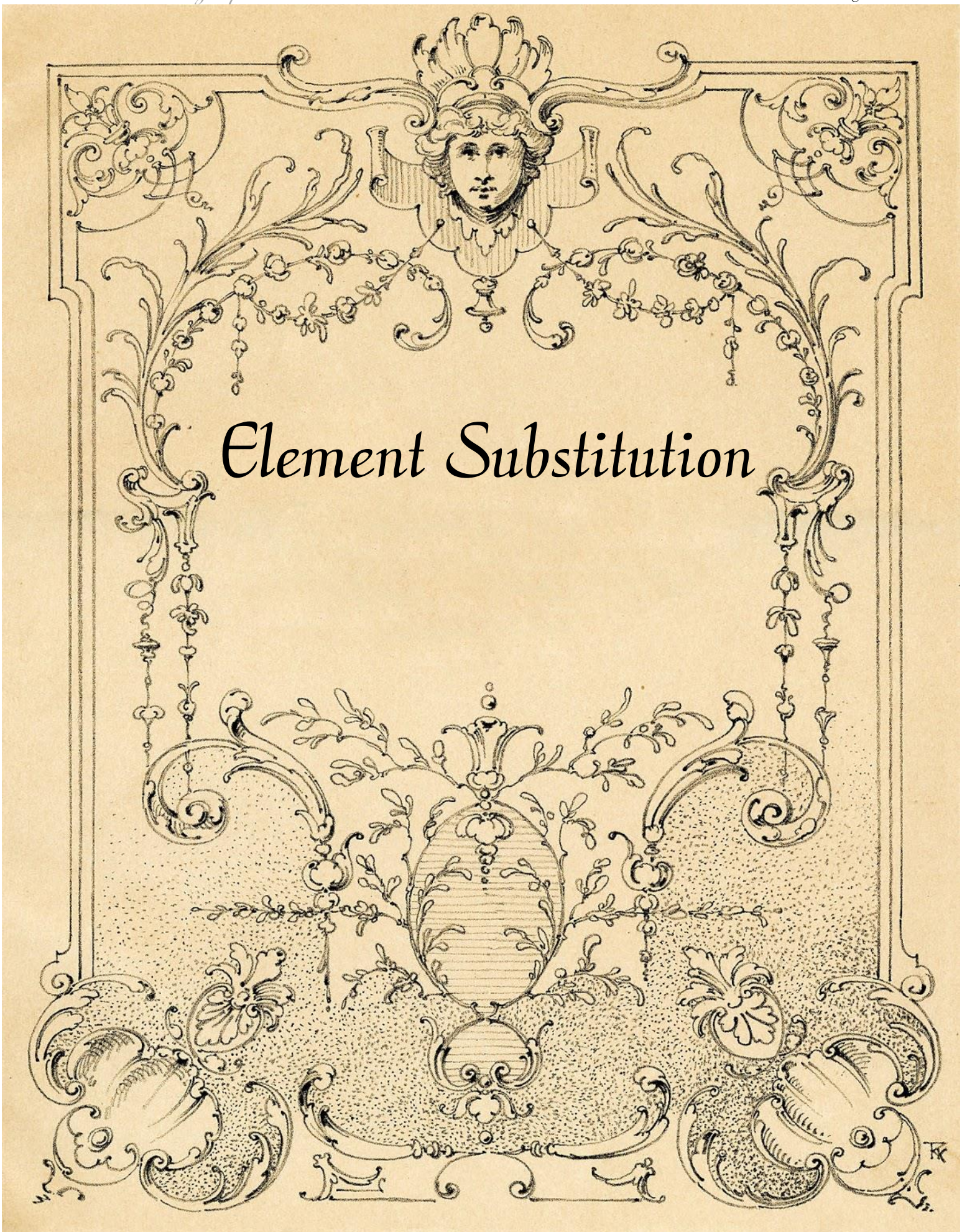
   Replace: \2_\1\2\3

   to -

   Replace: \1\2\3_\2

Name	New Name
ABC_DEF_GHI-123-45	ABC_DEF_GHI-123-45_B
Begin_RAIL_End	Begin_RAIL_End_e
TAIL_end	TAIL_end_A
Begin_SAIL	Begin_SAIL_e
8259438 some text	8259438 some text_2
ABC-123-45	ABC-123-45_B
Bahá'í Muñoz	Bahá'í Muñoz_a
My File v.1.2.345	My File v.1.2_y.345

3. So why all these exhaustive exercises about moving things from back to front or front to back? Because a lot of requests come in from users asking just that question. Hopefully this will give you ideas on how to go about it and adjust and tweak the RegEx as necessary to fit the patterns of your samples for the intended purposes.



Element Substitution

Element Substitution

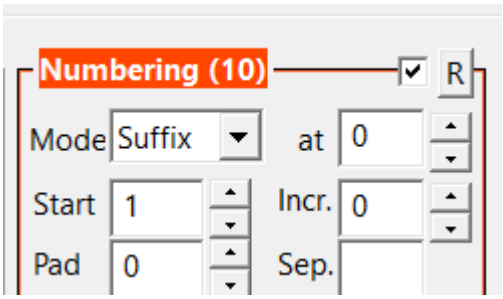
Convert a numeric identification preceded with underscore to (#)

342_236_197_0001.TIF to 342_236_197_(1).TIF

Two Runs:

First Run -

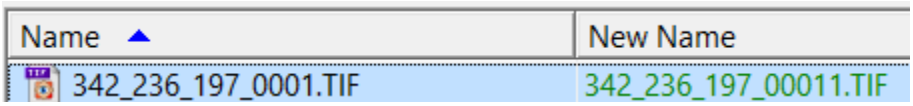
1. 'Section #10: Numbering' –



Mode: Suffix
Start: 1
Increment: 0

Tim Notes:

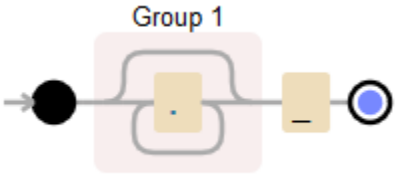
1. The Numbering is only to place the '1' at the end of the string, otherwise it will be removed during the RegEx.



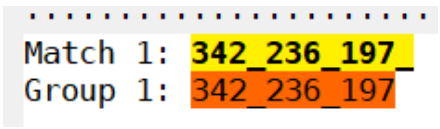
2. RegEx

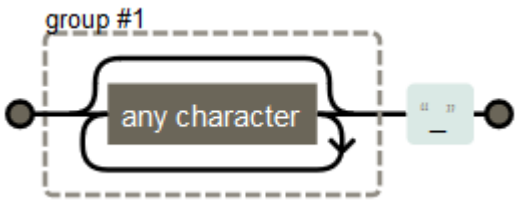
Match: (.*)_
Replace: \1_(

Don't forget the underscore character after \1



Sample:





Element Substitution

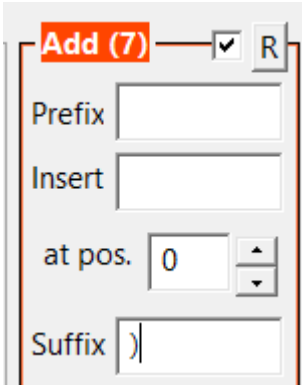
Convert a numeric identification preceded with underscore to (#) cont.

342_236_197_0001.TIF to 342_236_197_(1).TIF

Two Runs:

Second Run -

1. 'Section #7: Add'



Suffix:)

Forum Contributor: therube

Using: 342_236_197_0001.TIF

Match: (.*)_

Where:

\1 = 342_236_197

Tim Analysis:

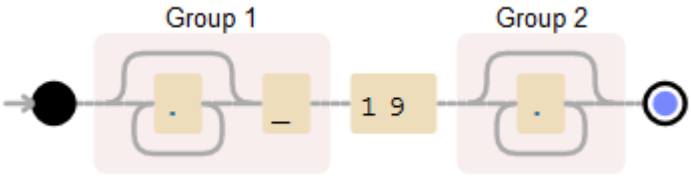
1..	Match against any character	Capture Group 1 = 3
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. _	Match against underscore. (Not Captured) Backtracks. This is the underscore After '197'	Changes Capture Group 1 = 342_236_197

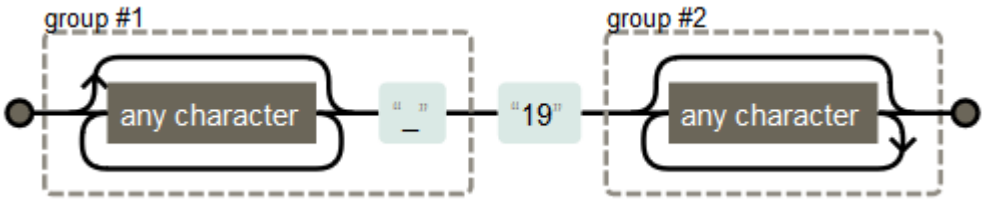
Element Substitution

Replace 19 with 2019 within filename

yyy xxxxxx zzz_190311_192559.dat to yyy xxxxxx zzz_20190311_192559.dat

Match: (.*?_)19(.*)
Replace: \12019\2





Sample:

```
........................................
Match 1: yyy xxxxxx zzz_190311_192559
Group 1: yyy xxxxxx zzz_
Group 2: 0311_192559
```

Forum Contributor: therube

Where:

\1 = yyy xxxxxx zzz_ \2 = 0311_192559

Tim Analysis:

1..	Match against any character	Capture Group 1 = y
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. _	Match against underscore	Capture Group 1 = yyy xxxxxx zzz_
5. 19	Match against string '19' (Not Captured) = '19'	
6. .*	Match against remaining string	Capture Group 2= 0311_192559

Element Substitution

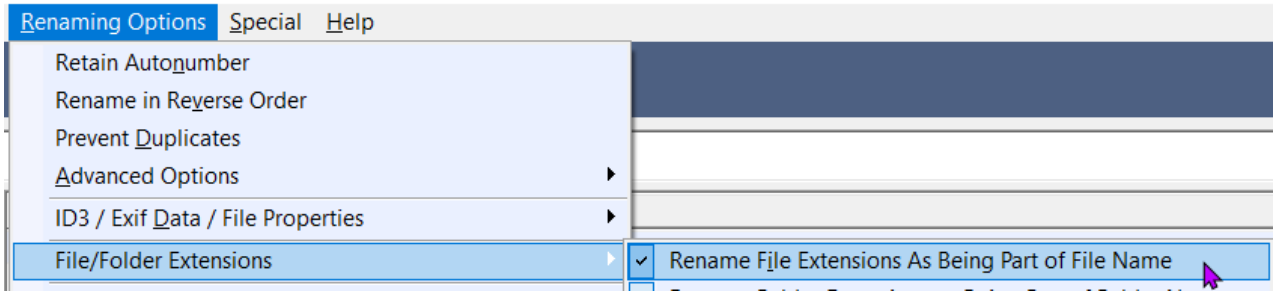
Move string to another location within and Replace dot with hyphen

Sample: A1234.BBBB.CCCC-DDDDDDDDD-EEEE

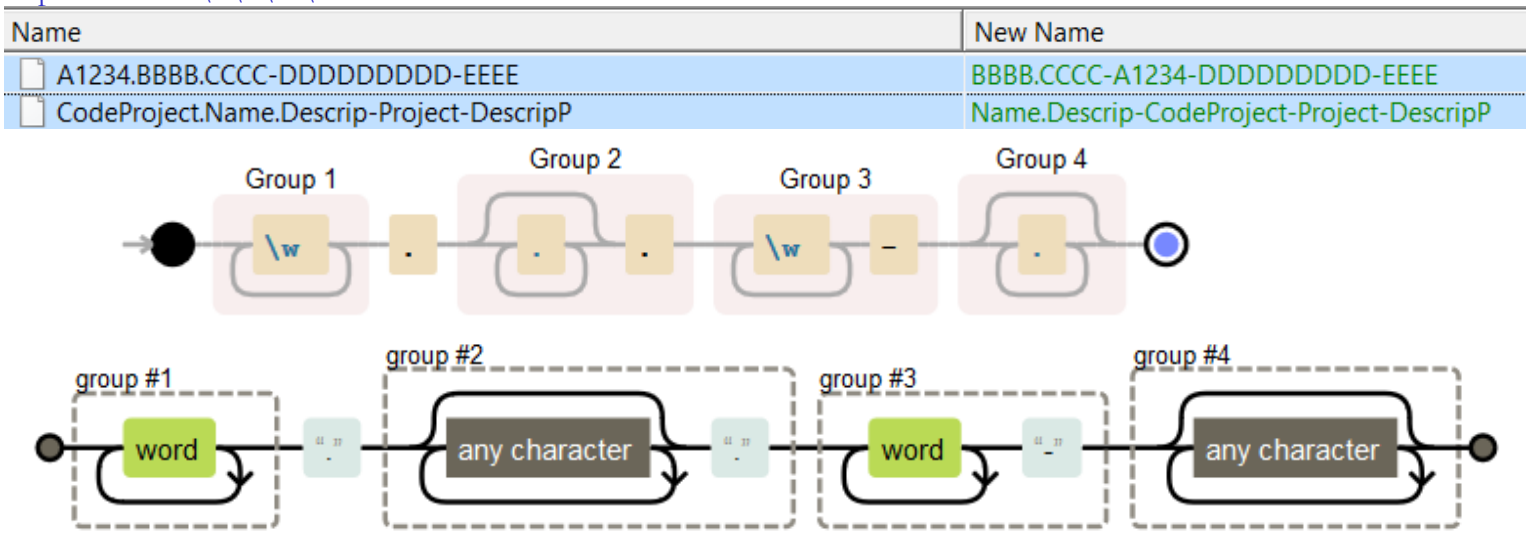
CodeProject.Name.Descrip-Project-DescripP
to
Name.Descrip-CodeProject-Project-DescripP

Because there are <dot> characters in ths string, you must enable 'Rename File Extensions As Being part of File Name' in the File/Folder/Extensions sub-menu of the Renaming Options Menu.

.



Match: (\w+)\.(.*\.)(\w+-)(.*)
Replace: \2\3\1-\4



Contributor Analysis:

1. find any number of "word characters" (\w+) followed by a (.) = [A1234]
2. find any number of "word characters" (.*) followed by a (.) = [BBBB.]
3. find any number of "word characters" (\w+) followed by a (-) = [DDDDDDDDD-]
4. gather up the rest = [EEEE]
5. rearrange things

Forum Contributor: therube

Element Substitution

Move string to another location within and Replace dot with hyphen cont.

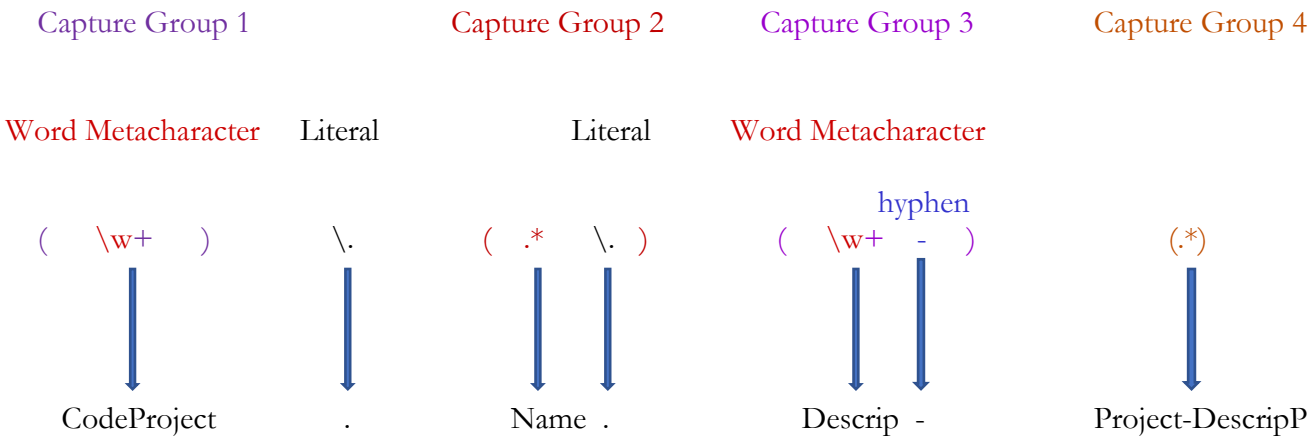
Using: CodeProject.Name.Descrip-Project-DescripP

Match: (\w+)\.(.*\.)(\w+-)(.*)

Sample:

```
.....................................................
Match 1: CodeProject.Name.Descrip-Project-DescripP
Group 1: CodeProject
Group 2: Name.
Group 3: Descrip-
Group 4: Project-DescripP
```

Breaks down:



Element Substitution

Move string to another location within and Replace dot with hyphen cont.

Using: CodeProject.Name.Descrip Descrip-Project-DescripP

Match: (\w+)\.(.*\.)(\w+-)(.*)

Where:

\1 = CodeProject \2 = Name. \3 = Descrip- \4 = Project-DescripP

Tim Analysis:

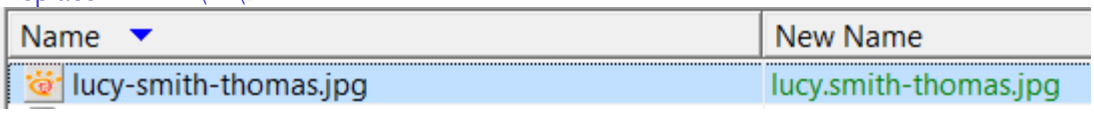
1. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 1 = C
2. +	Make it Greedy	Capture Group 1 = CodeProject
3. \.	Match against literal <dot> (Not Captured)	
4. .	Match against any character	Capture Group 2 = N
5. *	Make it Greedy Current position = EOL	Capture Group 2 = Name.Descrip-Project-DescripP
6. \.	Match against literal <dot> Backtracks.	Capture Group 2 = Name.
7. \w	Word Metacharacter. Match against any single letter, number or underscore	Capture Group 3 = D
8. +	Make it Greedy This is the 'Descript' following 'Name.'	Capture Group 3 = Descrip
9. -	Match against <hyphen>	Capture Group 3 = Descrip-
10. .*	Match against remaining string	Capture Group 4 = Project-DescripP

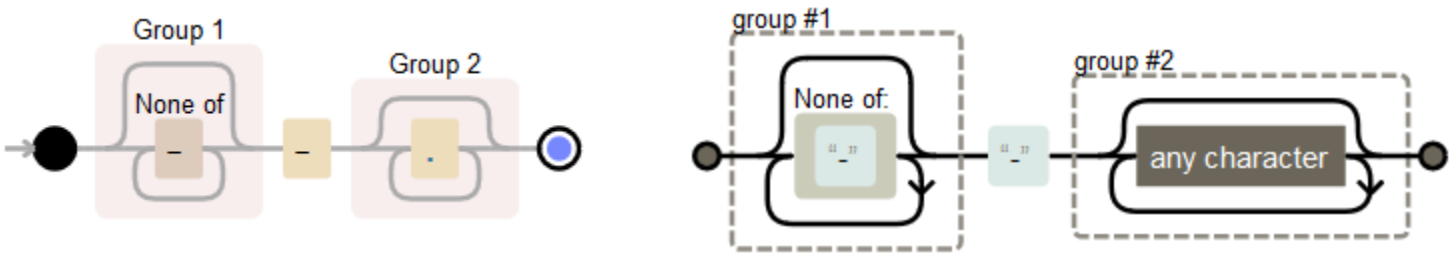
Element Substitution

Substitute dot for First hyphen in String

lucy-smith-thomas.jpg to lucy.smith-thomas.jpg

Match: ([^-]*)-(.*)
Replace: \1.\2





Sample:

```
.............................
Match 1: lucy-smith-thomas
Group 1: lucy
Group 2: smith-thomas
```

Forum Contributor: KenP

Using: lucy-smith-thomas.jpg

Where:

\1 = lucy \2 = smith-thomas

Tim Analysis:

1. [^-]	Match against a Negated Class consisting of a <hyphen>. Match against anything but a <hyphen>.	Capture Group 1 = l
2. *	Make it Greedy	Capture Group 1 = lucy
3. -	Match against a <hyphen> (Not Captured) This is the <hyphen> after 'lucy'	
4. .*	Match against remaining string	Capture Group 2 = smith-thomas

Element Substitution

Replace a space between numbers with a dot

3 0 44 1	to	3.0.44.1
3 00 4 1	to	3.00.4.1"
3 0 44 01	to	3.0.44.01"
3 00 44 001	to	3.00.44.001"
3 2	to	3.2"
3 0 4	to	3.0.4

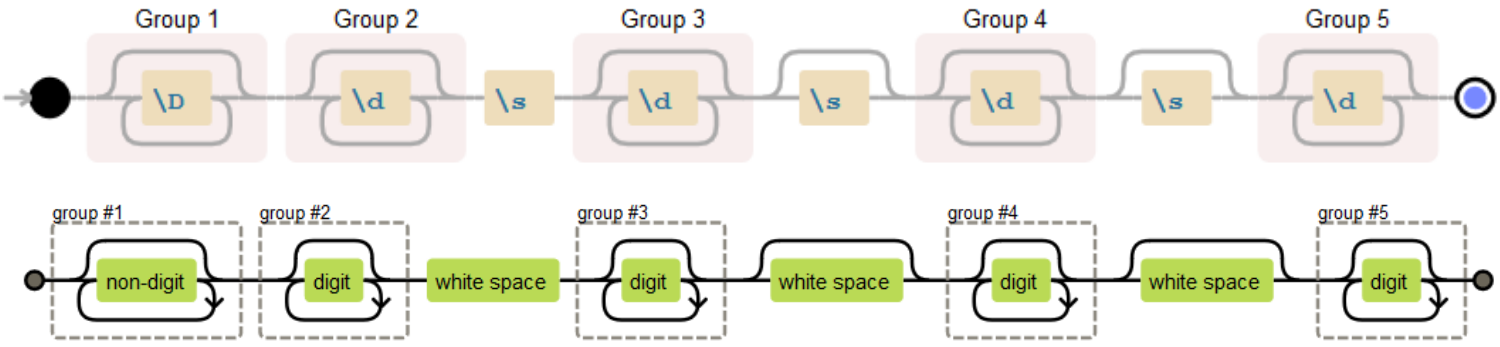
The original posting - Requires 2 Separate Runs:

First Run:

1. RegEx

Match: (\D*)(\d*)\s(\d*)\s?(\d*)\s?(\d*)
Replace: \1\2.\3.\4.\5

Name	New Name
3 2	3.2..
3 00 44 001	3.00.44.001
3 00 4 1	3.00.4.1
3 0 44 1	3.0.44.1
3 0 44 01	3.0.44.01
3 0 4	3.0.4.



Sample:

```
···················Start·Length···
Match 1: 3 0 44 1      0      8
Group 1:               0      0
Group 2: 3             0      1
Group 3: 0             2      1
Group 4: 44            4      2
Group 5: 1             7      1
```

Element Substitution

Replace a space between numbers with a dot cont.

First Run: cont.

Match: (\D*)(\d*)\s(\d*)\s?(\d*)\s?(\d*)

Contributor Notes:

1. The RegEx accounts for 4 groups of numbers each group separated with a <space>.
2. However, there are some files that have less than 4 groups. The Replace adds a dot after each group, If there are not 4 groups, the dot will still be added. Therefore the result will display those files in New Name with trailing dots. As you can see above, this includes two files, '3.2..' and '3.04.'

Tim Notes:

However, after a refresh:

Name	New Name
3.2	3.2
3.00.44.001	3.00.44.001
3.00.4.1	3.00.4.1
3.0.44.1	3.0.44.1
3.0.44.01	3.0.44.01
3.0.4	3.0.4

Windows automatically strips away trailing dots in a filename IF there is no extension.

Windows thinks it is folder. The samples above do not have extensions.

If extensions are added, then the filenames will display with the trailing dots:

Name	New Name
3 2.pdf	3.2...pdf
3 00 44 001.pdf	3.00.44.001.pdf
3 00 4 1.pdf	3.00.4.1.pdf
3 0 44 1.pdf	3.0.44.1.pdf
3 0 44 01.pdf	3.0.44.01.pdf
3 0 4.pdf	3.0.4..pdf

After a refresh, the files retain the trailing dots:

Name	New Name
3.2...pdf	3.2...pdf
3.00.44.001.pdf	3.00.44.001.pdf
3.00.4.1.pdf	3.00.4.1.pdf
3.0.44.1.pdf	3.0.44.1.pdf
3.0.44.01.pdf	3.0.44.01.pdf
3.0.4..pdf	3.0.4..pdf

Now the post is accurate in that it *will* require two runs. It does, though, offer an opportunity. You can perform the entire operation in one pass if you temporarily remove the file extensions and just perform the RegEx.

Element Substitution

Replace a space between numbers with a dot cont.

3 0 44 1	to	3.0.44.1
3 00 4 1	to	3.00.4.1"
3 0 44 01	to	3.0.44.01"
3 00 44 001	to	3.00.44.001"
3 2	to	3.2"
3 0 4	to	3.0.4

First Run: cont.

Match: (\D*)(\d*)\s(\d*)\s?(\d*)\s?(\d*)

Tim Notes: cont.

But let's continue as the post requires. One additional note worth mentioning is that the files '3.2…pdf' and '3.04..pdf' include the dot for the extensions, so actually the files are '3.2..' with two trailing dots and '3.04.' with one trailing dot.

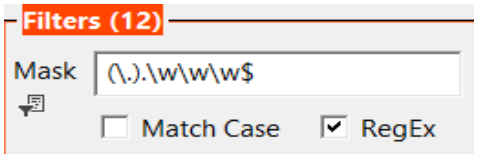
Post Continues -

Second Run:

So far we have some files that are fine as they are, and a couple that have trailing dots that need to be removed. This next run will display those files that need further modification and to apply the fixes.

1. 'Section #12: Filters' -

Mask: (\.).\w\w\w$
RegEx: selected



This filter will only display those files with trailing dots. The dots themselves are representative of the missing groups. Two files qualify. When you click the 'Refresh Filter Button', only those files will display.

Name	New Name
3.2...pdf	3.2...pdf
3.0.4..pdf	3.0.4..pdf

Element Substitution

Replace a space between numbers with a dot cont.

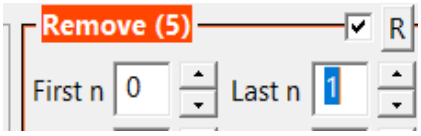
3 0 44 1	to	3.0.44.1
3 00 4 1	to	3.00.4.1"
3 0 44 01	to	3.0.44.01"
3 00 44 001	to	3.00.44.001"
3 2	to	3.2"
3 0 4	to	3.0.4

Second Run: cont.

2. 'Section #5: Remove'

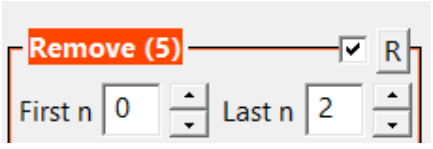
This will remove those trailing dots. This is done by using the 'LAST' data field of the Remove function and specifying the number of characters to remove. The value is dependent on how many trailing dots the filename contains.

Last n: 1

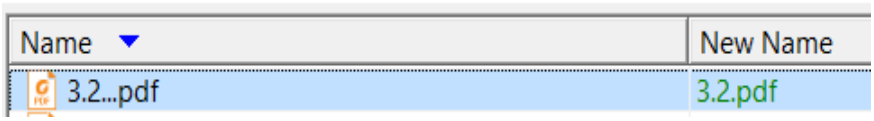


Use a value of 1 for those filenames that have 3 groups of numeric digits (remember, the RegEx accounted for 4 groups). In this example, there is only one. If you select the file you will see the final result.

Last n: 2



Use a value of 2 for those filenames that have 2 groups of numeric digits (remember, the RegEx accounted for 4 groups). In this example, there is only one. If you select the file you will see the final result.



Forum Contributor: KenP

Element Substitution

Replace a space between numbers with a dot cont.

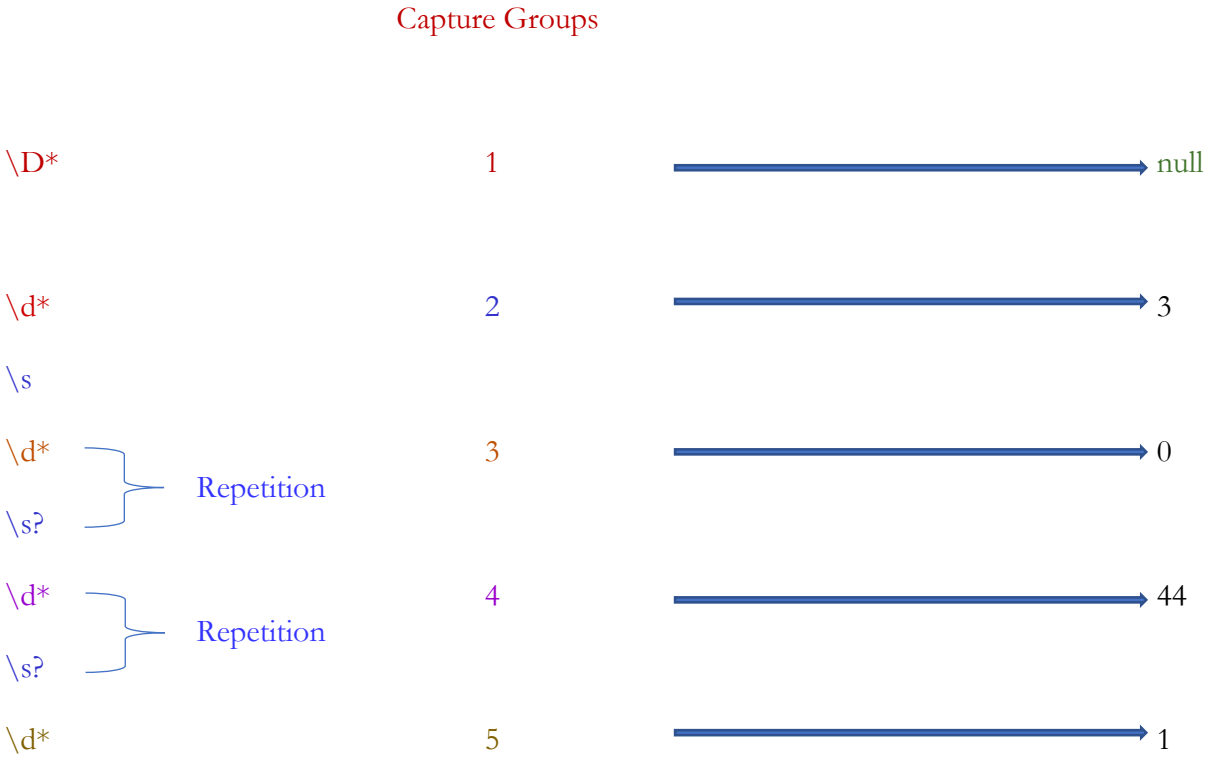
Using: 3 0 44 1

Match: (\D*)(\d*)\s(\d*)\s?(\d*)\s?(\d*)

Where:

\1 = null \2 = 3 \3 = 0 \4 = 44 \5 = 1

Breaks down:



Element Substitution

Replace a space between numbers with a dot cont.

Using: 3 0 44 1

Match: (\D*)(\d*)\s(\d*)\s?(\d*)\s?(\d*)

Where:

\1 = null \2 = 3 \3 = 0 \4 = 44 \5 = 1

Tim Analysis:

1. \D	Match against any non-digit numeric This is the <space> after '3'	Capture Group 1 = <space>
2. *	Make it Greedy Searches to EOL, then back to BOL	Capture Group 1 = null

Tim Notes:

1. This is one of those I find hard to evaluate on a per character basis. By itself, the \D will match against any non-digit, so it matches against the <space> after '3' initially. When made Greedy, however, it skips over all the non-digit characters, the <space> after '3', the <space> after '0' and the <space> after '44' hitting the EOL at which point it starts over at BOL.

3. \d	Match against Numeric Digit	Capture Group 2 = 3
4. *	Make it Greedy	Capture Group 2 = Unchanged
5. \s	Match against a <space> character (Not Captured) This is the <space> after '3'	
6. \d	Match against Numeric Digit	Capture Group 3 = 0
7. *	Make it Greedy	Capture Group 3 = Unchanged
8. \s	Match against a <space> character (Not Captured) This is the <space> after '0'	
9. ?	Make it optional.	All Capture Groups = Unchanged
10. \d	Match against Numeric Digit	Capture Group 4 = 4
11. *	Make it Greedy	Capture Group 4 = 44
12. \s	Match against a <space> character (Not Captured) This is the <space> after '44'	
13. ?	Make it optional.	All Capture Groups = Unchanged
14. \d	Match against Numeric Digit	Capture Group 5 = 1
15. *	Make it Greedy	Capture Group 5 = Unchanged

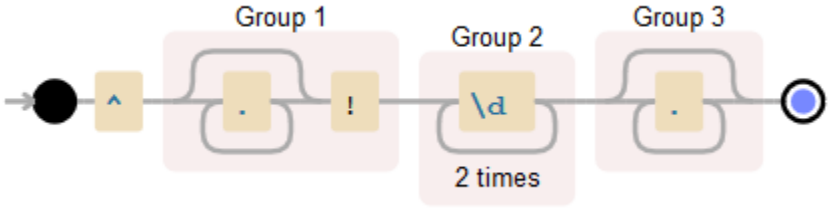
Element Substitution

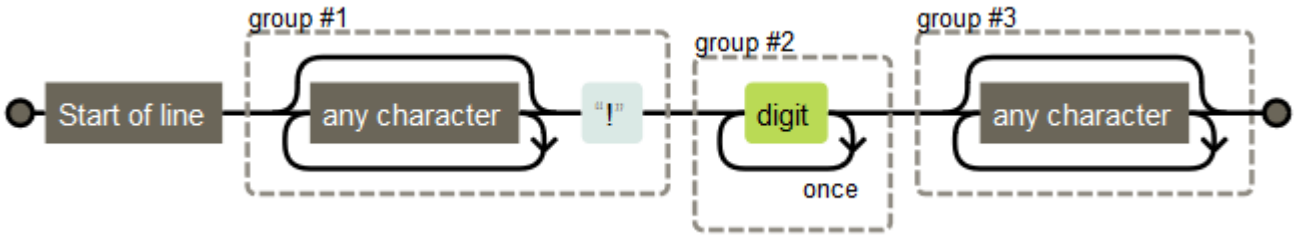
Substitute 'QB' for first two digits

First_Last_!303405.pdf to First_Last_!QB3405.pdf

Match: ^(.*!)(\d{2})(.*)
Replace: \1QB\3

Name	New Name
First_Last_!303405.pdf	First_Last_!QB3405.pdf





Forum Contributor: KenP

Where:

\1 = First_last_! \2 = 30 \3 = 3405

Sample:

```
Match 1: First_Last_!303405
Group 1: First_Last_!
Group 2: 30
Group 3: 3405
```

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. .	Match against any character	Capture Group 1 = f
3. *	Make it Greedy	Capture Group 1 = <entire string>
	Current position = EOL	
4. !	Match against !	Capture Group 1 = First_last_!
	Backtracks.	
5. \d{2}	Match against 2 numeric digits	Capture Group 2 = 30
6. .*	Match against remaining string	Capture Group 3 = 3405

Element Substitution

Replace Parentheses ( ) with Brackets [ ]

FolderName (nnnn) to FolderName [nnnn]

Where:

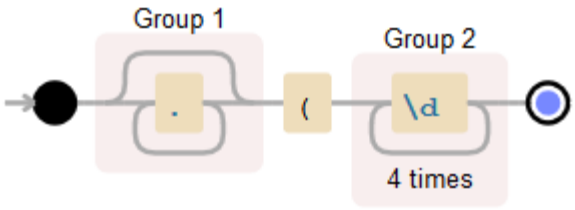
nnnn = 4 numeric digit year

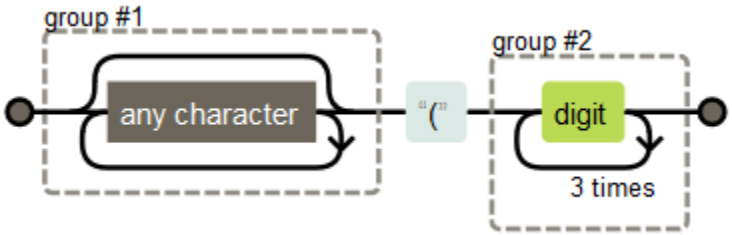
Match: (.*)\((\d{4})
Replace: \1[\2]

Name	New Name
FolderName (1980)	FolderName [1980]

Sample:

```
.............................
Match 1: FolderName (1980
Group 1: FolderName
Group 2: 1980
```





Forum Contributor: KenP

Using: FolderName (1980)

Where:

\1 = FolderName <space> \2 = 1980

Tim Analysis:

1..	Match against any character	Capture Group 1 =
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. \(	Match against a left parenthesis (Not Captured) Backtracks.	Changes Capture Group 1 = FolderName <space>
4. \d {4}	Match against 4 numeric digits	Capture Group 2 = 1980

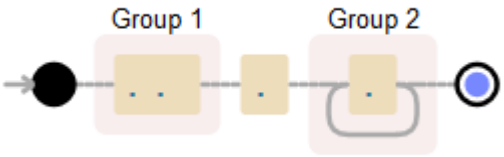
Element Substitution

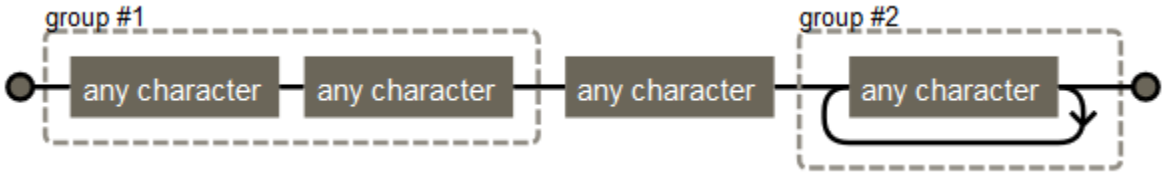
Simple Substitution – Substitute 'M' for 4th character in String

BBB001.jpg	to	BBM001.jpg
BBB002.jpg	to	BBM002.jpg
BBB003 48x12.jpg	to	BBM003 48x12.jpg

Match: (..).(.+)
Replace: \1M\2

Name	New Name
BBB003 48x12.jpg	BBM003 48x12.jpg
BBB002.jpg	BBM002.jpg
BBB001.jpg	BBM001.jpg





Sample:

```
.................
Match 1: BBB001
Group 1: BB
Group 2: 001
```

Contributor Analysis:

1. Capture Group 1 - first three characters = i.e., BB
2. Do not capture 4th character (B)
3. Capture Group 2 - rest of string to EOL (End of Line) = i.e., 001

Forum Contributor: Stefan

Uncategorized

NOTE:

The following contains both RegEx and BRU Uncategorized Entries.

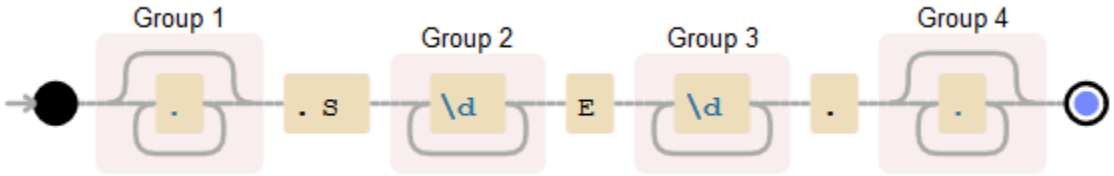
Uncategorized

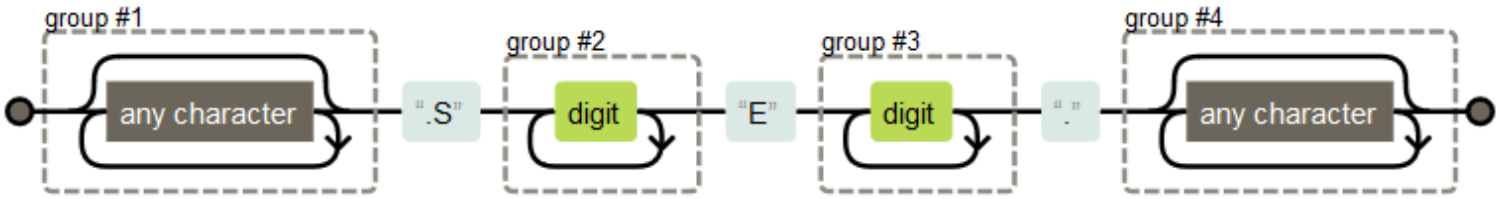
Downloaded TV Shows

The Office.S01E01.Pilot.mp4 to The Office - s01e01 - Pilot.mp4

Match: (.*)\.S(\d+)E(\d+)\.(.*)
Replace: \1 -s\2e\3- \4

Name	New Name
The Office.S01E01.Pilot.mp4	The Office -s01e01- Pilot.mp4





Sample:

```
.....................................
Match 1: The Office.S01E01.Pilot
Group 1: The Office
Group 2: 01
Group 3: 01
Group 4: Pilot
```

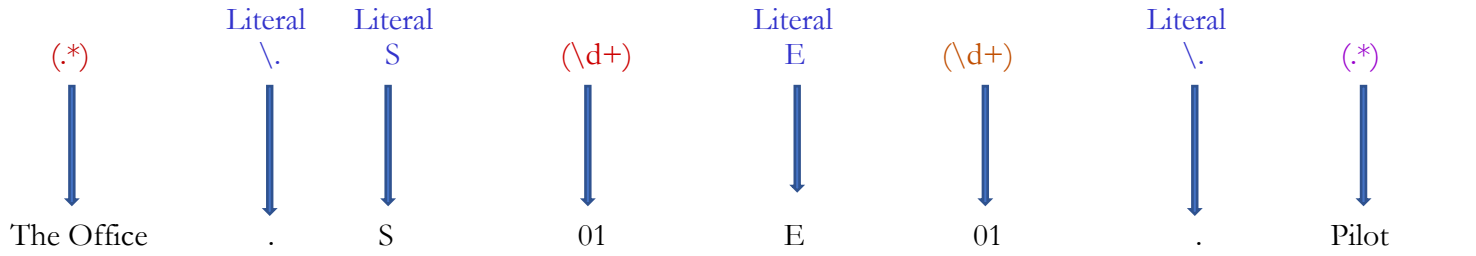
Uncategorized

Downloaded TV Shows cont.

The Office.S01E01.Pilot.mp4 to The Office - s01e01 - Pilot.mp4

Match: (.*)\.S(\d+)E(\d+)\.(.*)

Breaks down:



Where:

\1 = The Office \2 = 01 \3 = 01 \4 = Pilot

Tim Analysis:

1..	Match against any character	Capture Group 1 = T
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. \.	Match against literal <dot> (Not Captured) This is the <dot> after 'S01E01'	Changes Capture Group 1 = The Office.S01E01
4. S	Match against string 'S' (Not Captured) Changes step 3 value to <dot> after 'Office'	Changes Capture Group 1 = The Office
5. \d	Match against Numeric Digit This is the '0' of 'S01'	Capture Group 2= 0
6. +	Make it Greedy	Capture Group 2 = 01
7. E	Match against string 'E' (Not Captured)	
8. \d	Match against Numeric Digit This is the '0' of 'E01'	Capture Group 3 = 0
9. +	Make it Greedy	Capture Group 3 = 01
10. \.	Match against literal <dot> (Not Captured) This is the <dot> after 'S01E01'	
11. .*	Match against remaining string	Capture Group 4 = Pilot

Uncategorized

Downloaded TV Shows cont.

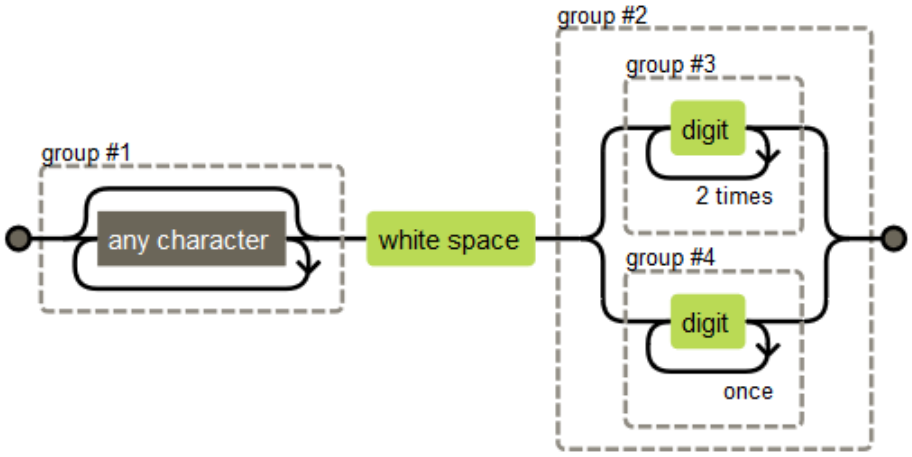
ShowName 01.mkv to Showname S01-E01.mkv
ShowName 02.mkv to Showname S01-E02.mkv
Showname 100.mkv to Showname S01-E100.mkv

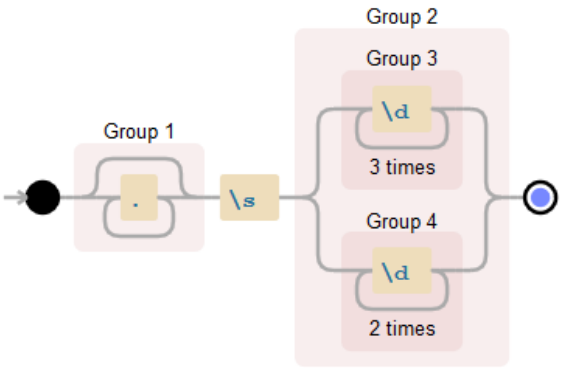
Match: (.*)\s((\d{3})|(\d{2}))
Replace: \1 S01-E\2

Name	New Name
Showname 100.mkv	Showname S01-E100.mkv
ShowName 02.mkv	ShowName S01-E02.mkv
ShowName 01.mkv	ShowName S01-E01.mkv

Contributor Notes:

1. You'll need to change the series number (colored red) to suit the series, i.e., S02 for season 2, etc.





Sample:

```
.....................Start.Length.....
Match 1: ShowName 01      0     11
Group 1: ShowName         0      8
Group 2: 01               9      2
Group 3 did not participate in the match
Group 4: 01               9      2
```

Sample:

```
.....................Start.Length.....
Match 1: Showname 100     0     12
Group 1: Showname         0      8
Group 2: 100              9      3
Group 3: 100              9      3
Group 4 did not participate in the match
```

Contributor Analysis:

1. (.*) Capture Group 1 captures everything up-to the 2 numbers which are the episode numbers.

2. ((\d{3})|(\d{2})) Looks for **either** a two numeric digit or three numeric digit episode number and if True will capture the value (episode numbers) in Capture Group 2.

Forum Contributor: KenP

Uncategorized

Downloaded TV Shows cont.

ShowName 01.mkv to Showname S01-E01.mkv
ShowName 02.mkv to Showname S01-E02.mkv
Showname 100.mkv to Showname S01-E100.mkv

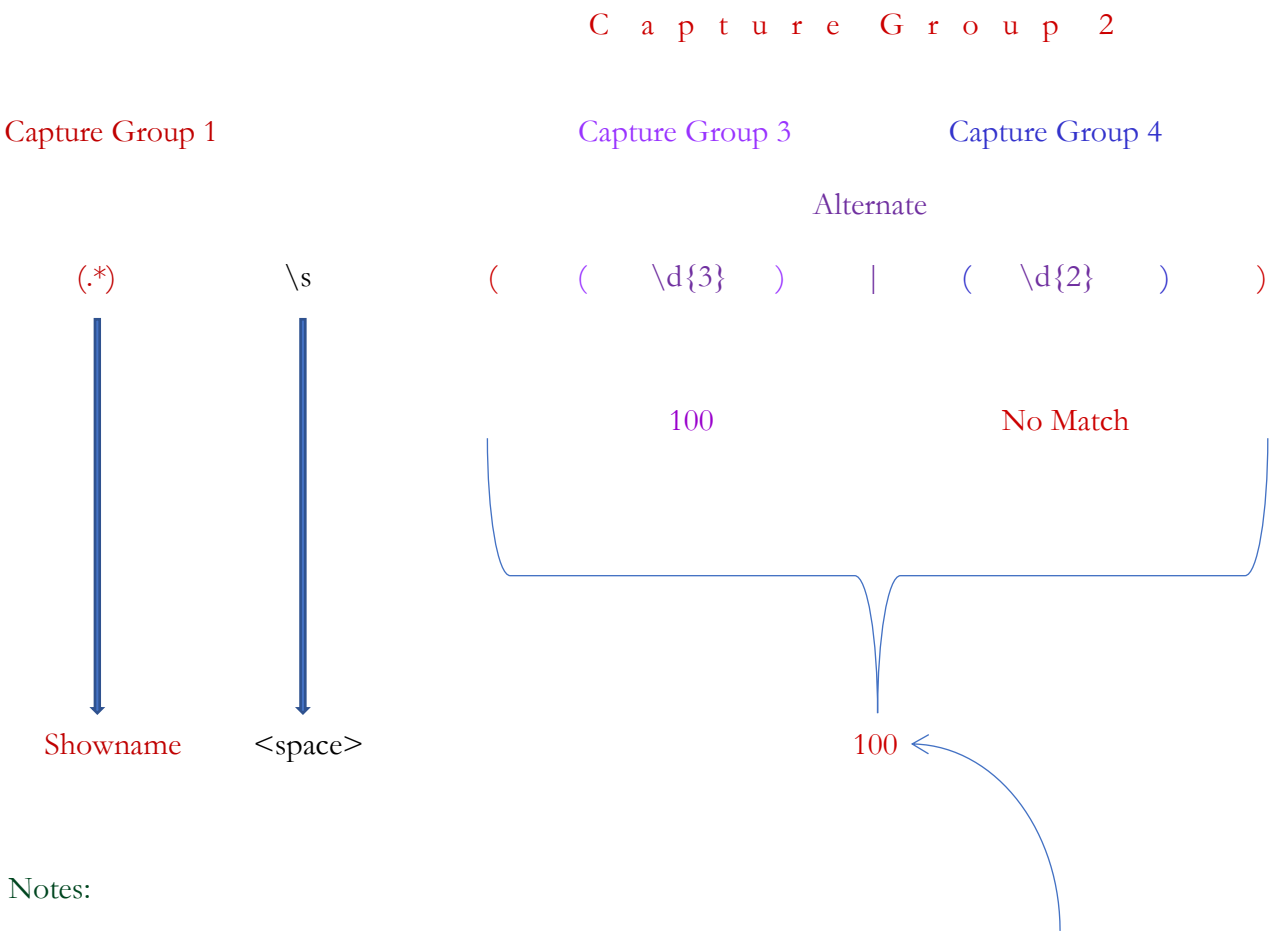
Match: (.*)\s((\d{3})|(\d{2}))

This uses an Alternate for Capture Group 2

This OR This

\d{3} | \d{2}

Breaks down:



Tim Notes:

1. A nested Capture Group takes its value inclusive of all values of the Capture Groups within.

Uncategorized

Downloaded TV Shows cont.

Match: (.*)\s((\d{3})|(\d{2}))

Using: Showname 100.mkv

Where:

\1 = Showname \2 = 100 \3 = 100 \4 = No Value

Tim Analysis:

1. 1..	Match against any character	Capture Group 1 =
2. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. \s	Match against a <space> character (Not Captured) This is the <space> after 'Showname'	Changes Capture Group 1 = Showname
4. \d {3}|\d{2}	Alternate.	Capture Group 2 = 100 Capture Group 3 = 100
	\d {3} Match against 3 numeric digits = 100 = True	
	or	
	\d{2} Match against 2 numeric digits	Capture Group 4 = No Value

Uncategorized

Isolate non-ASCII characters

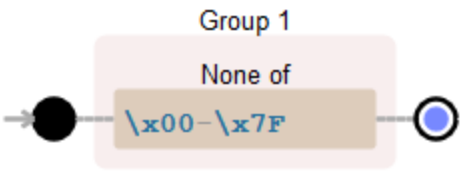
Samples:

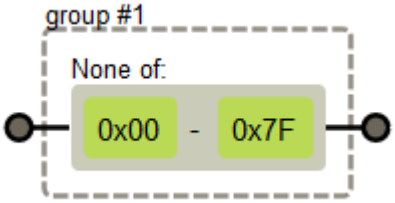
Amy Mußul (Mussul) Amparo Muñoz
Bahá'í Muñoz Ferrá'í

Match: ([^\x00-\x7F])
Replace: \1

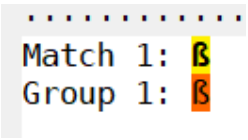
Name	New Name
Ferrá'í	á
Bahá'í Muñoz	á
Amy Mußul (Mussul)	ß
Amparo Muñoz	ñ

ASCII covers the bytes from 0 (x00) to 127 (x7F)
Syntax = \xYY
where \x = Hexidecimal Special Metacharacter

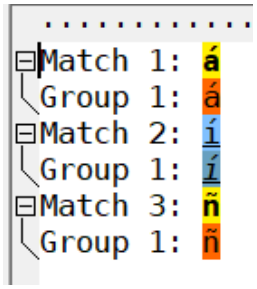




Sample: Amy Mußul (Mussul)



Sample: Bahá'í Muñoz



Contributor Notes:

1. Character class needs to be negated to search for characters NOT in the range, otherwise it will only detect if the string has *any* ASCII characters.

Tim Notes:

1. Limited – finds first occurrence only (refers only to Bru. The Regex Buddy Tester displays all matches)

Forum Contributor: xavier5678, therube

Uncategorized

Isolate non-ASCII characters cont.

Using: Amy Mußul (Mussul)

Match: ([^\x00-\x7F])

Where:

\1 = ß

Tim Analysis:

1. [^\x00-\x&F] Match against a Negated Class consisting of: hex – x00 through x&F. Match against anything but these hex codes — Capture Group 1 = ß

Tim Notes:

1. The hex codes x00 – x&F represent the ASCII codes 0 through 127. These codes correspond to the regular ASCII set as displayed below. They include the Alpha-numerics, general symbols, and some non-printable characters.

2. By setting these codes in a Negated Class, it directs theRegEx to search the string for the first occurrence of a hex code that is higher than ASCII 127.

Code	Char	Code	Char	Code	Char	Code	Char
0	NUL	32	SPACE	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	TAB	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o

Code	Char	Code	Char	Code	Char	Code	Char
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Uncategorized

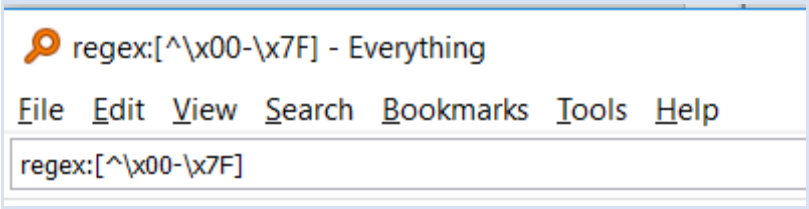
Isolate non-ASCII characters cont.

Samples:

Amy Mußul (Mussul) Amparo Muñoz
Bahá'í Muñoz Ferrá'í

If you want to search for all matching files, the Search Program, 'Everything', provides a RegEx feature:

1. Enter in the search field, regex:[^\x00-\x7F]



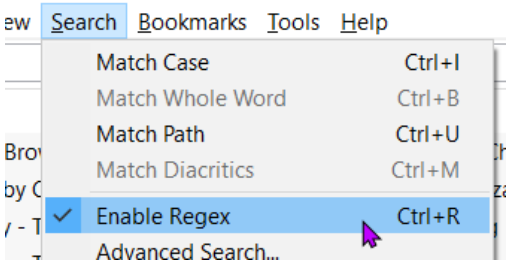
It will find all matching files and folders. Then all you need to do is to drag them into BRU.

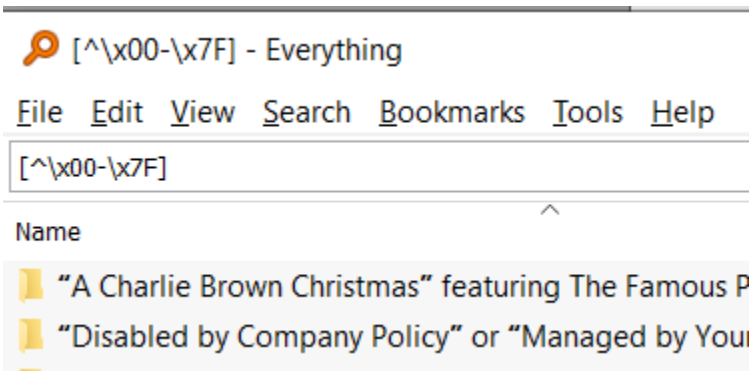
Forum Contributor: therube, KenP

Tim Notes:

In the program, Everything, you don't have to use the syntax, 'regex: <regex>', instead you can enable RegEx directly in the Search Menu:

.. Then just enter the RegEx in the Search data field.





This is most often preferable, because certain metacharacters, i.e. the EOL anchor, $, won't work correctly otherwise (speaking for myself, this is what I have found to be true).

Uncategorized

Isolate non-ASCII characters cont.

Samples:

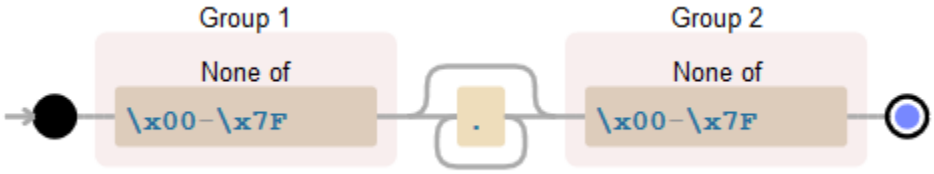
Amy Mußul (Mussul) Amparo Muñoz
Bahá'í Muñoz Ferrá'í

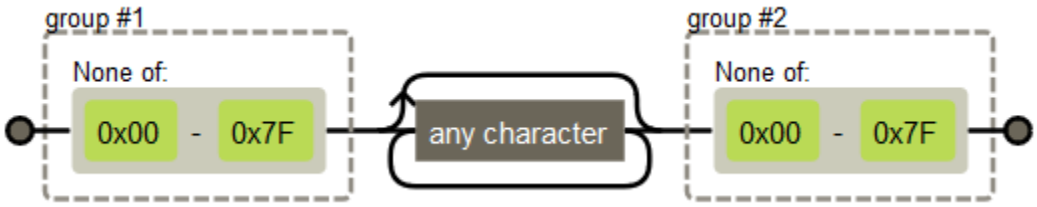
Tim Contribution –

This will find first and second occurrences:

Match: ([^\x00-\x7F]).*?([^\x00-\x7F])
Replace: \1\2

Name	New Name
Ferrá'í	áí
Bahá'í Muñoz	áí
Amy Mußul (Mussul)	Amy Mußul (Mussul)
Amparo Muñoz	Amparo Muñoz





Sample: Bahá'í Muñoz

```
.............
Match 1: á'í
Group 1: á
Group 2: í
```

Sample: Ferrá'í

```
.............Start·Length··
Match 1: á'í      4       5
Group 1: á        4       2
Group 2: í        7       2
```

… But will only work on strings that contain two or more occurrences.

Uncategorized

Isolate non-ASCII characters cont.

Using: Ferrá'í

Tim Contribution – cont.

This will find first and second occurrences: cont.

Match: ([^\x00-\x7F]).*?([^\x00-\x7F])

Where:

\1 = á \2 = í

Tim Analysis:

1. [^\x00-\x&F]	Match against a Negated Class consisting of: hex – x00 through x&F. Match against anything but these hex codes	Capture Group 1 = á
2..	Match against any character (Not Captured)	
3. *	Make it Greedy (Not Captured) = ' í ' Current position = EOL	
4. [^\x00-\x&F]	Match against a Negated Class consisting of: hex – x00 through x&F. Backtracks.	Capture Group 2 = í

Uncategorized

Isolate non-ASCII characters cont.

Samples:

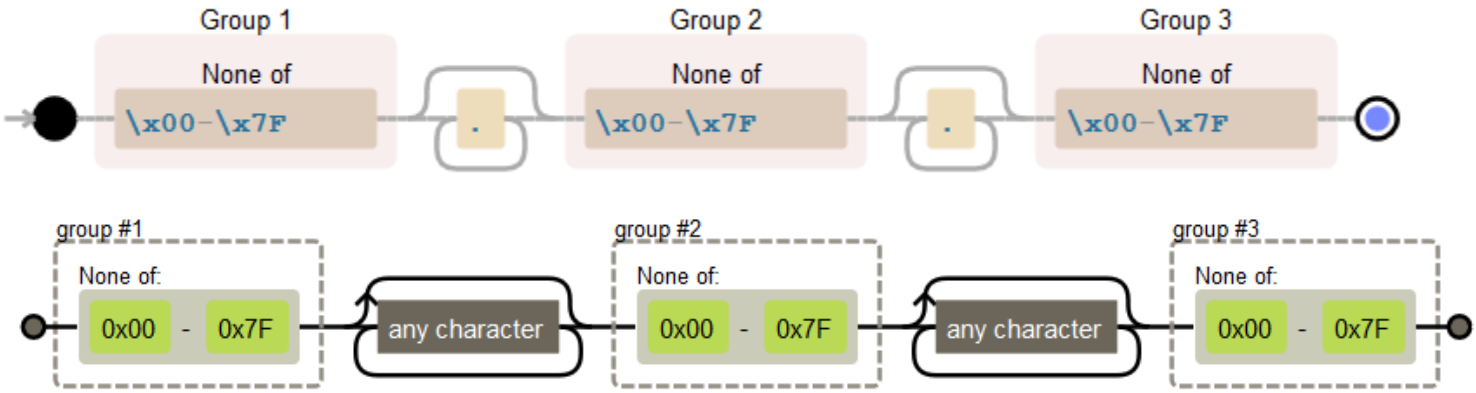
Amy Mußul (Mussul) Amparo Muñoz
Bahá'í Muñoz Ferrá'í

Tim Contribution – cont.

This will find three occurrences:

Match: ([^\x00-\x7F]).*?([^\x00-\x7F]).*?([^\x00-\x7F])
Replace: \1\2\3

Name	New Name
Ferrá'í	Ferrá'í
Bahá'í Muñoz	áíñ
Amy Mußul (Mussul)	Amy Mußul (Mussul)
Amparo Muñoz	Amparo Muñoz



Sample: Bahá'í Muñoz

	Start	Length
Match 1:	á'í Muñ	3 10
Group 1:	á	3 2
Group 2:	í	6 2
Group 3:	ñ	11 2

… But will only work on strings that contain three or more occurrences. Still, these give you an idea of how it works.

Uncategorized

Isolate non-ASCII characters cont.

Using: Bahá'í Muñoz

Tim Contribution – cont.

This will find three occurrences: cont.

Match: ([^\x00-\x7F]).*?([^\x00-\x7F]).*?([^\x00-\x7F])

Where:

\1 = á \2 = í \3 = ñ

Tim Analysis:

1. [^\x00-\x&F]	Match against a Negated Class consisting of: hex – x00 through x&F. Match against anything but these hex codes	Capture Group 1 = á
2..	Match against any character (Not Captured) = ' í '	
3. *	Make it Greedy (Not Captured) = ' í Muñoz ' Current position = EOL	
4. ?	But not too Greedy Changes Not Captured to just before the ' í '	
5. [^\x00-\x&F]	Match against a Negated Class consisting of: hex – x00 through x&F. Backtracks. Changes Not Captured to the apostrophe ' of ' Bahá' '	Capture Group 2 = í
6..	Match against any character (Not Captured) = <space> This is the <space> after ' Bahá'í '	
7. *	Make it Greedy (Not Captured) = ' <space> Muñoz ' Current position = EOL	
8. ?	But not too Greedy Changes not Captured to just before the <space> preceding ' Muñoz '	
9. [^\x00-\x&F]	Match against a Negated Class consisting of: hex – x00 through x&F. Backtracks. Changes Not Captured = '<space>Mu'	Capture Group 3 = ñ

Uncategorized

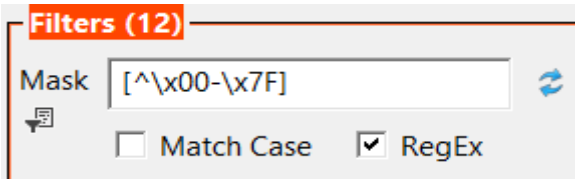
Isolate non-ASCII characters cont.

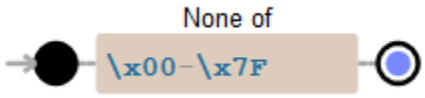
Tim Contribution – cont.

If you want to find matching files – then use the first RegEx in a filter:

'Section #12: Filters' -

Mask: [^\x00-\x7F]
RegEx: Selected

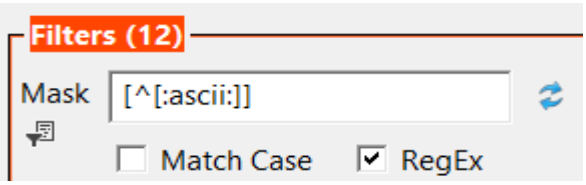


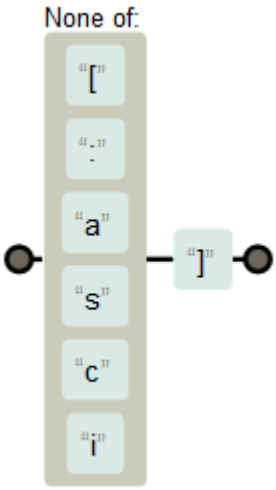


Or:

'Section #12: Filters' -

Mask: [^[:ascii:]]
RegEx: Selected





Click on the Filter Refresh Button (blue button to the right of the Mask data field)

Name	New Name
Ferrá'í	Ferrá'í
Bahá'í Muñoz	Bahá'í Muñoz
Amy Mußul (Mussul)	Amy Mußul (Mussul)
Amparo Muñoz	Amparo Muñoz

Tim Tidbit:

1. To Match all ASCII characters from 32 to 127, you can use: [-~] which is a case that consists of all printable characters from <space> through <tilde> . Uses the range ' - ' to do this. = [<space> - ~]
2. To Match *any* ASCII character, you can use the ASCII Class, [:ascii:] or [\x00-\x7F]

3. To match a specific ASCII character, enter it's hex value. i.e. ([\xF1]) (Decimal 241) to match éAmparo Muñoz. Refer to the Latin-1 ASCII Chart found in the Appendix.

Uncategorized

Isolate non-ASCII characters cont.

Samples:

Amy Mußul (Mussul) éAmparo Muñoz
Bahá'í Muñoz Ferrá'í

This will find all occurrences:

Observations:

1. Universal.
2. Will match up to 8 occurrences.
3. Multiple runs will do 8 more each time.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Replace: \1\2\3\4\5\6\7\8\9

	New Name
Amy Mußul (Mussul)	ß
Bahá'í Muñoz	áíñ
Ferrá'í	áí
éAmparo Muñoz	éñ

Sample:

```
·····························Start·Length··
Match 1: Amy Mußul (Mussul)        0      19 C L R F
Group 1: ß                         6       2 C L R F
Group 2:                          19       0 C L R F
Group 3:                          19       0 C L R F
Group 4:                          19       0 C L R F
Group 5:                          19       0 C L R F
Group 6:                          19       0 C L R F
Group 7:                          19       0 C L R F
Group 8:                          19       0 C L R F
Group 9:                          19       0
```

```
Beginning match attempt at character 0
1  ok
2  Amy Mu
3  Amy Muß
4  Amy Muß
5  Amy Mußul (Mussul)
6  Amy Mußul (Mussul)ok
7  Amy Mußul (Mussul)ok
8  Amy Mußul (Mussul)ok
9  Amy Mußul (Mussul)ok
10 Amy Mußul (Mussul)ok
11 Amy Mußul (Mussul)ok
12 Amy Mußul (Mussul)ok
13 Amy Mußul (Mussul)ok
14 Amy Mußul (Mussul)ok
15 Amy Mußul (Mussul)ok
16 Amy Mußul (Mussul)ok
17 Amy Mußul (Mussul)ok
18 Amy Mußul (Mussul)ok
19 Amy Mußul (Mussul)ok
20 Amy Mußul (Mussul)ok
21 Amy Mußul (Mussul)ok
22 Amy Mußul (Mussul)ok
23 Amy Mußul (Mussul)ok
24 Amy Mußul (Mussul)ok
25 Amy Mußul (Mussul)ok
26 Amy Mußul (Mussul)ok
27 Amy Mußul (Mussul)ok
28 Amy Mußul (Mussul)ok
   Match found in 28 steps
```

Uncategorized

Isolate non-ASCII characters cont.

Samples:

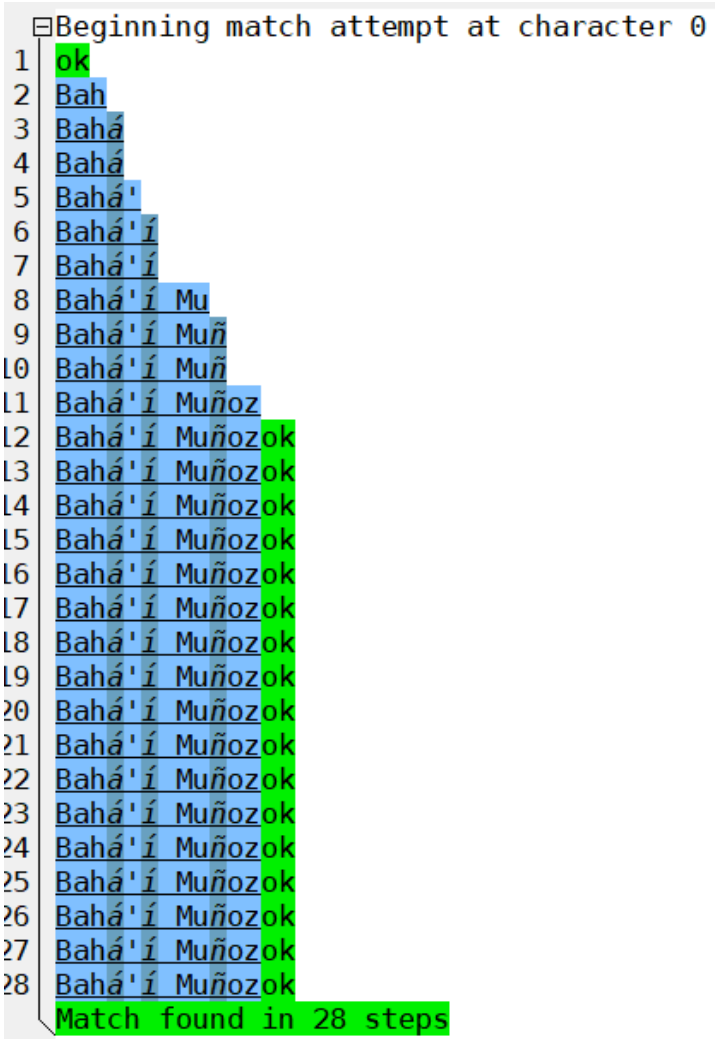
Amy Mußul (Mussul)	éAmparo Muñoz
Bahá'í Muñoz	Ferrá'í

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Sample:

		Start	Length
Match 1:	Bahá'í Muñoz	0	15
Group 1:	á	3	2
Group 2:	í	6	2
Group 3:	ñ	11	2
Group 4:		15	0
Group 5:		15	0
Group 6:		15	0
Group 7:		15	0
Group 8:		15	0
Group 9:		15	0



Uncategorized

Isolate non-ASCII characters cont.

Samples:

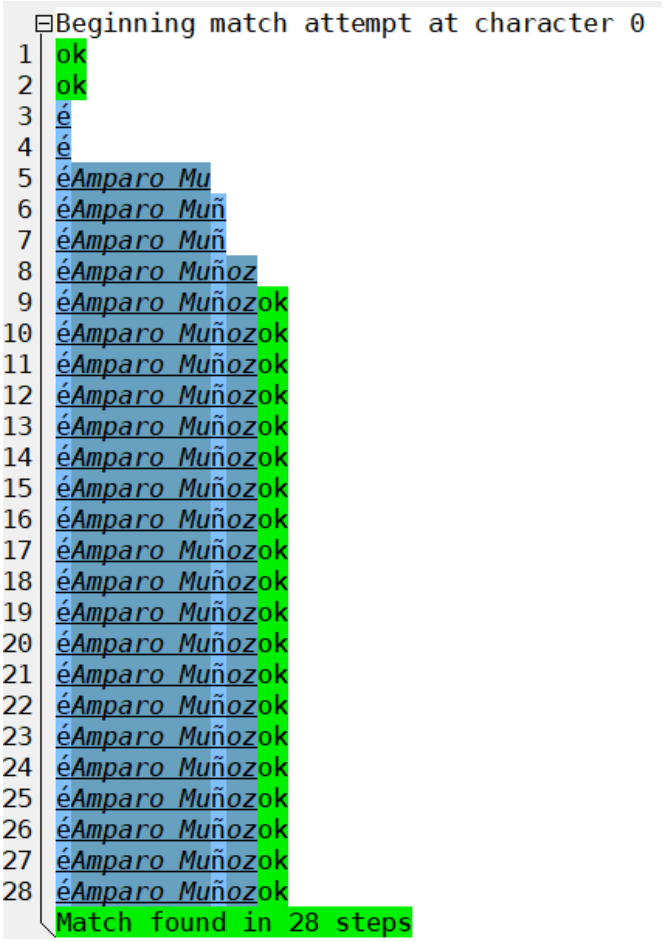
Amy Mußul (Mussul) éAmparo Muñoz
Bahá'í Muñoz Ferrá'í

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Sample:

	Start	Length
Match 1: **Amparo Muñoz**	0	13
Group 1: ñ	9	2
Group 2:	13	0
Group 3:	13	0
Group 4:	13	0
Group 5:	13	0
Group 6:	13	0
Group 7:	13	0
Group 8:	13	0
Group 9:	13	0



Uncategorized

Isolate non-ASCII characters cont.

Samples:

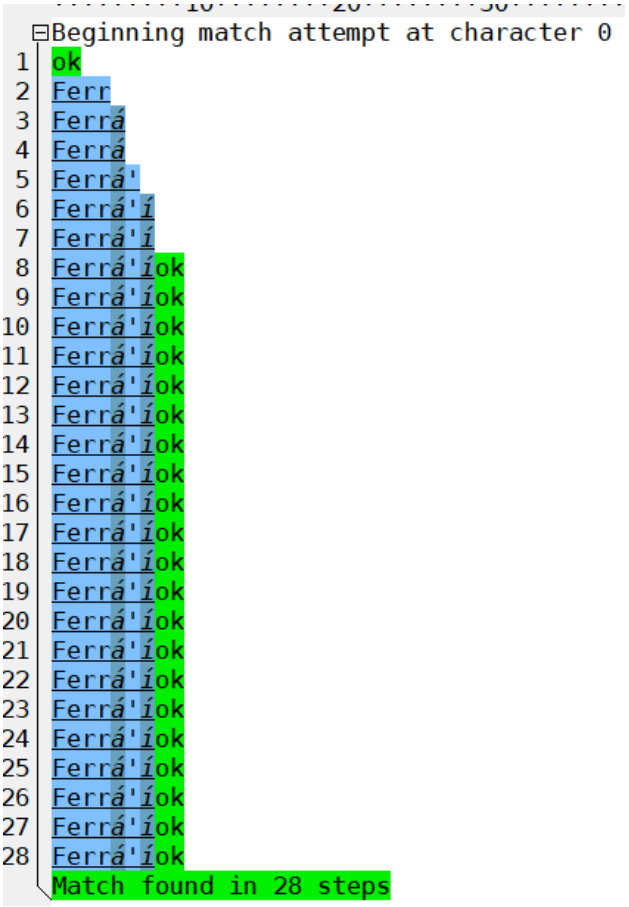
Amy Mußul (Mussul) éAmparo Muñoz
Bahá'í Muñoz Ferrá'í

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Sample:

	Start	Length
Match 1:	Ferrá'í	0 9
Group 1:	á	4 2
Group 2:	í	7 2
Group 3:		9 0
Group 4:		9 0
Group 5:		9 0
Group 6:		9 0
Group 7:		9 0
Group 8:		9 0
Group 9:		9 0



Uncategorized

Isolate non-ASCII characters cont.

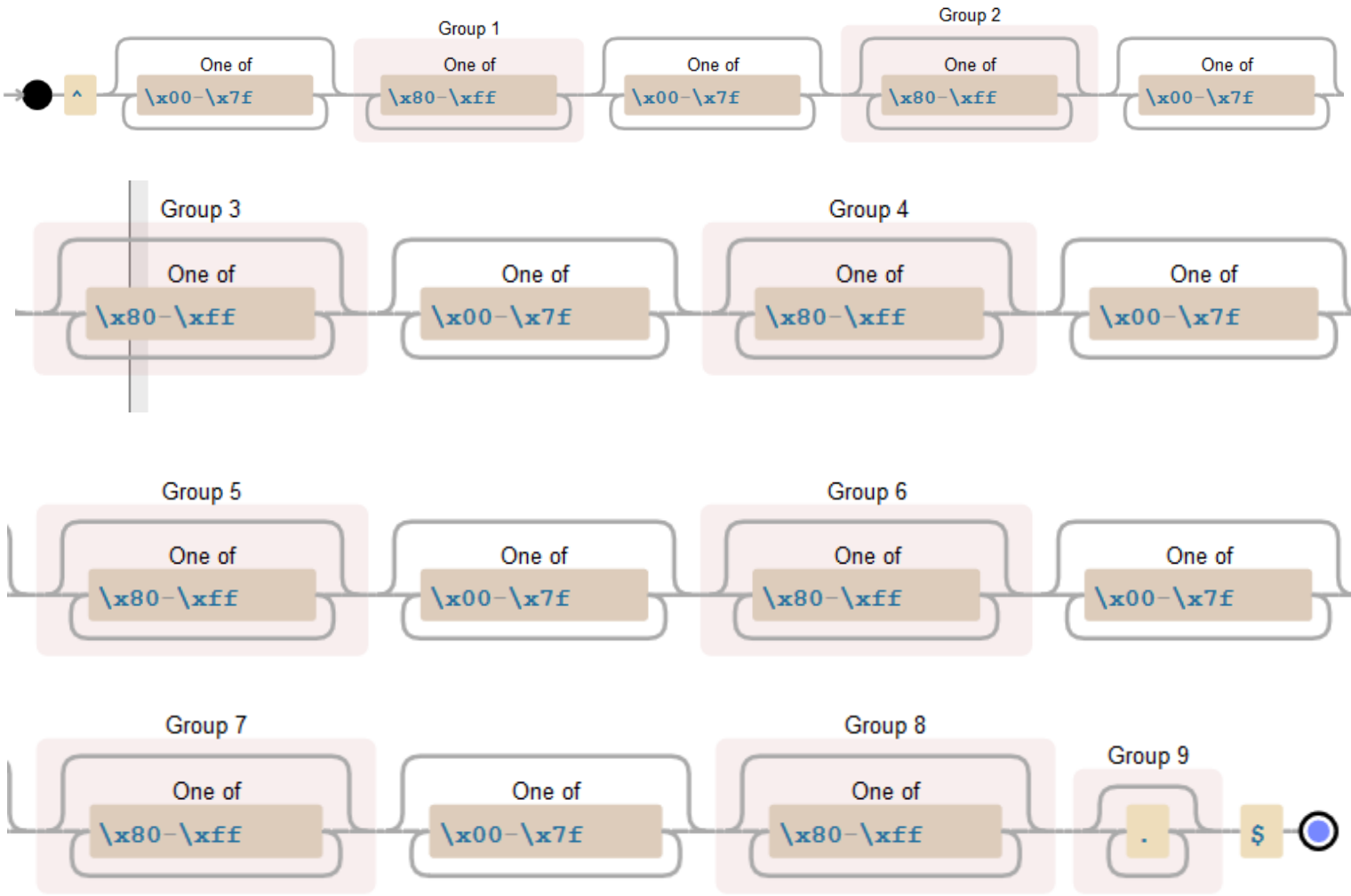
Samples:

Amy Mußul (Mussul)	éAmparo Muñoz
Bahá'í Muñoz	Ferrá'í

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Expanded:



Uncategorized

Isolate non-ASCII characters cont.

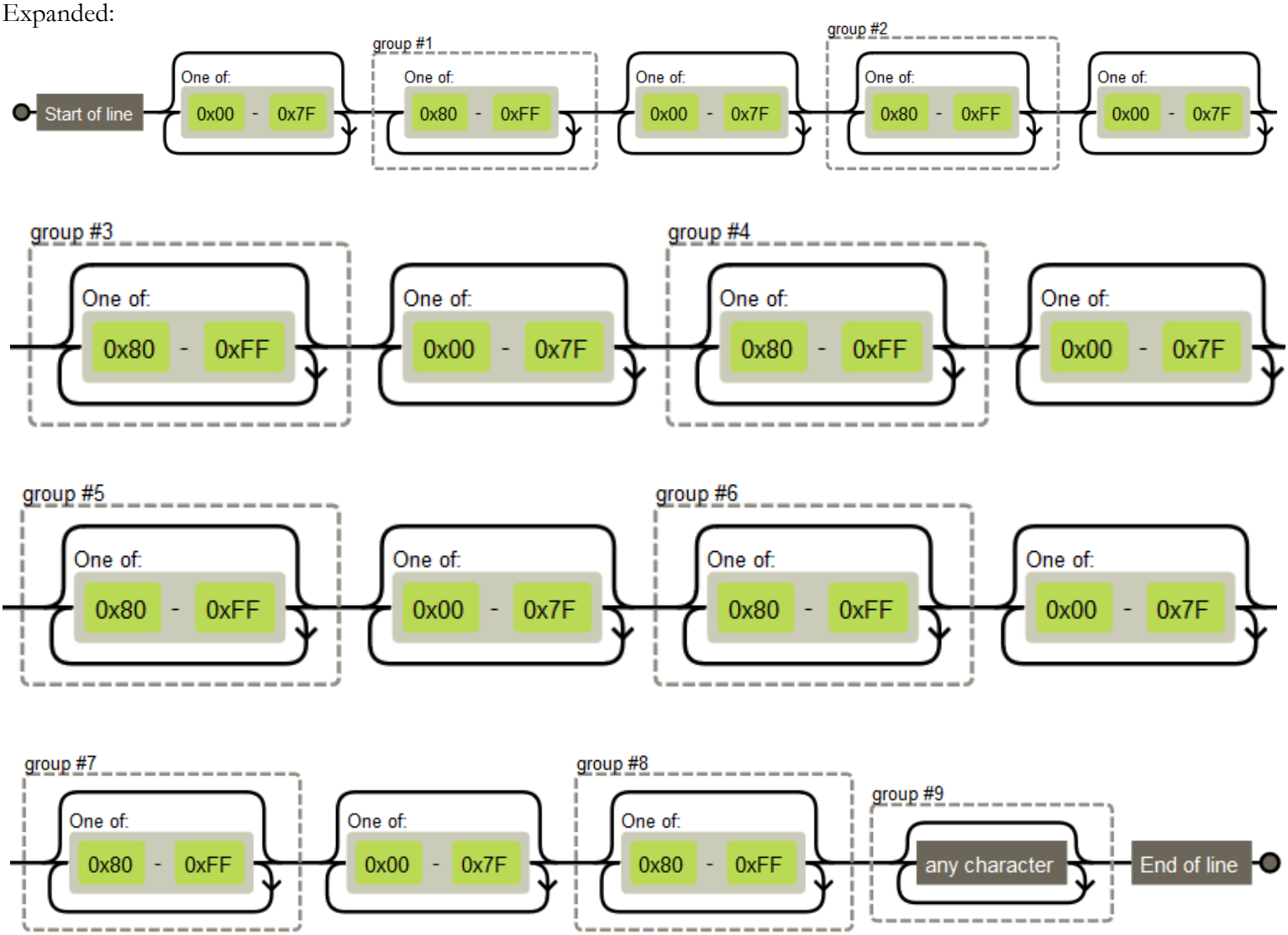
Samples:

Amy Mußul (Mussul)	éAmparo Muñoz
Bahá'í Muñoz	Ferrá'í

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Expanded:



Forum Contributor: RegexNinja

Uncategorized

Isolate non-ASCII characters cont.

Using: Bahá'í Muñoz

This will find all occurrences: cont.

Match:

```
^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$
```

Breaks down:

Capture Groups

^

[\x00-\x7f]*	Repetition	Not Captured	Bah
([\x80-\xff]+)		1	á
[\x00-\x7f]*	Repetition	Not Captured	'
([\x80-\xff]*)		2	í
[\x00-\x7f]*	Repetition	Not Captured	<space> Mu
([\x80-\xff]*)		3	ñ
[\x00-\x7f]*	Repetition	Not Captured	oz
([\x80-\xff]*)		4	null
[\x00-\x7f]*	Repetition	Not Captured	No Value
([\x80-\xff]*)		5	null
[\x00-\x7f]*	Repetition	Not Captured	No Value
([\x80-\xff]*)		6	null
[\x00-\x7f]*	Repetition	Not Captured	No Value
([\x80-\xff]*)		7	null
[\x00-\x7f]*	Repetition	Not Captured	No value
([\x80-\xff]*)		8	null
(.*)		9	null

$

Uncategorized

Isolate non-ASCII characters cont.

Using: Bahá'í Muñoz

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Tim Analysis:

This looks a lot more difficult than it actually is.

Where:

[\x00-\x7f]* - 0 – 127 decimal (printable characters ASCII)

([\x80-\xff]*) - 128 – 255 decimal (high ASCII 8-bit)

1. Go through the string matching against the regular ASCII until it encounters a high ASCII, capture it move through the string again each time until there is no more Capture Groups or EOL which ever comes first.

2. Gather up any remaining text (.*)

3. Can capture up to 8 High ASCII characters in each run.

4. The RegEx can be run multiple times until the string is exhausted.

Tim Notes:

1. (.*) will only contain a value if there are more characters left in the string. New name will reflect this so multiple runs can be permitted to isolate more High ASCII.

Uncategorized

Isolate non-ASCII characters cont.

Using: Bahá'í Muñoz

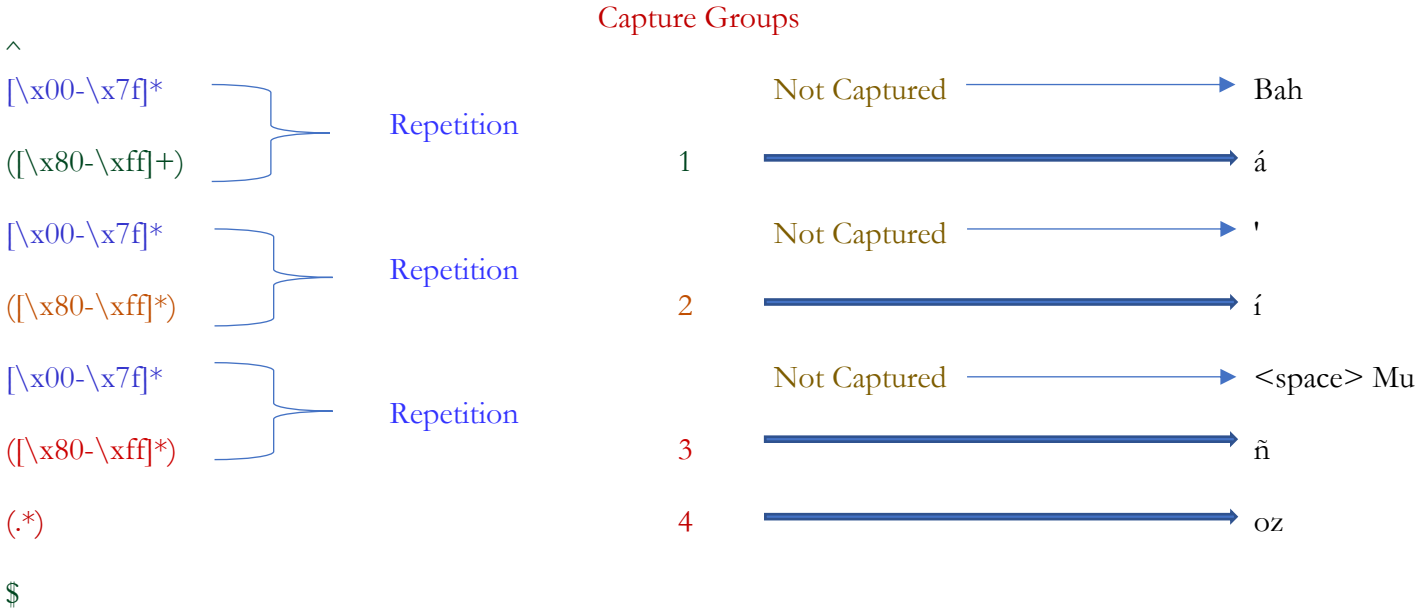
This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)(.*)$

Modifications:

1. Since the RegEx can be run multiple times, you can simplify it by removing any repetitive sequences that are excessive to your needs.

   For example, the RegEx is set up for eight captures, but if you only need a maximum of three captures you can simplify it down to:



And adjust the Replace String accordingly:

Match:

^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)[\x00-\x7f]*([\x80-\xff]*)

Replace:

\1\2\3\4

Uncategorized

Isolate non-ASCII characters cont.

Using: Bahá'í Muñoz

Using only two capture groups:

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)(.*)$
Replace: \1\2\3

First run:

	New Name
Amy Mußul (Mussul)	ß
Bahá'í Muñoz	áí Muñoz
Ferrá'í	áí
éAmparo Muñoz	éñoz

This captures characters 1-2.

The (.*) captures the remainder of the string so the next run can continue processing for characters 3 – 4.

In the above example, this includes:

áí Muñoz
éñoz

Second run:

Name	New Name
éñoz	éñ
áí Muñoz	áíñoz
áí	áí
ß	ß

éAmparo Muñoz string exhausted.
Bahá'í Muñoz remains.

Uncategorized

Isolate non-ASCII characters cont.

Using: Bahá'í Muñoz

This will find all occurrences: cont.

Match: ^[\x00-\x7f]*([\x80-\xff]+)[\x00-\x7f]*([\x80-\xff]*)(.*)$
Replace: \1\2\3

Using only two capture groups: cont.

Third run:

Name	New Name
éñ	éñ
áíñoz	áíñ
áí	áí
ß	ß

After third run - all strings are exhausted and no more matches can be made:

Name	New Name
éñ	éñ
áíñ	áíñ
áí	áí
ß	ß

Uncategorized

Isolate non-ASCII characters cont.

Using: 1ñ2ñ3ñ4ñ5ñ6ñ7ñ8ñ <space>

This will find all occurrences: cont.

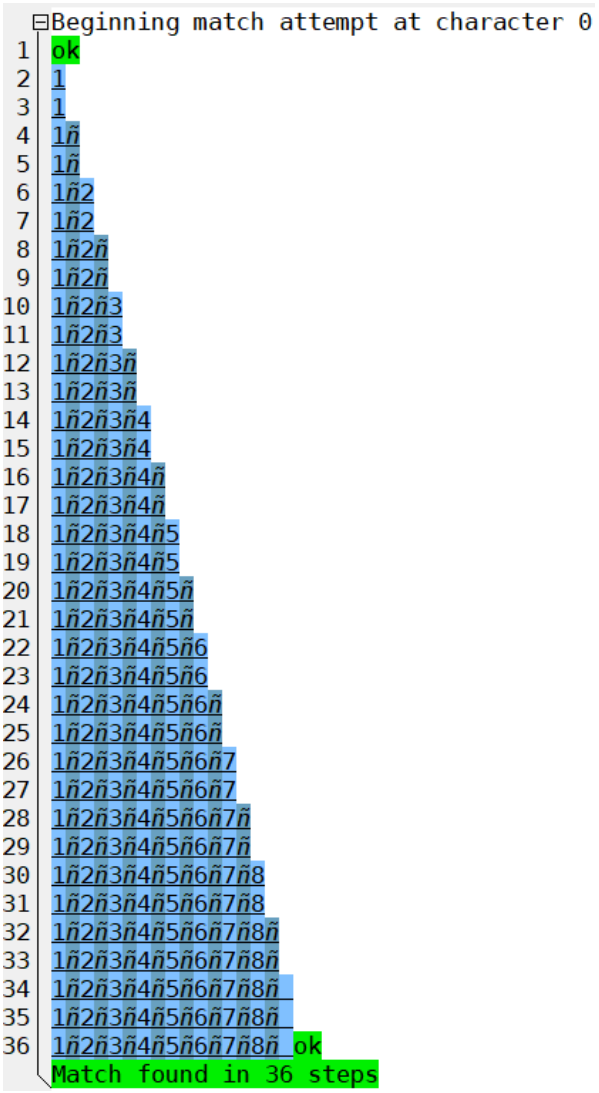
Here is an Atomic version by same Contributor:

Match:

```
^(?>[\x00-\x7f]*)([\x80-\xff]+)(?>[\x00-\x7f]*)([\x80-\xff]*)(?>[\x00-\x7f]*)([\x80-
\xff]*)(?>[\x00-\x7f]*)([\x80-\xff]*)(?>[\x00-\x7f]*)([\x80-\xff]*)(?>[\x00-\x7f]*)([\x80-
\xff]*)(?>[\x00-\x7f]*)([\x80-\xff]*)(?>[\x00-\x7f]*)([\x80-\xff]*)(.*)$
```

Sample:

	Start	Length
Match 1: 1ñ2ñ3ñ4ñ5ñ6ñ7ñ8ñ	0	25
Group 1: ñ	1	2
Group 2: ñ	4	2
Group 3: ñ	7	2
Group 4: ñ	10	2
Group 5: ñ	13	2
Group 6: ñ	16	2
Group 7: ñ	19	2
Group 8: ñ	22	2
Group 9:	24	1



Tim Notes:

1. The only difference is that Atomic Groups (Non-Marking) Are used for each of the 0-127 decimal matches.

2. This just demonstrates another method to accomplish the same goal. This is a popular theme in RegEx – providing different solutions.

3. Normally Atomic Groups are used to prevent or eliminate Backtracking, but the original RegEx is already very efficient.

4. For more information refer to Atomic Groups in the RegEx Manual Volume II found in the Appendix.

Uncategorized

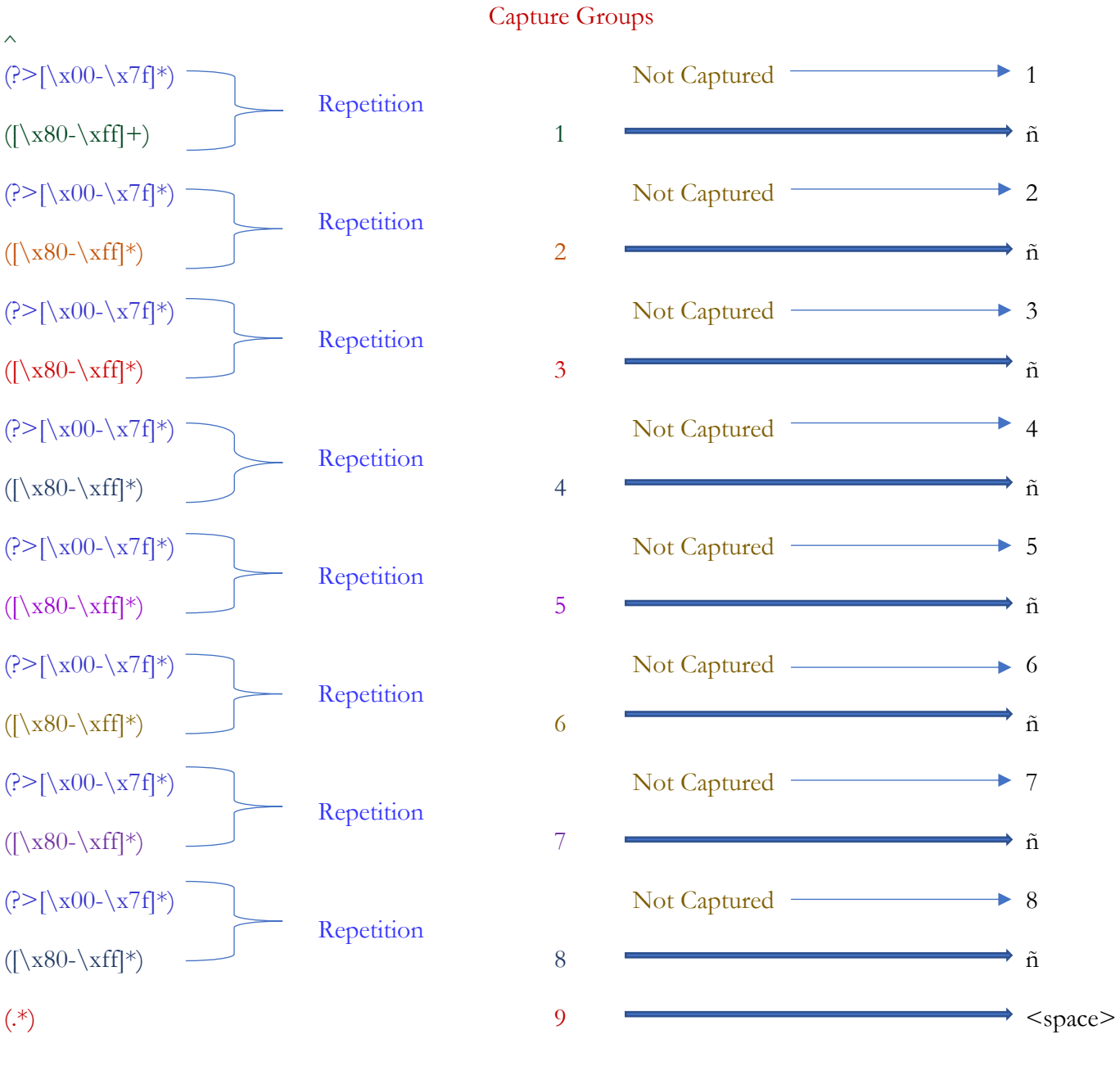
Isolate non-ASCII characters cont.

Using: 1ñ2ñ3ñ4ñ5ñ6ñ7ñ8ñ <space>

This will find all occurrences: cont.

Here is an Atomic version by same Contributor: cont.

Breaks down:



Uncategorized

Isolate non-ASCII characters cont.

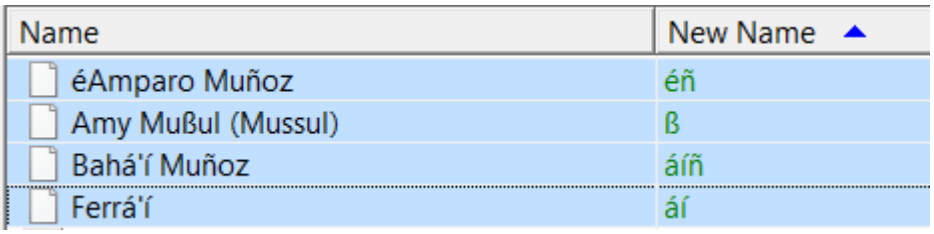
Samples:

Amy Mußul (Mussul) éAmparo Muñoz
Bahá'í Muñoz Ferrá'í

This will find all occurrences: cont.

Here is a JavaScript version by same Contributor:

```
newName=name.replace(/[\x00-\x7f]/g,'')
```



Contributor Notes:

1. This will only match files with at least one 8-bit ASCII (High ASCII decimal 128-255)

2. JavaScipt RegEx supports Global, so its simple.. Just replace 'lower-ASCII's (0-127 decimal) with nothing.

Tim Notes:

1. Global refers to the Global switch /g that allows all matches within a string to be performed rather than limited to just the first match. BRU RegEx and BRC do not currently support a Global switch, so they are limited to the first single match.

   BRC does, however, allow more than one RegEx to be run one after the other, so even though it does not support a Global switch, it can, in fact, match multiple times by running the same RegEx repeatedly within the same single BRC statement. The technique has been illustrated several times in the book. Search the PDF for 'max10' for an example.

   Not only that but RegEx' can be run prior to the BRC statement that will perform a rename in memory only, that will in a sense, set up a format for the final actual renaming process. This technique is illustrated several times in the book. For more information refer to these examples – search the PDF for BRC since I have no page references at this time.

   Unfortunately at this time, BRC does not support High ASCII, although both BRU and JavaScript do.

2. For a listing of High ASCII characters refer to the ASCII charts found in the Appendix.

Uncategorized

Isolate non-ASCII characters cont.

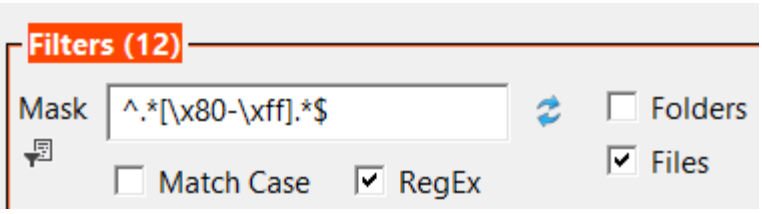
Samples:

Amy Mußul (Mussul) éAmparo Muñoz
Bahá'í Muñoz Ferrá'í

This will find all occurrences: cont.

Another method:

1. 'Section #12: Filters' –

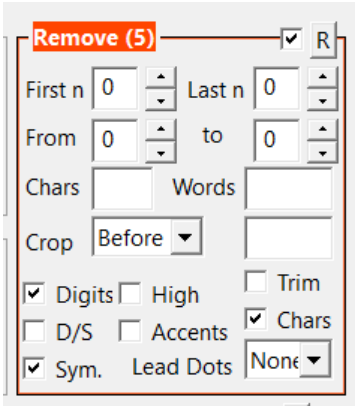


Mask = ^.*[\x80-\xff].*$
RegEx: Enabled

Click on the Filter Refresh button (blue button next to Mask field) or press F5

Name	New Name
éAmparo Muñoz	éAmparo Muñoz
Ferrá'í	Ferrá'í
Bahá'í Muñoz	Bahá'í Muñoz

2. 'Section #5: Remove' –



Digits: Enabled:
Sym: Enabled
Chars: Enabled
Chars = <space>

Name	New Name
éAmparo Muñoz	éñ
Ferrá'í	áí
Bahá'í Muñoz	áíñ

Uncategorized

More Downloaded TV Shows

Samples:

Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-.mp4
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.AAC.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv

To:

Poster wants the following – can't all be done:

Breaking Bad S02E13 720p.mp4	(Missing Episode title)
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.mp4	
30.rock.s03e08.flu.shot.720p.mkv	
Dexter S02 E11.mkv	(Missing Episode title, no 720p)

Observations:

1. Pattern is <name> <Season SS> <Episode EE> <720p>
2. Some shows have title of episode, some do not. Ie. , Breaking bad and Dexter.
3. Dexter does not follow any pattern, therefore will have to be dealt with separately.

Match: (.+)[Ss]\d\d\s*[Ee]\d\d
Replace: \1

Name	New Name
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.A...	Marvel.Agents.of.S.H.I.E.L.D..mp4
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv	Dexter .mkv
Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-....	Breaking Bad .mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv	30.rock..mkv

Not close to what the poster wanted.

Sample:

```
.............................
Match 1: Breaking Bad S02E13
Group 1: Breaking Bad
```

Sample:

```
........................
Match 1: Dexter S02 E11
Group 1: Dexter
```

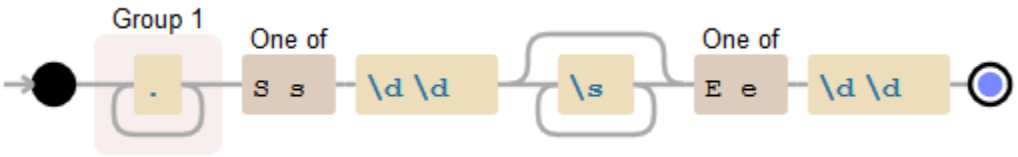
Sample:

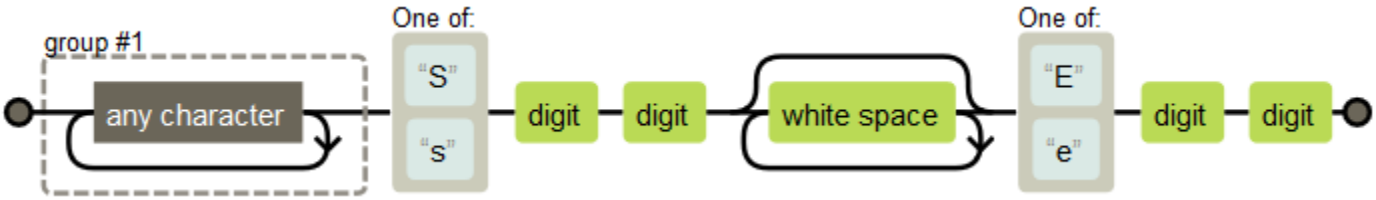
```
.........................................
Match 1: Marvel.Agents.of.S.H.I.E.L.D.S03E17
Group 1: Marvel.Agents.of.S.H.I.E.L.D.
```

Uncategorized

More Downloaded TV Shows cont.

Match: (.+)[Ss]\d\d\s*[Ee]\d\d





Forum Contributor: therube

Using: Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.AAC.mp4

Match: (.+)[Ss]\d\d\s*[Ee]\d\d

Where:

\1 = Marvel.Agents.of.S.H.I.E.L.D.

Tim Analysis:

1..	Match against any character	Capture Group 1 =
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [Ss]	Match against a class consisting of an 'S' or 's' (Not Captured) = 'S' This is the 'S' of 'S03E17'	Changes Capture Group 1 = Marvel.Agents.of.S.H.I.E.L.D.
4. \d\d	Match against 2 numeric digits (Not Captured) = '03'	
5. \s	Match against a <space> character = False (Not Captured)	
6. *	Make it Greedy. Restores Not Captured = '03'	
7. [Ee]	Match against a class consisting of an 'E' or 'e' (Not Captured) = 'E' Not Captured now = '03E'	
8. \d\d	Match against 2 numeric digits (Not Captured) = '17'	

Uncategorized

More Downloaded TV Shows cont.

Samples:

Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-.mp4
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.AAC.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv

To:

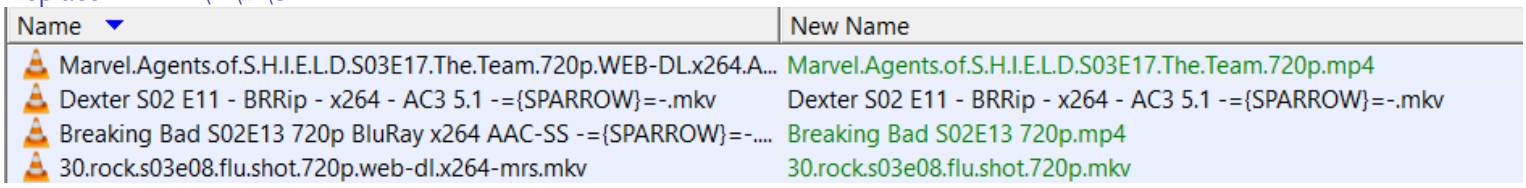
Poster wants the following – can't all be done:

Breaking Bad S02E13 720p.mp4	(Missing Episode title)
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.mp4	
30.rock.s03e08.flu.shot.720p.mkv	
Dexter S02 E11.mkv	(Missing Episode title, no 720p)

Tim Contribution –

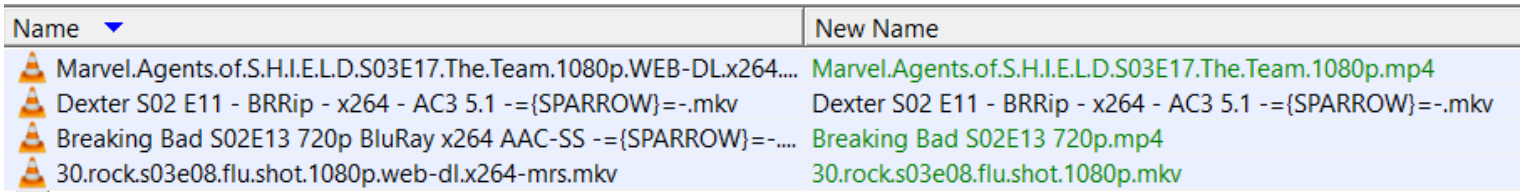
Here is a better solution – just a minor modification of therube's solution, but that only covered the pattern of the 720p and therefore Dexter did not qualify, however this is the closest that will come to what the poster wants (not including Dexter):.

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(.*\dp)
Replace: \1\2\3



Name	New Name
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.A...	Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.mp4
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv	Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv
Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-....	Breaking Bad S02E13 720p.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv	30.rock.s03e08.flu.shot.720p.mkv

This also works for 1080p (any number with p) as well:



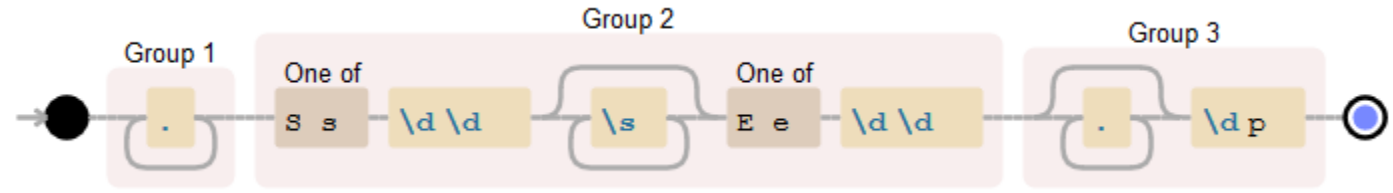
Name	New Name
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.1080p.WEB-DL.x264....	Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.1080p.mp4
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv	Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv
Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-....	Breaking Bad S02E13 720p.mp4
30.rock.s03e08.flu.shot.1080p.web-dl.x264-mrs.mkv	30.rock.s03e08.flu.shot.1080p.mkv

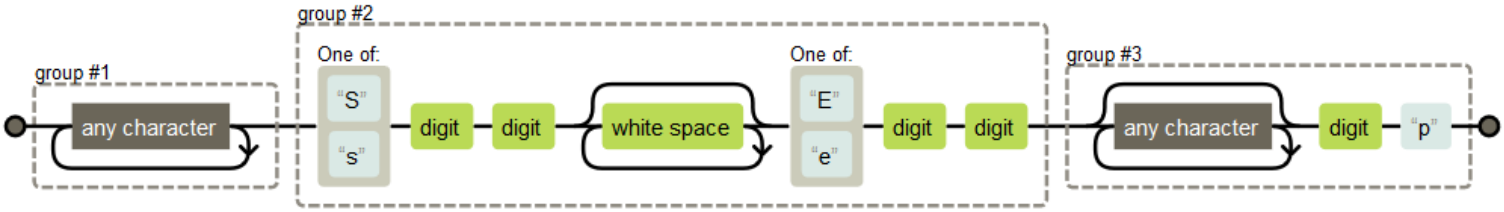
Uncategorized

More Downloaded TV Shows cont.

Tim Contribution – cont.

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(.*\dp)





Sample:

```
........................................
Match 1: 30.rock.s03e08.flu.shot.720p
Group 1: 30.rock.
Group 2: s03e08
Group 3: .flu.shot.720p
```

Uncategorized

More Downloaded TV Shows cont.

Tim Contribution – cont.

Using: 30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(.*\dp)

Where:

\1 = 30.rock. \2 = s03e08 \3 = .flu.shot.720p

Tim Analysis:

1..	Match against any character	Capture Group 1 = 3
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [Ss]	Match against a class consisting of an 'S' or 's'. Backtracked to match the 's' of 'mrs' Current position = EOL	Capture Group 2 = s Changes Capture Group 1 = 30.rock.s03e08.flu.shot.720p.web-dl.x264-mr
4. \d\d	Match against 2 numeric digits Changes the 's' of step 3 to the 's' of 's03e08'	Capture Group 2 = s03 Changes Capture Group 1 = 30.rock.

Tim Notes:

1. Can't move forward. Already at EOL (step 2). Already backtracked to match the 's' of 'mrs' (step 3), so the RegEx Engine returns to the beginning to reevaluate and matches the '03' of 's03e08' (step 4). This in turn changes the 's' that was captured in step 3 from the 's' of 'mrs' to the 's' of 's03e08'.

5. \s	Match against a <space> character No Match – there no <spaces> in this string.	All Capture Groups = No Value
6. *	Make it Greedy. Current position is the '3' of '03'	All Capture Groups = Restored
7. [Ee]	Match against a class consisting of an 'E' or 'e'. This is the 'e' of 's03e08'	Capture Group 2 = s03e
8. \d\d	Match against 2 numeric digits	Capture Group 2 = s03e08
9. .	Match against any character This is the <dot> after 's03e08'	Capture Group 3 = <dot>
10. *	Make it Greedy Current position = EOL	Capture Group 3 = .flu.shot.720p.web-dl.x264-mrs
11. \d	Match against Numeric Digit Backtracks to match against the '4' of '264'	Capture Group 3 = .flu.shot.720p.web-dl.x264
12. p	Match against string 'p' Backtracks to match against the 'p' of '720p'	Capture Group 3 = .flu.shot.720p

Uncategorized

More Downloaded TV Shows cont.

Samples:

Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-.mp4
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.AAC.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv

To:

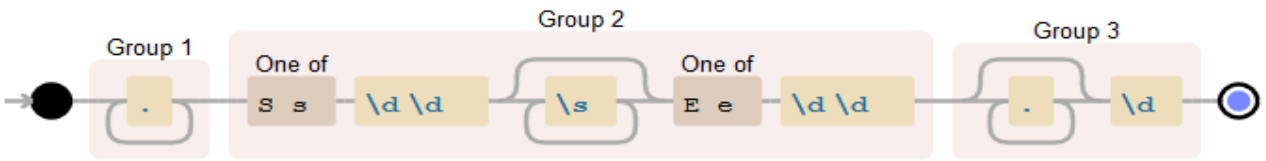
Breaking Bad S02E13 720p.mp4	(Missing Episode title)
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.mp4	
30.rock.s03e08.flu.shot.720p.mkv	
Dexter S02 E11.mkv	(Missing Episode title, no 720p)

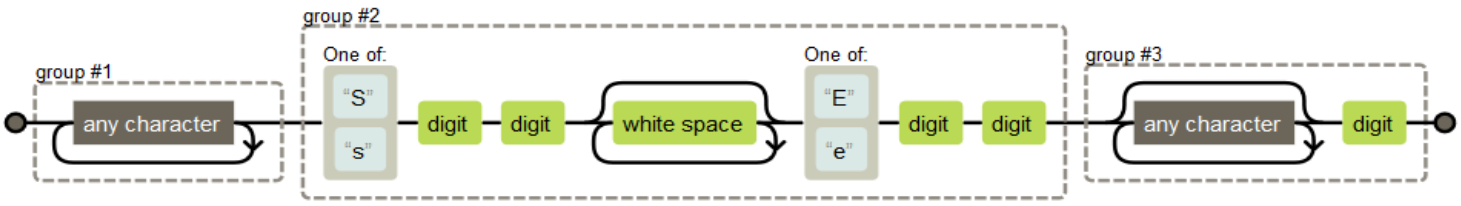
Tim Contribution – cont.

For inclusion of Dexter pattern or without the 720p or if you only want the name of the program and episode, use this:

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(.*\d)
Replace: \1\2

Name	New Name
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.A...	Marvel.Agents.of.S.H.I.E.L.D.S03E17.mp4
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv	Dexter S02 E11.mkv
Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-....	Breaking Bad S02E13.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv	30.rock.s03e08.mkv





Sample:

```
.....................................................
Match 1: Dexter S02 E11 - BRRip - x264 - AC3 5.1
Group 1: Dexter
Group 2: S02 E11
Group 3:  - BRRip - x264 - AC3 5.1
```

Poster cannot get all that is requested.

Uncategorized

More Downloaded TV Shows cont.

Tim Contribution – cont.

Using: Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(.*\d)
Where:

\1 = Dexter <space> \2 = s02 E11 \3 = <space> - BRRip - x264 - AD3 5.1

Tim Analysis:

1..	Match against any character	Capture Group 1 = D
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [Ss]	Match against a class consisting of an 'S' or 's'. This is the 'S' os "SPARROW' Current position = EOL	Capture Group 2 = S Changes Capture Group 1 = Dexter S02 E11 - BRRip - x264 <space> - AC3 5.1 -={
4. \d\d	Match against 2 numeric digits	Capture Group 2 = S02 Changes Capture Group 1 = Dexter <space>

Tim Notes:

1. Can't move forward. Already at EOL (step 2). Already backtracked to match the 'S' of 'SPARROW' (step 3), so the RegEx Engine starts at the beginning of the string and matches the '02' of 'S02 E11' (step 4). This in turn changes the 'S' that was captured in step 3 to the 'S' of 'S02 E11'.

5. \s	Match against a <space> character	Capture Group 2 = S02 <space>
6. *	Make it Greedy.	All Capture Groups = Unchanged
7. [Ee]	Match against a class consisting of an 'E' or 'e'. This is the 'E' of 'S02 E11'	Capture Group 2 = S02 E
8. \d\d	Match against 2 numeric digits	Capture Group 2 = S02 E11
9. .	Match against any character This is the <space> after 'E 11'	Capture Group 3 = <space>
10. *	Make it Greedy Current position = EOL	Capture Group 3 = <space> - BRRip - x264 - AC3 5.1 – ={SPARROW}=-
11. \d	Match against Numeric Digit Backtracks to match against the '1' of '5.1'	Capture Group 3 = <space> - BRRip - x264 - AC3 5.1

Uncategorized

More Downloaded TV Shows cont.

Samples:

Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-.mp4
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.AAC.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv

To:

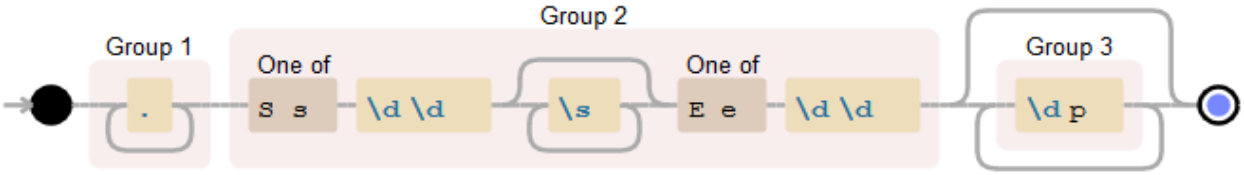
Breaking Bad S02E13 720p.mp4	(Missing Episode title)
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.mp4	
30.rock.s03e08.flu.shot.720p.mkv	
Dexter S02 E11.mkv	(Missing Episode title, no 720p)

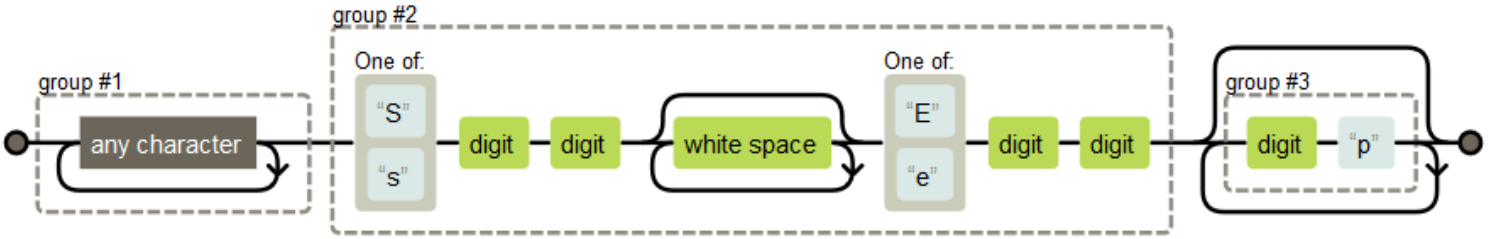
Tim Contribution – cont.

Or this:

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(\dp)*
Replace: \1\2

Name	New Name
Marvel.Agents.of.S.H.I.E.L.D.S03E17.The.Team.720p.WEB-DL.x264.A...	Marvel.Agents.of.S.H.I.E.L.D.S03E17.mp4
Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv	Dexter S02 E11.mkv
Breaking Bad S02E13 720p BluRay x264 AAC-SS -={SPARROW}=-....	Breaking Bad S02E13.mp4
30.rock.s03e08.flu.shot.720p.web-dl.x264-mrs.mkv	30.rock.s03e08.mkv





Sample:

```
.........................Start·Length····
Match 1: Dexter S02 E11       0      14
Group 1: Dexter               0       7
Group 2: S02 E11              7       7
Group 3 did not participate in the match
```

Poster can't get all that is requested.

Uncategorized

More Downloaded TV Shows cont.

Using: Dexter S02 E11 - BRRip - x264 - AC3 5.1 -={SPARROW}=-.mkv

Tim Contribution – cont.

Or this: cont.

Match: (.+)([Ss]\d\d\s*[Ee]\d\d)(\dp)*

Where:

\1 = Dexter <space> \2 = s02 E11 \3 = No Value

Tim Analysis:

1..	Match against any character	Capture Group 1 = D
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. [Ss]	Match against a class consisting of an 'S' or 's'. This is the 'S' os "SPARROW' Current position = EOL	Capture Group 2 = S Changes Capture Group 1 = Dexter S02 E11 - BRRip - x264 <space> - AC3 5.1 -={
4. \d\d	Match against 2 numeric digits	Capture Group 2 = S02 Changes Capture Group 1 = Dexter <space>

Tim Notes:

1. Can't move forward. Already at EOL (step 2). Already backtracked to match the 'S' of 'SPARROW' (step 3), so the RegEx Engine starts at the beginning of the string and matches the '02' of 'S02 E11' (step 4). This in turn changes the 'S' that was captured in step 3 to the 'S' of 'S02 E11'.

5. \s	Match against a <space> character	Capture Group 2 = S02 <space>
6. *	Make it Greedy.	All Capture Groups = Unchanged
7. [Ee]	Match against a class consisting of an 'E' or 'e'. This is the 'E' of 'S02 E11'	Capture Group 2 = S02 E
8. \d\d	Match against 2 numeric digits	Capture Group 2 = S02 E11
9. .	Match against any character This is the <space> after 'E 11'	Capture Group 3 = <space>
10. *	Make it Greedy Current position = EOL	Capture Group 3 = <space> - BRRip - x264 - AC3 5.1 – ={SPARROW}=-
11. \d	Match against Numeric Digit Backtracks to match against the '1' of '5.1'	Capture Group 3 = <space> - BRRip - x264 - AC3 5.1
12. p	Match against the string 'p' No Match – there no 'p' in this string.	Capture Group 3 = No Value
13. *	Make the (.*\dp) expression Greedy	Restores Capture Groups 1- 2

Uncategorized

Separate Run-On Sentences with spacing #1

IFeelTheEarthMove.mp3 to I Feel The Earth Move.mp3

Observations:

1. All words are uppercase

The original RegEx did not work at all so…

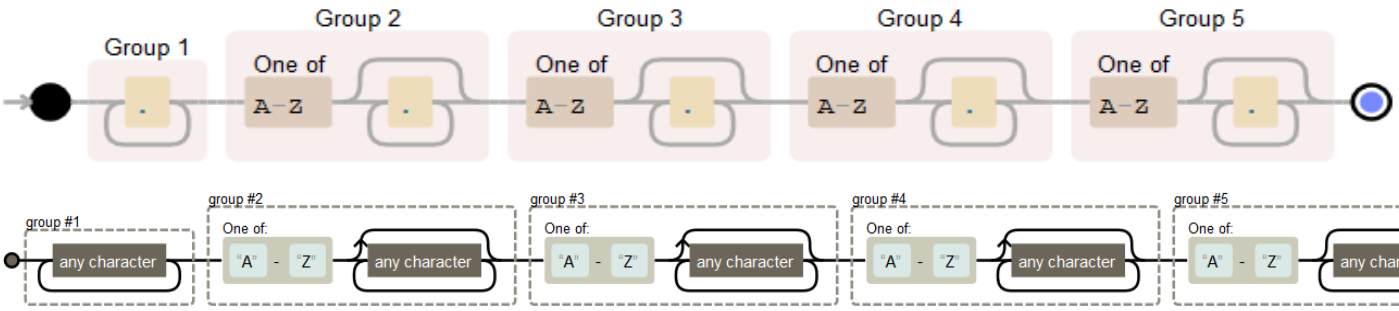
Tim Contribution –

Method #1: RegEx

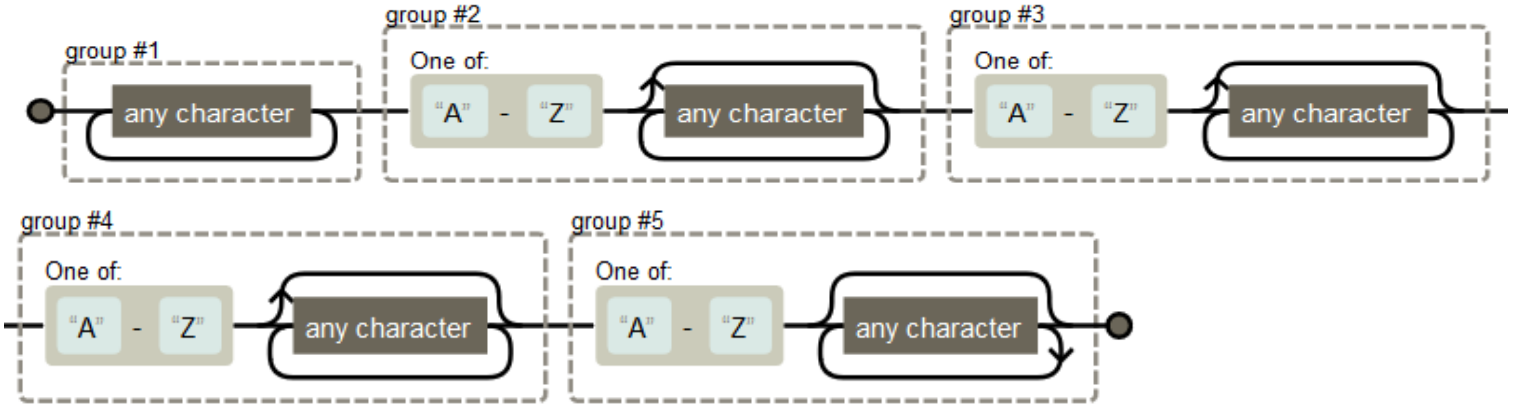
This is a variation of the RegEx I employed under, 'Separate Run on Names, Remove extraneous hyphens…'
Each Capture Group captures a full part of each word:

Match: (.+?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*)
Replace: \1 \2 \3 \4 \5

Name	New Name
IFeelTheEarthMove.mp3	I Feel The Earth Move.mp3



Expanded:



Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3 to I Feel The Earth Move.mp3

Tim Contribution – cont.

Method #1: RegEx cont.

Match: (.+?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*)

Sample:

```
.............................
Match 1: IFeelTheEarthMove
Group 1: I
Group 2: Feel
Group 3: The
Group 4: Earth
Group 5: Move
```

This works but it is very limited. It will only work on strings that consist of 5 words. Sure it can be adapted for up to a total of 9 words (the total number of Capture Groups) but it is too specific - just add an additional ([A-Z].*?) before the ([A-Z].*) at the end for each additional word.

Whenever you have a Parsing such as this, the RegEx looks intimidating but it is actually not. It is repetitive in nature. So if you break it down into its repetitive groups you can better analyze how it works or doesn't work.

Breaks down:

(.+?)	Capture Group 1	
([A-Z].*?)	Capture Group 2	Repetitive Capture Groups
([A-Z].*?)	Capture Group 3	
([A-Z].*?)	Capture Group 4	
([A-Z].*)	Capture Group 5	

Looks simpler doesn't it?

From this I can see that the first expression is unique followed by 3 identical expressions followed by a unique last expression.

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

Tim Contribution – cont.

IFeelTheEarthMove.mp3 to I Feel The Earth Move.mp3

Match: (.+?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*)

Where:

\1 = I \2 = Feel \3 = The \4 = Earth \5 = Move

Tim Analysis:

1. .	Match against single character	Capture Group 1 = I
2. +	Make it Greedy	Capture Group 1= entire string to EOL
3. ?	But not too Greedy (Non-Greedy) until …	Capture Group 1= I
	RegEx engine is backtracking	
4. [A-Z]	.. a class consisting of an uppercase letter	Capture Group 2 = F
	RegEx engine moving forward again	Capture Group 1 = Unchanged
5. .	Match against a single character	Capture Group 2 = Fe
6. *	Make it Greedy	Capture Group 2 = FeelTheEarthMove
7. ?	But not too Greedy (Non-Greedy)	Capture Group 2 = F
	RegEx engine is backtracking	

Tim Notes:

1. Steps 4-7 make up the analysis of the repetitive expressions. The expression will repeat but the values will change.
2. It is recommended by RegEx Style Guide to begin with the ^ anchor to force the RegEx Engine to the start of the string, even when it is unnecessary and would match regardless. This is because when you tell the engine to find a complex pattern at a given location, it doesn't waste time trying to find that pattern at many other locations.
3. Capture Group 1 unchanged in step 4 because there are no lowercase letters after 'I'.

i.e.,

Capture Group 1 = F Currently -

Capture Group 1
F eel The

When the uppercase letter is matched [A-Z] it changes Capture Group 1…

Capture Group 1 2
Feel The

Capture Group 1 gathers up any characters up to the match of Capture Group 2

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

Tim Contribution – cont.

IFeelTheEarthMove.mp3 to I Feel The Earth Move.mp3

Match: (.+?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*)

Where:

\1 = I \2 = Feel \3 = The \4 = Earth \5 = Move

Tim Analysis: cont:

8. [A-Z]	.. a class consisting of an uppercase letter	Capture Group 3 = T
	RegEx engine moving forward again	Changes Capture Group 2 = Feel
9. .	Match against a single character	Capture Group 3 = Th
10. *	Make it Greedy	Capture Group 3 = TheEarthMove
11. ?	But not too Greedy (Non-Greedy)	Capture Group 3 = T
	RegEx engine is backtracking	
12. [A-Z]	.. a class consisting of an uppercase letter	Capture Group 4 = E
	RegEx engine moving forward again	Changes Capture Group 3 =The
13. .	Match against a single character	Capture Group 4 = Ea
14. *	Make it Greedy	Capture Group 4 = EarthMove
15. ?	But not too Greedy (Non-Greedy)	Capture Group 5 = E
	RegEx engine is backtracking	
16. [A-Z]		Capture Group 5 = M
		Changes Capture Group 4 = Earth
17. .*	Capture remainder of string.	Capture Group 5 = Move

Expression satisfied.

Tim Notes:

1. Because this is the end of the string, there is no need to separate out the ‘dot’ from the ‘*‘ in the analysis, but rather state ‘to capture remainder of string’. For this same reason the ‘?‘ lazy quantifier could not be added because this would change Capture Group 5 backtracked to ‘M’ instead of ‘Move’.

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

Tim Contribution – cont.

Tim Analysis: cont:

A Word on Backtracking

I illustrate how the RegEx engine moves back and forth through the string attempting to satisfy the evaluation.

Think of the engine as a train on a track – and the track is the string. It has a beginning and it has an end – the end is what is called EOL or end of line, or using BRU, it would probably be more appropriate to say, 'end of filename' or 'end of folder name', but it is EOL.

The EOL is right before the extension. (Unless a Rename option is used that would include the extension as part of the filename). If the expression has not been satisfied, the engine can still backtrack, but when the extension is reached, or the $ anchor is used in the RegEx to signify that the EOL must be met, the evaluation stops.

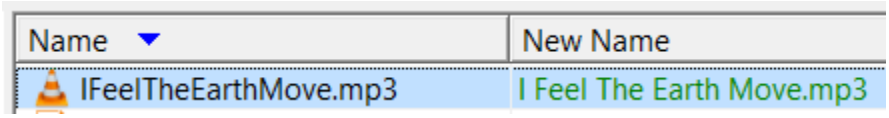
It is hard to predict what the engine does because it has a mind of its own it seems, at least to me – it will really try to satisfy the equation moving back and forth through the string. You can direct the train as is done here – using the Greedy quantifier * to go to EOL and then pull it back by making it Lazy using the Non-Greedy quantifier ?

But it is this back and forth that can slow down processing if you have a lot of files to rename. That is why possessive quantifiers are sometimes used in place of Greedy or Lazy quantifiers (for more on this refer to the RegEx Manual Volume I found in the Appendix of 'Volume I Bulk Rename Utility Operations Manual' and the RegEx Manual Volume II found in the Appendix of this book).

I am by no means an expert and these are my observations, not fact.

Method #2: 'Section #7: Add'

Simply set 'Word Space'



Forum Contributor: therube

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3 to I Feel The Earth Move.mp3

Method #3: JavaScript

```
newName = name.replace(/([A-Z])/g, ' $1').trim()
```

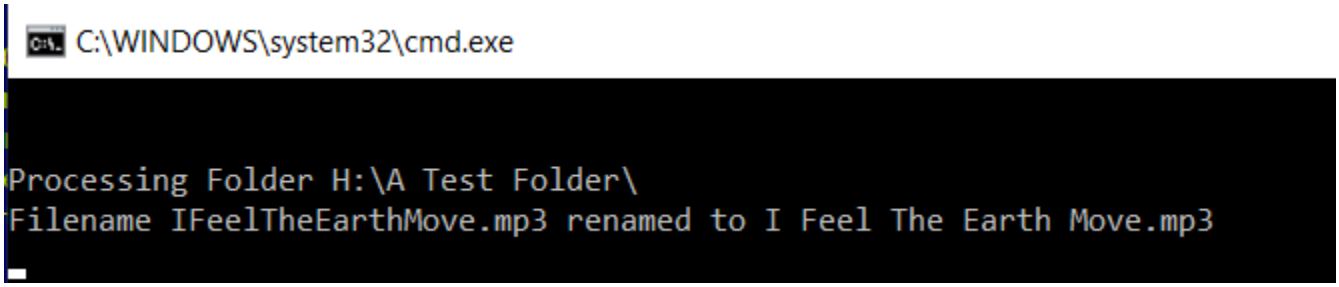
Name	New Name
IFeelTheEarthMove.mp3	I Feel The Earth Move.mp3

Forum Contributor: Admin

Method #4: BRC

Batch file:

```
@echo off
Set reg=/Regexp:"^(.[^ ]*)([A-Z].*):\1 \2"
Set max10=%reg% %reg% %reg% %reg% %reg% %reg% %reg% %reg% %reg% %reg%
brc64 /Dir:"YourFolderPath" /Pattern:"*.mp3" %max10% %max10% /Execute
pause>nul
```



Uncategorized

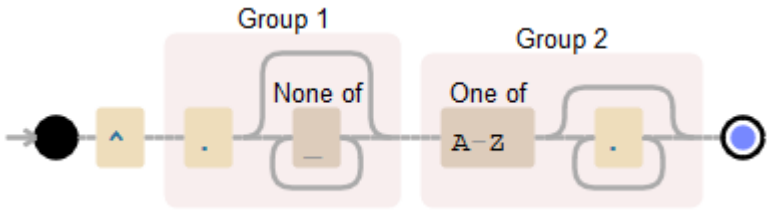
Separate Run-On Sentences with spacing #1 cont.

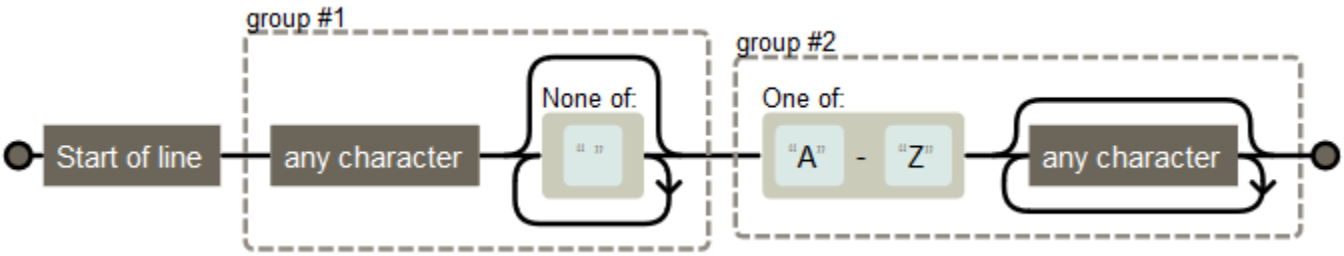
IFeelTheEarthMove.mp3 to I Feel The Earth Move.mp3

Method #4: BRC cont.

Breaks down to:

Match: ^(.[^]*)([A-Z].*)
Replace: \1 \2





Sample: first run

```
.............................
Match 1: IFeelTheEarthMove
Group 1: IFeelTheEarth
Group 2: Move
```

Dir: Changes directory to the directory containing the files to be renamed.

Tim Notes:

1. I prefer instead, i.e., ‘ cd/d "H:\Level1" ’ at the top of the batch file, for directory change

Contributor Notes:

1. BRC64 (or BRC32) executable must be in your path, if not must be explicitly stated, e.g. c:\BRU64\BRU64.exe
2. The regex just inserts <space> in between a final (NonSpace)(Uppercase).
3. BRC runs it 20 times, so as-written handles up to 20 'words'.. Throw in another %max10% for 30, etc.
4. You can remove /Execute to preview the results.

Forum Contributor: bru

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

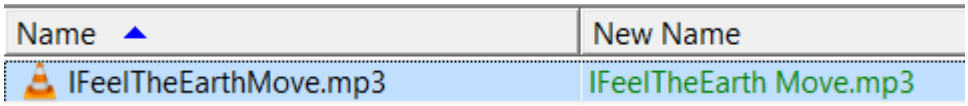
IFeelTheEarthMove.mp3

Match: ^(.[^]*)([A-Z].*)
Replace: \1 \2

Capture Group 1 Capture Group 2

(.[^]*) ([A-Z].*)

Run #1



Where:

\1 = IFeelTheEarth \2 = Move

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. .	Match against any character	Capture Group 1 = I
3. [^]	Negated Class. Ignore <space> character	Capture Group 1 = IF
4. *	Make it Greedy until…	Capture Group 1 = entire string to EOL
5. [A-Z]	.. Match against a class consisting of an uppercase letter	Capture Group 2 = M
	RegEx Engine is backtracking	Changes Capture Group 1= IFeelTheEarth
6. .*	Capture remainder of string.	Capture Group 2 = Move

By parsing out the string, it separates the beginning text from each word at the end for each subsequent run. The Replace string adds the <space>.

Tim Notes:

1. In a previous example, a run-on word group was hyphenated so the BRC RegEx 'locked on' to the <hyphen> character to separate the word groups moving forward, but here backtracking was used to capture the last word preceding the final <space> for each subsequent run using (.[^]*).

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

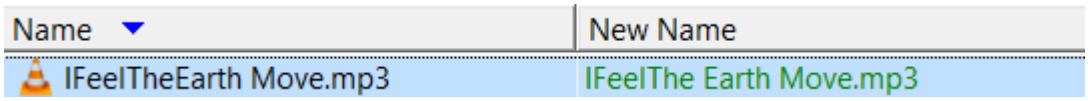
IFeelTheEarthMove.mp3

Match: ^(.[^]*)([A-Z].*)
Replace: \1 \2

Capture Group 1 Capture Group 2

(.[^]*) ([A-Z].*)

Run #2



Where:

\1 = IFeelThe \2 = Earth Move

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. .	Match against any character	Capture Group 1 = I
3. [^]	Negated Class. Ignore <space> character	Capture Group 1 = IF
4. *	Make it Greedy until…	Capture Group 1 = IFeelTheEarth
5. [A-Z]	.. Match against a class consisting of an uppercase letter	Capture Group 2 = E
	RegEx Engine is backtracking	Changes Capture Group 1= IFeelThe
6. .*	Capture remainder of string.	Capture Group 2 = Earth Move

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

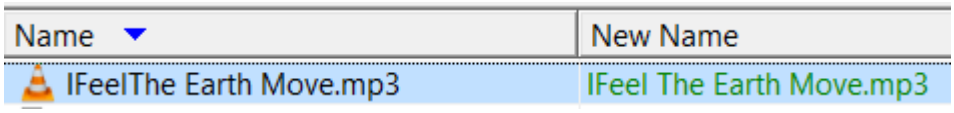
IFeelTheEarthMove.mp3

Match: ^(.[^]*)([A-Z].*)
Replace: \1 \2

Capture Group 1 Capture Group 2

(.[^]*) ([A-Z].*)

Run #3:



Where:

\1 = IFeel \2 = The Earth Move

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. .	Match against any character	Capture Group 1 = I
3. [^]	Negated Class. Ignore <space> character	Capture Group 1 = IF
4. *	Make it Greedy until…	Capture Group 1 = IFeelThe
5. [A-Z]	.. Match against a class consisting of an uppercase letter	Capture Group 2 = T
	RegEx Engine is backtracking	Changes Capture Group 1= IFeel
6. .*	Capture remainder of string.	Capture Group 2 = The Earth Move

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

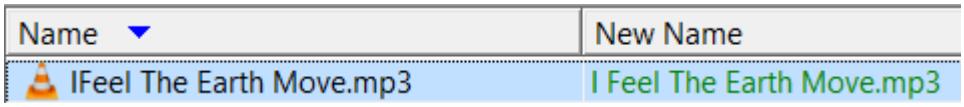
IFeelTheEarthMove.mp3

Match: ^(.[^]*)([A-Z].*)
Replace: \1 \2

Capture Group 1 Capture Group 2

(.[^]*) ([A-Z].*)

Run #4:



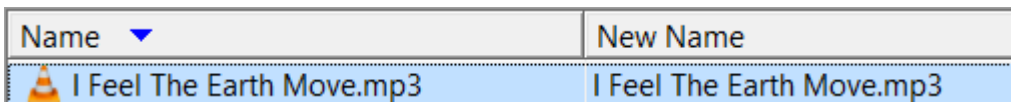
Where:

\1 =I \2 = Feel The Earth Move

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. .	Match against any character	Capture Group 1 = I
3. [^]	Negated Class. Ignore <space> character	Capture Group 1 = IF
4. *	Make it Greedy until…	Capture Group 1 = IFeel
5. [A-Z]	.. Match against a class consisting of an uppercase letter	Capture Group 2 = F
	RegEx Engine is backtracking	Changes Capture Group 1= I
6. .*	Capture remainder of string.	Capture Group 2 = Feel The Earth Move

String Exhausted – no more renames possible, RegEx Fails:



Uncategorized

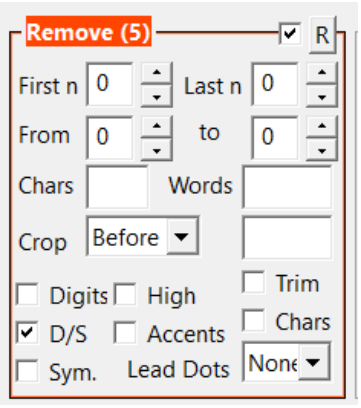
Separate Run-On Sentences with spacing #1 cont.

Finally, here's another RegEx that takes a slightly different approach from mine. It also uses repetitive groups.

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)(.*)

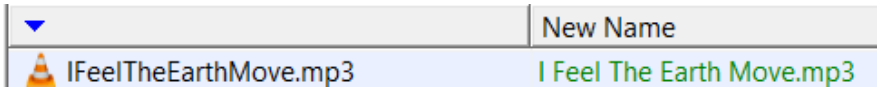
Replace: \1 \2 \3 \4 \5 \6 \7 \8 \9

'Section #5: Remove' -



D/S: Enable

Contributor and Tim Notes:
1. The RegEx will separate anywhere from 2-9 'words' (9 grabs everything-else)
2. The RegEx leaves extra <spaces> as a result of the replacement string
3. Requires enabling D/S in 'Section #5: Remove' to correct this.
4. The {0,1}-match protects against non-occurrences in Groups1-6 (for names of few words), the <spaces> are hard-coded. This is the same as the ' ? ' (optional) quantifier.



Tim Notes:

1. Restrictive, like mine. Also handles 9 words.
2. Unlike mine, where I can add or subtract the repetitive groups to tailor it to specific strings, because the <spaces> are hard coded into the RegEx, the entire RegEx has to be used.
3. Unlike mine it can handle 9 words without the tailoring. For example if my RegEx is used on a string with 5 words, I must use the exact RegEx that would match 5 words, otherwise the RegEx would fail.

e.g. This matches IFeelTheEarthMove: (.+?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*)
This will fail: (.+?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*?)([A-Z].*)

By adding the additional repetitive group ([A-Z].*?) at the end, the RegEx can now handle a six word string – BUT only a six word string. The contributor's is more flexible in this area.

Conclusion: There are sometimes tradeoffs that have to be made.

Mine does not require the D/S to remove excess <spaces>, but this contributor's does.
Mine MUST match the number of words in the string, the contributor's does not.
Both are limited by the number of capture groups allowed = 9

Again, it should also be noted that this only applies to using BRU, BRC and JavaScript do not have these limitations.

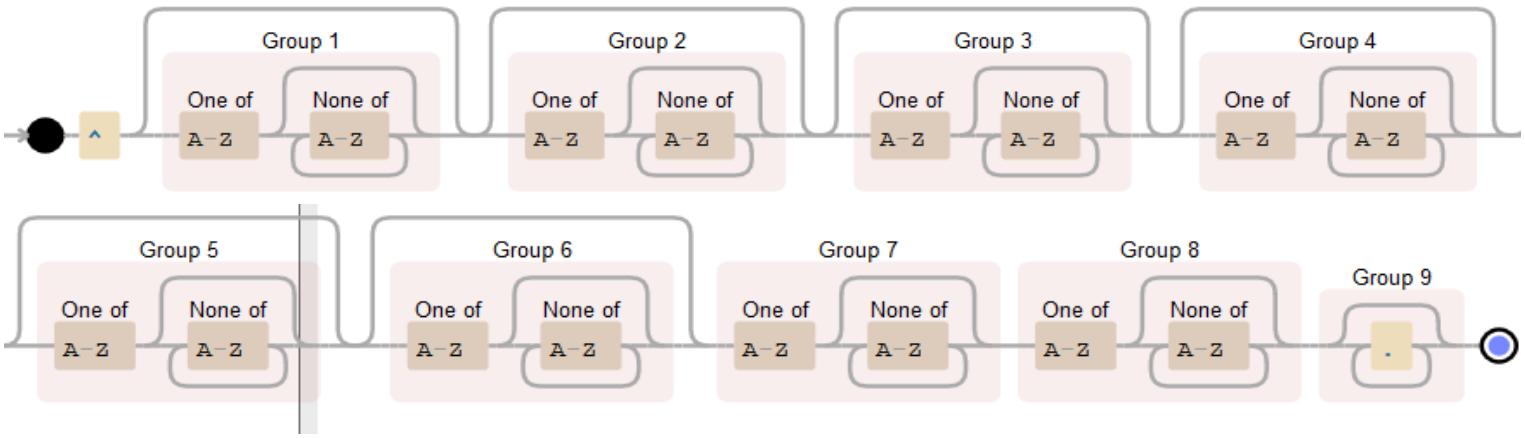
Forum Contributor: bru

Uncategorized

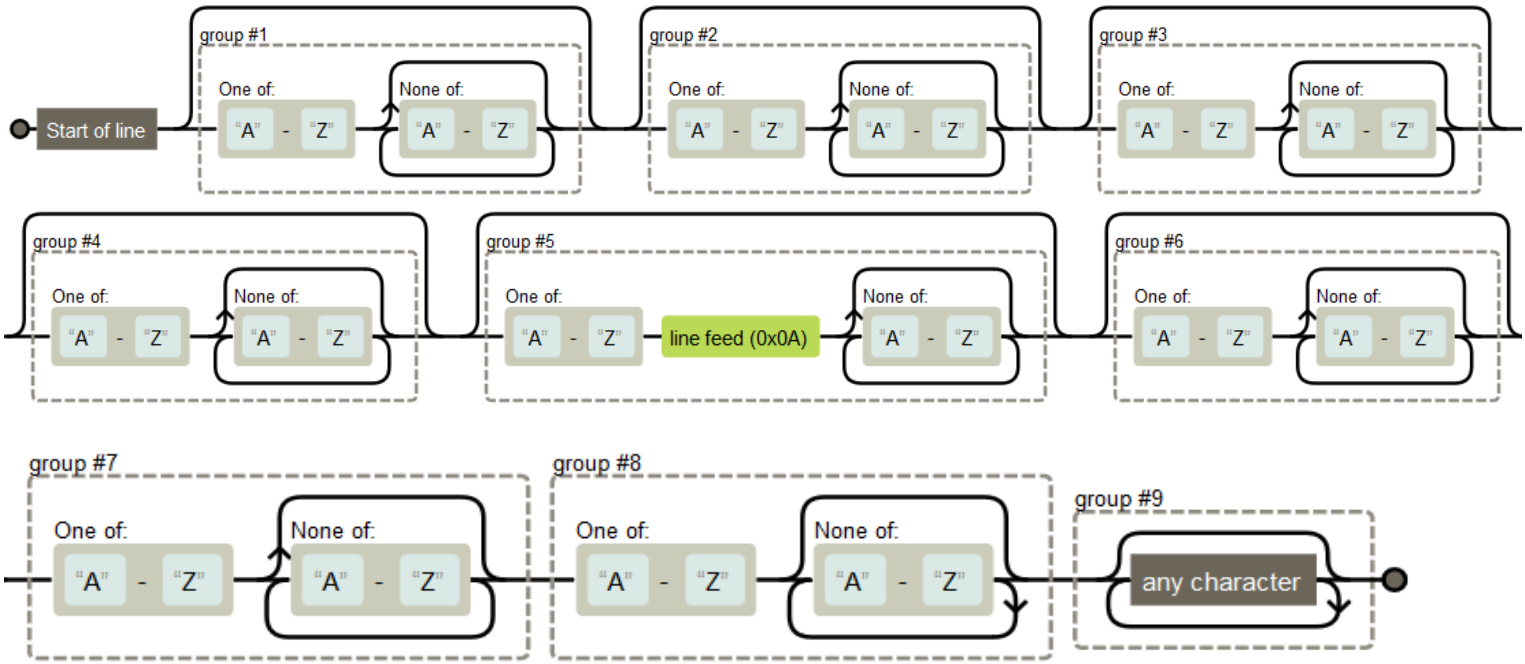
Separate Run-On Sentences with spacing #1 cont.

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)(.*)

Expanded:



Expanded:



Sample:

```
·····························Start·Length···
Match 1: IFeelTheEarthMove       0      17 CLRF
Group 1: I                       0       1 CLRF
Group 2: Feel                    1       4 CLRF
Group 3: The                     5       3 CLRF
Group 4 did not participate in the match CLRF
Group 5 did not participate in the match CLRF
Group 6 did not participate in the match CLRF
Group 7: Earth                   8       5 CLRF
Group 8: Move                   13       4 CLRF
Group 9:                        17       0
```

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)(.*)

Where:

\1 = I \2 = Feel \3 = The \4 = null \5 = null \6 = null

\7 = Earth \8 = Move \9 = null

Tim Analysis:

Breaks down:

^
([A-Z][^A-Z]*?){0,1} = Capture Group 1 + {0,1}
([A-Z][^A-Z]*?){0,1} = Capture Group 2 + {0,1}
([A-Z][^A-Z]*?){0,1} = Capture Group 3 + {0,1}
([A-Z][^A-Z]*?){0,1} = Capture Group 4 + {0,1}
([A-Z][^A-Z]*?){0,1} = Capture Group 5 + {0,1}
([A-Z][^A-Z]*?){0,1} = Capture Group 6 + {0,1}
([A-Z][^A-Z]*?) = Capture Group 7

Repetitive Capture Groups

([A-Z][^A-Z]*) = Capture Group 8
(.*) = Capture Group 9

Observations:

1. There are 7 repetitive expressions in the Capture Groups.
2. 6 of them have an added {minimum, maximum} Range Quantifier specification
3. Although I analyze each step, the expression of each group is looking for an uppercase letter followed by a lower case letter and if not, the value is null. At the end of each group the Quantifier {0,1) gives up the match (removes values) having not satisfied the expression, thus invalidating previous values. It isn't until Group 7, that matches are made beyond Group 1 and 2. My Analysis is nowhere perfect. Doing a piece by piece analysis, I may miss what the overall effect is, but sometimes you can't just look at the 'big' picture. You have to look at the parts. It may not be 100% accurate, but the values are, and if you understand that, then maybe later on you can think in terms of the 'whole'.

CAPTURE GROUP 1

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 1 =I
3. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine ignores the 'F'.	Capture Group 1 = RegEx stopped
4. *	Make it Greedy	Capture Group 1 = I
5. ?	But not too Greedy (Non-Greedy)	Capture Group 1 = Unchanged
6. {0,1}	Specify that the last expression, ([A-Z][^A-Z]*?) run a minimum of Zero to 1 time. Runs Zero.	Capture Group 1 = Unchanged

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)([A-Z][^A-Z]*)(.*)

Tim Analysis: cont.

Tim Notes:

1. Steps 2-5 make up the analysis of the repetitive expressions. The expression will repeat but the values will change as you will see.

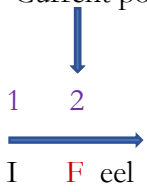
2. Step 6 is a Minimum, Maximum limitation on the match repetitions. By specifying this, the last expression will repeat itself based on the values herein. In this case, it is to run at least one time and at minimum not at all. For more information, refer to the RegEx Manual found in the Appendix of 'Volume I Bulk Rename Utility Operations Manual'.

CAPTURE GROUP 2

7. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 =F
8. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'e' and captures it.	Capture Group 2 = Fe Capture Group 1 = Unchanged = I
9. *	Make it Greedy	Capture Group 2 = Feel
10. ?	But not too Greedy (Non-Greedy) RegEx engine backtracking Current position = 'F' of 'Feel'	Changes Capture Group 2 = F Capture Group 1 = Unchanged = I

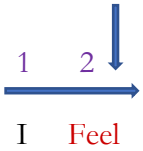
Visualize:

Step 7 - Current position is 'F' of 'Feel'

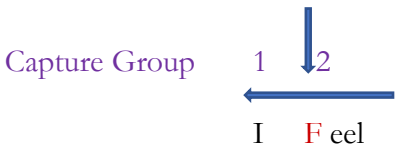


Step 8 - Current position is 'e' of 'Feel'

Step 9 - Current position is 'l' of 'Feel'



Step 10 - Current position is 'F' of 'Feel'



11. {0,1}	Specify that the last expression, ([A-Z][^A-Z]*?) run a minimum of Zero to 1 time. Runs Zero. Current position = 'F' of 'Feel'	Capture Group 2 = Unchanged = F Capture Group 1 = Unchanged = I

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)([A-Z][^A-Z]*)(.*)

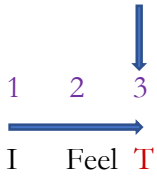
Tim Analysis: cont.

CAPTURE GROUP 3

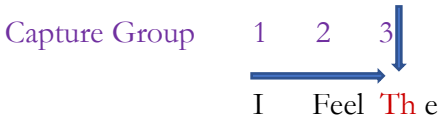
12. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 =T Changes Capture Group 2 = Feel Capture Group 1 = Unchanged = I
13. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'h' and captures it.	Capture Group 3 = Th
14. *	Make it Greedy	Capture Group 3 = The
15. ?	But not too Greedy (Non-Greedy) RegEx engine backtracking Current position = 'T' of 'The'	Changes Capture Group 3 = T
16. {0,1}	Specify that the last expression, ([A-Z][^A-Z]*?) Run a minimum of Zero to 1 time. Runs Zero. Current position = 'F' of 'Feel'	Capture Group 3 = \3 Invalid Changes Capture Group 2 = F Capture Group 1 = Unchanged = I

Visualize:

Step 12 - Current position is 'T' of 'The'

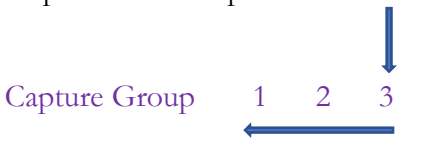


Step 13 - Current position is 'h' of 'The'

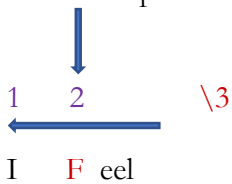


Step 14 - Current position is 'e' of 'The'

Step 15 - Current position is 'T' of 'The'



Step 16 – Current position is 'F' of Feel



Running zero times (optional), it returns to 'F' invalidating the value held by Capture Group 3

Tim Notes:

1. {0,1} is equivalent to the Quantifier ?. When applied against an expression, makes that expression optional.

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)(.*)

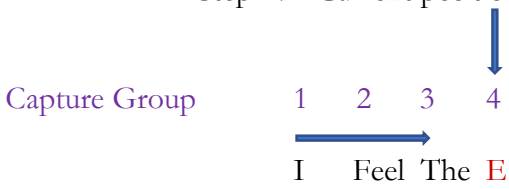
Tim Analysis: cont.

CAPTURE GROUP 4

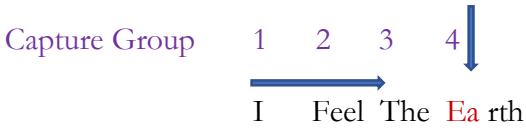
17. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 4 =E Changes Capture Group 3 = The Changes Capture Group 2 = Feel Capture Group 1 = Unchanged = I
18. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'a' and captures it.	Capture Group 4 = Ea
19. *	Make it Greedy	Capture Group 4 = Earth
20. ?	But not too Greedy (Non-Greedy) RegEx engine backtracking Current position is 'E' of 'Earth'	Capture Group 4 = E
21. {0,1}	Specify that the last expression, ([A-Z][^A-Z]*?) Run a minimum of Zero to 1 time. Runs Zero. Current position = 'F' of 'Feel'	Capture Group 4,3 = \4 \3 Invalid Changes Capture Group 2 = F Capture Group 1 = Unchanged = I

Visualize:

Step 17 - Current position is 'E' of 'Earth'

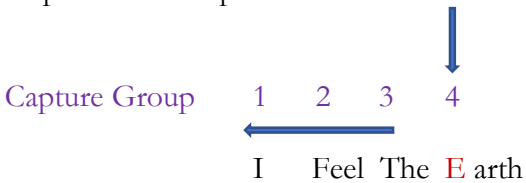


Step 18 - Current position is 'a' of 'Earth'

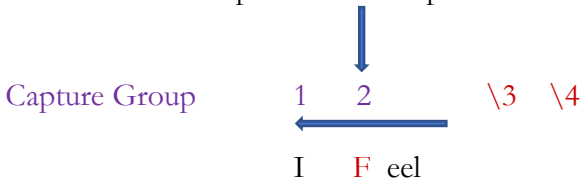


Step 19 - Current position is 'h' of 'Earth'

Step 20- Current position is 'E' of 'Earth'



Step 21 – Current position is 'F' of Feel



Running zero times, it returns to 'F' invalidating all of the values held by Capture Groups 4-3

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)([A-Z][^A-Z]*)(.*)

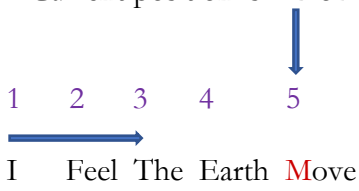
Tim Analysis: cont.

CAPTURE GROUP 5

22. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 5 =M Changes Capture Group 4 = Earth Changes Capture Group 3 = The Changes Capture Group 2 = Feel Capture Group 1 = Unchanged = I
23. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'o' and captures it.	Capture Group 5 = Mo
24. *	Make it Greedy	Capture Group 5 = Move
25. ?	But not too Greedy (Non-Greedy) RegEx engine backtracking Current position = 'M' of 'Move'	Capture Group 5 = M
26. {0,1}	Specify that the last expression, ([A-Z][^A-Z]*?) run a minimum of Zero to 1 time. Runs Zero. Current position = 'F' of "Feel'	Capture Group 5,4,3 = \5\4\3 Invalid Changes Capture Group 2 = F Capture Group 1 = Unchanged = I

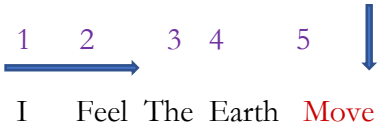
Visualize:

Step 22 - Current position is 'M' of 'Move'

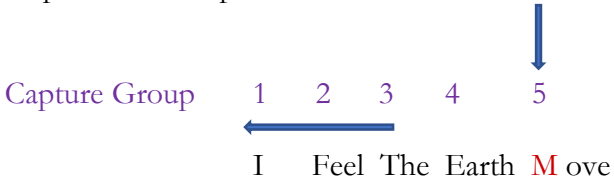


Step 23 - Current position is 'o' of 'Move'

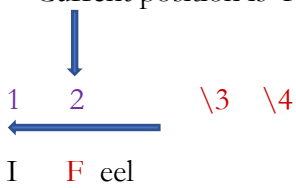
Step 24 - Current position is 'e' of 'Move'



Step 25- Current position is 'M' of 'Move'



Step 26 – Current position is 'F' of Feel



Running zero times, it returns to 'F' invalidating all of the values held by Capture Groups 4-3

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

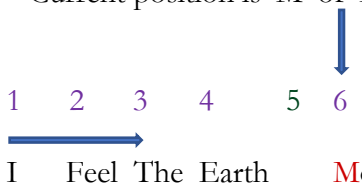
Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)([A-Z][^A-Z]*)(.*)

Tim Analysis: cont.

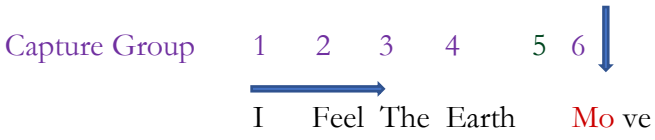
		CAPTURE GROUP 6
27. [A-Z]	Match against a class consisting of an uppercase letter Forces Capture Group 6 = M, so previous capture of M in Capture Group 5 is null. All other Capture Group's values are reevaluated to reflect the backtracking. This leaves Capture Group 5 with a null value.	Capture Group 6 =M Changes Capture Group 5 = null Changes Capture Group 4 = Earth Changes Capture Group 3 = The Changes Capture Group 2 = Feel Capture Group 1 = Unchanged = I
28. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'o' and captures it.	Capture Group 6 = Mo
29. *	Make it Greedy	Capture Group 6 = Move
30. ?	But not too Greedy (Non-Greedy) RegEx engine backtracking	Capture Group 6 = M
31. {0,1}	Specify that the last expression, ([A-Z][^A-Z]*?) run a minimum of Zero to 1 time. Runs Zero. Current position is 'F' of 'Feel'	Capture Group 6,5,4,3 = Invalid Changes Capture Group 2 = F Capture Group 1 = Unchanged = I

Visualize:

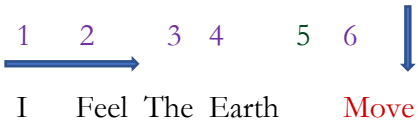
Step 27 - Current position is 'M' of 'Move'



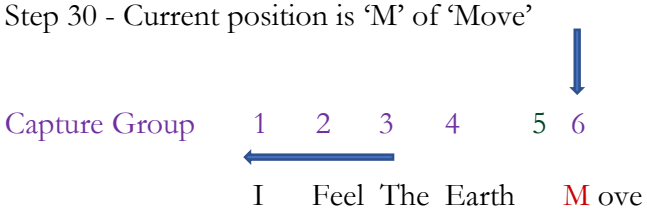
Step 28 - Current position is 'o' of 'Move'



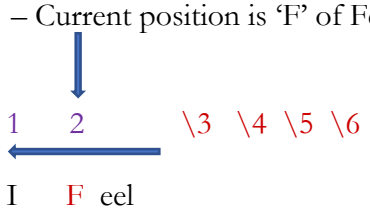
Step 29 - Current position is 'e' of 'Move'



Step 30 - Current position is 'M' of 'Move'



Step 31 – Current position is 'F' of Feel



Running zero times, it returns to 'F' invalidating all of the values held by Capture Groups 6-3

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

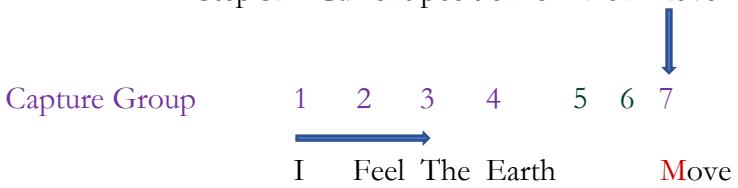
Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)([A-Z][^A-Z]*)(.*)

Tim Analysis: cont.

		CAPTURE GROUP 7
32. [A-Z]	Match against a class consisting of an uppercase letter Forces Capture Group 7 = M, so previous capture of M in Capture Group 6 is null. All other Capture Group's values are reevaluated to reflect the backtracking. This leaves Capture Group 6-5 with a null value.	Capture Group 7 =M Changes Capture Group 6 = null Changes Capture Group 5 = null Changes Capture Group 4 = Earth Changes Capture Group 3 = The Capture Group 2 = Unchanged = Feel Capture Group 1 = Unchanged = I
33. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'o' and captures it. RegEx engine is moving forward again	Capture Group 7 = Mo
34. *	Make it Greedy	Capture Group 7 = Move
35. ?	But not too Greedy (Non-Greedy) RegEx engine backtracking. Current position is 'M' of 'Move'	Capture Group 7 = M Capture Group 1 = Unchanged = I

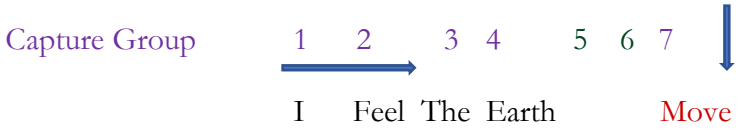
Visualize:

Step 32 - Current position is 'M' of 'Move'

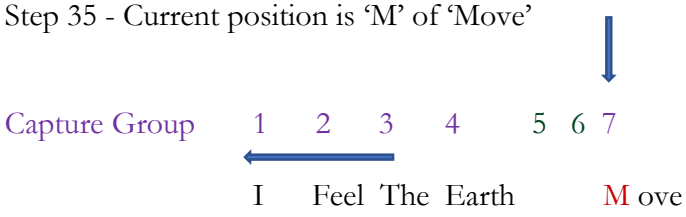


Step 33 - Current position is 'o' of 'Move'

Step 34 - Current position is 'e' of 'Move'



Step 35 - Current position is 'M' of 'Move'



Tim Notes:

1. There is no Minimum Maximum limitation {0,1}this time, so expression ([A-Z][^A-Z]*?) is allowed to run without being forced to return to 'F' as in 'Feel'. This continues to move the RegEx engine forward at this point.

Uncategorized

Separate Run-On Sentences with spacing #1 cont.

IFeelTheEarthMove.mp3

Match: ^([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*?){0,1}([A-Z][^A-Z]*)([A-Z][^A-Z]*)(.*)

Tim Analysis: cont.

CAPTURE GROUP 8

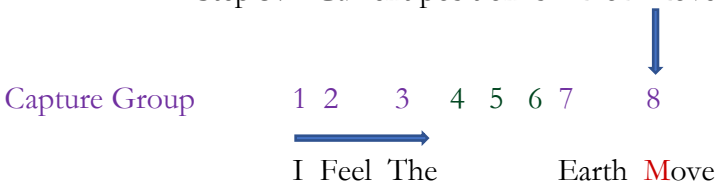
37. [A-Z]	Match against a class consisting of an uppercase letter Forces Capture Group 8 = M, so previous capture of M in Capture Group 7 is reevaluated to 'Earth'. Capture Group 6-5 remain unchanged with null values. Capture Group 4 previously held 'Earth changed to null.	Capture Group 8 =M Changes Capture Group 7 = Earth Changes Capture Group 6-4 = null Changes Capture Group 3 = The Changes Capture Group 2 = Feel Capture Group 1 = Unchanged = I
38. [^A-Z]	Ignore next uppercase letter What changed is the RegEx engine finds the lowercase 'o' and captures it. RegEx engine is moving forward again	Capture Group 8 = Mo
39. *	Make it Greedy Current position = 'e' of 'Move'	Capture Group 8 = Move Capture Group 1 = Unchanged = I

CAPTURE GROUP 9

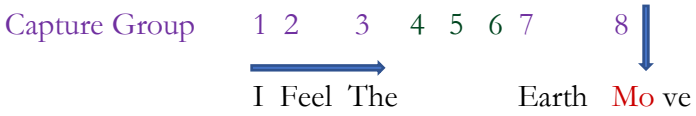
40. .*	Capture remainder of string to EOL (End of Line) Already at end of string so value is EOL = null.	Capture Group 9 = null

Visualize:

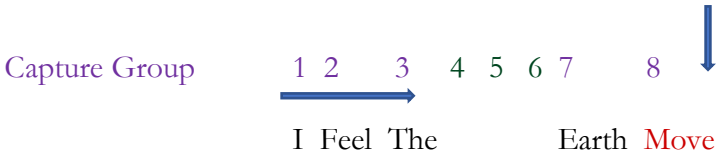
Step 37 - Current position is 'M' of 'Move'



Step 38 - Current position is 'o' of 'Move'

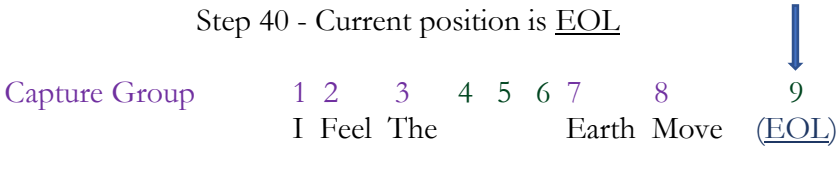


Step 39 - Current position is 'e' of 'Move'



Because {0,1} isn't present, the expression is run one time without returning to 'F' in 'Feel' as it did previously reflecting the minimum, run zero times. This results in nulling Capture Group 4 and reevaluating Capture Group 7's value to 'Earth'.

Step 40 - Current position is EOL



Uncategorized

Even More Downloaded TV Shows

Pielikers - 01x01 - I Like Pie.DIMENSION.English.C.updated.Addic7ed.com.srt
to
Pielikers S01E01 I Like Pie.srt

Person of Interest - 04x13 - M.I.A..DIMENSION.English.C.updated.Addic7ed.com.srt
to
Person of Interest S04E13 M.I.A..srt

Series Name - 02x02 - F.B.I..something.somethingmore.C.status.Addic7ed.com.srt
to
Series Name S02E02 F.B.I..srt

Silicon Valley - 02x05 - Server Space.IMMERSE.English.C.updated.Addic7ed.com.srt
to
Silicon Valley S02E05 Server Space.srt

Silicon Valley - 02x04 - The Lady.ASAP.English.C.updated.Addic7ed.com.srt
to
Silicon Valley S02E04 The Lady.srt

Observation:

1. Capture until first dot.
2. Add 'S' to first numeric for Season abbreviation, i.e. S02
3. Substitute 'E' for 'x' for Episode abbreviation, i.e. E04
4. Remove ' – ' and replace with one <space>

Match: (.+) - (\d\d)x(\d\d) - (.+)(\..*){6}
Replace: \1 S\2E\3 \4

Name	New Name
Silicon Valley - 02x05 - Server Space.IMMERSE.English....	Silicon Valley S02E05 Server Space.srt
Silicon Valley - 02x04 - The Lady.ASAP.English.C.updat...	Silicon Valley S02E04 The Lady.srt
Series Name - 02x02 - F.B.I..something.somethingmore....	Series Name S02E02 F.B.I..srt
Pielikers - 01x01 - I Like Pie.DIMENSION.English.C.upda...	Pielikers S01E01 I Like Pie.srt
Person of Interest - 04x13 - M.I.A..DIMENSION.English.C...	Person of Interest S04E13 M.I.A..srt

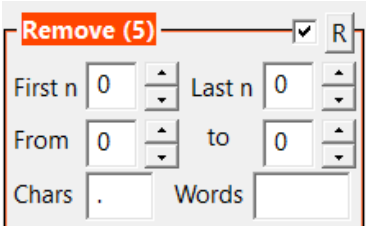
Uncategorized

Even More Downloaded TV Shows cont.

Tim Contribution –

Creates an excess dot at the end of three filenames above. To remove these, combine RegEx with –

'Section # 5: Remove'



Chars: . <dot>

Name	New Name
Silicon Valley - 02x05 - Server Space.IMMERSE.English....	Silicon Valley S02E05 Server Space.srt
Silicon Valley - 02x04 - The Lady.ASAP.English.C.updat...	Silicon Valley S02E04 The Lady.srt
Series Name - 02x02 - F.B.I..something.somethingmore....	Series Name S02E02 FBI.srt
Pielikers - 01x01 - I Like Pie.DIMENSION.English.C.upda...	Pielikers S01E01 I Like Pie.srt
Person of Interest - 04x13 - M.I.A..DIMENSION.English.C...	Person of Interest S04E13 MIA.srt

Uncategorized

Even More Downloaded TV Shows cont.

Using: Silicon Valley - 02x05 - Server Space.IMMERSE.English.C.updated.Addic7ed.com.srt

Match: (.+) - (\d\d)x(\d\d) - (.+)(\..*){6}

```
Match 1: Silicon Valley - 02x05 - Server Space.IMMERSE.English.C.updated.Addic7ed.com
Group 1: Silicon Valley
Group 2: 02
Group 3: 05
Group 4: Server Space
Group 5: .com
```

Where:

\1 = Silicon Valley \2 = 02 \3 = 05 \4 = Server Space \5 = .com

Tim Analysis:

1..	Match against any character	Capture Group 1 =
2. +	Make it Greedy	Capture Group 1 = <entire string>
3. <space>	Match against a <space> character (Not Captured) This is the <space> after 'Server'	Changes Capture Group 1 = Silicon Valley - 02x05 - Server
4. <hyphen>	Match against <hyphen> (Not Captured) This is the <hyphen> after '02x05 <space>' Changes <space> from step 2 to <space> After '02x05'	Changes Capture Group 1 = Silicon Valley - 02x05
5. <space>	Match against a <space> character (Not Captured) This is the <space> after '02x05 <space><hyphen>'	Capture Group 1 = Unchanged
6. \d\d	Match against 2 numeric digits Changes the <space> character from step 3 to the <space> character after 'Valley' and subsequently changes the <hyphen> in step 4 and the <space> in step 5 accordingly.	Capture Group 2 = 02 Changes Capture Group 1 = Silicon Valley
7. x	Match against a string 'x' (Not Captured) = 'x'	
8. \d\d	Match against 2 numeric digits	Capture Group 3 = 05
9. <space>	Match against a <space> character (Not Captured) This is the <space> after '02x05'	
10. <hyphen>	Match against <hyphen> (Not Captured) This is the <hypen> after '02x05<space>'	
11. <space>	Match against a <space> character (Not Captured) This is the <space> after '02x05 <space> <hypen>'	
12..	Match against any character	Capture Group 4 = S
13. +	Make it Greedy Current position at EOL	Capture Group 4 = Server Space.IMMERSE.English. C.updated.Addic7ed.com

Uncategorized

Even More Downloaded TV Shows cont.

Using: Silicon Valley - 02x05 - Server Space.IMMERSE.English.C.updated.Addic7ed.com.srt

Match: (.+) - (\d\d)x(\d\d) - (.+)(\..*){6}

Where:

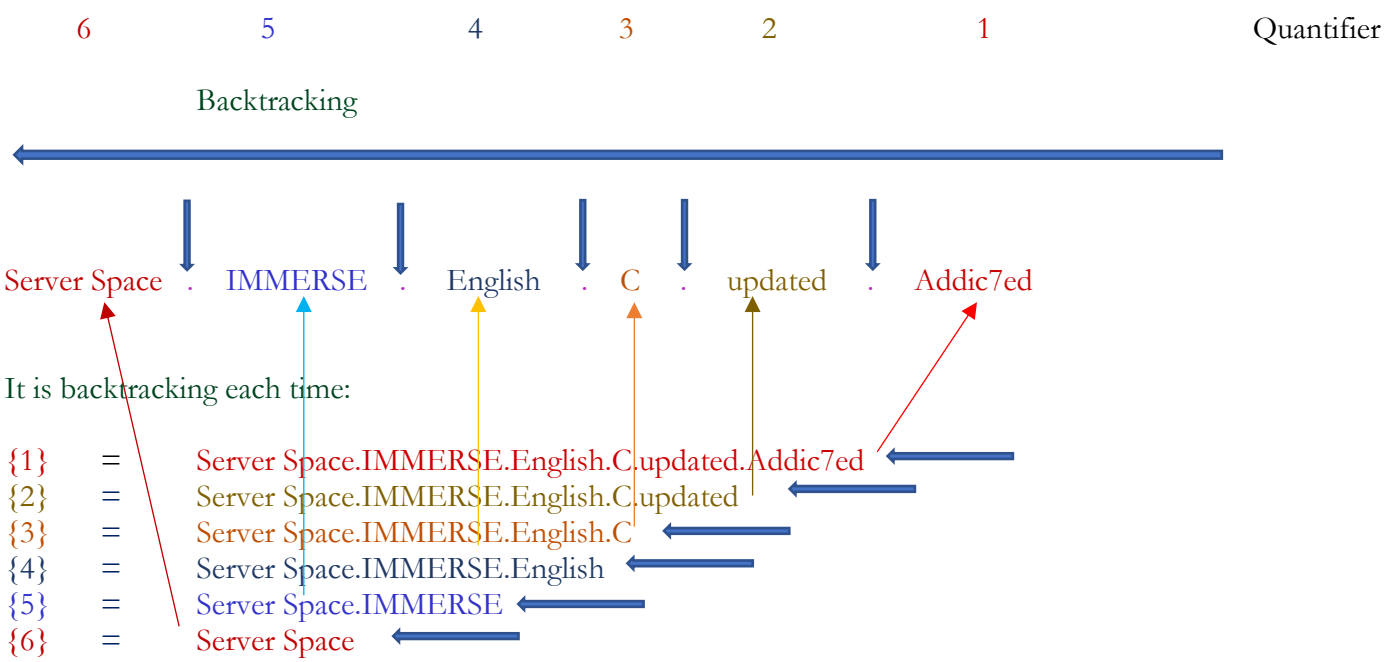
\1 = Silicon Valley \2 = 02 \3 = 05 \4 = Server Space \5 = .com

Tim Analysis: cont.

14. \.	Match against literal Backtracks to match against the <dot> Preceding '.com'	Capture Group 5 = <dot> Changes Capture Group 4 = Server Space.IMMERSE.English.C. updated.Addic7ed
15. .	Match against any character	Capture Group 5 = .c
16. *	Make it Greedy	Capture Group 5 = .com
17 {6}	Repeat last expression to match 6 iterations.	Capture Group 5 = Unchanged Changes Capture Group 4 = Server Space

Tim Notes:

1. By repeating the expression (\..*) 6 iterations, it matches against –



It is backtracking each time:

{1} = Server Space.IMMERSE.English.C.updated.Addic7ed
{2} = Server Space.IMMERSE.English.C.updated
{3} = Server Space.IMMERSE.English.C
{4} = Server Space.IMMERSE.English
{5} = Server Space.IMMERSE
{6} = Server Space

Each time it is removing more characters from Capture Group 4 until on the 6th iteration, the value is 'Server Space'

Uncategorized

Example of a Very Specific Rename of a Run-On Word Group

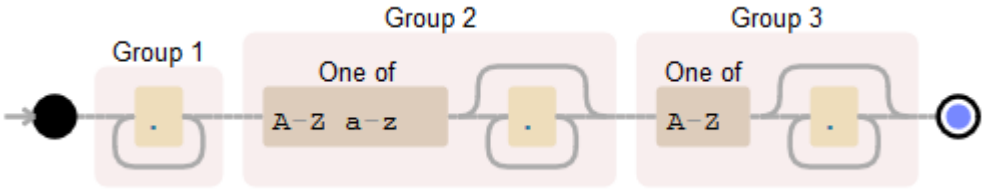
Tim Contribution –

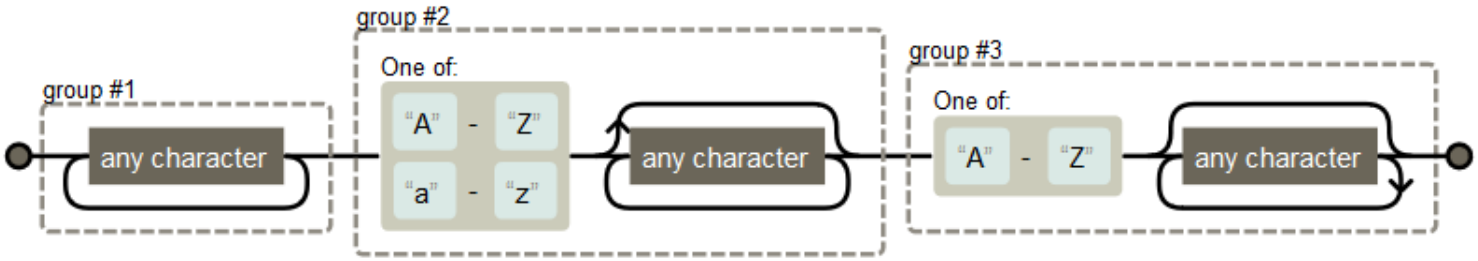
abcU.pdf to abc U.pdf
abcUnlimited to abc Unlimited

Observations:

1. Restrictive pattern - lowercase letters followed by uppercase letter

Match: (.+?)([A-Za-z].*?)([A-Z].*)
Replace: \1\2 \3





Sample:

```
.....................
Match 1: abcUnlimited
Group 1: a
Group 2: bc
Group 3: Unlimited
```

Tim Notes:

1. Could easily have been perfomed and less restrictive enabling Wordspace option of 'Section #7: Add'.
2. Sometimes when you have a very specfic task, rather than go for universal, it is better to be restrictive. Just be careful in selection of files to be renamed or use a filter.

Uncategorized

Example of a Very Specific Rename of a Run-On Word Group cont.

Tim Contribution – cont.

Using: abcUnlimited

Match: (.+?)([A-Za-z].*?)([A-Z].*)

Where:

\1 = a \2 = bc \3 = Unlimited

Tim Analysis:

1..	Match against any character	Capture Group 1 = a
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = a
4. [A-Za-z]	Match against a class consisting of an uppercase or lowercase letter	Capture Group 2 = b
5..	Match against any character	Capture Group 2 = bc
6. *	Make it Greedy	Capture Group 2 = bcUnlimited
7. ?	But not too Greedy	Capture Group 2 = b
8. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = U Changes Capture Group 2 = bc
9. .*	Match against remaining string	Capture Group 3 = Unlimited

Uncategorized

Renaming long filenames

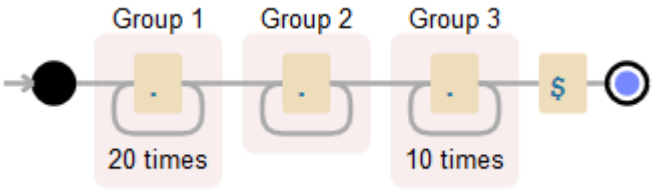
Sample:

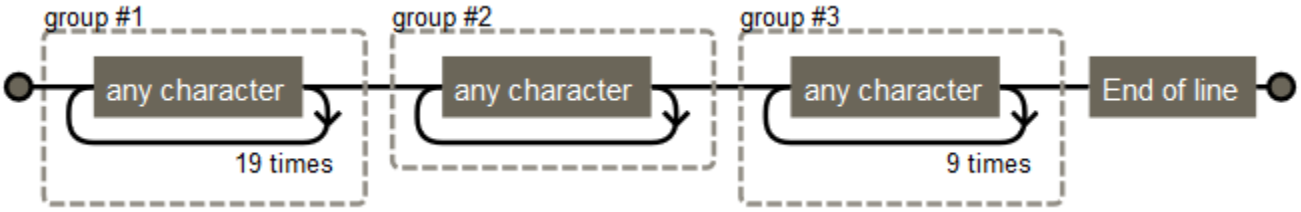
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >30 characters by keeping 1st-20 & last-10 characters

Match: (.{20})(.{1,})(.{10})$
Replace: \1\3

Name	New Name
1234567890abcdefghijklmnopqrstuvwxyzABCDE...	1234567890abcdefghijuvwxyz.pdf





Tim Notes:

Where:

\1 = 1234567890abcdefghij

\2 = klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop

\3 = qrstuvwxyz

2. BRU does *not* support long file pathnames that exceed 260 characters. This includes the filename with path. BRU can display the filename but will generate an error if you attempt to rename the file. The above filname is 164 characters and was placed in a test directory. Full path length was 181 characters, well below the 260 characters threshold.

Uncategorized

Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >30 characters by keeping 1st-20 & last-10 characters cont.

Match: (.{20})(.{1,})(.{10})$

```
Group 1: 1234567890abcdefghij
Group 2: klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop
Group 3: qrstuvwxyz
```

Expanded:

```
Match 1: 1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdef
Group 1: 1234567890abcdefghij
Group 2: klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz
Group 3: qrstuvwxyz
```

	Start	Length
ghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz	0	160
	0	20
1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop	20	130
	150	10

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 1
2. {20}	Repeat last expression to match 20 iterations.	Capture Group 1 = 1234567890abcdefghij
3. .	Match against any character	Capture Group 2 = k
4. {1,}	Specifies a minimum of 1 and maximum of unlimited iterations. Current position = EOL	

Capture Group 2 =

klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Uncategorized

Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >30 characters by keeping 1st-20 & last-10 characters cont.

Match: (.{20})(.{1,})(.{10})$

Tim Analysis: cont.

5. . Match against any character — Capture Group 3 = z
Already at EOL, backtracks to capture 'z'

Changes Capture Group 2 =

klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy

6. {10} Repeat last expression to match 10 iterations. — Capture Group 3 = qrstuvwxyz
Continues backtracking to take last 10 characters.

Changes Capture Group 2 =

klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop

Uncategorized

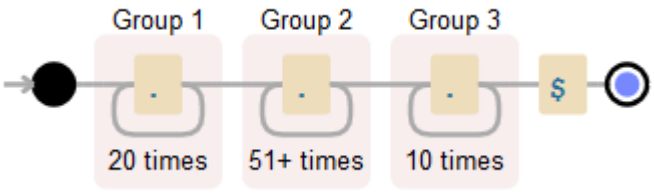
Renaming long filenames cont.

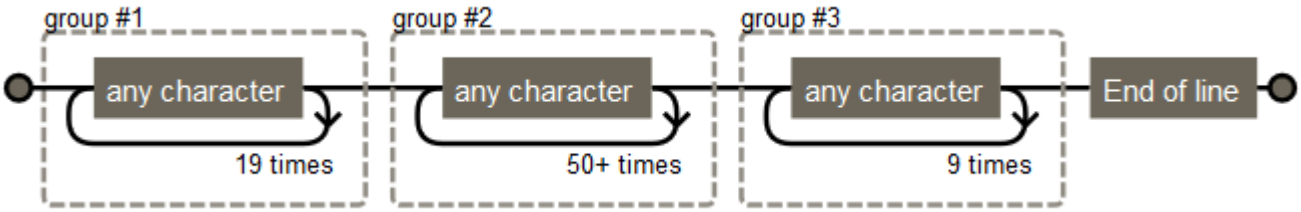
Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >80 characters by keeping 1st-20 & last-10 characters

Match: (.{20})(.{51,})(.{10})$
Replace: \1\3

Name	New Name
1234567890abcdefghijklmnopqrstuvwxyzABCDE...	1234567890abcdefghijuvwxyz.pdf





Tim Notes:

Where:

\1 = 1234567890abcdefghij

\2 = klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrst

\3 = uvwxyz

2. BRU does *not* support long file pathnames that exceed 260 characters. This includes the filename with path. BRU can display the filename but will generate an error if you attempt to rename the file. The above filname is 164 characters and was placed in a test directory. Full path length was 181 characters, well below the 260 characters threshold.

Uncategorized

Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >80 characters by keeping 1st-20 & last-10 characters cont.

Match: (.{20})(.{51,})(.{10})$

```
Group 1: 1234567890abcdefghij
Group 2: klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop
Group 3: qrstuvwxyz
```

Expanded:

```
Match 1: 1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz
Group 1: 1234567890abcdefghij
Group 2: klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJ
Group 3: qrstuvwxyz
```

	Start	Length
1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz	0	160
	0	20
KLMNOPQRSTUVWXYZabcdefghijklmnop	20	130
	150	10

Tim Analysis:

1. .	Match against any character	Capture Group 1 = 1
2. {20}	Repeat last expression to match 20 iterations.	Capture Group 1 = 1234567890abcdefghij
3. .	Match against any character	Capture Group 2 = k
4. {51,}	Specifies a minimum of 51 and maximum of unlimited iterations. Current position = EOL	

Capture Group 2 =

klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Uncategorized

Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >80 characters by keeping 1st-20 & last-10 characters cont.

Match: (.{20})(.{51,})(.{10})$

Tim Analysis: cont.

5. . Match against any character
Already at EOL, backtracks to capture 'z'
Capture Group 3 = z

Changes Capture Group 2 =

klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy

6. {10} Repeat last expression to match 10 iterations.
Continues backtracking to take last 10 characters.
Capture Group 3 = qrstuvwxyz

Changes Capture Group 2 =

klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop

Tim Notes:

1. The analysis on the given sample provides the same results as in the previous example but the sample size is what determines the difference in outcomes between the two RegEx.

   The first sample requires at least 31 characters (the initial 20 + 1 character + last 10 characters = 31 characters minimum).

   The second requires at least 81 characters (the initial 20 characters + 51 characters + last 10 characters = 81 minimum).

   Since the same sample was used on both, and the sample size was greater than the 31 characters required for the first RegEx and greater than the 81 characters required for the second RegEx, this would explain why there is no difference in the analysis results between the two.

Uncategorized

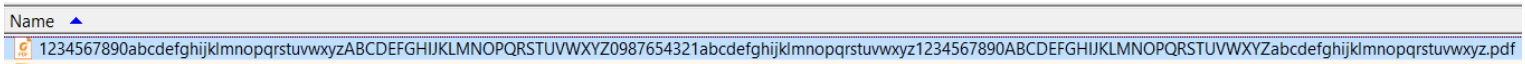
Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >80 characters by keeping 1st-80 characters

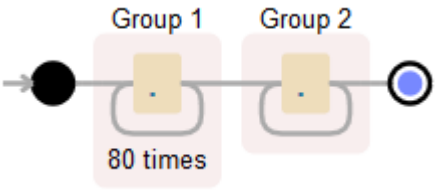
Match: (.{80})(.+)
Replace: \1

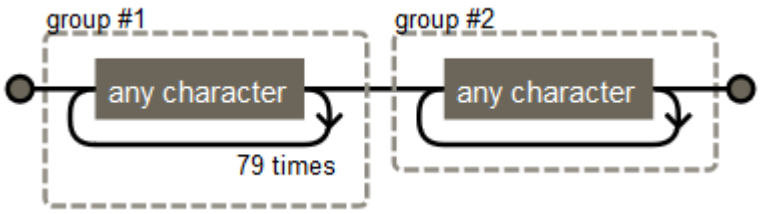
NAME



NEW NAME

1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh.pdf





Tim Notes:

Where:

\1 = 1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh

\2 = ijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

2. BRU does *not* support long file pathnames that exceed 260 characters. This includes the filename with path. BRU can display the filename but will generate an error if you attempt to rename the file. The above filname is 164 characters and was placed in a test directory. Full path length was 181 characters, well below the 260 characters threshold.

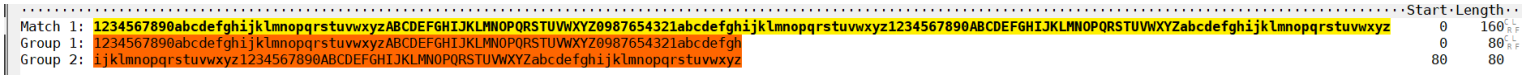
Uncategorized

Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >80 characters by keeping 1st-80 characters cont.

Match: (.{80})(.+)



Expanded:

```
Match 1: 1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijk
Group 1: 1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh
Group 2: ijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

```
                                                                    Start  Length
lmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz      0     160
                                                                                0      80
                                                                               80      80
```

Tim Analysis:

1. . Match against any character Capture Group 1 = 1
2. {80} Repeat last expression to match 80 iterations.

Capture Group 1 =

1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh

3. .+ Match against remaining string.

Capture Group 2 =

ijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Uncategorized

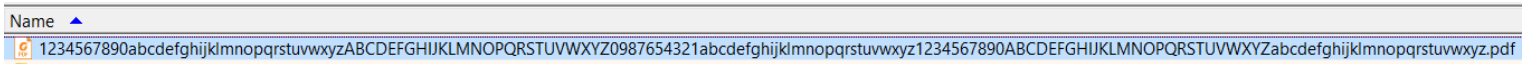
Renaming long filenames cont.

Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

Rename files >80 characters by keeping last-80 characters

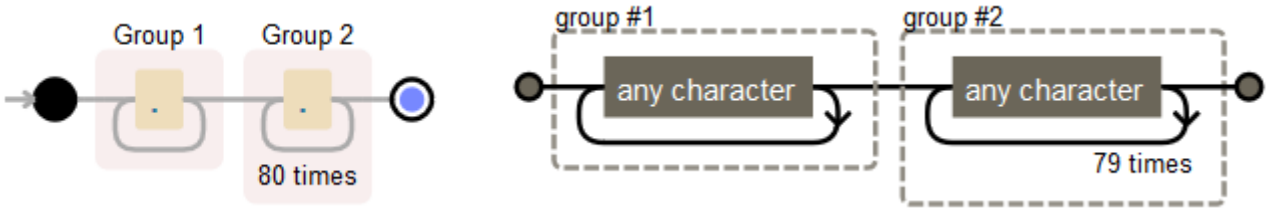
Match: (.+)(.{80})
Replace: \2

NAME



NEW NAME

1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh.pdf



Tim Notes:

1. Can't display Regex Buddy sample run photo– graphic too long to fit properly. Instead –

Where:

\1 = 1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh

\2 = ijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

2. BRU does not support long file pathnames that exceed 260 characters. This includes the filename with path. BRU can display the filename but will generate an error if you attempt to rename the file. The above filname is 164 characters and was placed in a test directory. Full path length was 181 characters, well below the 260 characters threshold.

Forum Contributor: truth

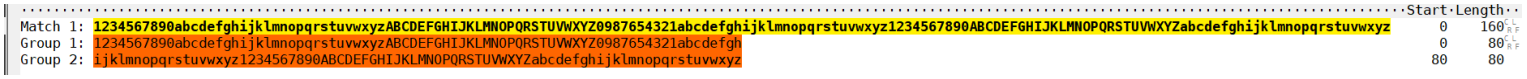
Uncategorized

Renaming long filenames cont.

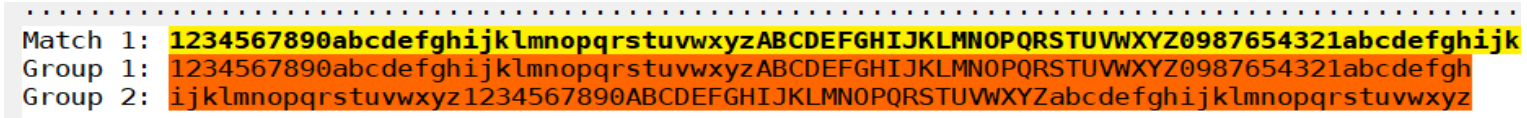
Using:
1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz.pdf

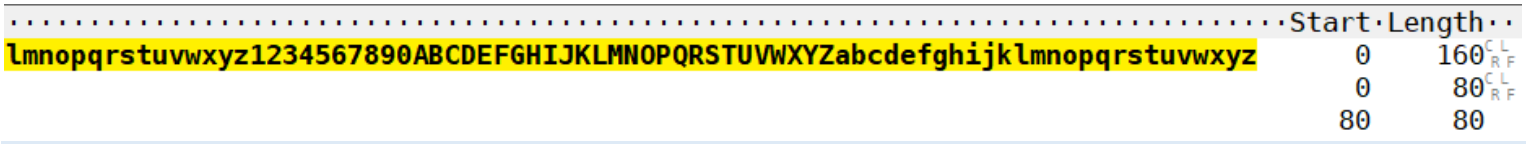
Rename files >80 characters by keeping last-80 characters cont.

Match: (.+)(.{80})



Expanded:





Tim Analysis:

1. .	Match against any character	Capture Group 1 = 1
2. +	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
3. .	Match against any character Already at EOL, backtracks to capture 'z'	Capture Group 2 = z

Changes Capture Group 1 =

1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy

4. {80}	Repeat last expression to match 80 iterations. Still backtracking. Takes the last 80 characters from the end.

Capture Group 2 =

ijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

Changes Capture Group 1 =

1234567890abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0987654321abcdefgh

Uncategorized

Identify Every 'Nth' File from a Folder Containing Hundreds of Files

Will allow you to more easily identify the 'Nth' file for possible later selection or renaming or other action. The first starting file will always be counted as the beginning file of series –
i.e., every 7th file with start = 1: file # 1, first file of the 7 or i.e., every 7th file with start = 3: file #3, first file of the 7.

1. The procedure will be as follows:
   a. Use Numbering to prefix an autonumber
   b. Increment value will be '1'
   b. Use a pad dependent upon the value of the 'Nth'
   c. Use a Break of 1
   d. Sep (arator character) will be <space> for clarification

2. The 'Nth' file number will determine the pad value. Remember that pad will pad zeros so you have to think in terms of decimal places - pad 2 for 10's (2 decimal places), pad 3 for 100's (3 decimal places), pad 4 for 1000's (4 decimal places) etc.

For example,

If you wanted to be able to identify every 7th file in a folder, the closest value would be 10, so pad = 2 for 2 decimal places.
If you wanted every 15th file, the closest value would be 100, so pad = 3 for 3 decimal places
If you wanted every 205th file, the closest value would be 1000 so pad = 4 for 4 decimal places

3. Determine starting value for the autonumber.

   a. Subtract the value for 'Nth' from the closest value equal to the number of required decimal places (as determined by the pad value above, again being 10, 100, 1000 etc.)

For example,

If you wanted to be able to identify every 9th file in a folder, the starting number is 10 - 1 = 1
If you wanted to be able to identify every 7th file in a folder, the starting number is 10 - 7 = 3
If you wanted every 15th file, the starting number is 100 - 15 = 85
If you wanted every 205th file, the starting number is 1000 - 205 = 795

3. The Break determines when the autonumber counter resets back to it's starting value. Using a value of '1' indicates that when the first character position of the autonumber changes, then reset.

i.e.

Both of these examples have the break set to occur when autonumber reaches 10 because that is when the counter will go from 09 to 10, thus changing the zero value, the first character in the autonumber, from zero to one

If the Autonumber starting value is set at 1, the sequence will be:
01, 02, 03, 04, 05, 06, 07, 08, 09, 01, … etc.

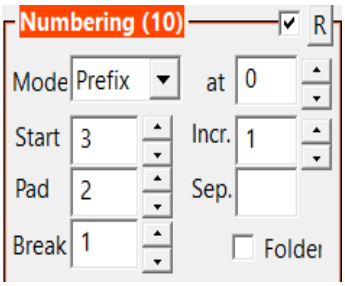
If the Autonumber starting value is set at 3, the sequence will be:
03, 04, 05, 06, 07, 08, 09, 03, 04, 05, … etc.

Uncategorized

Identify Every 'Nth' File from a Folder Containing Hundreds of Files cont.

Conclusion:

In this example, every filename that begins with a prefix of 09 will be every seventh file in the directory:

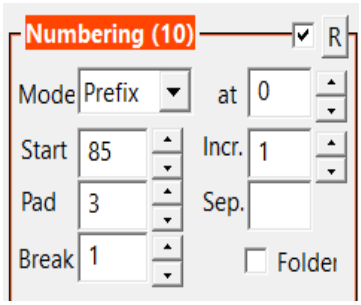


Mode: Prefix
Pad: 2 (10 is the closest decimal value for 7)
Start: 3 (10-7 = 3)
Break: 1
Sep: <space>

Every filename that begins with a prefix of 09 will be every seventh file in the directory. Not including the first file that begins the series.

	New Name
Windows XP Video Decoder Checkup Utility.txt	03 Windows XP Video Decoder Checkup Utility.txt
Windows XP Video Decoder Checkup Utility 1.0.0.1(freew...	04 Windows XP Video Decoder Checkup Utility 1.0.0.1(freeware).EXE
Windows XP AVI Fix.txt	05 Windows XP AVI Fix.txt
Windows Media Runtimes (if you don't have Windows Me...	06 Windows Media Runtimes (if you don't have Windows Media Player Installed) -wmfdist.exe
winamp iPod_Support_v2_03(freeware).zip	07 winamp iPod_Support_v2_03(freeware).zip
winamp ipod plugin.txt	08 winamp ipod plugin.txt
WAVTools v1.10(freeware).zip	09 WAVTools v1.10(freeware).zip
waveatmp3_setup v3.26(freeware).exe	03 waveatmp3_setup v3.26(freeware).exe

Another example using 15 for the 'Nth' value:



Mode: Prefix
Pad: 3 (100 is the closest decimal place value for 15)
Start: 85 (100-15 = 85)
Break: 1
Sep: <space>

Every filename that begins with a prefix of 085 will be every fifteenth file in the directory. Not including the first file that begins the series.

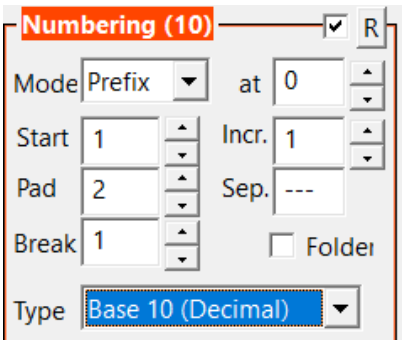
	New Name
YAMB.txt	085 YAMB.txt
Yamb-2.0-Preview(alpha) (freeware).zip	086 Yamb-2.0-Preview(alpha) (freeware).zip
Yamb-1.6+mp4tags(freeware).zip	087 Yamb-1.6+mp4tags(freeware).zip
YAAI_2.0.3.488(freeware).zip	088 YAAI_2.0.3.488(freeware).zip
YAAI - Yet Another Avi Info.txt	089 YAAI - Yet Another Avi Info.txt
xvidcodec.txt	090 xvidcodec.txt
XviD-1.1.2-01112006(freeware).exe	091 XviD-1.1.2-01112006(freeware).exe
xpcodec pack.txt	092 xpcodec pack.txt
XP_Codec_Pack_2.0.6.1(freeware).zip	093 XP_Codec_Pack_2.0.6.1(freeware).zip
x264 Video Codec.txt	094 x264 Video Codec.txt
x264 codec rev 634(freeware).exe	095 x264 codec rev 634(freeware).exe
wolfram research workbench.txt	096 wolfram research workbench.txt
winxpavifix v1.01(freeware).zip	097 winxpavifix v1.01(freeware).zip
winlame.txt	098 winlame.txt
winLAME-prerelease4(freeware).msi	099 winLAME-prerelease4(freeware).msi
Windows XP Video Decoder Checkup Utility.txt	085 Windows XP Video Decoder Checkup Utility.txt
Windows XP Video Decoder Checkup Utility 1.0.0.1(freew...	086 Windows XP Video Decoder Checkup Utility 1.0.0.1(freeware).EXE

Uncategorized

Identify Every 'Nth' File from a Folder Containing Hundreds of Files cont.

Here's another interesting method:

1. 'Section #10: Numbering' -

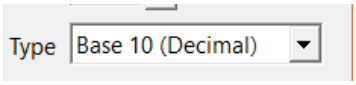


Mode: Prefix
Start: 1
Pad: 2
Break: 1
Incr: 1
Sep: ---
Type: Base 10 (Decimal)

2. This method uses the 'Type' option as the key:
   a. Take the value of 'n' as in 'nth' as was used in the examples on the previous pages, and add 1 to that value.
   b. Set the Type to the base equal to that result.

i.e.

For every 9th file, use (9 +1) = Base 10 (Decimal)



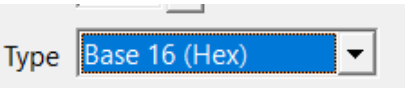
	New Name
Word1-Word2.com - 2009 r.ext	01---Word1-Word2.com - 2009 r.ext
Word1-Word2.com - 2009 r.ext	02---Word1-Word2.com - 2009 r.ext
Windows 7 Quick Reference Card.pdf	03---Windows 7 Quick Reference Card.pdf
Welcome.doc	04---Welcome.doc
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6.docx	05---UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6.docx
The.Thaw.aka.Arctic.Outbreak.2009	06---The.Thaw.aka.Arctic.Outbreak.2009
test.txt	07---test.txt
Strange problem with Internet Explorer - works only when r...	08---Strange problem with Internet Explorer - works...
some name again.ext	09---some name again.ext
some name again and more.ext	01---some name again and more.ext
some name [brackets1] again and [brackets 2] more.ext	02---some name [brackets1] again and [brackets 2] ...
some [name].ext	03---some [name].ext
some [name] again.ext	04---some [name] again.ext
SOLVED- Why is the washing machine getting out of balanc...	05---SOLVED- Why is the washing machine getting ...
Solved - Internet Explorer 11 Only Opens As Administrator ...	06---Solved - Internet Explorer 11 Only Opens As A...
Receipt_php.txt	07---Receipt_php.txt
Receipt_php.mht	08---Receipt_php.mht
Receipt_php.htm	09---Receipt_php.htm
prefs.js	01---prefs.js
old.txt	02---old.txt
new.txt	03---new.txt
nc_025213.pdf	04---nc_025213.pdf
nc_017733.pdf	05---nc_017733.pdf
Microsoft.doc	06---Microsoft.doc
localstore.rdf	07---localstore.rdf
LinkClick.pdf	08---LinkClick.pdf
Internet Explorer only opens as Administrator Solved - Win...	09---Internet Explorer only opens as Administrator S...
Internet Explorer 11 will only open if I right click and then s...	01---Internet Explorer 11 will only open if I right clic...

Uncategorized

Identify Every 'Nth' File from a Folder Containing Hundreds of Files cont.

i.e.

For every 15th file, use (15 +1) = Base 16 (Hex)



	New Name
Word1-Word2.com - 2009 r.ext	01---Word1-Word2.com - 2009 r.ext
Word1-Word2.com - 2009 r.ext	02---Word1-Word2.com - 2009 r.ext
Windows 7 Quick Reference Card.pdf	03---Windows 7 Quick Reference Card.pdf
Welcome.doc	04---Welcome.doc
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6.docx	05---UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6.docx
The.Thaw.aka.Arctic.Outbreak.2009	06---The.Thaw.aka.Arctic.Outbreak.2009
test.txt	07---test.txt
Strange problem with Internet Explorer - works only when r...	08---Strange problem with Internet Explorer - works...
some name again.ext	09---some name again.ext
some name again and more.ext	0A---some name again and more.ext
some name [brackets1] again and [brackets 2] more.ext	0B---some name [brackets1] again and [brackets 2] ...
some [name].ext	0C---some [name].ext
some [name] again.ext	0D---some [name] again.ext
SOLVED- Why is the washing machine getting out of balanc...	0E---SOLVED- Why is the washing machine getting ...
Solved - Internet Explorer 11 Only Opens As Administrator ...	0F---Solved - Internet Explorer 11 Only Opens As Ad...
Receipt_php.txt	01---Receipt_php.txt
Receipt_php.mht	02---Receipt_php.mht
Receipt_php.htm	03---Receipt_php.htm
prefs.js	04---prefs.js
old.txt	05---old.txt
new.txt	06---new.txt
nc_025213.pdf	07---nc_025213.pdf
nc_017733.pdf	08---nc_017733.pdf
Microsoft.doc	09---Microsoft.doc
localstore.rdf	0A---localstore.rdf
LinkClick.pdf	0B---LinkClick.pdf
Internet Explorer only opens as Administrator Solved - Win...	0C---Internet Explorer only opens as Administrator ...
Internet Explorer 11 will only open if I right click and then s...	0D---Internet Explorer 11 will only open if I right cli...
IE11 only opens when run as Administrator - Spiceworks.png	0E---IE11 only opens when run as Administrator - S...
IE will only open as administrator.png	0F---IE will only open as administrator.png
IE 11 ONLY WORKS IF I PRESS RUN ADMINISTRATOR WHY ...	01---IE 11 ONLY WORKS IF I PRESS RUN ADMINISTR...
Hassle.doc	02---Hassle.doc

Tim Notes:

1. For more information, refer to page 82, 'Section #10: Numbering' from 'Volume I Bulk Utility Operations Manual'.

Forum Contributors: JMM, bru, with added material by Tim

Uncategorized

Rename TV shows from 1x01 to s01e01

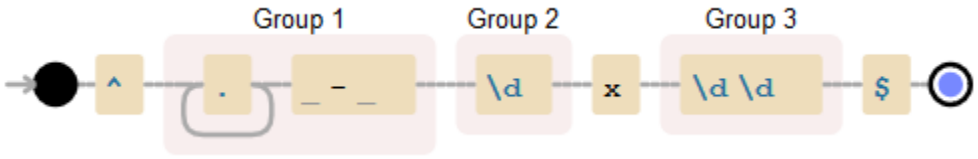
tvshow - 1x01.avi to tvshow - s01e01.avi
tvshow - 2x01.avi to tvshow - s02e01.avi

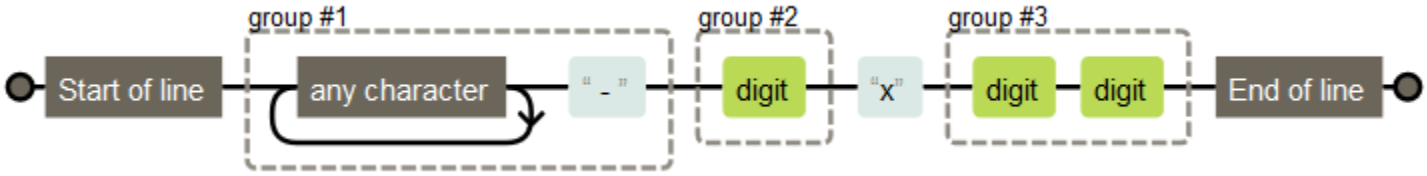
Observations:

1. Pattern <text> <Space> <hyphen> <space> #x##

Match: ^(.+ -)(\d)x(\d\d)$
Replace: \1s0\2e\3

	New Name
tvshow - 1x01.avi	tvshow - s01xe01.avi
tvshow - 2x01.avi	tvshow - s02xe01.avi





Sample:

```
.........................
Match 1: tvshow - 1x01
Group 1: tvshow -
Group 2: 1
Group 3: 01
```

Forum Contributor: Stefan

Uncategorized

Rename TV shows from 1x01 to s01e01 cont.

Using: tvshow - 1x01.avi to tvshow - s01e01.avi

Match: ^(.+ -)(\d)x(\d\d)$

Where:

\1 = tv show<space> - <space> \2 =1 \3 = 01

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL)	
2. .	Match gainst any character	Capture Group 1 = t
3. +	Make it Greedy until …	Capture Group 1 = entire string
4. <space>	.. <space> followed by …	Capture Group 1 = tv show <space> - <space>
5. -	.. <hyphen> followed by …	Capture Group 1 = tv show <space> -
6. <space>	.. <space> followed by …	Capture Group 1 = tv show <space> - <space>
7. \d	.. numeric digit followed by …	Capture Group 2 = 1
8. x	.. literal 'x' followed by …	Capture Group 2 = 1x
9. \d\d	.. two numeric digits	Capture Group 3 = 01
10. $	Last expression must have matched at EOL (must be at End of Line)	

Uncategorized

Tim's Tips & Tidbits:

Filtering out Filenames/Foldernames containing Spaces

It may seem straight forward, but if you want to filter out files/folders that contain <space> characters, the suggested approach by the original author, Jim Willsher will not work. He instructed users to enclose the string in the 'Mask' field of 'Section #12: Filters' in double quotations similarily to how you have to specify this in Windows.

i.e.

If you have several files:

do rename1.ext
do rename2.ext
do reanme3.ext

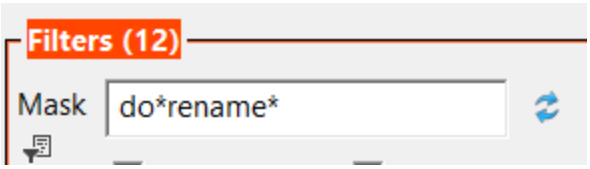
And you wish to filter them in the Content Pane (the File List Window):

Mask: do rename
Mask: do rename*
Mask: "do rename"

These will not work.

This will:

'Section #12: Filters' -



Mask: do*rename*

Work-Around:

1. Replace any occurrence of a <space> within the filter mask with a wildcard character.
2. And of course account for any extension or other extraneous characters of the filename that are not specified within the Mask with a wildcard character as well.
3. The above example shows a wildcard character accounting for the <space> character after 'do' and a secondary wildcard character that accounts for the '#.ext' at the end of the filename. Note that '?' will also work.

Uncategorized

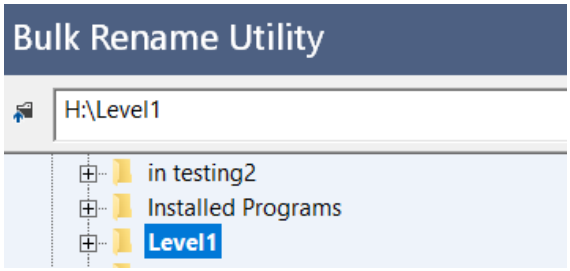
Move Files based on Creation Date to another Directory

Although not directly supported, you can use BRU to move files based on a creation date to another directory for archival purposes.

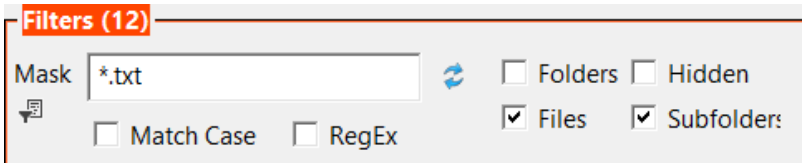
Observations:

1. You have text files of varying dates in a folder, 'Level1'.
2. You want to move the files older than March 2020 to another directory.

1. In the Navigation Pane (the folder tree, aka tree hierarchy) select the 'Level1' folder.



2. 'Section #12: Filters' –



Files: Enabled
Subfolders: Enabled (turns on Recursion)
Mask: *.txt

a. Click the Blue Filters Refresh button to the right of the Mask field or Press F5

Name	New Name
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt
abc123.....txt	abc123.....txt
out3.txt	out3.txt
out4.txt	out4.txt
out5.txt	out5.txt
1863_22_10_2019.txt	1863_22_10_2019.txt
dome.txt	dome.txt
out.txt	out.txt
out2.txt	out2.txt
test.txt	test.txt
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt
help.txt	help.txt
mike.txt	mike.txt
new.txt	new.txt
old.txt	old.txt
newnames.txt	newnames.txt
oldnames.txt	oldnames.txt
__BRU_TextFile.txt	__BRU_TextFile.txt
Receipt_php.txt	Receipt_php.txt

The Content Pane only displays those files that meet the filter's criteria. In this case, it displays only files with a '.txt. extension located inside the 'Level1' folder along with any other text files that reside in any sub-directories beneath 'Level1' (the recursion).

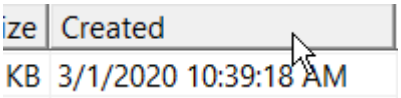
Uncategorized

Move Files based on Creation Date to another Directory cont.

3. Sort the files by 'Creation date'

Name	New Name		Size	Created
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt	Text Document	1 KB	3/1/2020 10:39:18 AM
abc123.....txt	abc123.....txt	Text Document	4 KB	3/25/2020 12:38:58 PM
out3.txt	out3.txt	Text Document	2 KB	3/27/2020 9:08:49 PM
out4.txt	out4.txt	Text Document	6 KB	3/27/2020 11:42:24 PM
out5.txt	out5.txt	Text Document	30 KB	3/28/2020 7:56:33 AM
1863_22_10_2019.txt	1863_22_10_2019.txt	Text Document	6 KB	1/13/2020 5:50:00 PM
dome.txt	dome.txt	Text Document	248 B	3/27/2020 10:25:03 AM
out.txt	out.txt	Text Document	2 KB	3/27/2020 3:52:44 PM
out2.txt	out2.txt	Text Document	4 KB	3/27/2020 4:02:56 PM
test.txt	test.txt	Text Document	1 KB	2/7/2020 11:10:50 PM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt	Text Document	1 KB	3/1/2020 10:39:18 AM
help.txt	help.txt	Text Document	5 KB	3/21/2020 12:04:51 PM
mike.txt	mike.txt	Text Document	6 KB	1/13/2020 5:50:00 PM
new.txt	new.txt	Text Document	124 B	2/6/2020 2:24:15 PM
old.txt	old.txt	Text Document	168 B	2/6/2020 2:22:56 PM
newnames.txt	newnames.txt	Text Document	134 B	2/17/2020 10:22:21 PM
oldnames.txt	oldnames.txt	Text Document	126 B	2/17/2020 9:00:35 PM
__BRU_TextFile.txt	__BRU_TextFile.txt	Text Document	123 B	2/17/2020 11:06:57 PM
Receipt_php.txt	Receipt_php.txt	Text Document	2 KB	8/20/2012 8:27:17 AM

a. The Creation date can be seen in the column 'Created' in the Content Pane.
b. To sort by Creation date, click on the 'Created' Title Bar. This is a toggle that will change the sort by ascending or descending order.



Name	New Name	Type	Size	Created
Receipt_php.txt	Receipt_php.txt	Text Document	2 KB	8/20/2012 8:27:17 AM
1863_22_10_2019.txt	1863_22_10_2019.txt	Text Document	6 KB	1/13/2020 5:50:00 PM
mike.txt	mike.txt	Text Document	6 KB	1/13/2020 5:50:00 PM
old.txt	old.txt	Text Document	168 B	2/6/2020 2:22:56 PM
new.txt	new.txt	Text Document	124 B	2/6/2020 2:24:15 PM
test.txt	test.txt	Text Document	1 KB	2/7/2020 11:10:50 PM
newnames.txt	newnames.txt	Text Document	134 B	2/17/2020 10:22:21 PM
oldnames.txt	oldnames.txt	Text Document	126 B	2/17/2020 9:00:35 PM
__BRU_TextFile.txt	__BRU_TextFile.txt	Text Document	123 B	2/17/2020 11:06:57 PM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt	Text Document	1 KB	3/1/2020 10:39:18 AM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt	Text Document	1 KB	3/1/2020 10:39:18 AM
help.txt	help.txt	Text Document	5 KB	3/21/2020 12:04:51 PM
abc123.....txt	abc123.....txt	Text Document	4 KB	3/25/2020 12:38:58 PM
dome.txt	dome.txt	Text Document	248 B	3/27/2020 10:25:03 AM
out.txt	out.txt	Text Document	2 KB	3/27/2020 3:52:44 PM
out2.txt	out2.txt	Text Document	4 KB	3/27/2020 4:02:56 PM
out3.txt	out3.txt	Text Document	2 KB	3/27/2020 9:08:49 PM
out4.txt	out4.txt	Text Document	6 KB	3/27/2020 11:42:24 PM
out5.txt	out5.txt	Text Document	30 KB	3/28/2020 7:56:33 AM

Files sorted in Ascending Order.

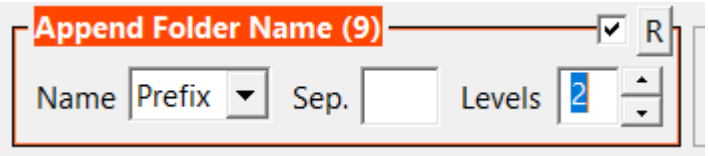
Uncategorized

Move Files based on Creation Date to another Directory cont.

4. Select (manually) all files that are older than March 2020.

Name	New Name	Sub Dir.	Type	Size	Created
Receipt_php.txt	Receipt_php.txt	\test b\adjus...	Text Document	2 KB	8/20/2012 8:27:17 AM
1863_22_10_2019.txt	1863_22_10_2019.txt	\test e	Text Document	6 KB	1/13/2020 5:50:00 PM
mike.txt	mike.txt	\test d	Text Document	6 KB	1/13/2020 5:50:00 PM
old.txt	old.txt	\test d	Text Document	168 B	2/6/2020 2:22:56 PM
new.txt	new.txt	\test d	Text Document	124 B	2/6/2020 2:24:15 PM
test.txt	test.txt	\test e	Text Document	1 KB	2/7/2020 11:10:50 PM
newnames.txt	newnames.txt	\test b\Sub ...	Text Document	134 B	2/17/2020 10:22:21 PM
oldnames.txt	oldnames.txt	\test b\Sub ...	Text Document	126 B	2/17/2020 9:00:35 PM
__BRU_TextFile.txt	__BRU_TextFile.txt	\test b\Sub ...	Text Document	123 B	2/17/2020 11:06:57 PM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt		Text Document	1 KB	3/1/2020 10:39:18 AM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt	\test d	Text Document	1 KB	3/1/2020 10:39:18 AM
help.txt	help.txt	\test d	Text Document	5 KB	3/21/2020 12:04:51 PM
abc123.....txt	abc123.....txt		Text Document	4 KB	3/25/2020 12:38:58 PM
dome.txt	dome.txt	\test e	Text Document	248 B	3/27/2020 10:25:03 AM
out.txt	out.txt	\test e	Text Document	2 KB	3/27/2020 3:52:44 PM
out2.txt	out2.txt	\test e	Text Document	4 KB	3/27/2020 4:02:56 PM
out3.txt	out3.txt		Text Document	2 KB	3/27/2020 9:08:49 PM
out4.txt	out4.txt		Text Document	6 KB	3/27/2020 11:42:24 PM
out5.txt	out5.txt		Text Document	30 KB	3/28/2020 7:56:33 AM

5. 'Selection #9: Append Folder Name' –



Name: Prefix
Levels: 2

a. This is the key to having BRU perform the Move. Typically, BRU couldn't do a move by Creation date or any other date criteria for that matter but by affixing the Directory names to the files, it does two things:

1. Allows BRU to move the files because BRU can move files after a renaming function.

2. By renaming the files using the original Directories' names, it provides a better archival system for keeping track of the original file's locations than simply just moving files out.

Uncategorized

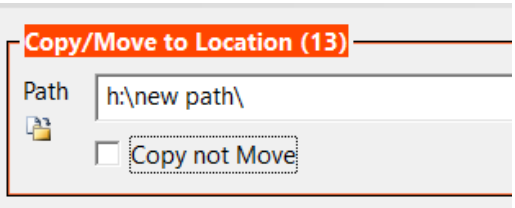
Move Files based on Creation Date to another Directory cont.

5. 'Selection #9: Append Folder Name' – cont.

b. Files showing the appended Folder names.

Name	New Name	Sub Dir.	Type	Size	Created
Receipt_php.txt	test badjustedReceipt_php.txt	\test b\adjus...	Text Document	2 KB	8/20/2012 8:27:17 AM
1863_22_10_2019.txt	Level1test e1863_22_10_2019.txt	\test e	Text Document	6 KB	1/13/2020 5:50:00 PM
mike.txt	Level1test dmike.txt	\test d	Text Document	6 KB	1/13/2020 5:50:00 PM
old.txt	Level1test dold.txt	\test d	Text Document	168 B	2/6/2020 2:22:56 PM
new.txt	Level1test dnew.txt	\test d	Text Document	124 B	2/6/2020 2:24:15 PM
test.txt	Level1test etest.txt	\test e	Text Document	1 KB	2/7/2020 11:10:50 PM
oldnames.txt	Sub Dir 1 Level 1Sub Dir Level 2oldnames.txt	\test b\Sub ...	Text Document	126 B	2/17/2020 9:00:35 PM
newnames.txt	Sub Dir 1 Level 1Sub Dir Level 2newnames.txt	\test b\Sub ...	Text Document	134 B	2/17/2020 10:22:21 PM
__BRU_TextFile.txt	Sub Dir 1 Level 1Sub Dir Level 2__BRU_TextFile.txt	\test b\Sub ...	Text Document	123 B	2/17/2020 11:06:57 PM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt		Text Document	1 KB	3/1/2020 10:39:18 AM
0 123 2 abc 3 456 7.txt	0 123 2 abc 3 456 7.txt	\test d	Text Document	1 KB	3/1/2020 10:39:18 AM
help.txt	help.txt	\test d	Text Document	5 KB	3/21/2020 12:04:51 PM
abc123.....txt	abc123.....txt		Text Document	4 KB	3/25/2020 12:38:58 PM
dome.txt	dome.txt	\test e	Text Document	248 B	3/27/2020 10:25:03 AM
out.txt	out.txt	\test e	Text Document	2 KB	3/27/2020 3:52:44 PM
out2.txt	out2.txt	\test e	Text Document	4 KB	3/27/2020 4:02:56 PM
out3.txt	out3.txt		Text Document	2 KB	3/27/2020 9:08:49 PM
out4.txt	out4.txt		Text Document	6 KB	3/27/2020 11:42:24 PM
out5.txt	out5.txt		Text Document	30 KB	3/28/2020 7:56:33 AM

6. 'Section #13: Copy/Move to Location' -



Path: <path you want to move files to>
Copy Not Move: Disable

Tim Notes:

1. This moves all of the renamed files to a single location.

Forum Contributor: GMA

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files

Using First Numeric Digit to base Even/Odd

Observations:

1#### to 1####_L.jpg,
3#### to 3####_L.jpg,
2#### to 2####_R.jpg
4#### to 4####_R.jpg

1. A file beginning with an odd number is renamed with a suffix of '_L' and an even number is renamed with '_R'
2. Pattern established: ### can be any number of numeric digits followed by any <text>.
3. The evaluation of an even or odd number is based on the first numeric digit of a numeric group (one or more numeric digits with following text).

Sample Base:

w f- 00086 jerry022	nc_017733	8259438 some text	1	2	2019-01-31_201149
123d56789abcde12345	b123456789adec	25032020_this-should	3	4	0 123 2 abc 3 456 7
100867	106	2018-12-04_142308	5	10	

044514_Freeware made by Emiel Wieldraaijer
081604_What is UPN (User Principal Name) in Office 365 - User Guide

To Rename using Entire String:

Match: ^(?=[02468])(.*)
Replace: \1_R

Name	New Name
w f- 00086 jerry022	w f- 00086 jerry022
0 123 2 abc 3 456 7	0 123 2 abc 3 456 7_R
044514_Freeware made by Emiel Wieldraaijer	044514_Freeware made by Emiel Wieldraaijer_R
081604_What is UPN (User Principal Name) in Office ...	081604_What is UPN (User Principal Name) in Office 365 - User Guide_R
1	1
10	10
100867	100867
106	106
123d56789abcde12345	123d56789abcde12345
2	2_R
2018-12-04_142308	2018-12-04_142308_R
2019-01-31_201149	2019-01-31_201149_R
25032020_this-should	25032020_this-should_R
3	3
4	4_R
5	5
8259438 some text	8259438 some text_R
b123456789adec	b123456789adec
nc_017733	nc_017733

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using First Numeric Digit to base Even/Odd cont.

To Rename using Entire String: cont.

Match: ^(?=[02468])(.*)

Sample:

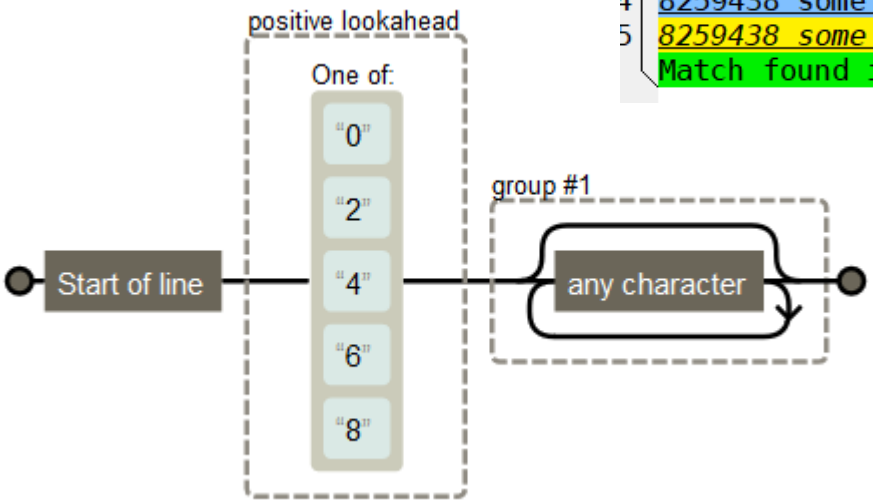
```
.............
Match 1: 206
Group 1: 206
```

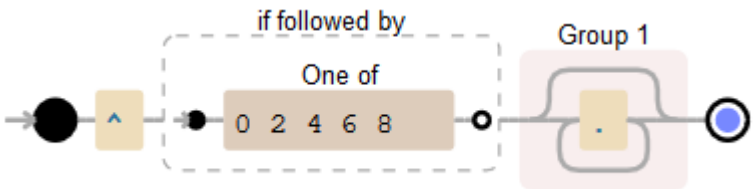
```
Beginning match attempt at character 0
1 ok
2 2
3 ok
4 206
5 206
Match found in 5 steps
```

Sample:

```
.............................
Match 1: 8259438 some text
Group 1: 8259438 some text
```

```
Beginning match attempt at character 0
1 ok
2 8
3 ok
4 8259438 some text
5 8259438 some text
Match found in 5 steps
```





Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using First Numeric Digit to base Even/Odd cont.

To Rename using Entire String: cont.

Using: 8259438 some text

Match: ^(?=[02468])(.*)

Where:

\1 = 8259438 some text

Tim Analysis:

1. ^	Start at beginning of string (BOL)
2. ?=[02468]	Uses a Look ahead to test first character position for class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. = '8' = True

Tim Notes:

1. This is how it determines whether or not the first numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the first character is not one of these values then the RegEx Fails.

3. (.*)	Match against the remainder of the string.	Capture Group 1 = 8259438 some text

Tim Notes:

1. You must remember that a Lookahead is a Non-Marking Group. It does not typically Capture values. The values held are temporary. That is why the (.*) is used to capture the value.

2. See RegEx Manual Volume I in Bulk Rename Utility Operations Manual Volume 1 and Volume II of the edition can be found in the Appendix here for further information.

3. Renaming using the entire string does not clearly 'display' the 'odd' or 'even' aspect of the example. This was done in the spirit of the poster's original request but I find it not practical. Read on.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using First Numeric Digit to base Even/Odd cont.

Observations:

1#### to 1####_L.jpg,
3#### to 3####_L.jpg,
2#### to 2####_R.jpg
4#### to 4####_R.jpg

To Rename using just first numeric digit:

Match: ^(?=[02468])(.)
Replace: \1_R

Name	New Name
w f- 00086 jerry022	w f- 00086 jerry022
0 123 2 abc 3 456 7	0_R
044514_Freeware made by Emiel Wieldraaijer	0_R
081604_What is UPN (User Principal Name) in Office ...	0_R
1	1
10	10
100867	100867
106	106
123d56789abcde12345	123d56789abcde12345
2	2_R
2018-12-04_142308	2_R
2019-01-31_201149	2_R
25032020_this-should	2_R
3	3
4	4_R
5	5
8259438 some text	8_R
b123456789adec	b123456789adec
nc_017733	nc_017733

This captures only the first character.

Tim Notes:

1. Visualizations and other diagrams are virtually the same, so won't be repeated here. The only difference is that (.) is used to capture the first character rather than using the Greedy Quantifier, (.*).

2. Analysis is virtually the same.

3. This clearly displays the "odd' or 'even' aspect of the numeric.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Second Numeric Digit to base Even/Odd Example #1

Observations:

13### to 13###_L.jpg,
33### to 33###_L.jpg,
20### to 20###_R.jpg
46### to 46###_R.jpg

1. A file beginning with an odd number is renamed with a suffix of '_L' and an even number is renamed with '_R'
2. Pattern established: ### can be any number of numeric digits followed by any <text>.
3. The evaluation of an even or odd number is based on the second numeric digit of a numeric group.
4. There must be a minimum of two numeric digits to match.

Sample Base:

w f- 00086 jerry022	nc_017733	8259438 some text	1	2	2019-01-31_201149
123d56789abcde12345	b123456789adec	25032020_this-should	3	4	0 123 2 abc 3 456 7
100867	106	2018-12-04_142308	5	10	

044514_Freeware made by Emiel Wieldraaijer
081604_What is UPN (User Principal Name) in Office 365 - User Guide

Match: ^(?=\d[02468])(.{2})
Replace: \1_R

Name	New Name
w f- 00086 jerry022	w f- 00086 jerry022
0 123 2 abc 3 456 7	0 123 2 abc 3 456 7
044514_Freeware made by Emiel Wieldraaijer	04_R
081604_What is UPN (User Principal Name) in Office ...	08_R
1	1
10	10_R
100867	10_R
106	10_R
123d56789abcde12345	12_R
2	2
2018-12-04_142308	20_R
2019-01-31_201149	20_R
25032020_this-should	25032020_this-should
3	3
4	4
5	5
8259438 some text	82_R
b123456789adec	b123456789adec
nc_017733	nc_017733

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Second Numeric Digit to base Even/Odd Example #1 cont.

Match: ^(?=\d[02468])(.{2})

Sample: 8259438 some text

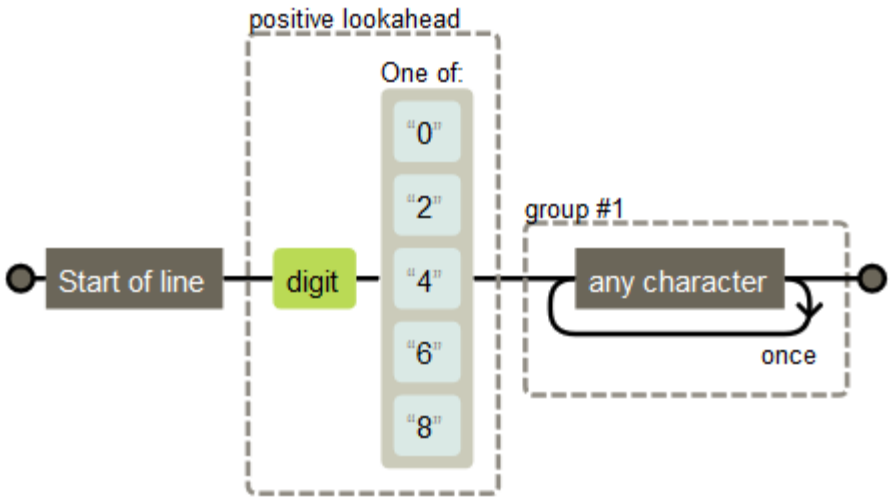
```
.............
Match 1: 82
Group 1: 82
```

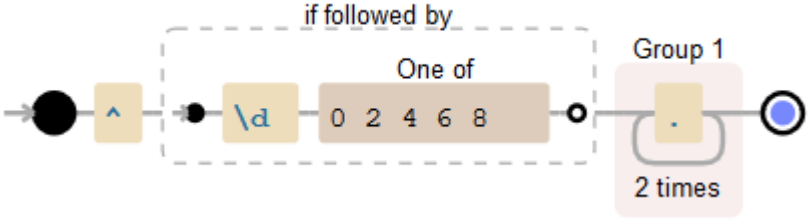
```
Beginning match attempt at character 0
1 ok
2 8
3 82
4 ok
5 82
6 82
Match found in 6 steps
```

Sample: 2018-12-04_142308

```
.............
Match 1: 20
Group 1: 20
```

```
Beginning match attempt at character 0
1 ok
2 2
3 20
4 ok
5 20
6 20
Match found in 6 steps
```





Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Second Numeric Digit to base Even/Odd Example #1 cont.

Using: 8259438 some text

Match: ^(?=\d[02468])(.{2})

Where:

\1 = 82

Tim Analysis:

1. ^	Start at beginning of string (BOL)
2. \d	Match against Numeric Digit = 8. This is the first character position
3. ?=[02468]	Uses a Look ahead to test second character position for class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. = '2' = True

Tim Notes:

1. This is how it determines whether or not the second numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the second character is not one of these values then the RegEx Fails.

4. (.)	Match against any character	Capture Group 1 = 8
5. {2}	Repeat last expression to match 2 iterations.	Capture Group 1 = 82

Tim Notes:

1. You must remember that a Lookahead is a Non-Marking Group. It does not typically Capture values. The values held are temporary. That is why the (.{2}) is used to capture the value.

2. See RegEx Manual Volume I in Bulk Rename Utility Operations Manual Volume 1 and Volume II of the edition can be found in the Appendix here for further information.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Second Numeric Digit to base Even/Odd Example #2

Observations:

13### to 13###_L.jpg,
33### to 33###_L.jpg,
20### to 20###_R.jpg
46### to 46###_R.jpg

1. A file beginning with an odd number is renamed with a suffix of '_L' and an even number is renamed with '_R'
2. Pattern established: ### can be any number of numeric digits followed by any <text>.
3. The evaluation of an even or odd number is based on the second numeric digit of a numeric group.
4. There must be a minimum of two numeric digits to match.

Sample Base:

w f- 00086 jerry022	nc_017733	8259438 some text	1	2	2019-01-31_201149
123d56789abcde12345	b123456789adec	25032020_this-should	3	4	0 123 2 abc 3 456 7
100867	106	2018-12-04_142308	5	10	

044514_Freeware made by Emiel Wieldraaijer
081604_What is UPN (User Principal Name) in Office 365 - User Guide

Match: ^(\d[02468])
Replace: _R

Name	New Name
w f- 00086 jerry022	w f- 00086 jerry022
0 123 2 abc 3 456 7	0 123 2 abc 3 456 7
044514_Freeware made by Emiel Wieldraaijer	04_R
081604_What is UPN (User Principal Name) in Office ...	08_R
1	1
10	10_R
100867	10_R
106	10_R
123d56789abcde12345	12_R
2	2
2018-12-04_142308	20_R
2019-01-31_201149	20_R
25032020_this-should	25032020_this-should
3	3
4	4
5	5
8259438 some text	82_R
b123456789adec	b123456789adec
nc_017733	nc_017733

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Second Numeric Digit to base Even/Odd Example #2 cont.

Match: ^(\d[02468])

Sample: 8259438 some text

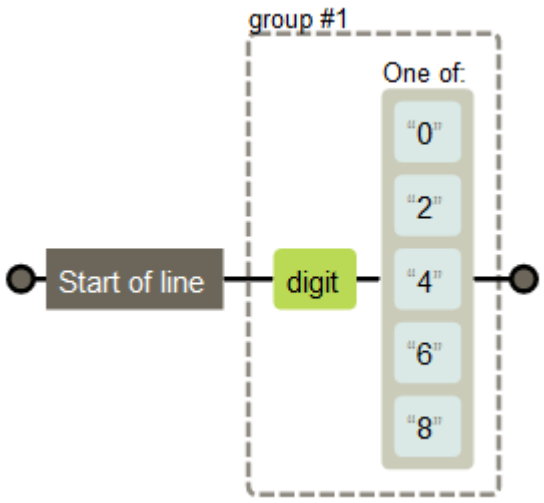
```
.............
Match 1: 82
Group 1: 82
```

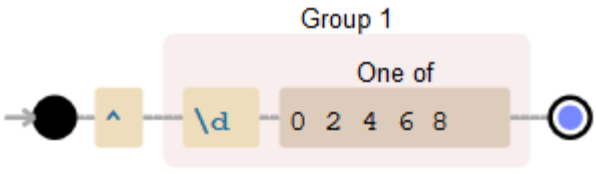
```
Beginning match attempt at character 0
1 ok
2 8
3 82
4 82
Match found in 4 steps
```

Sample: 2018-12-04_142308

```
.............
Match 1: 20
Group 1: 20
```

```
Beginning match attempt at character 0
1 ok
2 2
3 20
4 20
Match found in 4 steps
```





Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Second Numeric Digit to base Even/Odd Example #2 cont.

Using: 8259438 some text

Match: ^(\d[02468])

Where:

\1 = 82

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. \d	Match against Numeric Digit This is the first character position.	Capture Group 1 = 8
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'.	Capture Group 1 = 82

Tim Notes:

1. This is how it determines whether or not the second numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the second character is not one of these values then the RegEx Fails.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Observations:

167##	to	167##_L.jpg,
339##	to	339##_L.jpg,
538##	to	538##_R.jpg
460##	to	460##_R.jpg

1. A file beginning with an odd number is renamed with a suffix of '_L' and an even number is renamed with '_R'
2. Pattern established: ### can be any number of numeric digits followed by any <text>.
3. The evaluation of an even or odd number is based on the third numeric digit of a numeric group.
4. There must be a minimum of three numeric digits to match.

Sample Base:

w f- 00086 jerry022	nc_017733	8369438 some text	1	2	2019-01-31_201149
123d56789abcde12345	b123456789adec	25032020_this-should	3	4	0 123 2 abc 3 456 7
100867	106	2018-12-04_142308	5	10	

044514_Freeware made by Emiel Wieldraaijer
081604_What is UPN (User Principal Name) in Office 365 - User Guide

Using Third Numeric Digit to base Even/Odd

Match: ^(?=(\d{2})[02468])(.{3})
Replace: \1_R

Name	New Name
w f- 00086 jerry022	w f- 00086 jerry022
0 123 2 abc 3 456 7	0 123 2 abc 3 456 7
044514_Freeware made by Emiel Wieldraaijer	044_R
081604_What is UPN (User Principal Name) in Office ...	081604_What is UPN (User Principal Name) in Office 365 - User Guide
1	1
10	10
100867	100_R
106	106_R
123d56789abcde12345	123d56789abcde12345
2	2
2018-12-04_142308	2018-12-04_142308
2019-01-31_201149	2019-01-31_201149
25032020_this-should	250_R
3	3
4	4
5	5
8259438 some text	8259438 some text
b123456789adec	b123456789adec
nc_017733	nc_017733

Tim Notes:

1. This example is to show how easy it is to determine which numeric digits you want to base the evaluation by using a Lookahead to determine the value and the Capture Group at the end of the RegEx to capture the value.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Third Numeric Digit to base Even/Odd cont.

Match: ^(?=(\d{2})[02468])(.{3})

Sample: 8369438 some text.jpg

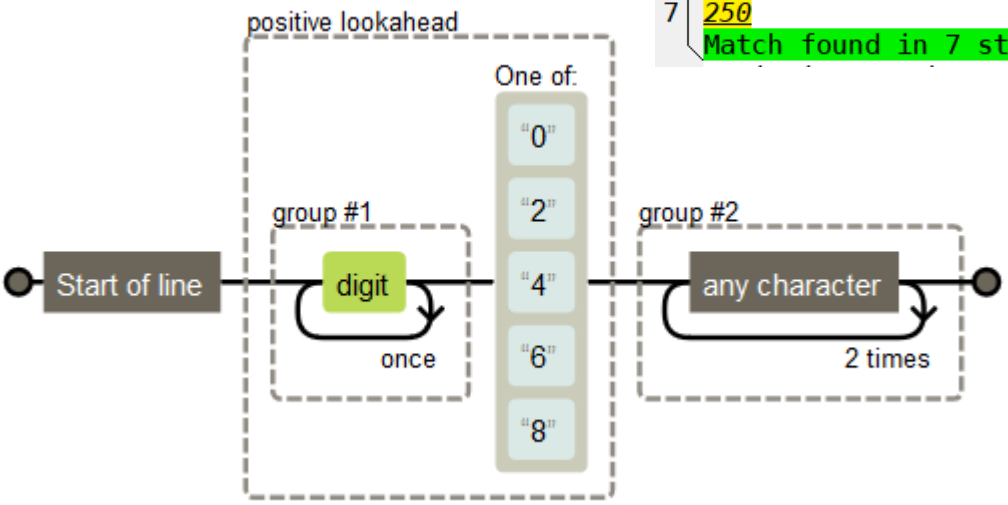
```
..............
Match 1: 836
Group 1: 83
Group 2: 836
```

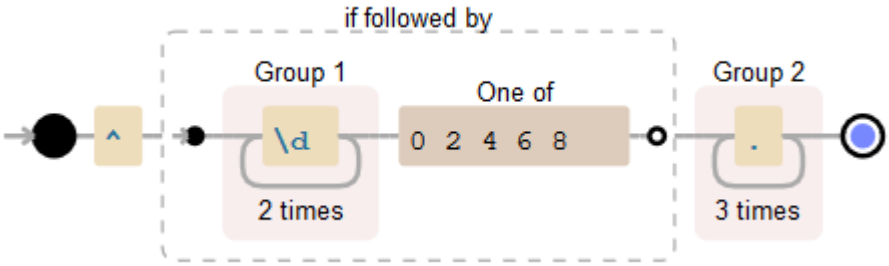
```
Beginning match attempt at character 0
1 ok
2 83
3 83
4 836
5 ok
6 836
7 836
Match found in 7 steps
```

Sample: 25032020_this-should

```
..............
Match 1: 250
Group 1: 25
Group 2: 250
```

```
Beginning match attempt at character 0
1 ok
2 25
3 25
4 250
5 ok
6 250
7 250
Match found in 7 steps
```





Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'
2. Use (.*) if you want to include entire string in New Name.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Using Third Numeric Digit to base Even/Odd cont.

Using: 8369438 some text.jpg

Match: ^(?=(\d{2})[02468])(.{3})

Where:

\1 = 836

Tim Analysis:

1. ^	Start at beginning of string (BOL)
2. \d	Match against Numeric Digit = 8
3. {2}	Repeat last expression to match 2 iterations = 83 This is the first and second character position.
4. ?=[02468]	Uses a Look ahead to test third character position for class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. = '6' = True

Tim Notes:

1. This is how it determines whether or not the third numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the third character is not one of these values then the RegEx Fails.

4. (.)	Match against any character	Capture Group 1 = 8
5. {3}	Repeat last expression to match 3 iterations.	Capture Group 1 = 836

Tim Notes:

1. You must remember that a Lookahead is a Non-Marking Group. It does not typically Capture values. The values held are temporary. That is why the (.{3}) is used to capture the value.

2. See RegEx Manual Volume I in Bulk Rename Utility Operations Manual Volume 1 and Volume II of the edition can be found in the Appendix here for further information.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Observations:

1	to	1_R	10	to	10_R
3	to	3_L	105	to	105_L
5	to	5_L			
4	to	4_R			

1. A file beginning with an odd number is renamed with a suffix of '_L' and an even number is renamed with '_R'
2. Pattern established: ### can be any number of numeric digits not preceded or followed by any <text>.
3. The evaluation of an even or odd number is based on the last numeric digit of a numeric group at EOL.
4. Numeric group can be made up of one or more numeric digits.

Sample Base:

1	5	9	106
2	6	10	78056
3	7	52	100867
4	8	105	

For Whole Numbers with no preceding or following text

i.e.,

10568 = even 10569 = odd 10 = even 5 = odd

Match: (^\d*[02468])$
Replace: \1_R

Name	New Name
1	1
2	2_R
3	3
4	4_R
5	5
6	6_R
7	7
8	8_R
9	9
10	10_R
52	52_R
105	105
106	106_R
78056	78056_R
100867	100867

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

For Whole Numbers with no preceding or following text cont.

Match: (^\d*[02468])$

Sample:

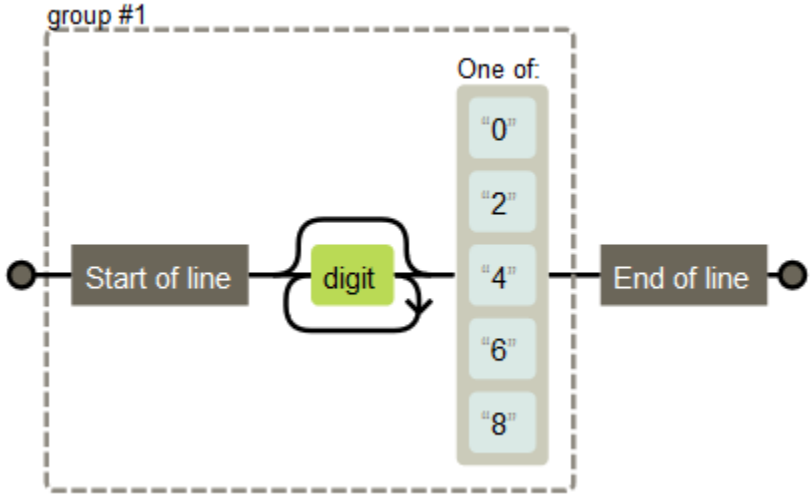
```
·············Start·Length····
Match 1: 2       0       1
Group 1: 2       0       1
```

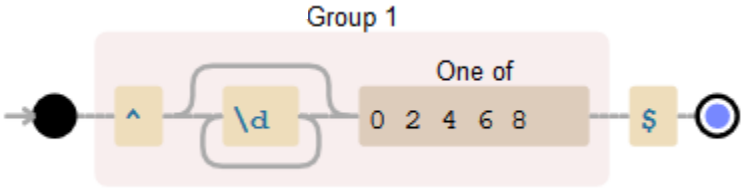
```
Beginning match attempt at character 0
1 ok
2 2
3 2backtrack
4 ok
5 2
6 2
7 2ok
Match found in 7 steps
```

Sample:

```
··············Start·Length··
Match 1: 106       0       3
Group 1: 106       0       3
```

```
Beginning match attempt at character 0
1 ok
2 106
3 106backtrack
4 10
5 106
6 106
7 106ok
Match found in 7 steps
```





Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

For Whole Numbers with no preceding or following text cont.

Using: 106

Match: (^\d*[02468])$

Where:

\1 = 106

Tim Analysis:

1. ^	Start at beginning of string (BOL) This is the verification that there is no preceding text before the numeric digits.	
2. \d	Match against Numeric Digit	Capture Group 1 = 1
3. *	Make it Greedy Current position = EOL	Capture Group 1 = 106 (entire string)
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. Backtracks to match against '6' = True	

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the last character is not one of these values then the RegEx Fails.

4. $	Must be at EOL This is the verification that there is no following text.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string

Observations:

<text>###1 <text> to ###1_R
<text>###3 <text> to ###3_L
<text>###5 <text> to ###5_L
<text>###4 <text> to ###4_R

1. A numeric group ending with an odd number is renamed with a suffix of '_L' and an even number with an '_R'
2. Pattern established: ### can be any number of numeric digits optionally preceded but must be followed by <text>.
3. The evaluation of an even or odd number is based on the last numeric digit of a numeric group.
4. Numeric group can be anywhere in string.
5. If there is more than one numeric group, only the first will be matched.
6. Numeric group must have a minimum of two numeric digits.

Sample Base:

some text 8259422 some more	8259419 some text	8259420 some text	8259422 some text
8259435 some text	8259436 some text	8259437 some text	8259438 some text
8259439 some text	8259440 some text	8259441 some text	8259442 some text
8259443 some text	8259444 some text	8259445 some text	8259446 some text
8259447 some text	8259448 some text	8259449 some text	8259450 some text
8259451 some text	some text -8259421	some text -8259423	some text -8259424

Match: (\d*[02468])+([^\d])
Replace: \1_R

Name	New Name
some text 8259422 some more.jpg	8259422_R.jpg
8259419 some text.jpg	8259419 some text.jpg
8259420 some text.jpg	8259420_R.jpg
8259422 some text.jpg	8259422_R.jpg
8259435 some text.jpg	8259435 some text.jpg
8259436 some text.jpg	8259436_R.jpg
8259437 some text.jpg	8259437 some text.jpg
8259438 some text.jpg	8259438_R.jpg
8259439 some text.jpg	8259439 some text.jpg
8259440 some text.jpg	8259440_R.jpg
8259441 some text.jpg	8259441 some text.jpg
8259442 some text.jpg	8259442_R.jpg
8259443 some text.jpg	8259443 some text.jpg
8259444 some text.jpg	8259444_R.jpg
8259445 some text.jpg	8259445 some text.jpg
8259446 some text.jpg	8259446_R.jpg
8259447 some text.jpg	8259447 some text.jpg
8259448 some text.jpg	8259448_R.jpg
8259449 some text.jpg	8259449 some text.jpg
8259450 some text.jpg	8259450_R.jpg
8259451 some text.jpg	8259451 some text.jpg
some text -8259421.jpg	some text -8259421.jpg
some text -8259423.jpg	some text -8259423.jpg
some text -8259424.jpg	some text -8259424.jpg

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string cont.

Match: (\d*[02468])+([^\d])

Sample: 8259436 some text.jpg

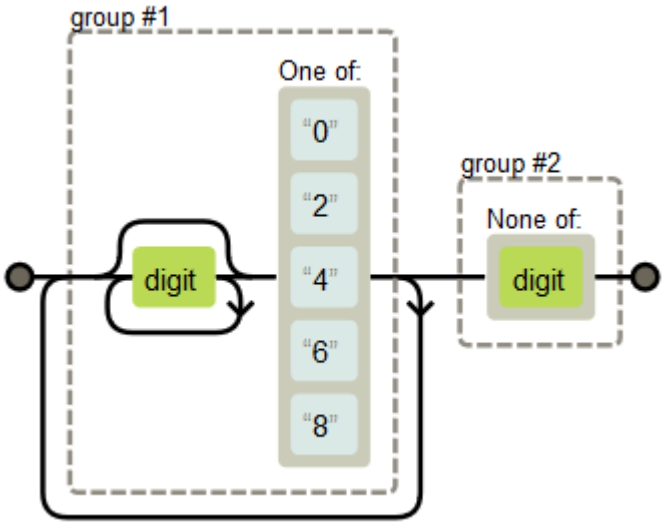
	Start	Length
Match 1:	8259436	0	8
Group 1:	8259436	0	7
Group 2:		7	1

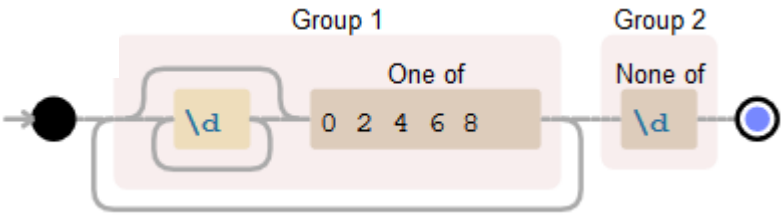
```
Beginning match attempt at character 0
1 8259436
2 8259436backtrack
3 825943
4 8259436
5 8259436ok
6 8259436backtrack
7 8259436
8 8259436
9 8259436
Match found in 9 steps
```

Sample: some text 8259420 some more.jpg

	Start	Length
Match 1:	8259420	10	8
Group 1:	8259420	10	7
Group 2:		17	1

```
Beginning match attempt at character 10
1 8259420
2 8259420backtrack
3 825942
4 8259420
5 8259420ok
6 8259420backtrack
7 8259420
8 8259420
9 8259420
Match found in 9 steps
```





Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string cont.

Using: some text 8259420 some more.jpg

Match: (\d*[02468])+([^\d])

Where:

\1 = 8259420 \2 = <space>

Tim Analysis:

1. \d	Match against Numeric Digit	Capture Group 1 = 8
2. *	Make it Greedy	Capture Group 1 = 8259420
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. Backtracks to match against '0' = True	

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx continues to backtrack.

4. +	Make the expression Greedy	Capture Group 1 = 825940

Tim Notes:

If the result does not change from step 3 to step 4, why add the next section, ([^\d])?

Because if this sub expression was not there, it would match against any even combination of numbers without regard to a numeric group.

i.e.,

Matches:

Etc.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string cont.

Using: some text 8259420 some more.jpg

Match: (\d*[02468])+([^\d])

Where:

\1 = 8259420 \2 = <space>

Tim Analysis: cont.

5. [^\d]	Match against a Negated Class consisting of a numeric digit, \d.	Capture Group 2 = <space>

Tim Notes:

1. This stops the matching at the first non-numeric digit. Thus the matching ends at the end of the numeric group.
2. Could also have used, (\d*[02468])([^\d]+) – the only difference is the placement of the Greedy Quantifier '+'. The result is the same.
3. If I want to capture the following text I can use a Greedy Quantifier after the sub expression, i.e. [^\d]* or [^\d]+.
4. If I want to capture both the preceding and following text, I can just apply the same sub-expression to both ends:

([^\d]*)(\d*[02468])([^\d]*)

some text 8259420 some more.jpg

Where:

\1 = some text <space> \2 = 8259420 \3 = <space> some more

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

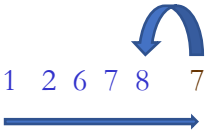
First even Numeric Digit matched located by moving back from end of First Numeric Group

2 Examples.

Observations:

1. both require minimum 2 numeric digits to match
2. locates first numeric group anywhere in string.
3. Matches against entire sequence then backtracks to match against first even numeric digit.

i.e. Using: 126787



Capture Group 2 = 12678
Capture Group 3 = 7

Sample Base:

w f- 00088 jerry028	0 123 2 abc 3 456 7	044543_Freeware made by Emiel Wieldraaijer
100867	106	1990-04-25_181845
2	2019-03-24_185117	some text 2019-09-28_135859
3	videofile99.eng	Word1-0002-Word2

Example #1

Match: (.*?)(\d*[02468])+(.*\w*)
Replace: \2

Where:

\1 = Preceding Text (optional)
\2 = Portion of numeric group that matched
\3 = Remainder of string including any remaining portion of numeric group after match

	New Name
w f- 00088 jerry028.jpg	00088 .jpg
0 123 2 abc 3 456 7	0
044543_Freeware made by Emiel Wieldra...	04454 .jpg
100867	10086
106	106
1990-04-25_181845.jpg	1990 .jpg
2	2
2019-03-24_185117.jpg	20 .jpg
3	3
some text 2019-09-28_135859.jpg	20 .jpg
videofile99.eng.srt	videofile99.eng.srt
Word1-0002-Word2.txt	0002 .txt

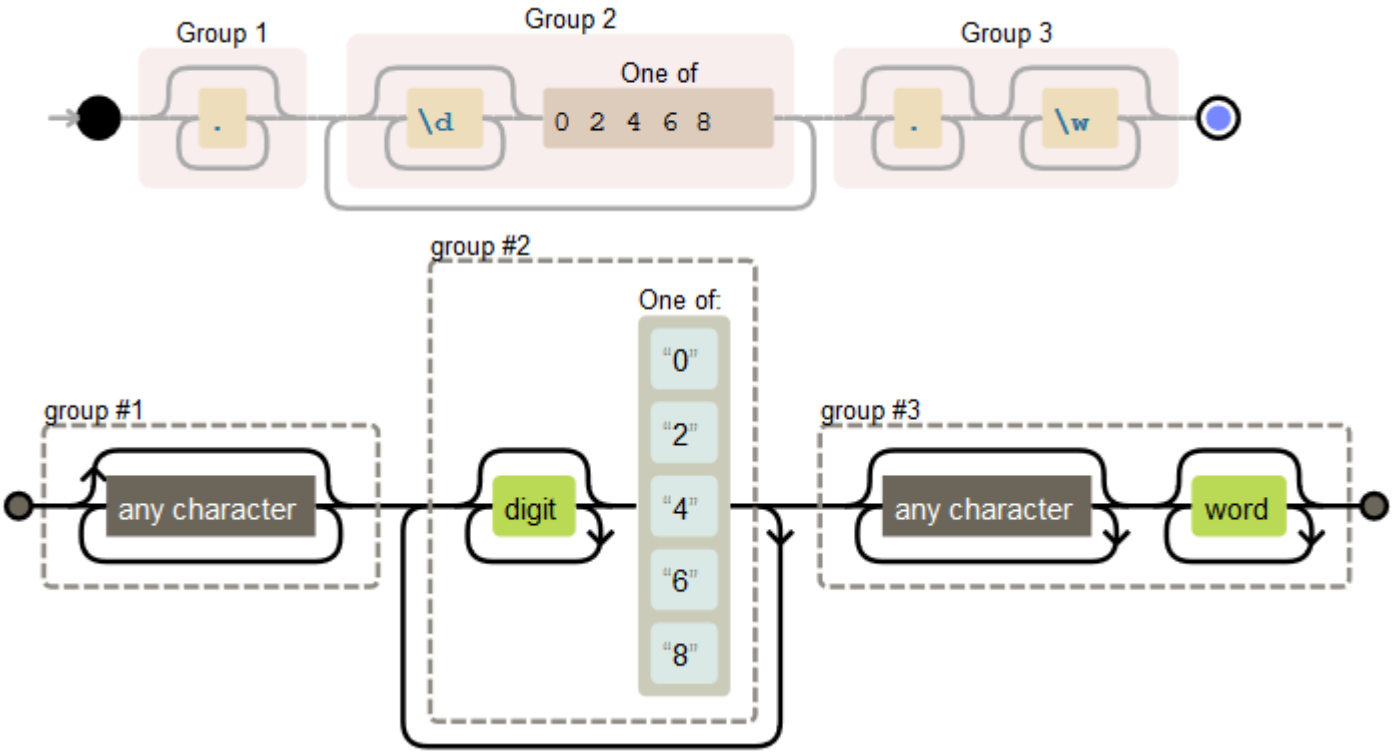
Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)



Sample:

```
·····Start·Length··
Match 1: 126787      0       6
Group 1:             0       0
Group 2: 12678       0       5
Group 3: 7           5       1
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  126787
4  126787backtrack
5  12678
6  12678backtrack
7  1267
8  12678
9  126787
10 126787backtrack
11 12678ok
12 12678backtrack
13 12678
14 126787
15 126787ok
16 126787
   Match found in 16 steps
```

Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]'.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: w f- 00088 jerry028

```
..............................
Match 1: w f- 00088 jerry028
Group 1: w f-
Group 2: 00088
Group 3:  jerry028
```

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = w f-<space> \2 = 00088 \3 = <space>jerry0028

Tim Analysis:

1..	Match against any character	Capture Group 1 = w
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \d	Match against Numeric Digit	Capture Group 2 = 0
		Changes Capture Group 1 = w f-<space>
2. *	Make it Greedy	Capture Group 2 = 00088
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', 6' or an '8'. Backtracks to match against '8' = True	Capture Group 2 = Unchanged

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx continues to backtrack.

4. +	Make the expression Greedy	Capture Group 2 = Unchanged
5..	Match against any character	Capture Group 3 = <space>
6. *	Make it Greedy	Capture Group 3 = <space>jerry028

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: w f- 00088 jerry028

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = w f-<space> \2 = 00088 \3 = <space>jerry0028

Tim Analysis: cont.

Tim Notes:

1. In order to match up against whole numbers, the following was added. The \w captures the first numeric digit after the initial match and the Greedy Quantifier captures the remainder (if any – if there is none, then Capture Group 3 = null):

7. \w Match against Word Character All Capture Groups = Unchanged
8. * Make it Greedy

Tim Notes:

1. When analyzing a RegEx, I look at the Captured values and I assign the Capture Groups that hold those values with descriptions of 'placeholders' – each 'reserved' for a particular piece of information. This not only helps me in determining what something is doing but also helps in debugging a RegEx that for whatever reason is not working as it should.

   In this case I can say:

   Capture Group 1 = Preceding text
   Capture Group 2 = Matched even value from first numeric group in string.
   Capture Group 3 = Remainder of string including any value left over after the match

2. At first it looks like a redundancy adding the \w+ when the (.*) by itself is enough to satisfy, but if you apply the RegEx against a string, '028', using just, .* will result in null value, whereas adding the \w+ results in the capture of '8' as it should be.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: 100867

	Start	Length
Match 1: 100867	0	6
Group 1:	0	0
Group 2: 10086	0	5
Group 3: 7	5	1

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = null \2 = 10086 \3 = 7

Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \d	Match against Numeric Digit	Capture Group 2 = 1
		Changes Capture Group 1 = null
2. *	Make it Greedy	Capture Group 2 = 100867
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', 6' or an '8'. Backtracks to match against '8' = True	Capture Group 2 = 10086

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx continues to backtrack.

4. +	Make the expression Greedy Current position = EOL	Capture Group 2 = Unchanged
5..	Match against any character	Capture Group 3 = No Value
6. *	Make it Greedy	Capture Group 3 = Unchanged

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: 100867

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = null \2 = 10086 \3 = 7

Tim Analysis: cont.

Tim Notes:

1. In order to match up against whole numbers, the following was added - the \w captures the first numeric digit after the initial match and the Greedy Quantifier captures the remainder (if any – if there is none, then Capture Group 3 = null):

7. \w	Match against Word Character	Capture Group 3 = 7
8. *	Make it Greedy	Capture Group 3 = Unchanged

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: 106

```
··············Start·Length··
Match 1: 106       0       3
Group 1:           0       0
Group 2: 106       0       3
Group 3:           3       0
```

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = null \2 = 106 \3 = null

Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \d	Match against Numeric Digit	Capture Group 2 = 1
		Changes Capture Group 1 = null
2. *	Make it Greedy	Capture Group 2 = 106
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', 6' or an '8'. Backtracks to match against '8' = True	Capture Group 2 = 106

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx continues to backtrack.

4. +	Make the expression Greedy Current position = EOL	Capture Group 2 = Unchanged
5..	Match against any character	Capture Group 3 = no value
6. *	Make it Greedy	Capture Group 3 = Unchanged

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: 106

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = null \2 = 106 \3 = null

Tim Analysis: cont.

Tim Notes:

1. In order to match up against whole numbers, the following was added - the \w captures the first numeric digit after the initial match and the Greedy Quantifier captures the remainder (if any – if there is none, then Capture Group 3 = null):

7. \w	Match against Word Character	Capture Group 3 = 6 Changes Capture Group 2 = 10
8. *	Make it Greedy	Capture Group 3 = null Changes Capture Group 2 = 106

Tim Notes:

1. Steps 7 and 8 seem redundant but without them, whole numbers would not be included in the match. And the Greedy Quantifier at the end is needed for whole numbers that matched on the last numeric digit of the number. Compare the '106' string with the '10867' string and you will see the necessity.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: some text 2019-09-28_135859.jpg

		Start	Length
Match 1:	some text 2019-09-28_135859	0	27
Group 1:	some text	0	10
Group 2:	20	10	2
Group 3:	19-09-28_135859	12	15

Example #1 cont.

Match: (.*?)(\d*[02468])+(.*\w*)

Where:

\1 = some text \2 = 20 \3 = 19-09-28_135859

Tim Analysis:

The analysis is no different than previous so a full analysis will not be needed.

1. This last sample shows how the match has left a portion of the numeric group. This will be captured in Capture Group 3 along with the remainder of the string.

2. Preceding text is captured in Capture Group 1 = some text

3. Capture Group 2 will capture the matched even numeric digit it finds in the first numeric group not from left to right but backtracking from right to left. How it matches is:

   a. Moves forward to match against the first numeric group = 2019 where the sequence is defined as ending with a non-numeric digit, i.e., the <hyphen>.
      1. Capture Group 2 = 2019

   b. Backtrack to match against the 9 for an even number = False.
      1. Capture Group 2 = 201

   c. Backtrack to match against the 1 for an even number = False.
      1. Capture Group 2 = 20

   d. Backtrack to match against the 0 for an even number = True.

   e. Capture Group 2 = '20' because it captures from the first numeric digit in the group to the matched numeric digit.

4. The Greedy Quantifier captures the last part of the numeric group that was not matched along with the remainder of the string. Capture Group 3 = 19-09-28_135859

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Sample Base:

w f- 00088 jerry028	0 123 2 abc 3 456 7	044543_Freeware made by Emiel Wieldraaijer
100867	106	1990-04-25_181845
2	2019-03-24_185117	some text 2019-09-28_135859
3	videofile99.eng	Word1-0002-Word2

Example #2

Match: (.*?)(\d*[02468])+([\d]*)(.*\w*)
Replace: \2 <space>

Where:

\1 = Preceding Text (optional)
\2 = Portion of numeric group that matched
\3 = portion of numeric group that remains after match if any
\4 = following text (optional)

	New Name
w f- 00088 jerry028.jpg	00088 .jpg
0 123 2 abc 3 456 7	0
044543_Freeware made by Emiel Wieldra...	04454 .jpg
100867	10086
106	106
1990-04-25_181845.jpg	1990 .jpg
2	2
2019-03-24_185117.jpg	20 .jpg
3	3
some text 2019-09-28_135859.jpg	20 .jpg
videofile99.eng.srt	videofile99.eng.srt
Word1-0002-Word2.txt	0002 .txt

Tim Notes.

1. <space> added to show match against whole numbers.

2. The difference between Example #1 and Example #2 is that I isolate the leftover numeric portion after the match from the numeric group – if any, and this is captured in Capture Group 3, rather than let it be swept up in Capture Group 4.

3. The analysis will be basically the same as in Example #1, with the exception of the isolation, so I won't bother with a full analysis on multiple examples as I did with Example #1.

4. Any Capture Group that does not have a text value will be null.

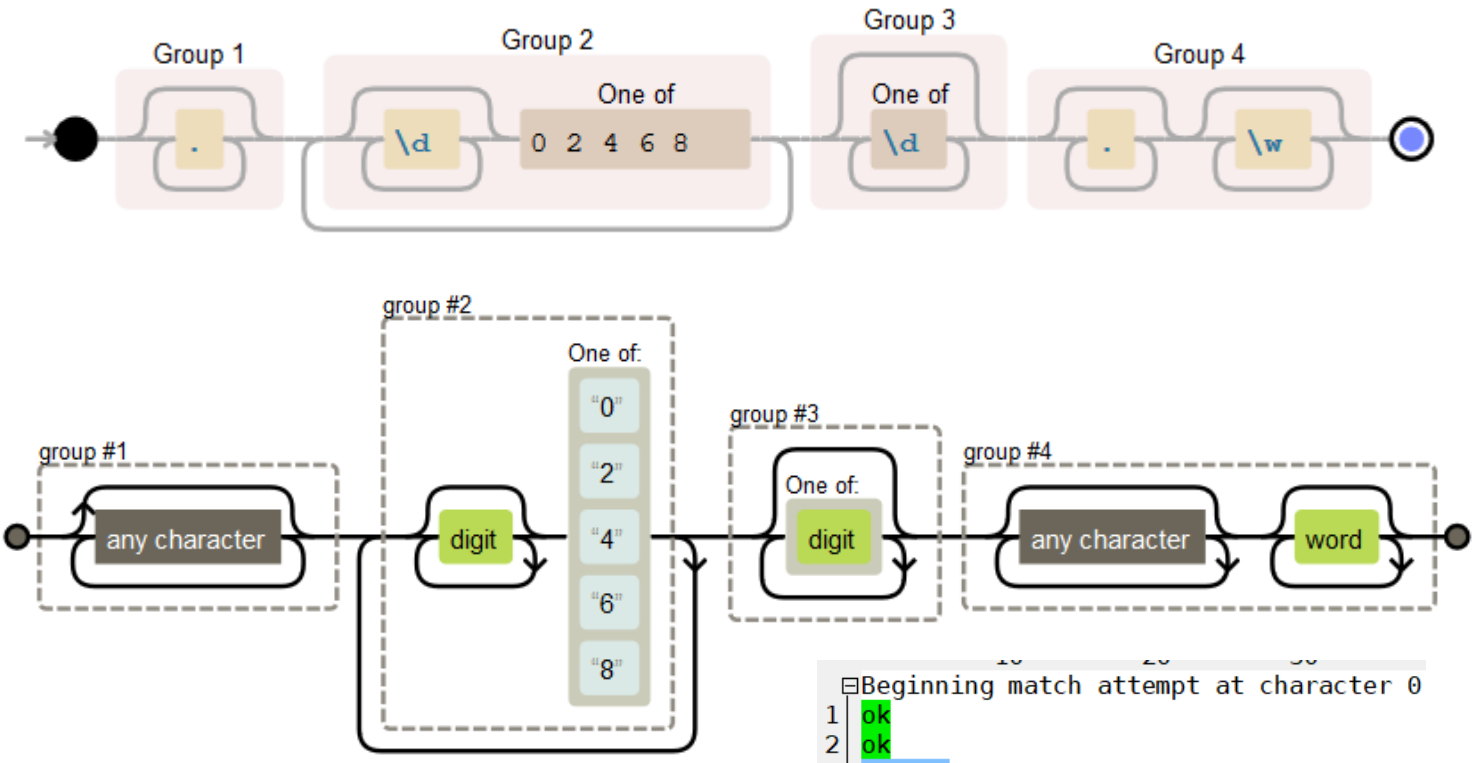
Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Example #2 cont.

Match: (.*?)(\d*[02468])+([\d]*)(.*\w*)



Sample:

```
                  Start  Length
Match 1: 126787       0       6
Group 1:              0       0
Group 2: 12678        0       5
Group 3: 7            5       1
Group 4:              6       0
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  126787
4  126787backtrack
5  12678
6  12678backtrack
7  1267
8  12678
9  126787
10 126787backtrack
11 12678ok
12 12678backtrack
13 12678
14 126787
15 126787
16 126787ok
17 126787ok
18 126787ok
Match found in 18 steps
```

Tim Notes:

1. Not much practicality but my position is to show you enough examples of these 'variations on a theme' progressions so you can really see what effect a Metacharacter's addition or subtraction from a given RegEx has. In doing so, you will learn how best to utilize them in your own programming.
2. To test for Odd Numbers, change the class to '[13579]'.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: 126787

Example #2 cont.

Match: (.*?)(\d*[02468])+([\d]*)(.*\w*)

Where:

\1 = null \2 = 12678 \3 = 7 \4 = null

Tim Analysis:

1..	Match against any character	Capture Group 1 = 1
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \d	Match against Numeric Digit	Capture Group 2 = 1
		Changes Capture Group 1 = null
2. *	Make it Greedy	Capture Group 2 = 126787
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', 6' or an '8'.	Capture Group 2 = 12678
	Backtracks to match against '7' = False	
	Backtracks to match against '8' = True	

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx continues to backtrack.

4. +	Make the expression Greedy	Capture Group 2 = Unchanged
	Current position = EOL	
5. [\d]	Match against a class consisting of a numeric digit. This represents the beginning of any leftover portion of the numeric group after the match in step 3.	Capture Group 3 = 7
6. *	Make it Greedy	Capture Group 3 = Unchanged
	This would match against any remaining numeric digits in this numeric group.	
7..	Match against any character	Capture Group 4 = no value
8. *	Make it Greedy	Capture Group 4 = Unchanged

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

First even Numeric Digit matched located by moving back from end of First Numeric Group cont.

Using: 126787

Example #2 cont.

Match: (.*?)(\d*[02468])+([\d]*)(.*\w*)

Where:

\1 = null \2 = 12678 \3 = 7 \4 = null

Tim Analysis: cont.

Tim Notes:

1. In order to match up against whole numbers, the following was added (the \w captures the first numeric digit after the initial match and the Greedy Quantifier captures the remainder (if any – if there is none, then Capture Group 3 = null):

9. \w	Match against Word Character	Capture Group 4 = Unchanged
10. *	Make it Greedy	Capture Group 4 = null

Tim Notes:

1. Steps 9 and 10 seem redundant but without them, whole numbers would not be included in the match. And the Greedy Quantifier at the end is needed for whole numbers that matched on the last numeric digit of the number.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

<text> ###1 <text> to ###1_R
<text> ###3 <text> to ###3_L
<text> ###5 <text> to ###5_L
<text> ###4 <text> to ###4_R

Sample Base:

Too many to list.

Let's take this even further:

Match against first numeric group anywhere in string and preserve any preceding OR following text

Observations:

1. A Numeric group ending with an odd number is renamed with a suffix of '_L' and an even number is renamed with '_R'
2. Pattern established: ### can be any number of numeric digits optionally preceded and/or followed by <text>.
3. The evaluation of an even or odd number is based on the last numeric digit of a numeric group.
4. Numeric group can be anywhere in string.
5. If there is more than one numeric group, only the first will be matched.
6. Numeric group can have one or more numeric digits.

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)
Replace: \2

Name	New Name
1	1
10	10_R
100867	100867
105	105
106	106_R
2	2_R
3	3
4	4_R
5	5
52	52_R
6	6_R
7	7
78056	78056_R
8	8_R

Name	New Name
8259420 some text.jpg	8259420_R.jpg
8259422 some text.jpg	8259422_R.jpg
8259435 some text.jpg	8259435 some text.jpg
8259436 some text.jpg	8259436_R.jpg
8259437 some text.jpg	8259437 some text.jpg
8259438 some text.jpg	8259438_R.jpg
8259439 some text.jpg	8259439 some text.jpg
8259440 some text.jpg	8259440_R.jpg
8259441 some text.jpg	8259441 some text.jpg
8259442 some text.jpg	8259442_R.jpg
8259443 some text.jpg	8259443 some text.jpg
8259444 some text.jpg	8259444_R.jpg
8259445 some text.jpg	8259445 some text.jpg
8259446 some text.jpg	8259446_R.jpg
8259447 some text.jpg	8259447 some text.jpg
8259448 some text.jpg	8259448_R.jpg
8259449 some text.jpg	8259449 some text.jpg
8259450 some text.jpg	8259450_R.jpg
8259451 some text.jpg	8259451 some text.jpg

some text -8259421.jpg	some text -8259421.jpg
some text -8259423.jpg	some text -8259423.jpg
some text -8259424.jpg	8259424_R.jpg
some text -8259425.jpg	some text -8259425.jpg
some text -8259426.jpg	8259426_R.jpg
some text -8259427.jpg	some text -8259427.jpg
some text -8259428.jpg	8259428_R.jpg
some text -8259429.jpg	some text -8259429.jpg
some text -8259430.jpg	8259430_R.jpg
some text -8259431.jpg	some text -8259431.jpg
some text -8259432.jpg	8259432_R.jpg
some text -8259433.jpg	some text -8259433.jpg
some text -8259434.jpg	8259434_R.jpg
some text 8259420 some ...	8259420_R.jpg
some text 8259420.jpg	8259420_R.jpg
some text 8259421 some ...	some text 8259421 some more.jpg

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

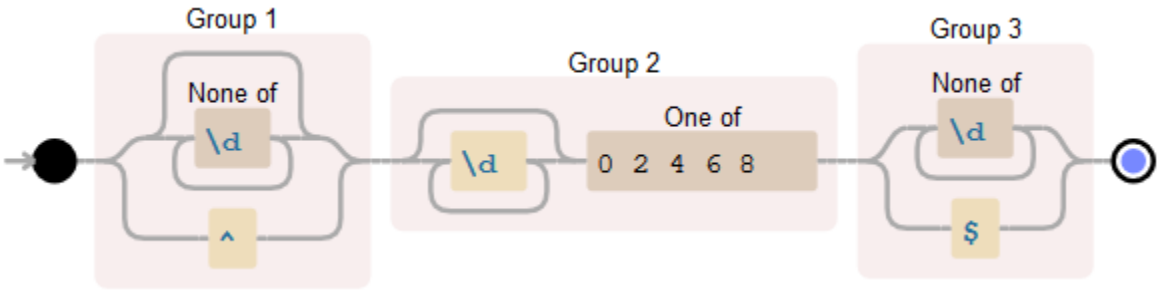
Match against first numeric group anywhere in string and preserve any preceding OR following text cont.

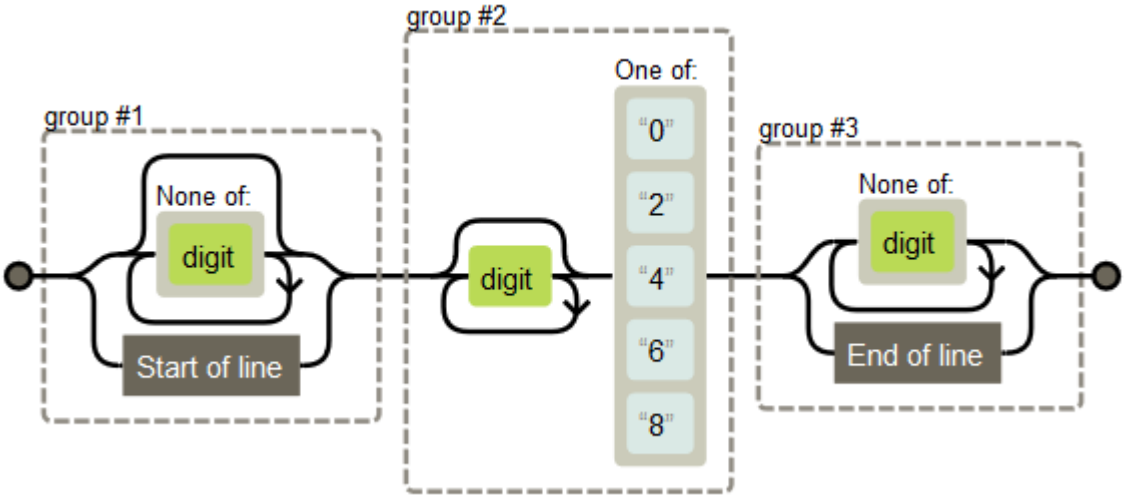
<text> ###1 <text> to ###1_R
<text> ###3 <text> to ###3_L
<text> ###5 <text> to ###5_L
<text> ###4 <text> to ###4_R

Sample Base:

Too many to list.

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)





Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string and preserve any preceding OR following text cont.

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)

Sample: Numeric Group followed by Text

```
·····························Start·Length···
Match 1: 8259450 some text      0      17
Group 1:                        0       0
Group 2: 8259450                0       7
Group 3:  some text             7      10
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  8259450
4  8259450backtrack
5  825945
6  8259450
7  8259450
8  8259450 some text
9  8259450 some text
Match found in 9 steps
```

Sample: Single Digit Number

```
·············Start·Length··
Match 1: 2      0       1
Group 1:        0       0
Group 2: 2      0       1
Group 3:        1       0
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  2
4  2backtrack
5  ok
6  2
7  2
8  2backtrack
9  2ok
10 2ok
Match found in 10 steps
```

Sample: Multiple Digit Number

```
···············Start·Length···
Match 1: 78056      0       5
Group 1:            0       0
Group 2: 78056      0       5
Group 3:            5       0
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  78056
4  78056backtrack
5  7805
6  78056
7  78056
8  78056backtrack
9  78056ok
10 78056ok
Match found in 10 steps
```

Uncategorized

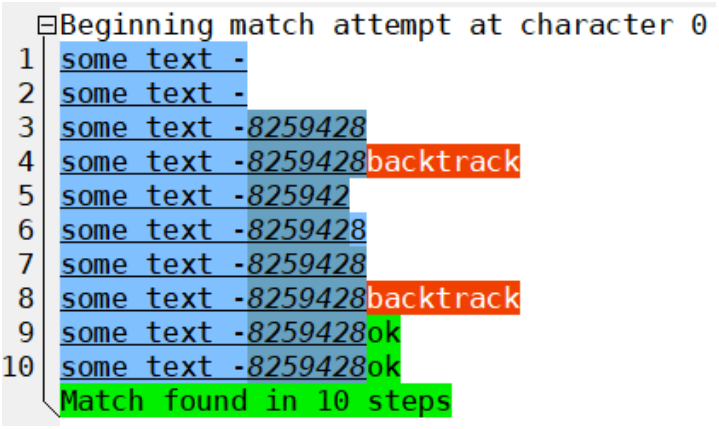
Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string and preserve any preceding OR following text cont.

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)

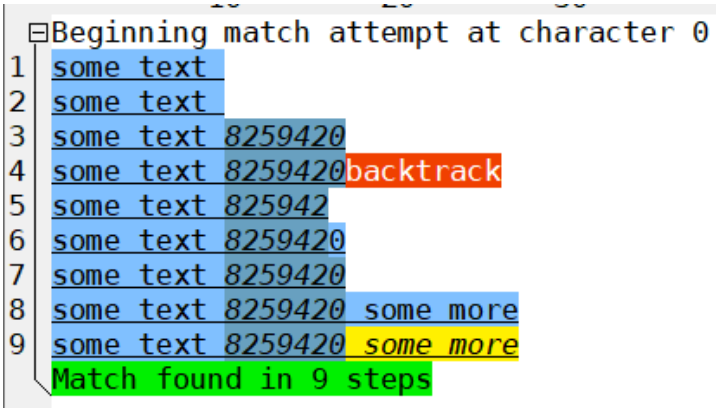
Sample: Text followed by Numeric Group

	Start	Length
Match 1:	some text -8259428	0	18
Group 1:	some text -	0	11
Group 2:	8259428	11	7
Group 3:		18	0



Sample: Numeric Group Preceded and Followed by Text

	Start	Length
Match 1:	some text 8259420 some more	0	27
Group 1:	some text	0	10
Group 2:	8259420	10	7
Group 3:	some more	17	10



Tim Notes:

1. Capture Group 1 is 'reserved' for preceding text
2. Capture Group 2 is 'reserved' for the numeric group
3. Capture Group 3 is 'reserved' for following text
4. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string and preserve any preceding OR following text cont.

Using: some text 8259420 some more.jpg

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)

Where:

\1 = some text <space> \2 = 8259420 \3 = <space> some more

Breaks down:

Capture Group 1 Capture Group 2 Capture Group 3

alternate alternate

anchor anchor

([^\d]* | ^) (\d* [02468]) ([^\d]+ | $)

This uses an Alternate for Capture Group 1

This OR This

[^\d]* | ^

Tim Notes:

1. The [^\d*] is used to capture any preceding text from those samples that have preceding text before the numeric group. What the Alternate does is provide a choice between those samples that do and those samples that don't. For those that do not, the first position would have a value of BOL (beginning of line or ' ^ '), followed by numeric digits.

This uses an Alternate for Capture Group 3

This OR This

[^\d]* | $

Tim Notes:

1. The [^\d*] is used to capture any following text from those samples that have following text after the numeric group. What the Alternate does is provide a choice between those samples that do and those samples that don't. For those that do not, the last position following the numeric digits would be EOL (End of Line or ' $ ').

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string and preserve any preceding OR following text cont.

Using: some text 8259420 some more.jpg

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)

Where:

\1 = some text <space> \2 = 8259420 \3 = <space> some more

Tim Analysis:

1. [^\d]*|^	Alternate (see Previous Page)		
	a. [^\d]	Match against a Negated Class consisting of a numeric digit, \d.	Capture Group 1 = s
	b. *	Make it Greedy	Capture Group 1 = some text <space>
	or		
	a. ^	Test if at beginning of string (BOL) This is the verification that there is no preceding text before the numeric digits.	
2. \d	Match against Numeric Digit		Capture Group 2 = 8
3. *	Make it Greedy		Capture Group 2 = 8259420
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. Backtracks to match against '0' = True		

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx continues to backtrack.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against first numeric group anywhere in string and preserve any preceding OR following text cont.

Using: some text 8259420 some more.jpg

Match: ([^\d]*|^)(\d*[02468])([^\d]+|$)

Where:

\1 = some text <space> \2 = 8259420 \3 = <space> some more

Tim Analysis: cont.

4. [^\d]+|$	Alternate (see Previous Page)		
	a. [^\d]	Match against a Negated Class consisting of a numeric digit, \d.	Capture Group 3 = <space>
	b. +	Make it Greedy	Capture Group 3 = <space> some more
	or		
	a. $	Test if at end of string (EOL) This is the verification that there is no following text after the numeric digits.	

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match only numeric digits at end of string with preceding text

Observations:

<text> ###1 to ###1_R
<text> ###3 to ###3_L
<text> ###5 to ###5_L
<text> ###4 to ###4_R

1. A Numeric Group ending with an odd number is renamed with a suffix of '_L' and an even number with '_R'
2. Pattern established: ### can be any number of numeric digits preceded by <text>.
3. The evaluation of an even or odd number is based on the last numeric digit of a numeric group
4. Numeric group must be at end of string.
5. If there is more than one numeric group, only the last will be matched.
6. Numeric group must have a minimum of two numeric digits.

Sample Base:

some text -8259421 some text -8259423 some text -8259424 some text -8259425
some text -8259426 some text -8259427 some text -8259428 some text -8259429
some text -8259430 some text -8259431 some text -8259432 some text -8259433
some text -8259434 some text 8259420

Match: (\d*[02468]+)$
Replace: \1_R

Name	New Name
some text -8259421.jpg	some text -8259421.jpg
some text -8259423.jpg	some text -8259423.jpg
some text -8259424.jpg	8259424_R.jpg
some text -8259425.jpg	some text -8259425.jpg
some text -8259426.jpg	8259426_R.jpg
some text -8259427.jpg	some text -8259427.jpg
some text -8259428.jpg	8259428_R.jpg
some text -8259429.jpg	some text -8259429.jpg
some text -8259430.jpg	8259430_R.jpg
some text -8259431.jpg	some text -8259431.jpg
some text -8259432.jpg	8259432_R.jpg
some text -8259433.jpg	some text -8259433.jpg
some text -8259434.jpg	8259434_R.jpg
some text 8259420.jpg	8259420_R.jpg

Tim Notes:

1. A Simple variation on 'For Whole Numbers with no preceding or following text', by simply removing the BOL anchor, it will match to end of string only.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match only numeric digits at end of string with preceding text cont.

Observations:

<text> ###1 to ###1_R
<text> ###3 to ###3_L
<text> ###5 to ###5_L
<text> ###4 to ###4_R

Sample Base:

some text -8259421 some text -8259423 some text -8259424 some text -8259425
some text -8259426 some text -8259427 some text -8259428 some text -8259429
some text -8259430 some text -8259431 some text -8259432 some text -8259433
some text -8259434 some text 8259420

Expanded Sample Base:

2018-12-02_050318 2018-12-02_051129 2018-12-02_103233 2018-12-02_104715
2018-12-02_104829 2018-12-02_105458 2018-12-02_105651 2018-12-02_105710
2018-12-02_105821 2018-12-02_110233 2018-12-02_110326 2018-12-02_110451
2018-12-04_142308 2018-12-11_121758 2018-12-11_123540 2018-12-11_181312

Match: (\d*[02468]+)$
Replace: \1_R

Name	New Name
2018-12-02_050318.jpg	050318_R.jpg
2018-12-02_051129.jpg	2018-12-02_051129.jpg
2018-12-02_103233.jpg	2018-12-02_103233.jpg
2018-12-02_104715.jpg	2018-12-02_104715.jpg
2018-12-02_104829.jpg	2018-12-02_104829.jpg
2018-12-02_105458.jpg	105458_R.jpg
2018-12-02_105651.jpg	2018-12-02_105651.jpg
2018-12-02_105710.jpg	105710_R.jpg
2018-12-02_105821.jpg	2018-12-02_105821.jpg
2018-12-02_110233.jpg	2018-12-02_110233.jpg
2018-12-02_110326.jpg	110326_R.jpg
2018-12-02_110451.jpg	2018-12-02_110451.jpg
2018-12-04_142308.jpg	142308_R.jpg
2018-12-11_121758.jpg	121758_R.jpg
2018-12-11_123540.jpg	123540_R.jpg
2018-12-11_181312.jpg	181312_R.jpg

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match only numeric digits at end of string with preceding text cont.

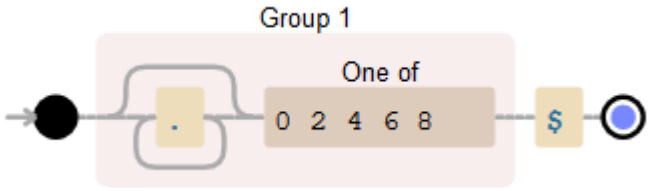
Sample Base:

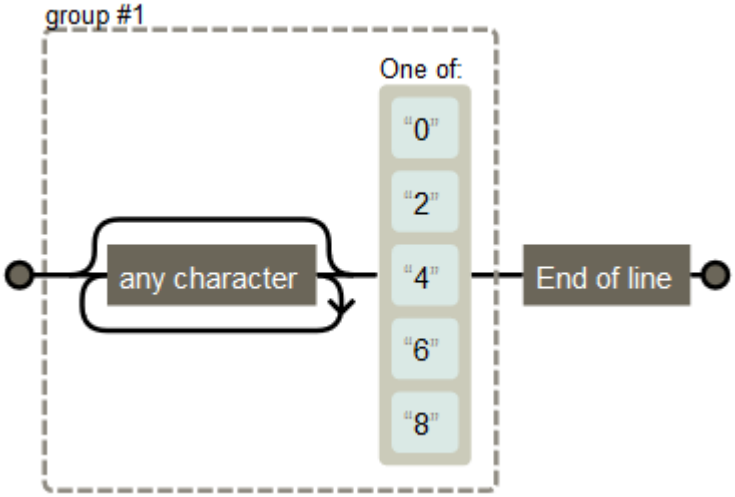
some text -8259421	some text -8259423	some text -8259424	some text -8259425
some text -8259426	some text -8259427	some text -8259428	some text -8259429
some text -8259430	some text -8259431	some text -8259432	some text -8259433
some text -8259434	some text 8259420		

Expanded Sample Base:

2018-12-02_050318	2018-12-02_051129	2018-12-02_103233	2018-12-02_104715
2018-12-02_104829	2018-12-02_105458	2018-12-02_105651	2018-12-02_105710
2018-12-02_105821	2018-12-02_110233	2018-12-02_110326	2018-12-02_110451
2018-12-04_142308	2018-12-11_121758	2018-12-11_123540	2018-12-11_181312

Match: (\d*[02468]+)$





Sample: some text -8259428.jpg

```
·····················Start·Length··
Match 1: 8259428      11       7
Group 1: 8259428      11       7
```

```
Beginning match attempt at character 11
1 8259428
2 8259428backtrack
3 825942
4 8259428
5 8259428
6 8259428ok
Match found in 6 steps
```

Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\1_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match only numeric digits at end of string with preceding text cont.

Using: some text -8259428.jpg

Match: (\d*[02468]+)$

Where:

\1 = 8259428

Tim Analysis:

1. \d	Match against Numeric Digit	Capture Group 1 = 82
2. *	Make it Greedy	Capture Group 1 = 8259428
3. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. Backtracks to match against '8' = True	Capture Group 1 = No Change

Tim Notes:

1. This is how it determines whether or not the first numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx Fails.
2. Although Step 3 shows no Change, without it, there would be no even match.

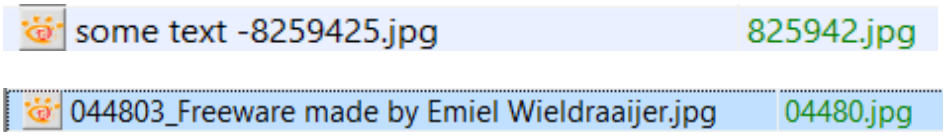
4. +	Make the expression Greedy	Capture Group 1 = 825940
5. $	Test if at end of string (EOL) This is the verification that there is no following text after the numeric digits.	

Tim Notes:

1. Without the EOL anchor, it would match against any even combination of numbers without regard to a numeric group.

   i.e.,

   Matches:



Etc.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month

Observations:

YYYY-MM-D1 <text> to YYYY-MM-D1_R
YYYY-MM-D3 <text> to YYYY-MM-D3_L
YYYY-MM-D5 <text> to YYYY-MM-D5_L
YYYY-MM-D4 <text> to YYYY-MM-D4_R

1. A file with a numeric group with a specific point within that group as determined to be an odd or even number is renamed with a suffix of '_L' or '_R', respectively.
2. Pattern established: YYYY-MM-DD_<text>
3. The evaluation of an even or odd number is based on the last numeric digit of the DD numeric group
4. Requires a more restrictive pattern.
5. Numeric group of YYYY-MM-DD can be anywhere in the string.
6. If there are more than one date numeric group, only the first will be matched.

Sample Base:

2018-12-02_050318 2018-12-02_051129 2018-12-04_142308 2018-12-11_121758 2018-12-11_123540
2018-12-11_181312 2018-12-16_111240 2018-12-17_110351 2019-03-24_185117 1990-04-24_100001
1990-04-25_181845 1990-05-05_103801 2019-05-05_103921 1990-05-08_115123 2019-05-08_115123
1990-05-15_130811 2019-05-15_130811 1990-05-17_085904 1990-06-01_113856 2019-06-01_113856
1990-06-07_174201 1990-06-24_185246 2019-06-24_185246 1990-06-24_185355 2019-06-24_185355
1990-06-25_105737

some text 1995-09-28_135859 some text 2018-09-28_135859 some text 2019-09-28_135859

Match: ^(([^\d]*)(20\d\d|19\d\d))(([- _])[01]\d)(\5[0123][02468])(.*)$
Replace: \3\4\6_R

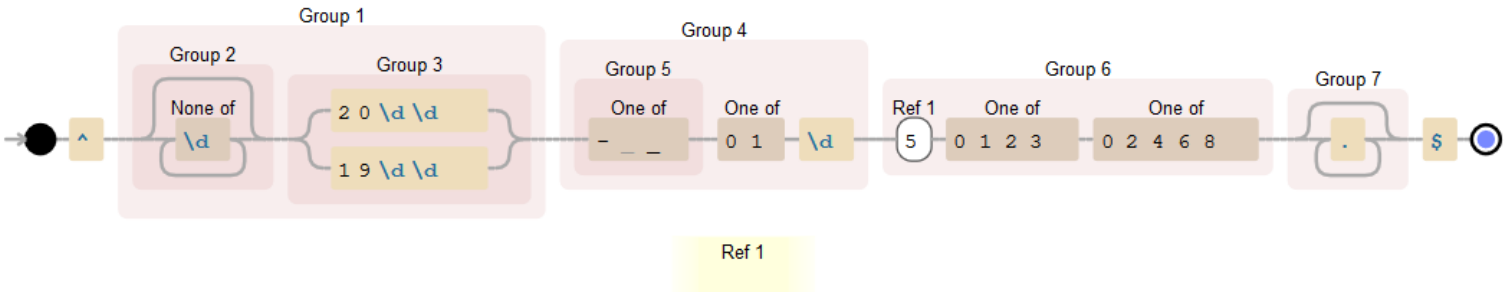
Name	New Name
1990-04-24_100001.jpg	1990-04-24_R.jpg
1990-04-25_181845.jpg	1990-04-25_181845.jpg
1990-05-05_103801.jpg	1990-05-05_103801.jpg
1990-05-08_115123.jpg	1990-05-08_R.jpg
1990-05-15_130811.jpg	1990-05-15_130811.jpg
1990-05-17_085904.jpg	1990-05-17_085904.jpg
1990-06-01_113856.jpg	1990-06-01_113856.jpg
1990-06-07_174201.jpg	1990-06-07_174201.jpg
1990-06-24_185246.jpg	1990-06-24_R.jpg
1990-06-24_185355.jpg	1990-06-24_R.jpg
1990-06-25_105737.jpg	1990-06-25_105737.jpg
2018-12-02_050318.jpg	2018-12-02_R.jpg
2018-12-02_051129.jpg	2018-12-02_R.jpg
2018-12-04_142308.jpg	2018-12-04_R.jpg
2018-12-11_121758.jpg	2018-12-11_121758.jpg
2018-12-11_123540.jpg	2018-12-11_123540.jpg
2018-12-11_181312.jpg	2018-12-11_181312.jpg
2018-12-16_111240.jpg	2018-12-16_R.jpg
2018-12-17_110351.jpg	2018-12-17_110351.jpg
2019-03-24_185117.jpg	2019-03-24_R.jpg
2019-05-05_103921.jpg	2019-05-05_103921.jpg
2019-05-08_115123.jpg	2019-05-08_R.jpg
2019-05-15_130811.jpg	2019-05-15_130811.jpg
2019-06-01_113856.jpg	2019-06-01_113856.jpg
2019-06-24_185246.jpg	2019-06-24_R.jpg
2019-06-24_185355.jpg	2019-06-24_R.jpg
some text 1995-09-28_135859.jpg	1995-09-28_R.jpg
some text 2018-09-28_135859.jpg	2018-09-28_R.jpg
some text 2019-09-28_135859.jpg	2019-09-28_R.jpg

Uncategorized

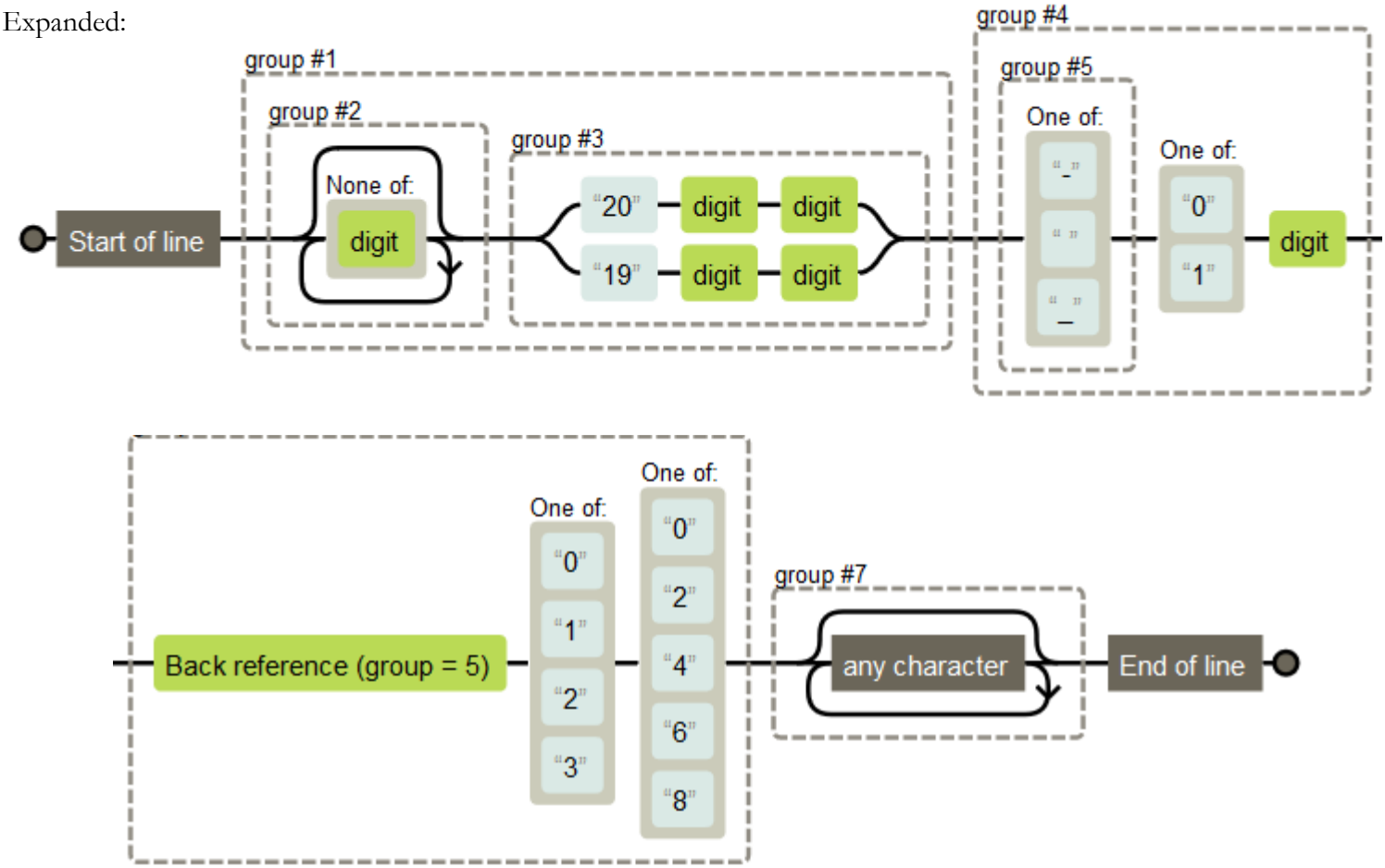
Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Match: ^(([^\d]*)(20\d\d|19\d\d))(([- _])[01]\d)(\5[0123][02468])(.*)$



Expanded:



Tim Notes:

1. To test for Odd Numbers, change the class to '[13579]' and use a Replacement String of '\3\4\6_L'

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Match: ^(([^\d]*)(20\d\d|19\d\d))(([- _])[01]\d)(\5[0123][02468])(.*)$

Sample:

```
.............................Start·Length···
Match 1: 2018-12-02_050318       0      17
Group 1: 2018                    0       4
Group 2:                         0       0
Group 3: 2018                    0       4
Group 4: -12                     4       3
Group 5: -                       4       1
Group 6: -02                     7       3
Group 7: _050318                10       7
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  ok
4  2
5  20
6  201
7  2018
8  2018
9  2018
10 2018-
11 2018-
12 2018-1
13 2018-12
14 2018-12
15 2018-12-
16 2018-12-0
17 2018-12-02
18 2018-12-02
19 2018-12-02_050318
20 2018-12-02_050318
21 2018-12-02_050318ok
   Match found in 21 steps
```

Sample:

```
.....................................
Match 1: some text 1995-09-28_135859
Group 1: some text 1995
Group 2: some text
Group 3: 1995
Group 4: -09
Group 5: -
Group 6: -28
Group 7: _135859
```

```
Beginning match attempt at character 0
1  ok
2  some text
3  some text
4  some text backtrack
5  some text 1
6  some text 19
7  some text 199
8  some text 1995
9  some text 1995
10 some text 1995
11 some text 1995-
12 some text 1995-
13 some text 1995-0
14 some text 1995-09
15 some text 1995-09
16 some text 1995-09-
17 some text 1995-09-2
18 some text 1995-09-28
19 some text 1995-09-28
20 some text 1995-09-28_135859
21 some text 1995-09-28_135859
22 some text 1995-09-28_135859ok
   Match found in 22 steps
```

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Match: ^(([^\d]*)(20\d\d|19\d\d))(([- _])[01]\d)(\5[0123][02468])(.*)$

Sample:

```
                              Start  Length
Match 1: 1990-06-24_185246        0      17
Group 1: 1990                     0       4
Group 2:                          0       0
Group 3: 1990                     0       4
Group 4: -06                      4       3
Group 5: -                        4       1
Group 6: -24                      7       3
Group 7: _185246                 10       7
```

```
Beginning match attempt at character 0
1  ok
2  ok
3  ok
4  backtrack
5  1
6  19
7  199
8  1990
9  1990
0  1990
1  1990-
2  1990-
3  1990-0
4  1990-06
5  1990-06
6  1990-06-
7  1990-06-2
8  1990-06-24
9  1990-06-24
0  1990-06-24_185246
1  1990-06-24_185246
2  1990-06-24_185246ok
   Match found in 22 steps
```

Sample:

```
Match 1: some text 2019-09-28_135859
Group 1: some text 2019
Group 2: some text
Group 3: 2019
Group 4: -09
Group 5: -
Group 6: -28
Group 7: _135859
```

```
Beginning match attempt at character 0
1  ok
2  some text
3  some text
4  some text 2
5  some text 20
6  some text 201
7  some text 2019
8  some text 2019
9  some text 2019
10 some text 2019-
11 some text 2019-
12 some text 2019-0
13 some text 2019-09
14 some text 2019-09
15 some text 2019-09-
16 some text 2019-09-2
17 some text 2019-09-28
18 some text 2019-09-28
19 some text 2019-09-28_135859
20 some text 2019-09-28_135859
21 some text 2019-09-28_135859ok
   Match found in 21 steps
```

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(([^\d]*)(20\d\d|19\d\d))(([- _])[01]\d)(\5[0123][02468])(.*)$

Where:

\1 = 2018 \2 = null \3 = 2018 \4 = -09 \5 = - \6 = -28 \7 = _135859

This uses an Alternate for Capture Group 1

This OR This

20\d\d | 19\d\d

Breaks down:

Using: some text 2019-09-28_135859.jpg

Capture Group 1 Capture Group 4

Capture Group 2 Capture Group 3 Capture Group 5

anchor Alternate

^ (([^\d]*) (20\d\d | 19\d\d)) (([- _]) [01] \d)

some text <space> 2019 - 09

Capture Group 6 Capture Group 7

anchor

(\5 [0123] [02468]) (.*) $

- 28 _135859

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(([^\d]*)(20\d\d|19\d\d))(([- _])[01]\d)(\5[0123][02468])(.*)$

Where:

\1 = 2018 \2 = null \3 = 2018 \4 = -09 \5 = - \6 = -28 \7 = _135859

Tim Analysis:

1. ^ Start at beginning of string (BOL)

The Preceding Text (Optional)

2. [^\d] Match against a Negated Class consisting of a numeric digit, \d.
   Capture Group 1 = no value
   Capture Group 2 = no value
3. * Make it Greedy
   Capture Group 1 = null
   Capture Group 2 = null

The Year

2. 20\d\d|19\d\d Alternate (see Previous Page)
   a. 20 Match against string '20' This is the '20' representing the century part of the year.
   Capture Group 1 = 20
   Capture Group 2 = Unchanged
   Capture Group 3 = 20
   b. \d\d Match against 2 numeric digits
   Capture Group 1 = 2018
   Capture Group 3 = 2018

   or

   a. 19 Match against string '19' = '20' = False
   b. \d\d Match against 2 numeric digits

Tim Notes:

1. Could also have used ' (20|19)(\d\d) ' to shorten the Alternate sub expression.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(([^\d]*)(20\d\d|19\d\d))([- _])[01]\d)(\5[0123][02468])(.*)$

Where:

\1 = 2018 \2 = null \3 = 2018 \4 = -09 \5 = - \6 = -28 \7 = _135859

Tim Analysis: cont.

The Month

3. [- _]	Match against a class consisting of a <hyphen> <space> or underscore	Capture Group 4 = - Capture Group 5 = -
4. [01]	Match against a class consisting of a numeric digit, '0' or '1' This is the '0' of the first part of the month, '09'	Capture Group 4 = -0 Capture Group 5 = Unchanged

Tim Notes:

Where:

0 represents months 01-09
1 represents months 10-12

5.\d	Match against Numeric Digit	Capture Group 4 = -09 Capture Group 5 = Unchanged

The Day Part 1

6. \3	Recall Capture Group's 5 value	Capture Group 6 = -
7. [0123]	Match against a class consisting of a numeric digit, '0', '1' '2' or '3'	Capture Group 6 = -2

Tim Notes:

Where:

0 represents days 01-09
1 represents days 10-19
2 represents days 20-29
3 represents days 30-31

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?20\d\d|19\d\d)([- _])[01]\d)(\3[0123][02468])(.*)$

Where:

\1 = 2018 \2 = -09 \3 = - \4 = -28 \5 = _135859

Tim Analysis: cont.

The Day Part 2: The Test

8. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. Backtracks to match against '8' = True	Capture Group 6 = -28

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx Fails.

2. To test against month, move the [02468] after the [01] instead.

The Following Text (Optional)

9. .*	Match against remaining string	Capture Group 7 = _135859

The End

10. $	Must be at end of string (EOL)

Tim Notes:

1. Uses a class to represent the possible months of the year and days of the month.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using Negative Lookbehinds to Restrict Range of Years Matched

Observations:

YYYY-MM-D1 <text> to YYYY-MM-D1_R
YYYY-MM-D3 <text> to YYYY-MM-D3_L
YYYY-MM-D5 <text> to YYYY-MM-D5_L
YYYY-MM-D4 <text> to YYYY-MM-D4_R

1. A file with a numeric group with a specific point within that group as determined to be an odd or even number is renamed with a suffix of '_L' or '_R', respectively.
2. Pattern established: YYYY-MM-DD_<text>
3. The evaluation of an even or odd number is based on the last numeric digit of the DD numeric group.
4. Requires a more restrictive pattern.
5. Numeric group of YYYY-MM-DD can be anywhere in the string.
6. If there are more than one date numeric group, only the first will be matched.

Sample Base:

2018-12-02_050318 2018-12-02_051129 2018-12-04_142308 2018-12-11_121758 2018-12-11_123540
2018-12-11_181312 2018-12-16_111240 2018-12-17_110351 2019-03-24_185117 1990-04-24_100001
1990-04-25_181845 1990-05-05_103801 2019-05-05_103921 1990-05-08_115123 2019-05-08_115123
1990-05-15_130811 2019-05-15_130811 1990-05-17_085904 1990-06-01_113856 2019-06-01_113856
1990-06-07_174201 1990-06-24_185246 2019-06-24_185246 1990-06-24_185355 2019-06-24_185355
1990-06-25_105737

some text 1995-09-28_135859 some text 2018-09-28_135859 some text 2019-09-28_135859

This restricts years from 1970-2020 only and months to 01-12 and days 0-31 (with verification checks)

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$
Replace: \1\2\3\5_R

Name	New Name
1990-04-24_100001.jpg	1990-04-24_R.jpg
1990-04-25_181845.jpg	1990-04-25_181845.jpg
1990-05-05_103801.jpg	1990-05-05_103801.jpg
1990-05-08_115123.jpg	1990-05-08_R.jpg
1990-05-15_130811.jpg	1990-05-15_130811.jpg
1990-05-17_085904.jpg	1990-05-17_085904.jpg
1990-06-01_113856.jpg	1990-06-01_113856.jpg
1990-06-07_174201.jpg	1990-06-07_174201.jpg
1990-06-24_185246.jpg	1990-06-24_R.jpg
1990-06-24_185355.jpg	1990-06-24_R.jpg
1990-06-25_105737.jpg	1990-06-25_105737.jpg
2018-12-02_050318.jpg	2018-12-02_R.jpg
2018-12-02_051129.jpg	2018-12-02_R.jpg
2018-12-04_142308.jpg	2018-12-04_R.jpg
2018-12-11_121758.jpg	2018-12-11_121758.jpg
2018-12-11_123540.jpg	2018-12-11_123540.jpg
2018-12-11_181312.jpg	2018-12-11_181312.jpg
2018-12-16_111240.jpg	2018-12-16_R.jpg
2018-12-17_110351.jpg	2018-12-17_110351.jpg
2019-03-24_185117.jpg	2019-03-24_R.jpg
2019-05-05_103921.jpg	2019-05-05_103921.jpg
2019-05-08_115123.jpg	2019-05-08_R.jpg
2019-05-15_130811.jpg	2019-05-15_130811.jpg
2019-06-01_113856.jpg	2019-06-01_113856.jpg
2019-06-24_185246.jpg	2019-06-24_R.jpg
2019-06-24_185355.jpg	2019-06-24_R.jpg
some text 1995-09-28_135859.jpg	1995-09-28_R.jpg
some text 2018-09-28_135859.jpg	2018-09-28_R.jpg
some text 2019-09-28_135859.jpg	2019-09-28_R.jpg

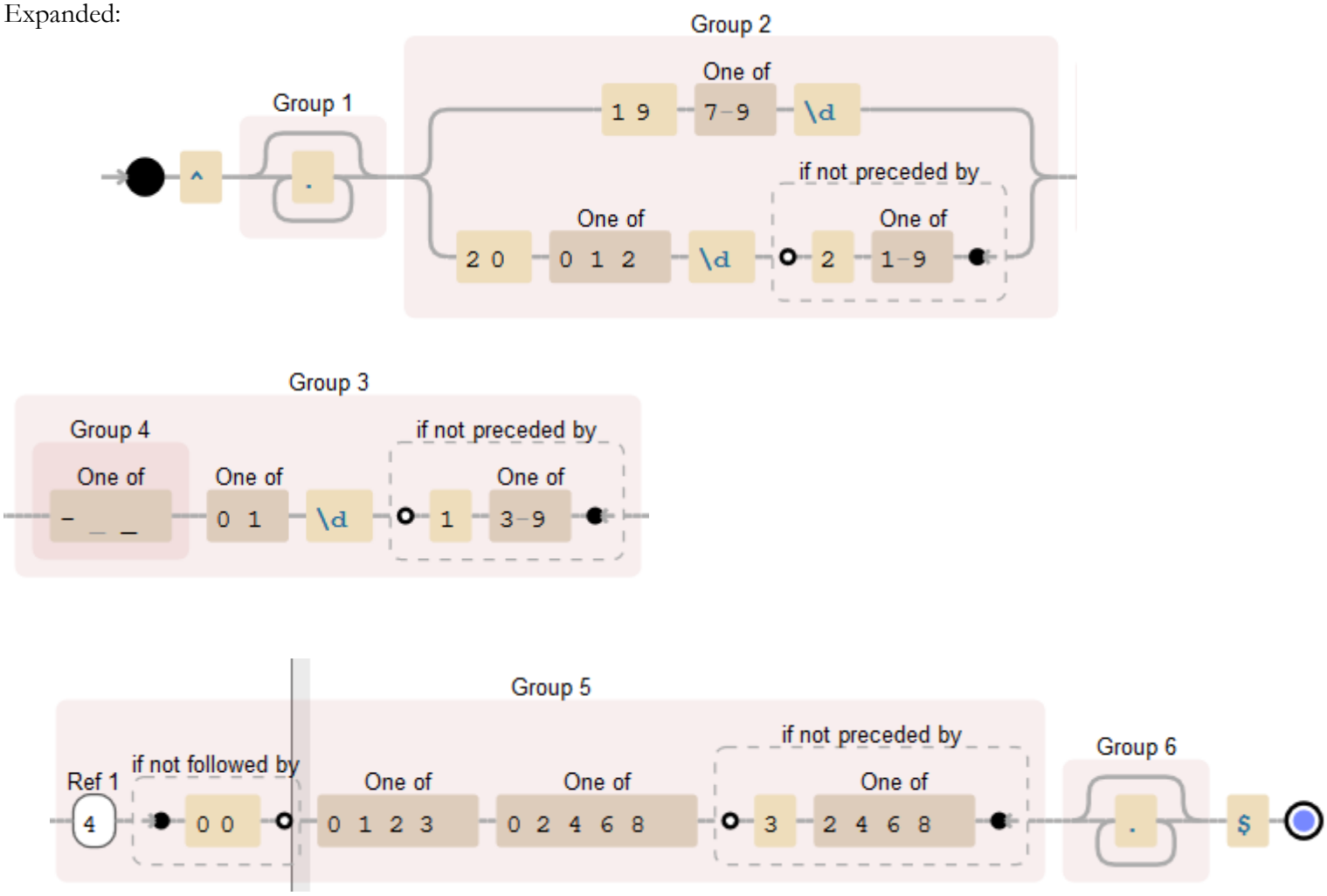
Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using Negative Lookbehinds to Restrict Range of Years Matched cont.

Expanded:



Tim Notes:

1. If you recall, The First Visualizer program does not support Negative Lookbehinds, therefore only the second Visualizer program can be utilized in diagramming the RegEx.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using Negative Lookbehinds to Restrict Range of Years Matched cont.

Sample:

	Start	Length
Match 1:	2019-06-24_185246	0	17
Group 1:		0	0
Group 2:	2019	0	4
Group 3:	-06	4	3
Group 4:	-	4	1
Group 5:	-24	7	3
Group 6:	_185246	10	7

Tim Notes:

1. Capture Group 1 is 'reserved' for preceding text.
2. Capture Group 2 is the YEAR
3. Capture Group 3 is the MONTH
4. Capture Group 4 is the Separator used between dates
5. Capture Group 5 is the DAY
6. Capture Group 6 is 'reserved' for following text

```
Beginning match attempt at character 0
1  ok
2  ok
3  ok
4  backtrack
5  2
6  20
7  201
8  2019
9  20backtrack
10 20ok
11 2019
12 2019-
13 2019-
14 2019-0
15 2019-06
16 2019-backtrack
17 2019-ok
18 2019-06
19 2019-06-
20 2019-06-backtrack
21 2019-06-ok
22 2019-06-2
23 2019-06-24
24 2019-06-backtrack
25 2019-06-ok
26 2019-06-24
27 2019-06-24_185246
28 2019-06-24_185246
29 2019-06-24_185246ok
   Match found in 29 steps
```

Uncategorized

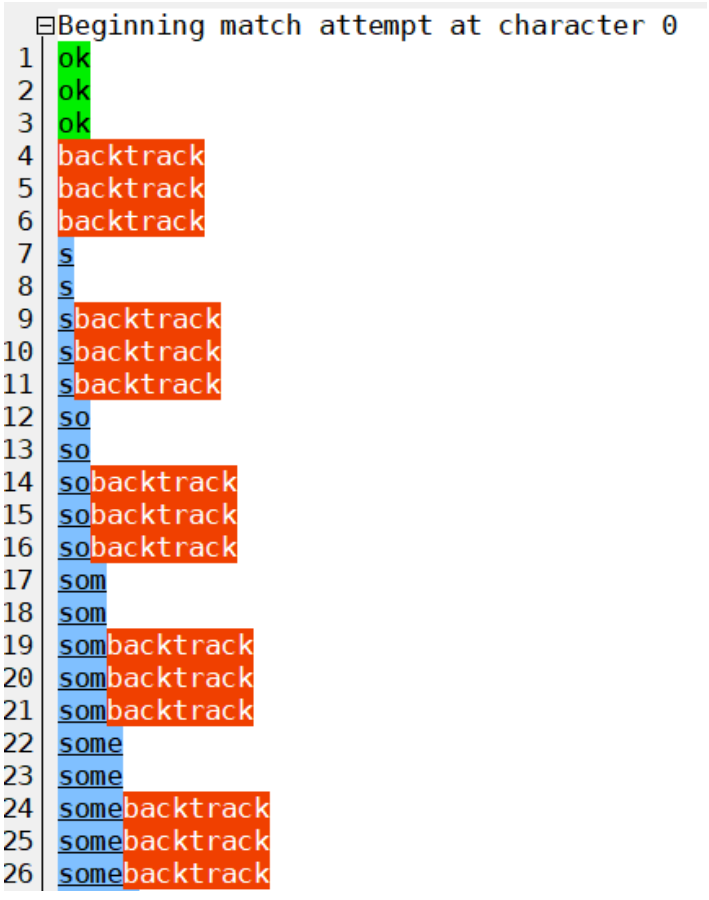
Add Left, Right Suffix for Even/Odd Numbered Files cont.

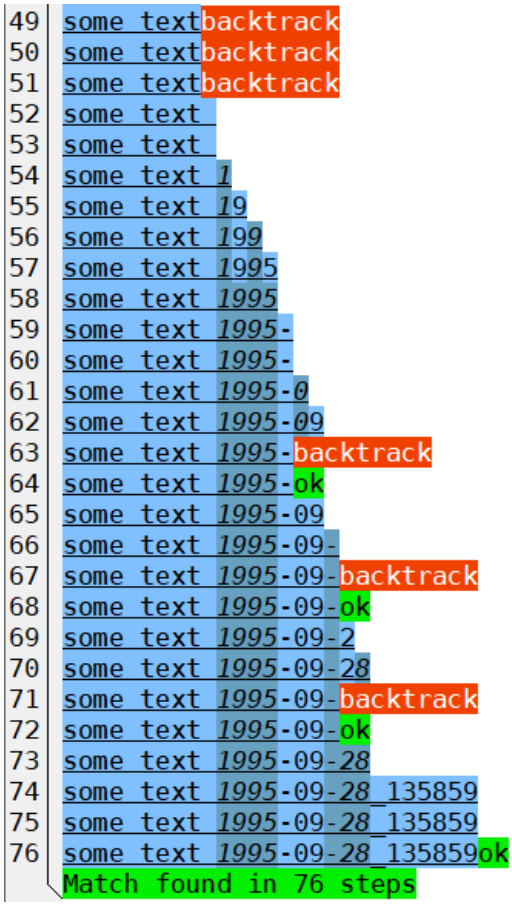
Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using Negative Lookbehinds to Restrict Range of Years Matched cont.

Sample:

```
.....................................
Match 1: some text 1995-09-28_135859
Group 1: some text
Group 2: 1995
Group 3: -09
Group 4: -
Group 5: -28
Group 6: _135859
```





.
.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using Negative Lookbehinds to Restrict Range of Years Matched cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

Where:

\1 = null \2 = 2018 \3 = -09 \4 = - \5 = -28 \6 = _135859

Breaks down:

Capture Group 2

Capture Group 1

anchor alternate Negative Lookbehind

^ (.*?) (19[7-9]\d | 20[012]\d (?<!2[1-9]))

Capture Group 3

Capture Group 4

Negative Lookbehind

(([- _]) [01]\d (?<!1[3-9]))

Capture Group 5

Capture Group 6

Negative Lookahead Negative Lookbehind anchor

(\4 (?!00) [0123] [02468] (?<!3[2468])) (.*) $

Tim Notes:

1. Capture Group 4 captures the separator used between dates. This can be a <hyphen>, <space> or underscore

Uncategorized

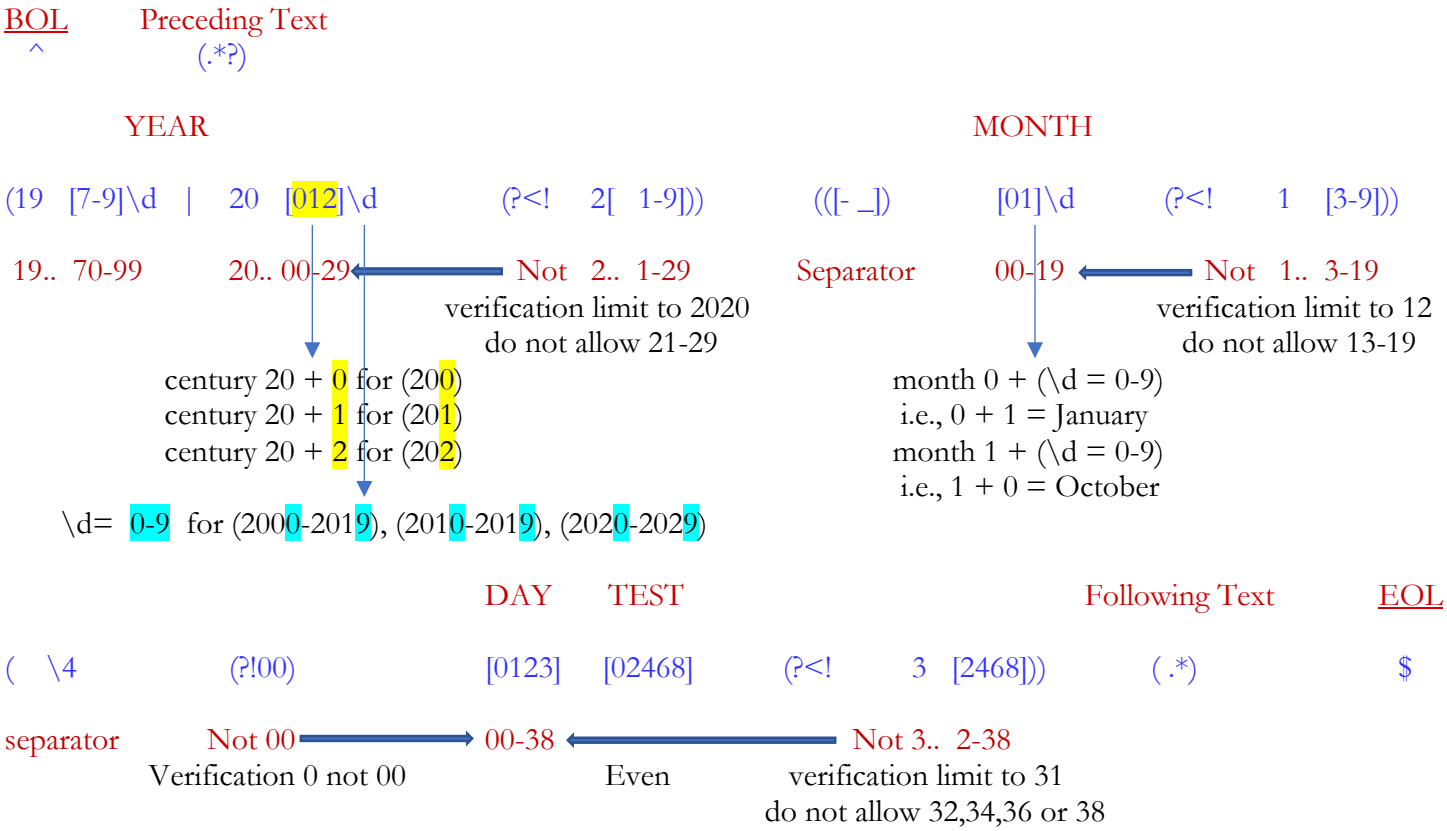
Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using Negative Lookbehinds to Restrict Range of Years Matched cont.

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

More Breakdown:



This uses an Alternate for Capture Group 2

This	OR	This
19[7-9]\d	|	20[012]\d(?<!2[1-9])

Tim Notes:

1. Change to Odd use: [0123][13579](?<!3[3579]) in place of, [0123][02468](?<!3[2468]) (DAY, TEST, Verification)

2. If you examine the Negative Lookbehind used in Test, the Lookbehind is set for only those values that would match the 'Even' Test – other than zero because 30 is a valid even day. It verifies against those days that would not be valid but would still match and this would include 32, 34, 36 and 38. This is why these values are excluded.

3. The Lookbehinds are used as verification limiters.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

Where:

\1 = null \2 = 2018 \3 = -09 \4 = - \5 = -28 \6 = _135859

Tim Analysis:

1. ^ Start at beginning of string (BOL)

The Preceding Text (Optional)

2. .	Match against any character	Capture Group 1 = 2
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL

The Year

2. 19[7-9]\d|20[012]\d Alternate (see Previous Page)
   - a. 19 Match against string '19' = '20' = False
     This is the '19' representing
     the century part of the year.
   - b. [7-9] Match against a class range consisting
     of a range from 7-9
   - c. \d Match against 1 numeric digit

Tim Notes:

1. The '19' (Century) with the class range [7-9] (first part of YY) and the added \d (second part of YY) will cover years 197(0-9) or 1970-1979, 198(0-9) or 1980-1989 and 199(0-9) or 1990-1999. The value of (0-9) is obtained because the last numeric digit of the YYYY captured using \d has to be a numeric value 0-9 (representing a single numeric digit).

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

Where:

\1 = null \2 = 2018 \3 = -09 \4 = - \5 = -28 \6 = _135859

Tim Analysis: cont.

The Year cont.

2. 19[7-9]\d|20[012]\d Alternate cont.

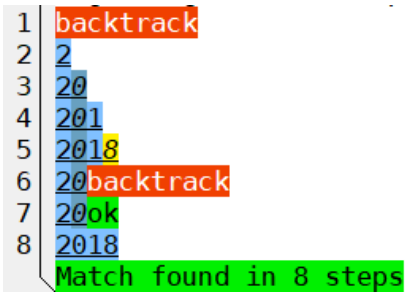
or

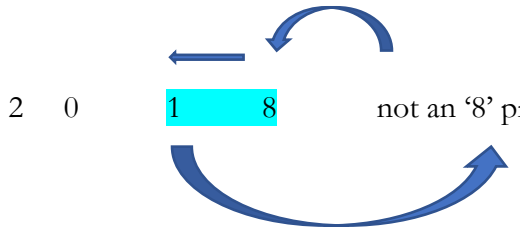
a. 20	Match against string '20'	Capture Group 2 = 20 Changes Capture Group 1 = null
b. [012]	Match against a class consisting of a '0', '1' or '2'	Capture Group 2 = 201
c. \d	Match against 1 numeric digit	Capture Group 2 = 2018

Tim Notes:

1. The '20' (Century) with the class [012] (first part of YY) and the added \d (second part of YY) will cover years 200(0-9) or 2000-2009, 201(0-9) or 2010-2019 and 202(0-9) or 2020-2099. The value of (0-9) is obtained because the last numeric digit of the YYYY captured using \d has to be a numeric value 0-9 (representing a single numeric digit).

d. (?<!2[1-9]) Negative Lookbehind.
Look to the left and test for a 1-9 *not preceded immediately* by a 2.
= '8' preceded by '1' True





not an '8' preceded by a '2' = True

Tim Notes:

1. Disqualifies years 2021 through 2099 leaving only the years 2000-2020 as valid.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

Where:

\1 = null \2 = 2018 \3 = -09 \4 = - \5 = -28 \6 = _135859

Tim Analysis: cont.

The Month

3. [- _]	Match against a class consisting of a <hyphen> <space> or underscore	Capture Group 3 = - Capture Group 4 = -
4. [01]	Match against a class consisting of a numeric digit, '0' or '1' This is the '0' of the first part of the month, '09'	Capture Group 3 = -0 Capture Group 4 = Unchanged

Tim Notes:

Where:

0 represents months 01-09
1 represents months 10-12

5.\d	Match against Numeric Digit	Capture Group 3 = -09 Capture Group 4 = Unchanged
6. (?<!1[3-9])	Negative Lookbehind. look to the left and test for a 3-9 *not immediately preceded* by a 1. = '9' preceded by '0' = True	

Tim Notes:

1. This essentially verifies that the preceding number for MONTH is a value between 0-12, disqualifying 13-19 as months.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

Where:

\1 = null \2 = 2018 \3 = -09 \4 = - \5 = -28 \6 = _135859

Tim Analysis: cont.

The Day Part 1

6. \4	Recall Capture Group's 4 value	Capture Group 5 = -
7. (?!00)	Negative Lookahead. Look to the right and ensure that there *are not* two Zeros sequentially. ='02' = True.	

Tim Notes:

1. Tests that there is a valid day that does not contain a leading '00'.

8. [0123]	Match against a class consisting of a numeric digit, '0', '1' '2' or '3'	Capture Group 5 = -2

Tim Notes:

Where:

0 represents days 01-09
1 represents days 10-19
2 represents days 20-29
3 represents days 30-31

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Match against a numeric not at the beginning or end of the string, i.e. Day of Month cont.

Using: 2018-09-28_135859.jpg

Match: ^(.*?)(19[7-9]\d|20[012]\d(?<!2[1-9]))(([- _])[01]\d(?<!1[3-9]))(\4(?!00)[0123][02468](?<!3[2468]))(.*)$

Where:

\1 = null \2 = 2018 \3 = -09 \4 = - \5 = -28 \6 = _135859

Tim Analysis: cont.

The Day Part 2: The Test

9. [02468]	Match against a class consisting of a numeric digit that is a '0', '2', '4', '6' or an '8'. Backtracks to match against '8' = True	Capture Group 5 = -28

Tim Notes:

1. This is how it determines whether or not the last numeric digit is an even number (divisible) using a test for the even numbers that appear in a numeric sequence 0-9. These are the '0','2', '4', '6' and '8'. If the character tested is not one of these values then the RegEx Fails.

10. (?<!3[2468])	Negative Lookbehind. look to the left and test for a 2, 4, 6 or 8 *not immediately preceded* by a 3. = '8' preceded by '2' = True

Tim Notes:

1. The Lookbehind is set for only those values that would match the 'Even' Test – other than zero because 30 is a valid 'even' day. It verifies against those days that would not be valid but would still match, and this would include 32, 34, 36 and 38. This is why these values are excluded.

The Following Text (Optional)

11. .*	Match against remaining string	Capture Group 6 = _135859

The End

10. $	Must be at end of string (EOL)

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

This is a tale of Biblical proportions!

There were once two RegEx, Cain and Abel.

Abel was even tempered and Cain was anything but. Cain, in fact, always found himself at odds with his brother.

In the age of PCRE, both had acquired samples of Ten strings to determine who was the better RegEx, but the stats showed it was a toss up.

But there came upon a time when Cain sought out a mercenary force, 'Everything', to amass a huge army of strings. Abel, fearing for his life, also enlisted the services of 'Everything'.

The battle was decidedly fought in the arena of the Content Pane in the Kingdom of BRU. The armies were strung together to wage the Greedy war that would end zero or more of them. It was a bloody campaign. Parsing of numbers flying every which way. Too many to count. The bookies were betting even odds, but for a while it looked like the odds were against Cain as his numbers dwindled down. But when the dust subsided, it seemed peace had reigned after all - or had it?

And it was said unto Cain, 'Where is Abel thy brother?' And he curtly replied, 'I know not: Am I my brother's abacus?'

'Wink, wink, nudge, nudge'

Read on, Dear Friends…

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Renaming for Even/Odd Using the 'Everything' Program in Conjunction with BRU

Observations:

3. Use the program. 'Everything' to match all files that begin with a number and determine if that number is odd or even.
4. This will be accomplished in two runs. One for even numbered files, and one for odd numbered files.
5. Enter each RegEx formula individually into the 'Everything' search field for each run.

Tim Notes:

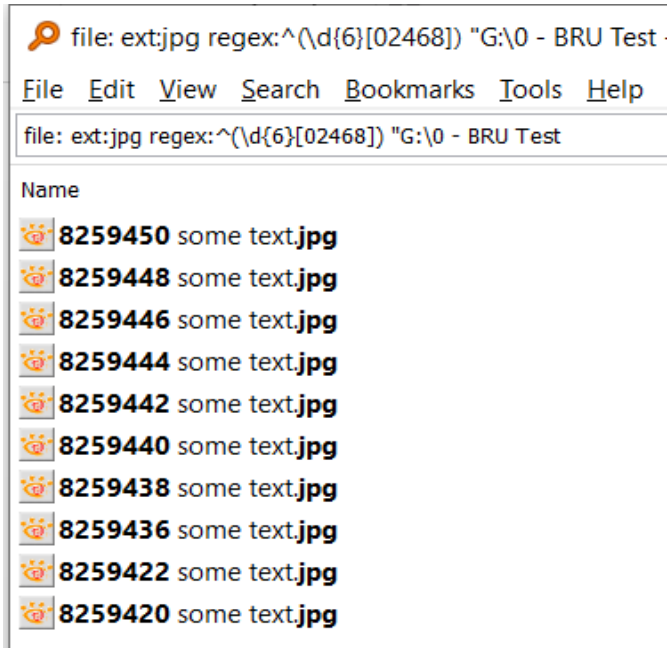
1. Using Everything is useful when you need to gather files to be renamed from different sources. You define a filter with what you want to include, then use Drag and Drop to enter the files into BRU.

Select from the previous pages which of the RegEx on those pages you want to use for the Everything filter (don't forget to have 'RegEx' Enabled under the 'Search Menu'), or use a different one. For these examples, I limited the sample list down and then restricted it further to numeric groups with 6 numeric digits and no preceding text, using:

^(\d{6}[02468])

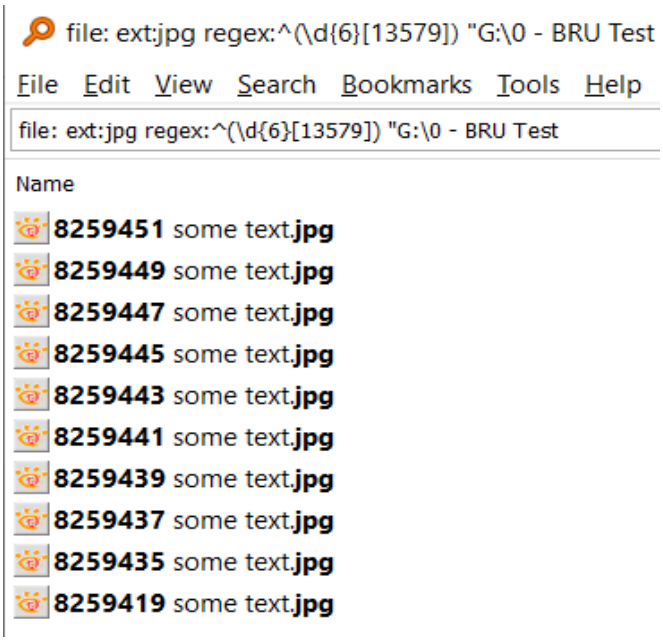
Even Numbered Files:

file: ext:jpg regex:^(\d{6}[02468]) "G:\0 - BRU Test



Odd Numbered Files:

file: ext:jpg regex:^(\d{6}[13579]) "G:\0 - BRU Test



ext:jpg	filter the extension in the search to just jpgs.
regex:	the following is treated as a RegEx string.
"G:\0..	limit the search to the following directory and include subdirectories (otherwise it would indicate 'parent').

This syntax is added by Everything from user selections of the Advanced Search facility under the Search Menu.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

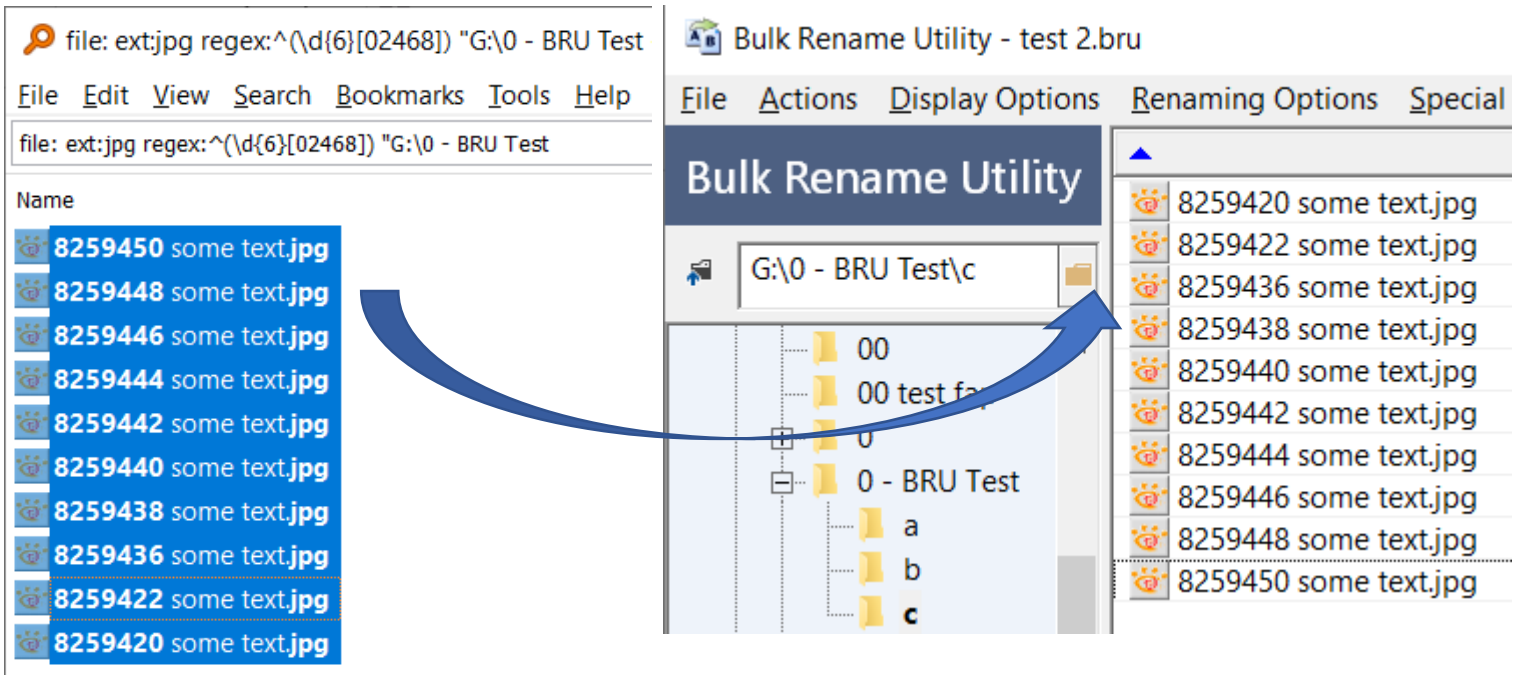
Renaming for Even/Odd Using the 'Everything' Program in Conjunction with BRU cont.

First run –

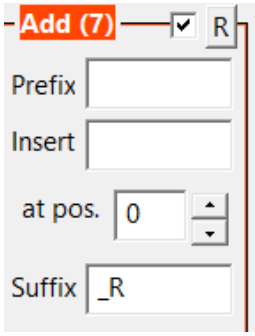
1. Enter the 'even number formula' RegEx into the 'Everything' search field to bring up all files that begin with an even number.

   file: ext:jpg regex:^(\d{6}[02468]) "G:\0 - BRU Test

2. Select and drag these into BRU. Due to an known issue, it may take more then one attempt. If they all didn't display in BRU's Content Panel, Clear (refresh files) and drag again (Hold down Ctrl key when dragging if you want only the new files to display). Update – Fixed in Newest version as of June 25, 2020.



3: 'Section #7: Add' –



Suffix: _R

'Section #5: Remove' – [1]

Last n: 9

Tim Notes:

1. For this example, I removed the extraneous text using 'Section #5: Remove' for clarification, but this is not required.

Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Renaming for Even/Odd Using the 'Everything' Program in Conjunction with BRU cont.

First run – cont.

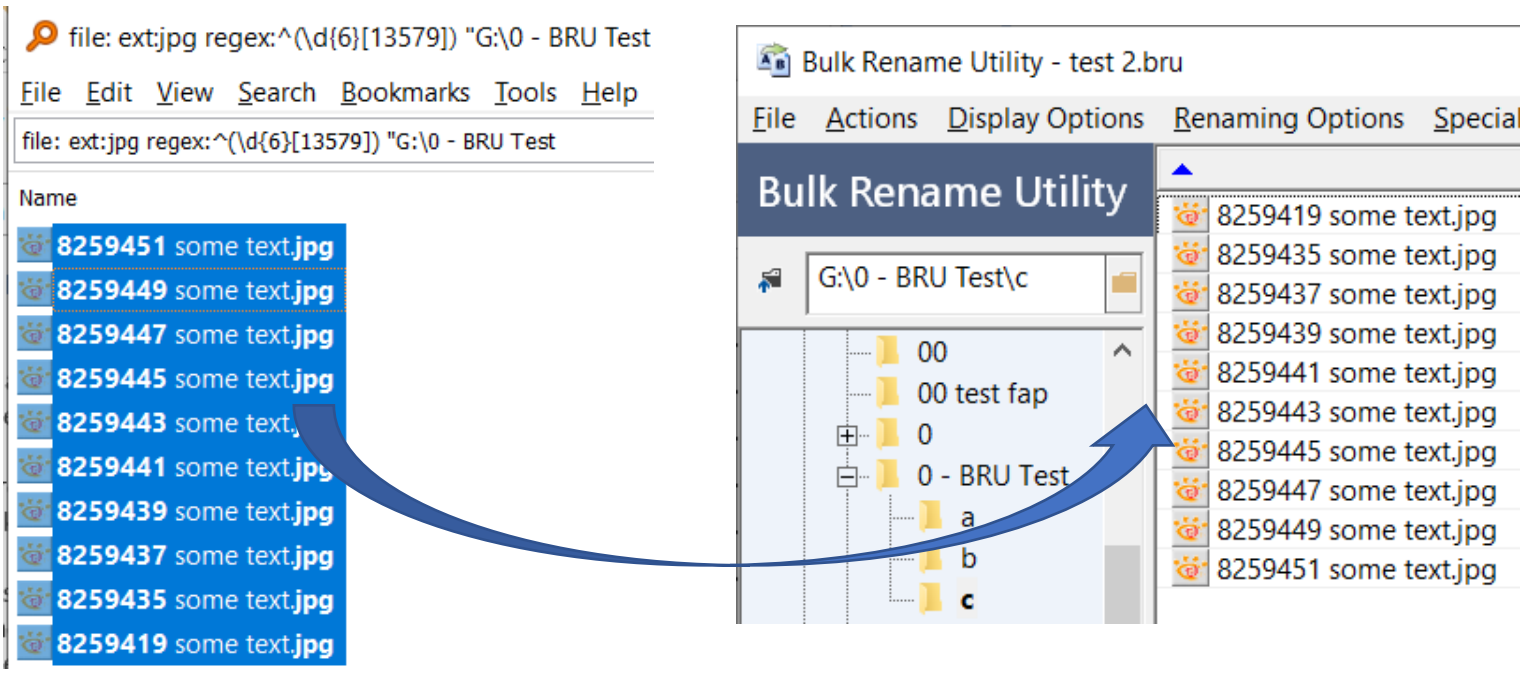
	New Name
8259420 some text.jpg	8259420 _R.jpg
8259422 some text.jpg	8259422 _R.jpg
8259436 some text.jpg	8259436 _R.jpg
8259438 some text.jpg	8259438 _R.jpg
8259440 some text.jpg	8259440 _R.jpg
8259442 some text.jpg	8259442 _R.jpg
8259444 some text.jpg	8259444 _R.jpg
8259446 some text.jpg	8259446 _R.jpg
8259448 some text.jpg	8259448 _R.jpg
8259450 some text.jpg	8259450 _R.jpg

Second run –

1. Enter the 'odd number formula' RegEx into the 'Everything' search field to bring up all files that begin with an odd number.

file: ext:jpg regex:^(\d{6}[13579]) "G:\0 - BRU Test

2. Select and drag these into BRU. Due to an known issue, it may take more then one attempt. If they all didn't display in BRU's Content Panel, Clear (refresh files) and drag again (Hold down Ctrl key when dragging if you want only the new files to display). Update – Fixed in Newest version as of June 25, 2020.



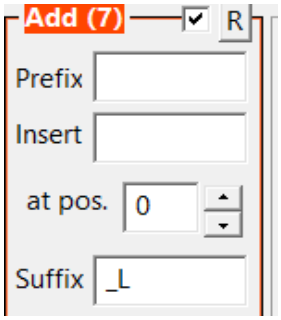
Uncategorized

Add Left, Right Suffix for Even/Odd Numbered Files cont.

Renaming for Even/Odd Using the 'Everything' Program in Conjunction with BRU cont.

Second run – cont.

3: 'Section #7: Add' –



Suffix: _L

'Section #5: Remove' – [1]

Last n: 9

	New Name
8259419 some text.jpg	8259419 _L.jpg
8259435 some text.jpg	8259435 _L.jpg
8259437 some text.jpg	8259437 _L.jpg
8259439 some text.jpg	8259439 _L.jpg
8259441 some text.jpg	8259441 _L.jpg
8259443 some text.jpg	8259443 _L.jpg
8259445 some text.jpg	8259445 _L.jpg
8259447 some text.jpg	8259447 _L.jpg
8259449 some text.jpg	8259449 _L.jpg
8259451 some text.jpg	8259451 _L.jpg

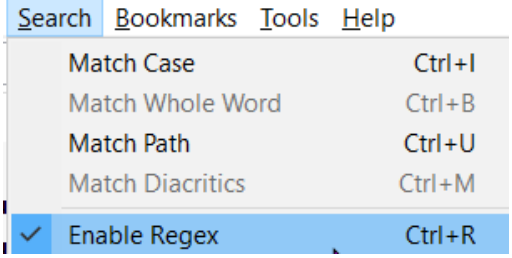
Tim Notes:

1. For this example, I removed the extraneous text using 'Section #5: Remove' for clarification, but this is not required.

2. 'Everything' is a fantastic Freeware Search program available from www.voidtools.com. Some of the examples throughout this book specifically mention this program.

3. This section has been a Tim Contribution with thanks to RegexNinja and original Forum Contributor: therube

4. To Use RegEx with Everything, you can set the option in the Search menu:
   a. This way you don't need to enter: file: ext:jpg regex in the Search Field. However, this will not allow you to filter the extension or specify a directory. When you enable RegEx the entire data field is treated as a RegEx.



Uncategorized

Specifying a Filename Containing a <space> in a BRC /Pattern Switch

12 34.log
This_is an-example

As already discussed, BRC's /Pattern switch has a limitation. Here is a second look at that discussion.

BRC will only process one /Pattern switch. If Multiple /Pattern switches are specified, only the last will be recognized.

This means there is not much flexibility when renaming single files using BRC.

A better implementation would have been:

/Pattern:file1 /Pattern:file2

rather than the current:

/Pattern: "file1 file2"

For example, to rename filenames that contain <spaces> is not possible using /Pattern in BRC because BRC uses the <space> as a delimiter in the /Pattern switch.

So "*12 34*"

Is interpreted in the /Pattern as two separate filespecs instead of one:

*12 – names ending with '12'
34* – names beginning with "34'

To specify a <space> use ? or *

i.e., /Pattern "*12?34*.log"

matches 12 34.log

but it will also match 12x34.log as well.

2. The Wildcard, ' ? ' means to match any single character.
   a. Can match against unintended files.

3. The Wildcard, ' * ' means to match any number of characters.
   a. This can be especially problematic because it will match against many unintended files.

4. Take heed that you weed out your source directory of unintended files before running.

Uncategorized

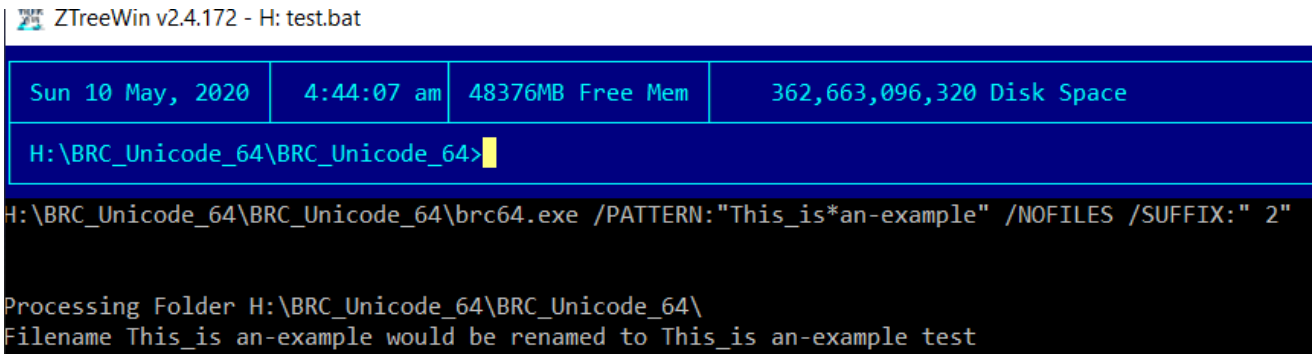
Specifying a Filename Containing a <space> in a BRC /Pattern Switch cont.

Using: This is a example

Tim Contribution –

In a Batch File –

```
@ECHO OFF
REM // the input:
SET PAT=This_is an-example
REM // replace <space> to Wildcard ?:
SET PAT=%PAT: =?%

REM // Specify path to BRC64.exe
SET BRC64=H:\BRC_Unicode_64\BRC_Unicode_64\brc64.exe
BRC64 /PATTERN:"%PAT%" /SUFFIX:" test"
```



Note the

%PAT:<space> = ?%

Instead of simply specifying the variable %PAT%

This is a Replace syntax.

The added syntax of : <xx> = <yy>

converts all-occurrences of xx within varNames value into yy

If varName=12xx34xx56, its value gets overwritten with: 12yy34yy56

Or simply:

replace character xx in string = with yy

So in this example, any <spaces> in the string are converted to ‘?’

Based on a RegEx by Stefan, with additional by RegexNinja

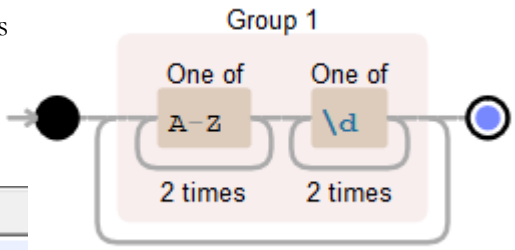
Uncategorized

Match Groupings

AA11BB22CC33DD44

Observations:

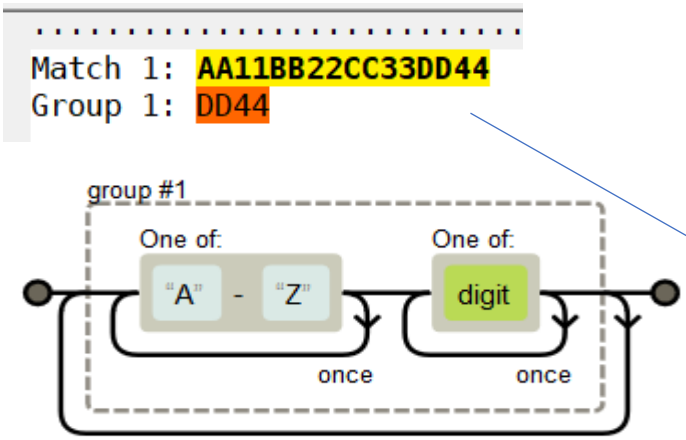
1. Given a string with duplicate patterns, selectively match against those patterns

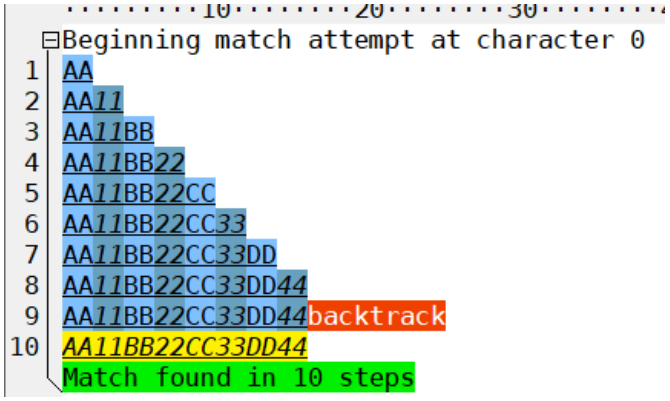


Match: ([A-Z]{2}[\d]{2})+
Replace: \1

	New Name
AA11BB22CC33DD44	DD44

Sample:





Tim Analysis:

1. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 1 = A
2. {2}	Repeat expression for two iterations	Capture Group 1 = AA
3. [\d]	Match against a class consisting of a numeric digit.	Capture Group 1 = AA1
4. {2}	Repeat expression for two iterations	Capture Group 1 = AA11
5. +	Make the entire Expression Greedy	Capture Group 1 = DD44

Why the change from 'AA11', the beginning of the string, to 'DD44', the end of the string? That is the heart of this RegEx. By placing the Greedy Quantifier outside of the Capture Group, instead of making just the last sub-expression Greedy within the Capture Group, it makes the 'entire' expression of the Capture Group Greedy. This means that it will search and match against each group consisting of two uppercase letters followed by two numeric digits until it reaches the EOL, at which time it backtracks to match against the 'DD44'.

So the Matches are:

AA11, BB22, CC33 and finally DD44.

Uncategorized

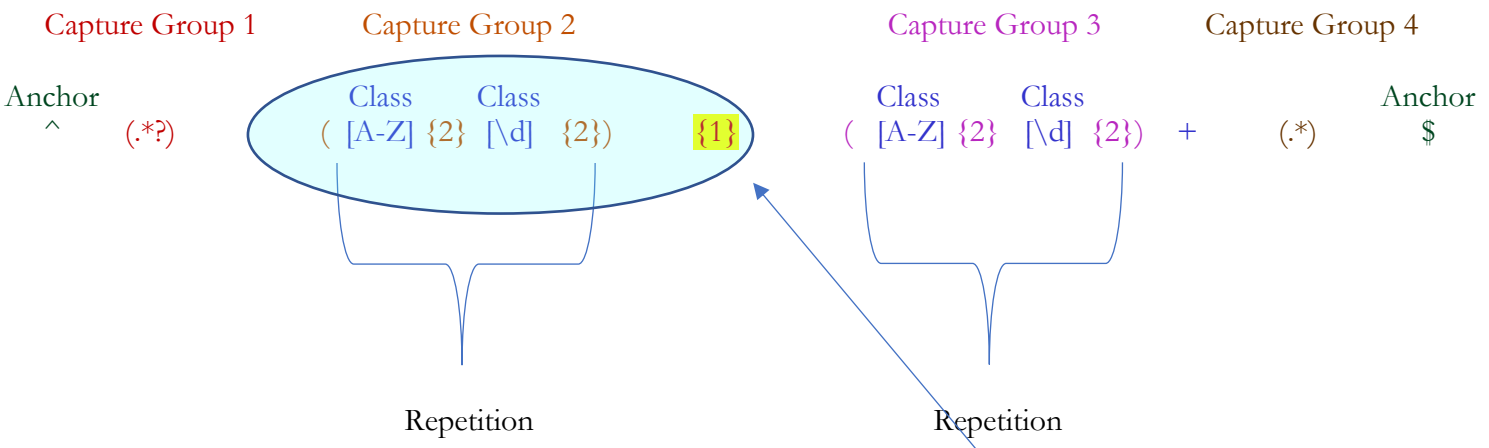
Match Groupings cont.

AA11BB22CC33DD44

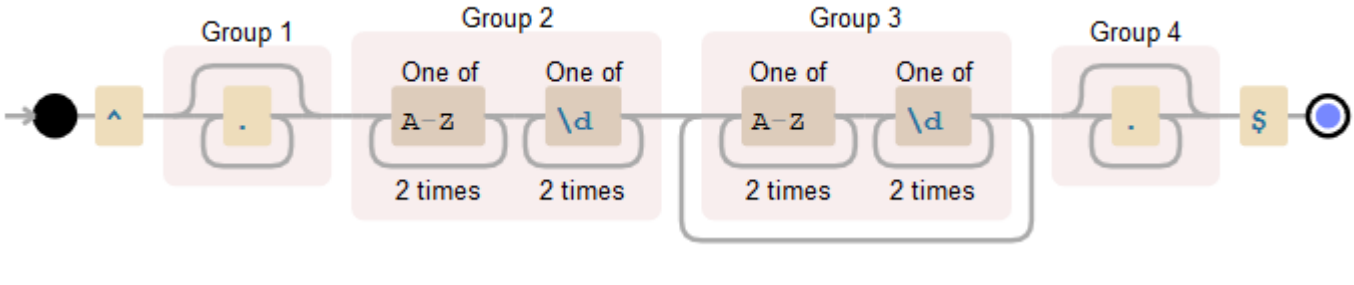
Match: ^(.*?)([A-Z]{2}[\d]{2}){1}([A-Z]{2}[\d]{2})+(.*)$
Replace: \2

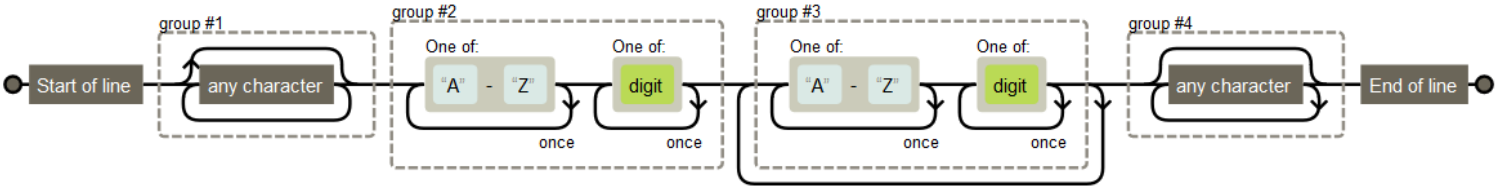
Name	New Name
AA11BB22CC33DD44	AA11

Breaks down:



You will notice that Capture Group 3 is the exact RegEx used in the first example that captured the final group. Therefore in this RegEx, Capture Group 3 will always have this value, but it is the {1} that is the key to the Selective Group Match technique.





Uncategorized

Match Groupings cont.

AA11BB22CC33DD44

Match: ^(.*?)([A-Z]{2}[\d]{2}){1}([A-Z]{2}[\d]{2})+(.*)$

```
                          Start  Length
Match 1: AA11BB22CC33DD44     0      16
Group 1:                      0       0
Group 2: AA11                 0       4
Group 3: DD44                12       4
Group 4:                     16       0
```

```
        10        20        30        40
Beginning match attempt at character 0
 1 ok
 2 ok
 3 ok
 4 AA
 5 AA11
 6 AA11
 7 AA11BB
 8 AA11BB22
 9 AA11BB22CC
10 AA11BB22CC33
11 AA11BB22CC33DD
12 AA11BB22CC33DD44
13 AA11BB22CC33DD44backtrack
14 AA11BB22CC33DD44
15 AA11BB22CC33DD44ok
16 AA11BB22CC33DD44ok
17 AA11BB22CC33DD44ok
Match found in 17 steps
```

Where:

\1 = null \2 = BB11 \3 = DD44 \4 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = A Capture Group 1 = null
6. {2}	Repeat expression for two iterations	Capture Group 2 = AA
7. [\d]	Match against a class consisting of a numeric digit.	Capture Group 2 = AA1
8. {2}	Repeat expression for two iterations	Capture Group 2 = AA11
9. {1}	Make the entire Expression repeat for one iteration.	Capture Group 2 = Unchanged
10. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = B
11. {2}	Repeat expression for two iterations	Capture Group 3 = BB
12. [\d]	Match against a class consisting of a numeric digit.	Capture Group 3 = BB1
13. {2}	Repeat expression for two iterations	Capture Group 3 = BB11
14. +	Make the entire Expression Greedy Current position = EOL	Capture Group 1 = DD44

Why the change from 'BB11' to 'DD44', the end of the string? Same explanation as before. By placing the Greedy Quantifier outside of the Capture Group, it makes the 'entire' expression of the Capture Group Greedy. It will search and match against each group consisting of two uppercase letters followed by two numeric digits until it reaches the EOL, at which time it backtracks to match against the 'DD44'.

15. .*	Match against remaining string	Capture Group 4 = null
16. $	Must be at EOL (End of Line)	

Uncategorized

Match Groupings cont.

AA11BB22CC33DD44

Match: ^(.*?)([A-Z]{2}[\d]{2}){1}([A-Z]{2}[\d]{2})+(.*)$

Understanding the significance of {1} in the Selective Group Match.

AA11 BB22 CC33 DD44

^(.*?)([A-Z]{2}[\d]{2}){1}([A-Z]{2}[\d]{2})+(.*)$ start at 1

AA11 BB22 CC33 DD44

^(.*?)([A-Z]{2}[\d]{2}){2}([A-Z]{2}[\d]{2})+(.*)$ change to 2

AA11 BB22 CC33 DD44

^(.*?)([A-Z]{2}[\d]{2}){3}([A-Z]{2}[\d]{2})+(.*)$ change to 3

AA11 BB22 CC33 DD44

^(.*?)([A-Z]{2}[\d]{2}){1}([A-Z]{2}[\d]{2})+(.*)$ Capture Group 3

By changing the value of the Quantifier after Capture Group 2, I select which of the duplicate pattens I want to match against. Capture Group 3, on the other hand, will always hold the value of the last duplicate pattern in the string.

Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

Keeping only the 10th occurrence of a repeating format.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

Match: ^(.*?)([A-Z]{2}[\d]{2}){10}([A-Z]{2}[\d]{2})+(.*)$
Replace: \2

Name	New Name
AA11BB22CC33DD44EE55FF66GG77HH88II9...	JJ00

AA11 BB22 CC33 DD44 EE55 FF66 GG77 HH88 II99 JJ00 KK11

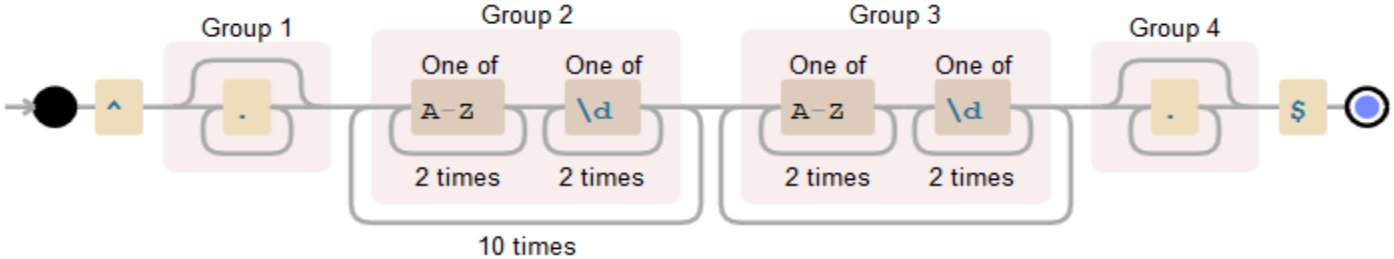
1st 2nd 3rd 4th 5th 6th 7th 8th 9th 10th 11th

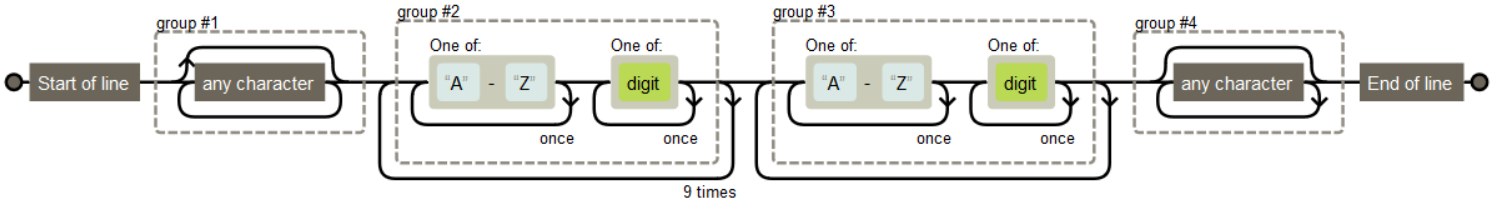
Occurrence
1st = AA11 2nd = BB22 3rd = CC33 4th = DD44
5th = EE55 6th = FF66 7th = GG77 8th = HH88
9th = II99 10th = JJ00 11th = KK11

Capture Group 1 Capture Group 2 Capture Group 3 Capture Group 4

Anchor ^ (.*?) (Class [A-Z] {2} Class [\d] {2}) {10} (Class [A-Z] {2} Class [\d] {2}) + (.*) Anchor $

Repetition Repetition





Uncategorized

Match Groupings cont.

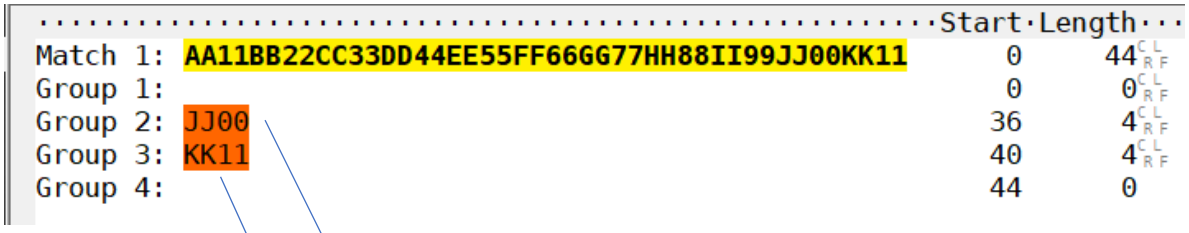
AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

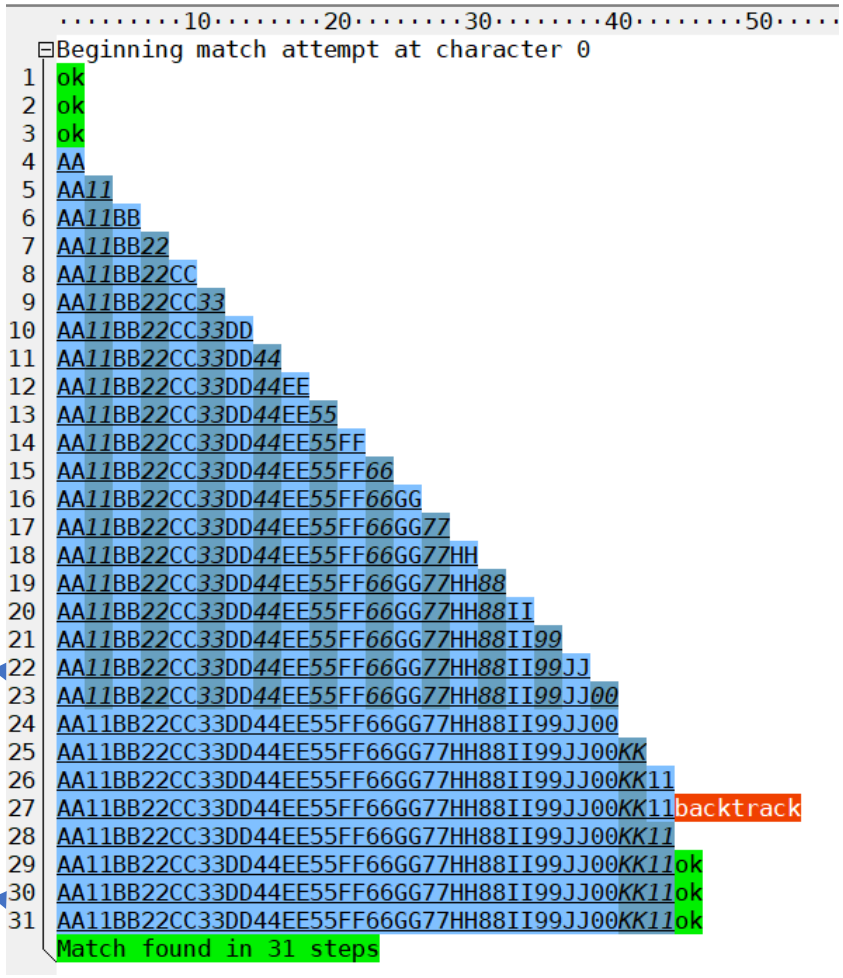
Keeping only the 10th occurrence of a repeating format. cont.

Match: ^(.*?)([A-Z]{2}[\d]{2}){10}([A-Z]{2}[\d]{2})+(.*)$

Where:

\1 = null \2 = JJ00 \3 = KK11 \4 = null





Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

Match: ^(.*?)([A-Z]{2}[\d]{2}){10}([A-Z]{2}[\d]{2})+(.*)$

Where:

\1 = null \2 = JJ00 \3 = KK11 \4 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 2 = A Capture Group 1 = null
6. {2}	Repeat expression for two iterations	Capture Group 2 = AA
7. [\d]	Match against a class consisting of a numeric digit.	Capture Group 2 = AA1
8. {2}	Repeat expression for two iterations	Capture Group 2 = AA11
9. {10}	Make the entire Expression repeat for ten iterations.	Capture Group 2 = JJ00
10. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = K
11. {2}	Repeat expression for two iterations	Capture Group 3 = KK
12. [\d]	Match against a class consisting of a numeric digit.	Capture Group 3 = KK1
13. {2}	Repeat expression for two iterations	Capture Group 3 = KK11
14. +	Make the entire Expression Greedy Current position = EOL	Capture Group 3 = Unchanged

Why the change from 'AA11' to 'JJ00' for Capture Group 2? Because of the {10}, it repeats 10 iterations of Capture Group 2. Capture Group 2 matches against duplicate patterns consisting of 2 uppercase letters followed by 2 numeric digits. So the RegEx Engine moves forward through the string to match ten potential times until satisfying the match at 'JJ00'.

As for Capture Group 4, same explanation as before. By placing the Greedy Quantifier outside of the Capture Group, it makes the 'entire' expression of the Capture Group Greedy. It will search and match against each group consisting of two uppercase letters followed by two numeric digits until it reaches the EOL.. From the previous match at 'JJ00', Capture Group 4 begins at the second position of '0' of 'JJ00'. I mention this because I did not want to leave you with the misconception that Capture Group 4 backtracked to the beginning of the string. It does not. It starts where the last sub-expression left off. It continues to the EOL, at which time it backtracks to match against the 'KK11'.

15. .*	Match against remaining string	Capture Group 4 = null
16. $	Must be at EOL (End of Line)	

Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences.

Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$
Replace: \1\2\5

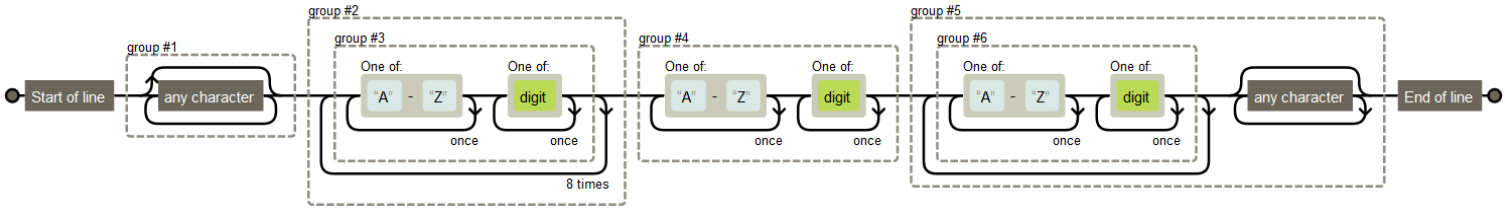
AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11	AA11BB22CC33DD44EE55FF66GG77HH88II99KK11

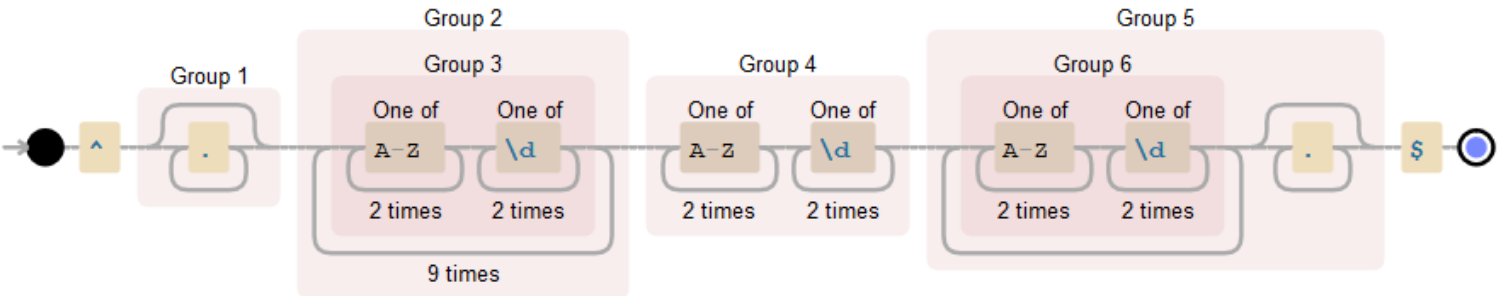
Note that the JJ00 is missing from NewName

Comparison with previous

```
^(.*?)      ([A-Z]{2}[\d]{2} ) {9}      ([A-Z]{2}[\d]{2})                    +  (  .*     )     $
^(.*?)  (   ([A-Z]{2}[\d]{2} ) {9}  )   ([A-Z]{2}[\d]{2})    (   ([A-Z]{2}[\d]{2})    +     .*     )     $
```

The first section of the RegEx that is used to selectively match against the duplicate pattern has now been nested together with the Quantifier (set to the 9th group) that determines which pattern is selected (Groups 2 and 3). The second repetition of the RegEx, ([A-Z]{2}[\d]{2}), is used to isolate the 10th duplicate pattern group (Group 4). It is nested along with Group 5. The last repetition pattern (now Group 5) is still used to capture the last duplicate pattern match. Capture Group 1 retains a null value.





Uncategorized

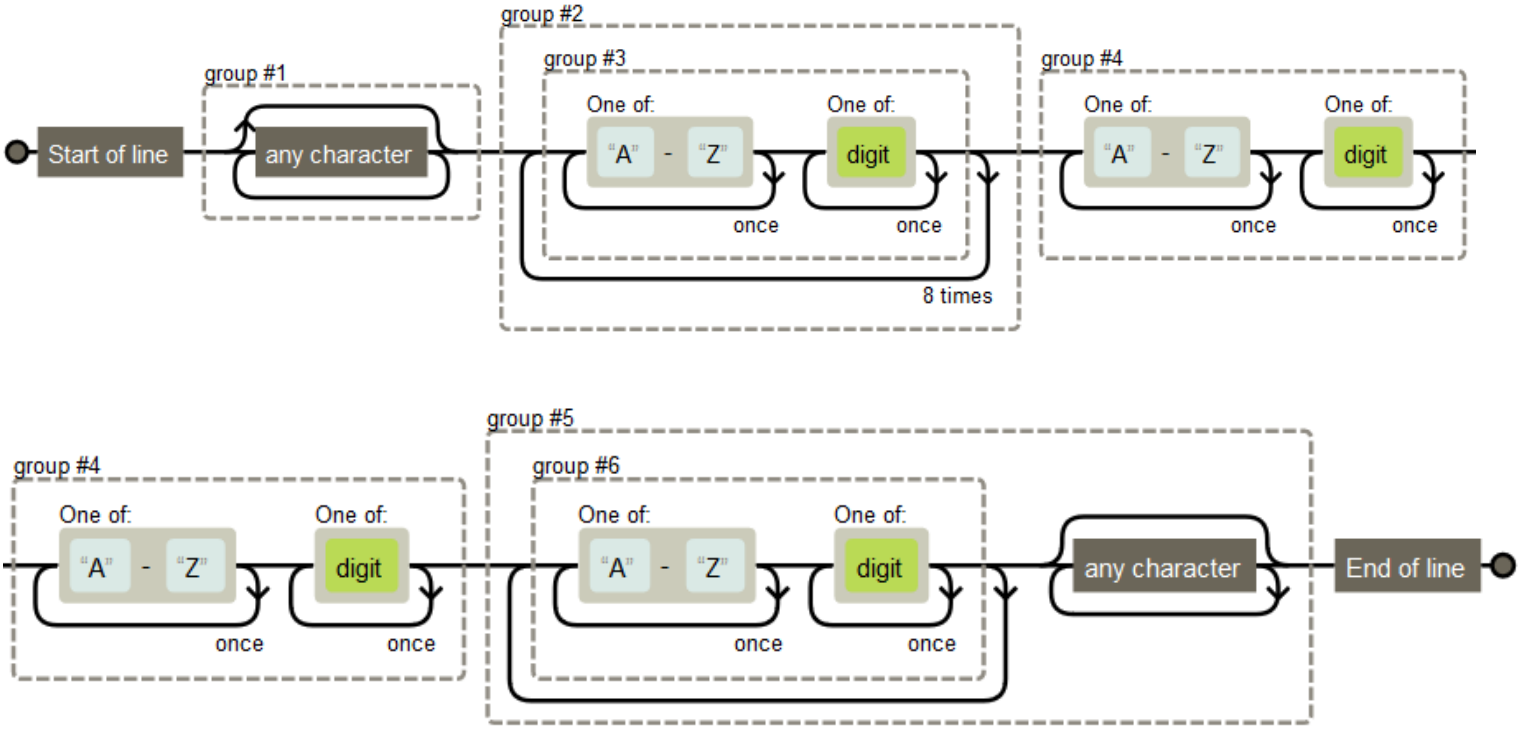
Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences. cont.

Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$

Expanded:



Tim Notes:

1. In order to remove a Duplicate Pattern, you need to capture the pattern before, the pattern to be removed and the pattern after. That is what Capture Groups 3, 4 and 6 do as illustrated on the next page. It captures Pattern # 9, Pattern # 10 and Pattern # 11 respectively.

Uncategorized

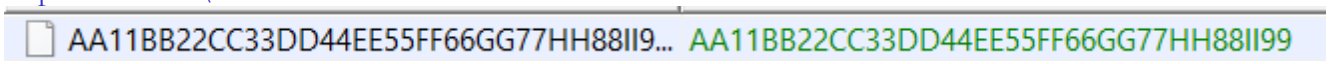
Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences. cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

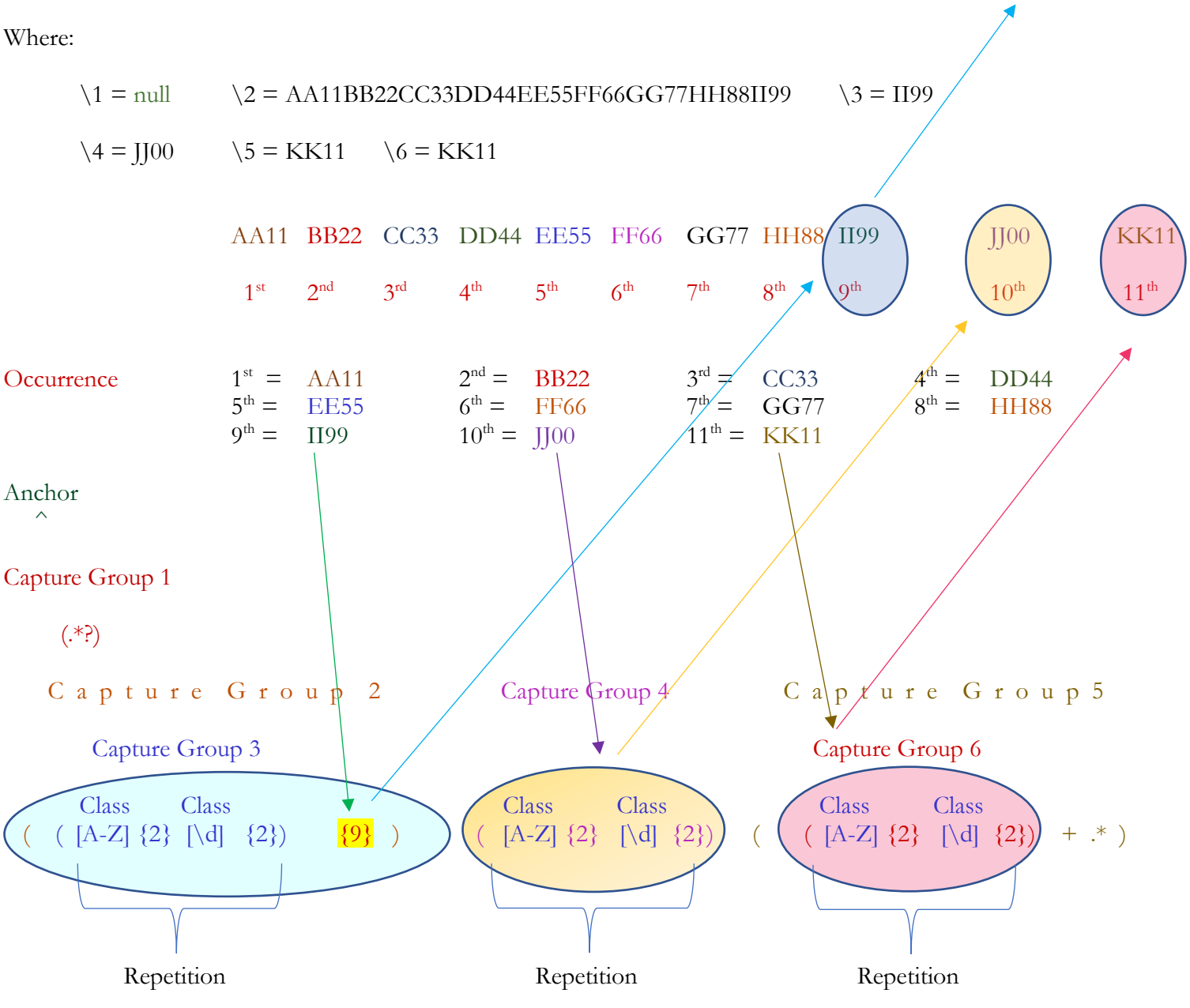
Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$
Replace: \2



Where:

\1 = null \2 = AA11BB22CC33DD44EE55FF66GG77HH88II99 \3 = II99

\4 = JJ00 \5 = KK11 \6 = KK11



Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

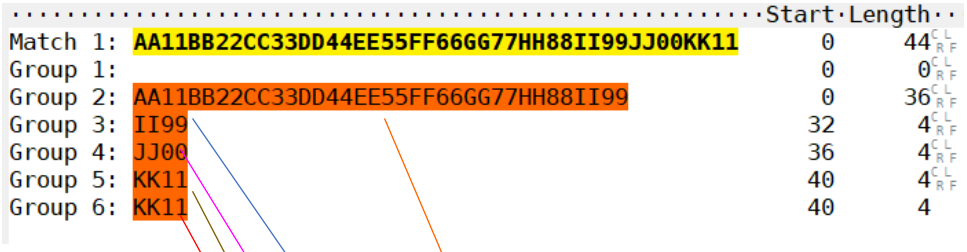
If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences. cont.

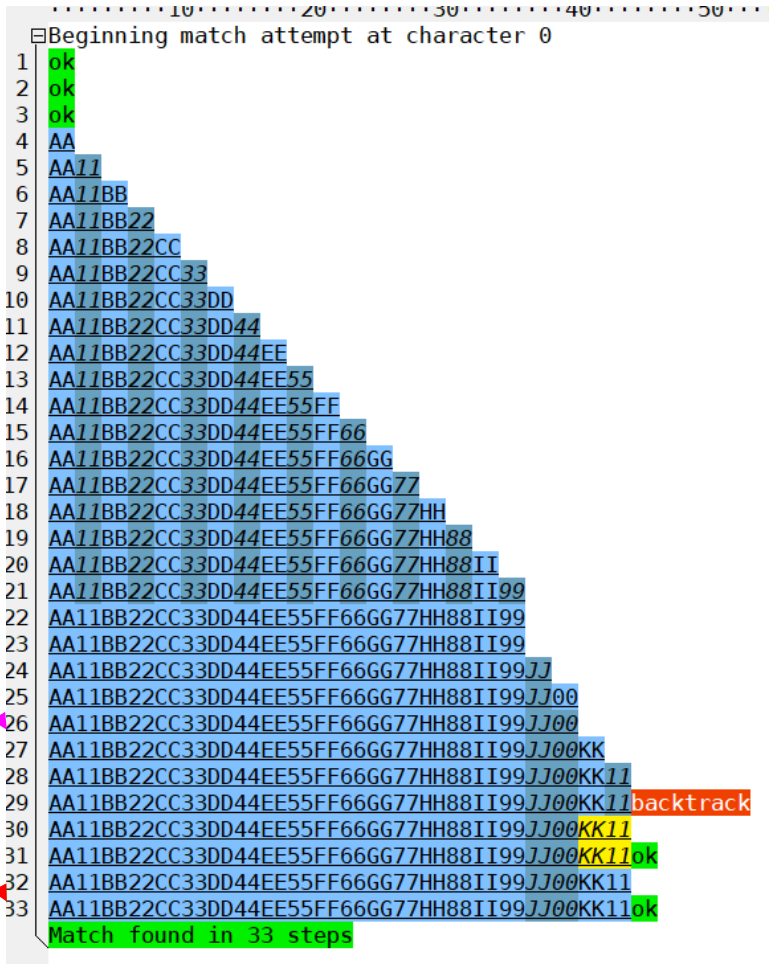
Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$

Where:

\1 = null \2 = AA11BB22CC33DD44EE55FF66GG77HH88II99 \3 = II99

\4 = JJ00 \5 = KK11 \6 = KK11





Tim Notes:

1. Where the arrows intersect are Nested Groups – Capture Groups 2 and 3, and Capture Groups 5 and 6.

2. Color codes for group identification are the same used on preceding page.

Contributor Notes:

1. You can use ([A-Z]{2}[\d]{2})+ in your Filter to show names with 1-or-more of the Pattern Duplicates.

Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences. cont.

Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$

Where:

\1 = null \2 = AA11BB22CC33DD44EE55FF66GG77HH88II99 \3 = II99

\4 = JJ00 \5 = KK11 \6 = KK11

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = A
3. *	Make it Greedy	Capture Group 1 = <entire string>
4. ?	But not too Greedy	Capture Group 1 = BOL
5. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 3 = A Capture Group 2 = A Capture Group 1 = null
6. {2}	Repeat expression for two iterations	Capture Group 3 = AA Capture Group 2 = AA
7. [\d]	Match against a class consisting of a numeric digit.	Capture Group 2 = AA1
8. {2}	Repeat expression for two iterations	Capture Group 2 = AA11
9. {9}	Make the entire Expression repeat for nine iterations.	Capture Group 3 = II99 Capture Group 2 = AA11BB22CC33DD44EE55FF66GG77HH88II99

Tim Notes:

1. The Outer Group's value (Capture Group 2) is inclusive of all Capture Groups within (Capture Group 3).

2. In this case, however, Capture Group 2 includes the value of 'II99' already (steps 5-9) , but you don't end up with 'AA11BB22CC33DD44EE55FF66GG77HH88II99II99' for the value of Capture Group 2.

   Like any other evaluation, when Capture Group 3 is evaluated to 'II99' (step 9), this takes away that value from Capture Group 2 leaving AA11BB22CC33DD44EE55FF66GG77HH88 and Capture Group 3 now has the value 'II99'.

   When this happens Capture Group 2's value is evaluated as including the nested Capture Group 3. Capture Group's 2 value becomes 'AA11BB22CC33DD44EE55FF66GG77HH88II99'.

3. Capture Group 1's value doesn't change from the BOL position, because Capture Group 2 captures the first character, 'A'. Therefore Capture Group 1's value is null. (EOL and BOL have a null value).

Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences. cont.

Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$

Where:

\1 = null \2 = AA11BB22CC33DD44EE55FF66GG77HH88II99 \3 = II99

\4 = JJ00 \5 = KK11 \6 = KK11

Tim Analysis: cont.

10. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 4 = J
11. {2}	Repeat expression for two iterations	Capture Group 4 = JJ
12. [\d]	Match against a class consisting of a numeric digit.	Capture Group 4 = JJ0
13. {2}	Repeat expression for two iterations	Capture Group 4 = JJ00

Why the change from 'AA11' to 'II99' for Capture Group 3? Because of the {9}, it repeats 9 iterations of Capture Group 2, whose singular value is captured by Capture Group 3 . Capture Group 2 matches against duplicate patterns consisting of 2 uppercase letters followed by 2 numeric digits.

The RegEx Engine moves forward through the string to match nine potential times until satisfying the match at 'II99'. Capture Group 2 captures the string up to and including 'II99' but when Capture Group 3 is evaluated, it captures the singular value of 'II99', thus removing it from Capture Group 2.

Capture Group 2 takes the nested (combined) value of 'AA11BB22CC33DD44EE55FF66GG77HH88' for Capture Group 2 and the value of 'II99' for Capture Group 3 resulting in 'AA11BB22CC33DD44EE55FF66GG77HH88I99' for the value of Capture Group 2.

The Greedy Quantifier is not outside of Capture Group 4. Therefore it is a straight move forward and Capture. The RegEx engine position is at the last '9' of 'II99' and moves forward to capture 2 uppercase letters followed by 2 numeric digits = 'JJ00' for Capture Group 4.

Tim Notes:

1. I use 'red' and 'orange' colors to differentiate when Capture Group 3 has the value of 'II99' and Capture Group 2 has the value of 'II99'.

Uncategorized

Match Groupings cont.

AA11BB22CC33DD44EE55FF66GG77HH88II99JJ00KK11

If you wanted to keep everything besides the 10th occurrence when there's 11-or-more occurrences. cont.

Match: ^(.*?)(([A-Z]{2}[\d]{2}){9})([A-Z]{2}[\d]{2})(([A-Z]{2}[\d]{2})+.*)$

Where:

\1 = null \2 = AA11BB22CC33DD44EE55FF66GG77HH88II99 \3 = II99

\4 = JJ00 \5 = KK11 \6 = KK11

Tim Analysis: cont.

14. [A-Z]	Match against a class consisting of an uppercase letter	Capture Group 6 = K Capture Group 5 = K
15. {2}	Repeat expression for two iterations	Capture Group 6 = KK Capture Group 5 = KK
16. [\d]	Match against a class consisting of a numeric digit.	Capture Group 6 = KK1 Capture Group 5 = KK1
17. {2}	Repeat expression for two iterations	Capture Group 6 = KK11 Capture Group 5 = KK11
18. +	Make the entire Expression Greedy Current position = EOL	Capture Group 6 = Unchanged Capture Group 5 = Unchanged

By placing the Greedy Quantifier outside of Capture Group 5, it makes the 'entire' expression of the Capture Group Greedy. It will search and match against each group consisting of two uppercase letters followed by two numeric digits until it reaches the EOL..

From the previous match at 'JJ00', Capture Group 5 begins at the second position of '0' of 'JJ00'. I mention this because I did not want to leave you with the misconception that Capture Group 5 backtracked to the beginning of the string. It does not. It starts where the last sub-expression left off. It continues to the EOL, at which time it backtracks to match against the 'KK11'.

Because it is nested, Capture Group 6 takes on the same values. The RegEx Engine has already reached the EOL by the time it evaluates the .*, but because it is inclusive to Capture Group 5, it does not produce a Zero Occurrence Match resulting in a null value, but instead retains the previous value of 'KK11'.

19. .*	Match against remaining string	Capture Group 6 = Unchanged Capture Group 5 = Unchanged
20. $	Must be at EOL (End of Line)	

Forum Contributor: RegexNinja

End section ~

Regular Expressions

[This Page Intentionally Blank]

Expert's Corner - JavaScript

Tim Note:

The following are taken from the public domain BRU Forums. Contributors of the final JavaScript used will be noted.

The best method of learning is to study from others. Here are many examples that you can try for yourself and do your own analysis. I have tested every single example and have included photos with each to demonstrate that they work. Forum posts have been selected based on their usefulness and tutorial benefit.

Multiple runs require that criteria settings be reset between runs using the 'Reset' button found next to the 'Rename' button located at the bottom right of the interface. It should be noted that using the Reset will remove the checkmark setting for the 'Section #14: JavaScript', that selects this criteria, but does not remove the script itself from the JavaScript Code form. Therefore to re-run the script, you only need to re-enable the criteria by setting the Checkmark again.

It is recommended to have JavaScript sugar.js and date.js libraries included depending on your script. This can be done easily through the 'JavaScript Libraries' sub-menu of the Special Menu. If a required library is not included , the script will generate an error message when run. If the library is not required, it is recommended to not have it included as it can slow down processing.

JavaScript requires an inexpensive commercial license to unlock the feature. Home users can certainly benefit without having to be in business and it helps support the company for further BRU development as well.

The JavaScript currently supported in BRU is ECMAScript 5 2009.

It should also be mentioned that any coloring is mainly specific to each section, and in many cases each page. Any meaning therein does not extend to other sections or pages. There are exceptions which will become obvious. The coloring is for clarification and to allow the user to more easily follow the topic of discussion.

JavaScript

Convert Written Date to YMD

For example, January 3, 2020

Sample:

1234567890 January 6, 2020.pdf to 1234567890 20200106.pdf

Observations:

1. It will assume that the date always starts after the first <space>.

Two Methods:

Method #1 -

```
function getSecondPart(str) {

  var index = str.indexOf(" ");                    // Gets the first index where a <space> occurs
  return name.substring(index);
}
function getFirstPart(str) {
  var index = str.indexOf(" ");                    // Gets the first index where a <space> occurs
  return name.substring(0,index);
}

var d = new Date(Date.parse(getSecondPart(name)));
var mm = d.getMonth() + 1;                         // getMonth() is zero-based
var dd = d.getDate();

var yyyymmdd =  [d.getFullYear(),
        (mm>9 ? '' : '0') + mm,
        (dd>9 ? '' : '0') + dd
        ].join('');

newName = getFirstPart(name) + ' ' + yyyymmdd;
```

Name	New Name
1234567890 January 6, 2020.pdf	1234567890 20200106.pdf
1234567890 February 15, 2020.pdf	1234567890 20200215.pdf

JavaScript

Convert Written Date to YMD cont.

1234567890 January 6, 2020.pdf to 1234567890 20200106.pdf

Method #2 -

```
Date.prototype.yyyymmdd = function() {
var mm = this.getMonth() + 1;                    // getMonth() is zero-based
var dd = this.getDate();

 return [this.getFullYear(),
 (mm>9 ? '' : '0') + mm,
 (dd>9 ? '' : '0') + dd
 ].join('');
 };

 function getSecondPart(str) {
 var index = str.indexOf(" ");                   // Gets the first index where a <space> occurs
 return name.substring(index);
 }
 function getFirstPart(str) {
 var index = str.indexOf(" ");                   // Gets the first index where a <space> occurs
 return name.substring(0,index);
 }

 var d = new Date(Date.parse(getSecondPart(name)));

 newName = getFirstPart(name) + ' ' + d.yyyymmdd();
```

Name	New Name
1234567890 January 6, 2020.pdf	1234567890 20200106.pdf
1234567890 February 15, 2020.pdf	1234567890 20200215.pdf

Forum Contributor: Admin, therube

Tim Notes:

Thanks to the Admin who really went out of his way to help me with a problem regarding this.

JavaScript

Convert YMD to Written Date

234567890 20200106.pdf to 1234567890 January 6, 2020.pdf

Observations:

1. It will assume that the date always starts after the first <space>.

```
function parseyyyymmdd(str) {
   if(!/^(\d){8}$/.test(str)) return "invalid date";
   var y = str.substr(0,4),
       m = str.substr(4,2),
       d = str.substr(6,2);
   return new Date(y,m,d);
}

function getSecondPart(str) {
   return str.split(' ')[1];
}

function getFirstPart(str) {
   return str.split(' ')[0];
}

var d = parseyyyymmdd(getSecondPart(name));

newName = getFirstPart(name) + ' ' + d.toString("MMMM dd, yyyy");

// for all toString formats see: https://github.com/datejs/Datejs
```

Name	New Name
234567890 20200106.pdf	234567890 February 06, 2020.pdf

Forum Contributor: Admin

JavaScript

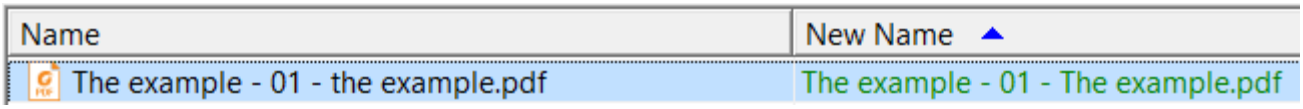
Change letter in any position in string from Lowercase to Uppercase

The example - 01 - the example.pdf

```
String.prototype.replaceAt=function(index, replacement) {
        return this.substr(0, index) + replacement+ this.substr(index + replacement.length);
}

var pos = 20;
pos--;
var str = name;
var strup = str.toUpperCase();
strup = str.replaceAt(pos,strup.substr(pos,1));

newName = strup;
```

Contributor Notes:

1. var pos = 20 specifies position and it can be changed as needed.



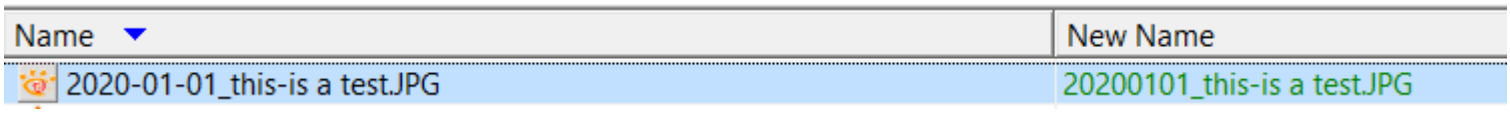
Forum Contributor: Admin

JavaScript

Remove hyphens between date

2020-01-01_this-is a test.JPG

```
ind = 10;                                                    // up to position
str_repl = "-";                                              // text to replace
str_with = "";                                               // replace with

firstPart = name.substring(0, ind);
secondPart = name.substring(ind);
firstPart = firstPart.replace(new RegExp(str_repl, 'gi'), str_with);

newName = firstPart + secondPart;
```



Forum Contributor: Admin

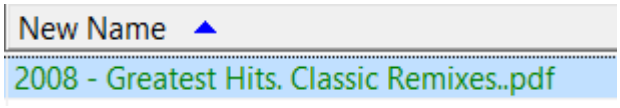
JavaScript

Remove Redundant Information

Sample:

2008 - Greatest Hits. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes.pdf

```
// remove multiple, leading or trailing <spaces>
function trim(s) {
   s = s.replace(/(^\s*)|(\s*$)/gi,"");
   s = s.replace(/[ ]{2,}/gi," ");
   s = s.replace(/\n /,"\n");
   return s;
}


var sep = ".";
str = name.split(".");
var result = [];
for(var i =0; i < str.length ; i++){
  var ent = trim(str[i]);
  if(result.indexOf(ent) == -1) {
   result.push(ent);
   }
}

newName = result.join(sep + " ");
newName += sep;
```



JavaScript

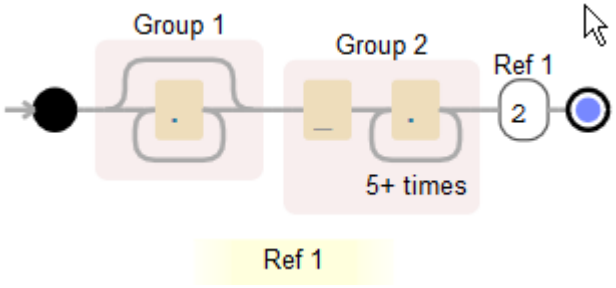
Remove Redundant Information cont.

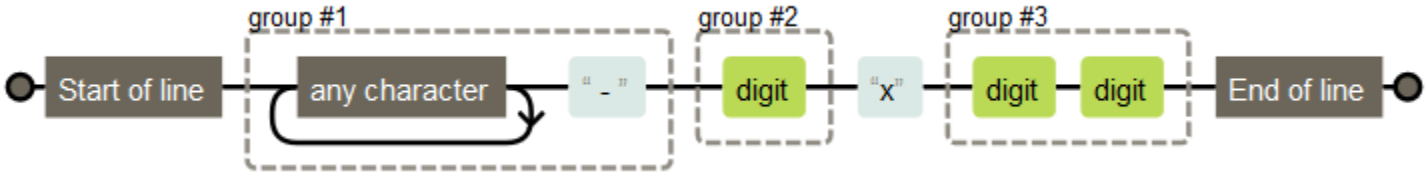
Sample:

2008 - Greatest Hits. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes. Classic Remixes.pdf

Here is a RegEx equivalent (although it is not universal)

Match: (.*?)(.{5,}?)\2
Replace: \1\2





Sample:

		Start	Length
Match 1:	2008 - Greatest Hits. Classic Remixes. Classic Remixes.	0	55
Group 1:	2008 - Greatest Hits.	0	21
Group 2:	Classic Remixes.	21	17
Match 2:	Classic Remixes. Classic Remixes.	55	34
Group 1:		55	0
Group 2:	Classic Remixes.	55	17
Match 3:	Classic Remixes. Classic Remixes.	89	34
Group 1:		89	0
Group 2:	Classic Remixes.	89	17
Match 4:	Classic Remixes. Classic Remixes.	123	34
Group 1:		123	0
Group 2:	Classic Remixes.	123	17
Match 5:	Classic Remixes. Classic Remixes.	157	34
Group 1:		157	0
Group 2:	Classic Remixes.	157	17
Match 6:	Classic Remixes. Classic Remixes.	191	34
Group 1:		191	0
Group 2:	Classic Remixes.	191	17

Forum Contributor: JavaScript – Admin, RegEx - glennrune

JavaScript

Convert High ASCII and Unicode to ASCII (1)

```
var defaultDiacriticsRemovalMap = [
```

```
{'base':'A',
'letters':'\u0041\u24B6\uFF21\u00C0\u00C1\u00C2\u1EA6\u1EA4\u1EAA\u1EA8\u00C3\u0100\u0102\u1EB0\u1EAE\u1EB4\u1EB2\u0226\u01E0\u00C4\u01DE\u1EA2\u00C5\u01FA\u01CD\u0200\u0202\u1EA0\u1EAC\u1EB6\u1E00\u0104\u023A\u2C6F'},

{'base':'AA','letters':'\uA732'},

{'base':'AE','letters':'\u00C6\u01FC\u01E2'},

{'base':'AO','letters':'\uA734'},

{'base':'AU','letters':'\uA736'},

{'base':'AV','letters':'\uA738\uA73A'},

{'base':'AY','letters':'\uA73C'},

{'base':'B', 'letters':'\u0042\u24B7\uFF22\u1E02\u1E04\u1E06\u0243\u0182\u0181'},

{'base':'C', 'letters':'\u0043\u24B8\uFF23\u0106\u0108\u010A\u010C\u00C7\u1E08\u0187\u023B\uA73E'},

{'base':'D',
'letters':'\u0044\u24B9\uFF24\u1E0A\u010E\u1E0C\u1E10\u1E12\u1E0E\u0110\u018B\u018A\u0189\uA779\u00D0'},

{'base':'DZ','letters':'\u01F1\u01C4'},

{'base':'Dz','letters':'\u01F2\u01C5'},

{'base':'E',
'letters':'\u0045\u24BA\uFF25\u00C8\u00C9\u00CA\u1EC0\u1EBE\u1EC4\u1EC2\u1EBC\u0112\u1E14\u1E16\u0114\u0116\u00CB\u1EBA\u011A\u0204\u0206\u1EB8\u1EC6\u0228\u1E1C\u0118\u1E18\u1E1A\u0190\u018E'},

{'base':'F', 'letters':'\u0046\u24BB\uFF26\u1E1E\u0191\uA77B'},

{'base':'G',
'letters':'\u0047\u24BC\uFF27\u01F4\u011C\u1E20\u011E\u0120\u01E6\u0122\u01E4\u0193\uA7A0\uA77D\uA77E'},

{'base':'H',
'letters':'\u0048\u24BD\uFF28\u0124\u1E22\u1E26\u021E\u1E24\u1E28\u1E2A\u0126\u2C67\u2C75\uA78D'},
```

JavaScript

Convert High ASCII and Unicode to ASCII cont. (2)

```
{'base':'I',
'letters':'\u0049\u24BE\uFF29\u00CC\u00CD\u00CE\u0128\u012A\u012C\u0130\u00CF\u1E2E\u1EC8\u01CF\
u0208\u020A\u1ECA\u012E\u1E2C\u0197'},

{'base':'J', 'letters':'\u004A\u24BF\uFF2A\u0134\u0248'},

{'base':'K',
'letters':'\u004B\u24C0\uFF2B\u1E30\u01E8\u1E32\u0136\u1E34\u0198\u2C69\uA740\uA742\uA744\uA7A2'},

{'base':'L',
'letters':'\u004C\u24C1\uFF2C\u013F\u0139\u013D\u1E36\u1E38\u013B\u1E3C\u1E3A\u0141\u023D\u2C62\u
2C60\uA748\uA746\uA780'},

{'base':'LJ','letters':'\u01C7'},

{'base':'Lj','letters':'\u01C8'},


{'base':'M', 'letters':'\u004D\u24C2\uFF2D\u1E3E\u1E40\u1E42\u2C6E\u019C'},

{'base':'N',
'letters':'\u004E\u24C3\uFF2E\u01F8\u0143\u00D1\u1E44\u0147\u1E46\u0145\u1E4A\u1E48\u0220\u019D\uA790\uA7A4'},

{'base':'NJ','letters':'\u01CA'},

{'base':'Nj','letters':'\u01CB'},

{'base':'O',
'letters':'\u004F\u24C4\uFF2F\u00D2\u00D3\u00D4\u1ED2\u1ED0\u1ED6\u1ED4\u00D5\u1E4C\u022C\u1E4E\u014C\u1E50\u1E5
2\u014E\u022E\u0230\u00D6\u022A\u1ECE\u0150\u01D1\u020C\u020E\u01A0\u1EDC\u1EDA\u1EE0\u1EDE\u1EE2\u1ECC\u1E
D8\u01EA\u01EC\u00D8\u01FE\u0186\u019F\uA74A\uA74C'},

{'base':'OI','letters':'\u01A2'},

{'base':'OO','letters':'\uA74E'},

{'base':'OU','letters':'\u0222'},

{'base':'OE','letters':'\u008C\u0152'},

{'base':'oe','letters':'\u009C\u0153'},

{'base':'P', 'letters':'\u0050\u24C5\uFF30\u1E54\u1E56\u01A4\u2C63\uA750\uA752\uA754'},

{'base':'Q', 'letters':'\u0051\u24C6\uFF31\uA756\uA758\u024A'},

{'base':'R',
'letters':'\u0052\u24C7\uFF32\u0154\u1E58\u0158\u0210\u0212\u1E5A\u1E5C\u0156\u1E5E\u024C\u2C64\uA75A\uA7A6\uA782'},

{'base':'S',
'letters':'\u0053\u24C8\uFF33\u1E9E\u015A\u1E64\u015C\u1E60\u0160\u1E66\u1E62\u1E68\u0218\u015E\u2C7E\uA7A8\uA784'},
```

JavaScript

Convert High ASCII and Unicode to ASCII cont. (3)


```
{'base':'T', 'letters':'\u0054\u24C9\uFF34\u1E6A\u0164\u1E6C\u021A\u0162\u1E70\u1E6E\u0166\u01AC\u01AE\u023E\uA786'},

{'base':'TZ','letters':'\uA728'},

{'base':'U',
'letters':'\u0055\u24CA\uFF35\u00D9\u00DA\u00DB\u0168\u1E78\u016A\u1E7A\u016C\u00DC\u01DB\u01D7\u01D5\u01D9\u1EE6
\u016E\u0170\u01D3\u0214\u0216\u01AF\u1EEA\u1EE8\u1EEE\u1EEC\u1EF0\u1EE4\u1E72\u0172\u1E76\u1E74\u0244'},

{'base':'V', 'letters':'\u0056\u24CB\uFF36\u1E7C\u1E7E\u01B2\uA75E\u0245'},

{'base':'VY','letters':'\uA760'},

{'base':'W', 'letters':'\u0057\u24CC\uFF37\u1E80\u1E82\u0174\u1E86\u1E84\u1E88\u2C72'},

{'base':'X', 'letters':'\u0058\u24CD\uFF38\u1E8A\u1E8C'},

{'base':'Y', 'letters':'\u0059\u24CE\uFF39\u1EF2\u00DD\u0176\u1EF8\u0232\u1E8E\u0178\u1EF6\u1EF4\u01B3\u024E\u1EFE'},

{'base':'Z', 'letters':'\u005A\u24CF\uFF3A\u0179\u1E90\u017B\u017D\u1E92\u1E94\u01B5\u0224\u2C7F\u2C6B\uA762'},


{'base':'a',
'letters':'\u0061\u24D0\uFF41\u1E9A\u00E0\u00E1\u00E2\u1EA7\u1EA5\u1EAB\u1EA9\u00E3\u0101\u0103\u1EB1\u1EAF\u1EB5\
u1EB3\u0227\u01E1\u00E4\u01DF\u1EA3\u00E5\u01FB\u01CE\u0201\u0203\u1EA1\u1EAD\u1EB7\u1E01\u0105\u2C65\u0250'},

{'base':'aa','letters':'\uA733'},

{'base':'ae','letters':'\u00E6\u01FD\u01E3'},

{'base':'ao','letters':'\uA735'},

{'base':'au','letters':'\uA737'},

{'base':'av','letters':'\uA739\uA73B'},

{'base':'ay','letters':'\uA73D'},

{'base':'b', 'letters':'\u0062\u24D1\uFF42\u1E03\u1E05\u1E07\u0180\u0183\u0253'},

{'base':'c', 'letters':'\u0063\u24D2\uFF43\u0107\u0109\u010B\u010D\u00E7\u1E09\u0188\u023C\uA73F\u2184'},

{'base':'d', 'letters':'\u0064\u24D3\uFF44\u1E0B\u010F\u1E0D\u1E11\u1E13\u1E0F\u0111\u018C\u0256\u0257\uA77A'},

{'base':'dz','letters':'\u01F3\u01C6'},

{'base':'e',
'letters':'\u0065\u24D4\uFF45\u00E8\u00E9\u00EA\u1EC1\u1EBF\u1EC5\u1EC3\u1EBD\u0113\u1E15\u1E17\u0115\u0117\u00EB\u
1EBB\u011B\u0205\u0207\u1EB9\u1EC7\u0229\u1E1D\u0119\u1E19\u1E1B\u0247\u025B\u01DD'},

{'base':'f', 'letters':'\u0066\u24D5\uFF46\u1E1F\u0192\uA77C'},

{'base':'g', 'letters':'\u0067\u24D6\uFF47\u01F5\u011D\u1E21\u011F\u0121\u01E7\u0123\u01E5\u0260\uA7A1\u1D79\uA77F'},

{'base':'h', 'letters':'\u0068\u24D7\uFF48\u0125\u1E23\u1E27\u021F\u1E25\u1E29\u1E2B\u1E96\u0127\u2C68\u2C76\u0265'},
```

JavaScript

Convert High ASCII and Unicode to ASCII cont. (4)

```
{'base':'hv','letters':'\u0195'},

{'base':'i',
'letters':'\u0069\u24D8\uFF49\u00EC\u00ED\u00EE\u0129\u012B\u012D\u00EF\u1E2F\u1EC9\u01D0\u0209\u020B\u1ECB\u012F\u1E2D\u0268\u0131'},

{'base':'j', 'letters':'\u006A\u24D9\uFF4A\u0135\u01F0\u0249'},

{'base':'k', 'letters':'\u006B\u24DA\uFF4B\u1E31\u01E9\u1E33\u0137\u1E35\u0199\u2C6A\uA741\uA743\uA745\uA7A3'},

{'base':'l',
'letters':'\u006C\u24DB\uFF4C\u0140\u013A\u013E\u1E37\u1E39\u013C\u1E3D\u1E3B\u017F\u0142\u019A\u026B\u2C61\uA749\uA781\uA747'},

{'base':'lj','letters':'\u01C9'},

{'base':'m', 'letters':'\u006D\u24DC\uFF4D\u1E3F\u1E41\u1E43\u0271\u026F'},

{'base':'n',
'letters':'\u006E\u24DD\uFF4E\u01F9\u0144\u00F1\u1E45\u0148\u1E47\u0146\u1E4B\u1E49\u019E\u0272\u0149\uA791\uA7A5'},


{'base':'nj','letters':'\u01CC'},

{'base':'o',
'letters':'\u006F\u24DE\uFF4F\u00F2\u00F3\u00F4\u1ED3\u1ED1\u1ED7\u1ED5\u00F5\u1E4D\u022D\u1E4F\u014D\u1E51\u1E53\u014F\u022F\u0231\u00F6\u022B\u1ECF\u0151\u01D2\u020D\u020F\u01A1\u1EDD\u1EDB\u1EE1\u1EDF\u1EE3\u1ECD\u1ED9\u01EB\u01ED\u00F8\u01FF\u0254\uA74B\uA74D\u0275'},

{'base':'oi','letters':'\u01A3'},

{'base':'ou','letters':'\u0223'},

{'base':'oo','letters':'\uA74F'},

{'base':'p','letters':'\u0070\u24DF\uFF50\u1E55\u1E57\u01A5\u1D7D\uA751\uA753\uA755'},

{'base':'q','letters':'\u0071\u24E0\uFF51\u024B\uA757\uA759'},

{'base':'r','letters':'\u0072\u24E1\uFF52\u0155\u1E59\u0159\u0211\u0213\u1E5B\u1E5D\u0157\u1E5F\u024D\u027D\uA75B\uA7A7\uA783'},

{'base':'s','letters':'\u0073\u24E2\uFF53\u00DF\u015B\u1E65\u015D\u1E61\u0161\u1E67\u1E63\u1E69\u0219\u015F\u023F\uA7A9\uA785\u1E9B'},

{'base':'t','letters':'\u0074\u24E3\uFF54\u1E6B\u1E97\u0165\u1E6D\u021B\u0163\u1E71\u1E6F\u0167\u01AD\u0288\u2C66\uA787'},

{'base':'tz','letters':'\uA729'},

{'base':'u','letters':
'\u0075\u24E4\uFF55\u00F9\u00FA\u00FB\u0169\u1E79\u016B\u1E7B\u016D\u00FC\u01DC\u01D8\u01D6\u01DA\u1EE7\u016F\u0171\u01D4\u0215\u0217\u01B0\u1EEB\u1EE9\u1EEF\u1EED\u1EF1\u1EE5\u1E73\u0173\u1E77\u1E75\u0289'},

{'base':'v','letters':'\u0076\u24E5\uFF56\u1E7D\u1E7F\u028B\uA75F\u028C'},
```

JavaScript

Convert High ASCII and Unicode to ASCII cont. (5)

```
{'base':'vy','letters':'\uA761'},

{'base':'w','letters':'\u0077\u24E6\uFF57\u1E81\u1E83\u0175\u1E87\u1E85\u1E98\u1E89\u2C73'},


{'base':'x','letters':'\u0078\u24E7\uFF58\u1E8B\u1E8D'},

{'base':'y','letters':'\u0079\u24E8\uFF59\u1EF3\u00FD\u0177\u1EF9\u0233\u1E8F\u00FF\u1EF7\u1E99\u1EF5\u01B4\u024F\u1EFF'},

{'base':'z','letters':'\u007A\u24E9\uFF5A\u017A\u1E91\u017C\u017E\u1E93\u1E95\u01B6\u0225\u0240\u2C6C\uA763'}
];
```

```
var diacriticsMap = {};
for (var i=0; i < defaultDiacriticsRemovalMap .length; i++){
  var letters = defaultDiacriticsRemovalMap [i].letters;
    for (var j=0; j < letters.length ; j++){
      diacriticsMap[letters[j]] = defaultDiacriticsRemovalMap [i].base;
    }
}

// "what?" version ...http://jsperf.com/diacritics/12

function removeDiacritics (str) {
  return str.replace(/[^\u0000-\u007E]/g, function(a){
  return diacriticsMap[a] || a;
  });
}

newName = removeDiacritics(origName);
```

Name	New Name
0 - also see 'filenames with embedded ÿ char	0 - also see 'filenames with embedded y char
Bérénice Marlohe	Berenice Marlohe

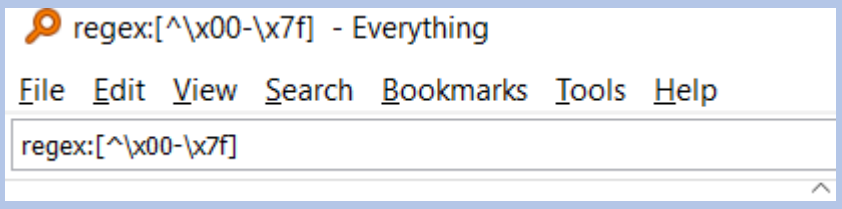
JavaScript

Convert High ASCII and Unicode to ASCII cont. (6)

Tim Notes:

1. You can find files that contain Unicode or high ASCII using the search program, 'Everything'. 'Everything' accepts RegEx:

Enter this in the search field: regex:[^\x00-\x7f]



Contributor Analysis:

1. RegEx: enable RegEx search
2. [ ] specify a set of characters
3. ^ NOT in this character set
4. \x00-\x7f the ASCII character range – extended ASCII in this case

Forum Contributor: Admin

JavaScript

Removing Repeated Words in File Names

tr43-Apocalyptic_Dreams_Stem1_Drums_Stem1_Drums.aiff to tr43-Apocalyptic_Dreams_Stem1_Drums.aiff
tr44-Apocalyptic_Dreams_Stem2_Choir_Stem2_Choir.aiff to tr44-Apocalyptic_Dreams_Stem2_Choir.aiff
tr133-Apocalyptic_Dreams_30_Second_30_Second.aiff to tr133-Apocalyptic_Dreams_30_Second.aiff

tr45-Apocalyptic_Dreams_Stem3_Percussion_Stem3_Percussion.aiff
to
tr45-Apocalyptic_Dreams_Stem3_Percussion.aiff

tr46-Apocalyptic_Dreams_Stem4_Strings_Brass_Stem4_Strings_Brass.aiff
to
tr46-Apocalyptic_Dreams_Stem4_Strings_Brass.aiff

```
newName=name.split('_').filter(function(item,i,allItems){
    return i==allItems.indexOf(item);
}).join('_');
```

Name	New Name
tr46-Apocalyptic_Dreams_Stem4_Strings_Brass_Stem4_Strings_Brass.aiff	tr46-Apocalyptic_Dreams_Stem4_Strings_Brass.aiff
tr45-Apocalyptic_Dreams_Stem3_Percussion_Stem3_Percussion.aiff	tr45-Apocalyptic_Dreams_Stem3_Percussion.aiff
tr44-Apocalyptic_Dreams_Stem2_Choir_Stem2_Choir.aiff	tr44-Apocalyptic_Dreams_Stem2_Choir.aiff
tr43-Apocalyptic_Dreams_Stem1_Drums_Stem1_Drums.aiff	tr43-Apocalyptic_Dreams_Stem1_Drums.aiff
tr133-Apocalyptic_Dreams_30_Second_30_Second.aiff	tr133-Apocalyptic_Dreams_30_Second.aiff

Forum Contributor: Admin

JavaScript

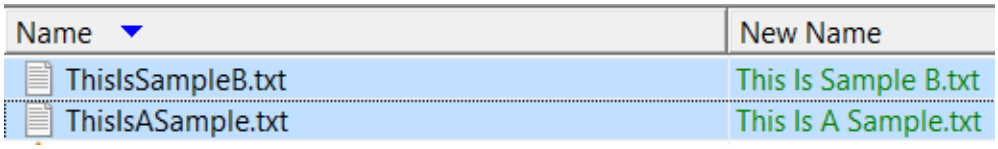
Separate Run-On Sentences with spacing #2 (refer to Uncategorized under RegEx section for #1)

ThisIsASample.txt to This Is A Sample.txt
ThisIsSampleB.txt to This Is Sample B.txt

Observations:

1. Each word is Uppercase providing a pattern. Run-on uppercase words is called camelCase.

```
newName = name.replace(/([A-Z])/g, ' $1').trim();
```



Forum Contributor: Admin

JavaScript

Replace a series of hyphens with spaces, but leave the last hyphen

Artist - Title	to	Ignore
Artist - Artist - Title	to	Artist, Artist - Title
Artist - Artist - Artist - Title	to	Artist, Artist, Artist - Title
Artist - Artist - Artist - Artist - Title	to	Artist, Artist, Artist, Artist - Title

```
function replaceAll(str, find, replace) {
    return str.replace(new RegExp(find, 'g'), replace);
}

var str = name, replacement = '<';
str = str.replace(/-([^-]*)$/,replacement+'$1');          //replace last - into <

str = replaceAll(str, ' -', ',');                         //replace all remaining  - into ,
str = replaceAll(str, '-', ',');                          //replace all remaining - into ,

str = str.replace(/<([^<]*)$/,'-'+'$1');                  //replace last < into -

newName = str;
```

Name	New Name
Artist - Title	Artist - Title
Artist - Artist - Title	Artist, Artist - Title
Artist - Artist - Artist - Title	Artist, Artist, Artist - Title
Artist - Artist - Artist - Artist - Title	Artist, Artist, Artist, Artist - Title

Forum Contributor: Admin

JavaScript

Uppercase first letter of name

150830_foldername01 to 150830_Foldername01

```
    // replace char in in position 8 with uppercase
var i = 7;

String.prototype.replaceAt=function(index, character) {
return this.substr(0, index) + character + this.substr(index+character.length);
 }

var str = name;
var ch = str.substr(i,1);
ch = ch.toUpperCase();
str = str.replaceAt(i, ch);

newName = str;
```

Name	New Name
150830_foldername01	150830_Foldername01

Forum Contributor: Admin

JavaScript

Example of using string.repeat(number) of JavaScript

sample: “Father”.repeat(3)

```
newName = "Father".repeat(3)
```

Name	New Name
192359_Way to import data from Excel sheets into Word (Software applic...	FatherFatherFather.png

Forum Contributor: Admin

JavaScript

Change date format

5JUL2001 FR-009 CAL	to	FR-009 CAL Y_2001 M_07 D_05
05JUN2013 FR-009-41.2 PM	to	FR-009-41.2 PM Y_2013 MON_06 D_05
01FEB2000 LQ-132 IMQ	to	LQ-132 IMQ Y_2000 M_02 D_01
1FEB2005 LBE-002-4 WO	to	LBE-002-4 WO Y_2005 M_02 D_01

Where date change is:

D <written Month> YYYY to YYYY MM DD i.e., 2001 M_07 D_05

```
n = name.indexOf(" ");
part1 = name.substr(0,n);
part2 = name.substr(n+1);
d = new Date();
d = Date.create(part1);

newName = part2 + ' ' + 'Y_' + d.getFullYear() + ' M_' + (d.getMonth()+1) + ' D_' + d.getDate();
```

Name	New Name
1FEB2005 LBE-002-4 WO	LBE-002-4 WO Y_2005 M_2 D_1
5JUL2001 FR-009 CAL	FR-009 CAL Y_2001 M_7 D_5
05JUN2013 FR-009-41.2 PM	FR-009-41.2 PM Y_2013 M_6 D_5
01FEB2000 LQ-132 IMQ	LQ-132 IMQ Y_2000 M_2 D_1

Forum Contributor: Admin

JavaScript

Change date format cont.

5JUL2001 FR-009 CAL	to	FR-009 CAL Y_2001 M_07 D_05
05JUN2013 FR-009-41.2 PM	to	FR-009-41.2 PM Y_2013 MON_06 D_05
01FEB2000 LQ-132 IMQ	to	LQ-132 IMQ Y_2000 M_02 D_01
1FEB2005 LBE-002-4 WO	to	LBE-002-4 WO Y_2005 M_02 D_01

Tim Contribution –

This is good but the Poster wanted MM and DD not M and D. The d.getMonth() function returns a value 0-11 resulting in single values for Jan through September. The same for d.getDate(). Here is a function I found that could help.

```
function formatDateToString(date){
                                                                    // 01, 02, 03, ... 29, 30, 31
var dd = (date.getDate() < 10 ? '0' : '') + date.getDate();
                                                                    // 01, 02, 03, ... 10, 11, 12
 var MM = ((date.getMonth() + 1) < 10 ? '0' : '') + (date.getMonth() + 1);
                                                                    // 1970, 1971, ... 2015, 2016, ...
 var yyyy = date.getFullYear();


                                                                  // create the format you want
   return (dd + "-" + MM + "-" + yyyy);
}
```

Here is the corrected script:

```
n = name.indexOf(" ");
part1 = name.substr(0,n);
part2 = name.substr(n+1);
d = new Date();
d = Date.create(part1);

var MM = ((d.getMonth() + 1) < 10 ? '0' : '') + (d.getMonth() + 1);
var dd = (d.getDate() < 10 ? '0' : '') + d.getDate();

newName = part2 + ' ' + 'Y_' + d.getFullYear() + ' M_' + MM + ' D_' + dd;
```

Name	New Name
1FEB2005 LBE-002-4 WO	LBE-002-4 WO Y_2005 M_02 D_01
5JUL2001 FR-009 CAL	FR-009 CAL Y_2001 M_07 D_05
05JUN2013 FR-009-41.2 PM	FR-009-41 Y_2013 M_06 D_05.2 PM
01FEB2000 LQ-132 IMQ	LQ-132 IMQ Y_2000 M_02 D_01

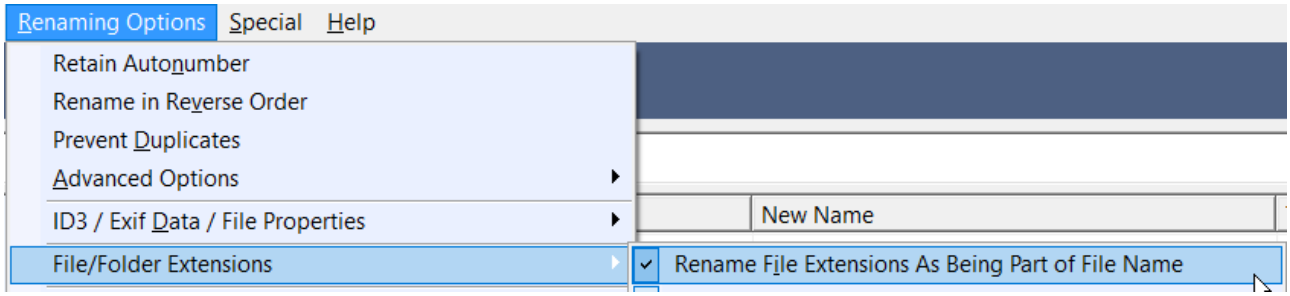
JavaScript

Change date format cont.

Tim Contribution – cont.

Tim Notes:

1. Because there is a dot character in the filename, '05JUN2013 FR-009-41.2 PM' make sure you have 'Rename File Extensions As Being Part of File Name' set from File/Folder Extensions sub menu under Renaming Options menu if you have similar files.



   a. I overlooked this in the first trial. Thanks to Admin for his assist.

Other Sources, Marcel (Stack Overflow)

JavaScript

Remove Multiple Hyphens leaving one

ABC-XYZ	to	Ignore
ABC--XYZ	to	ABC-XYZ
ABC----XYZ	to	ABC-XYZ

```
String.prototype.replaceAll = function(search, replacement) {
var target = this;
return target.split(search).join(replacement);
};

var str = name;
str = str.replaceAll(" ", "-");
str = str.replaceAll("-------", "-");
str = str.replaceAll("------", "-");
str = str.replaceAll("-----", "-")
str = str.replaceAll("----", "-");
str = str.replaceAll("---", "-");
str = str.replaceAll("--", "-");

newName = str;
```

Name	New Name
ABC-XYZ	ABC-XYZ
ABC--XYZ	ABC-XYZ
ABC----XYZ	ABC-XYZ

Forum Contributor: Admin

JavaScript

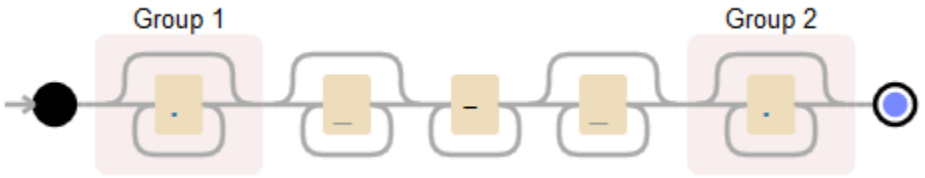
Remove Multiple Hyphens leaving one cont.

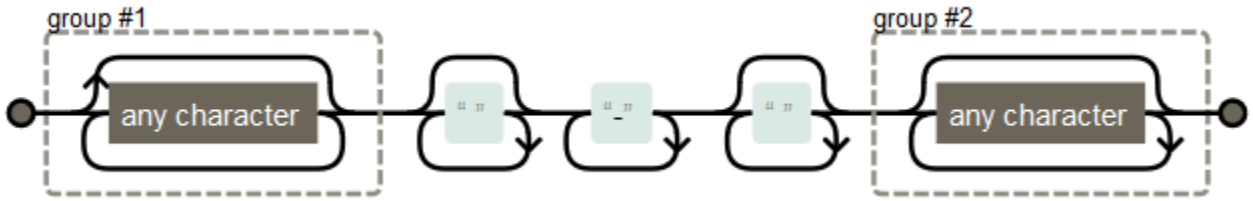
ABC-XYZ	to	Ignore
ABC--XYZ	to	ABC-XYZ
ABC----XYZ	to	ABC-XYZ

RegEx Equivalent:

Match: (.*?) *-+ *(.*)
Replace: \1-\2

Name	New Name
ABC-XYZ	ABC-XYZ
ABC--XYZ	ABC-XYZ
ABC----XYZ	ABC-XYZ





Sample:

```
.....................
Match 1: ABC--XYZ
Group 1: ABC
Group 2: XYZ
```

Forum Contributor: therube

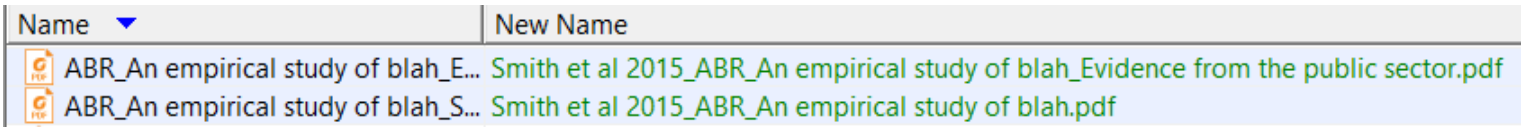
JavaScript

Move everything after name to beginning of filename and add underscore

ABR_An empirical study of blah_**Smith et al 2015**.pdf
to
Smith et al 2015_ABR_An empirical study of blah.pdf

ABR_An empirical study of blah_Evidence from the public sector_**Smith et al 2015**.pdf
to
Smith et al 2015_ABR_An empirical study of blah_Evidence from the public sector.pdf

```
var char2find = "_";
var index = name.lastIndexOf(char2find);

newName = name.substr(index+1) + char2find + name.substr(0,index);
```



Contributor Notes:

1. This JavaScript function has no constraints on file name format, e.g. it switches everything around the last '_' (underscore), or any other character of choice (just change the underscore in 'var char2find =' statement).

2. It works with any number of underscores in the file name or any position, also works for files like :

_testfile.ext

or

test_file_.ext

3. Bold was used to distinguish areas to be affected in samples above and are not part of renaming.

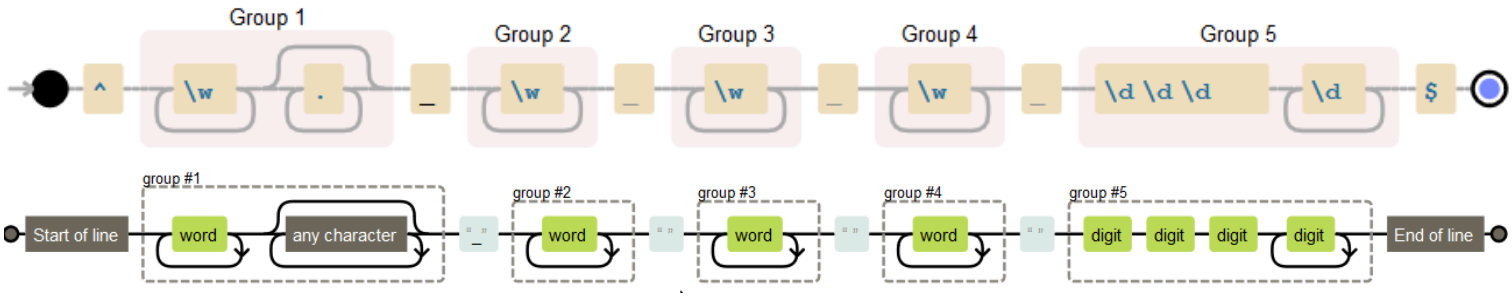
JavaScript

Move everything after name to beginning of filename and add underscore cont.

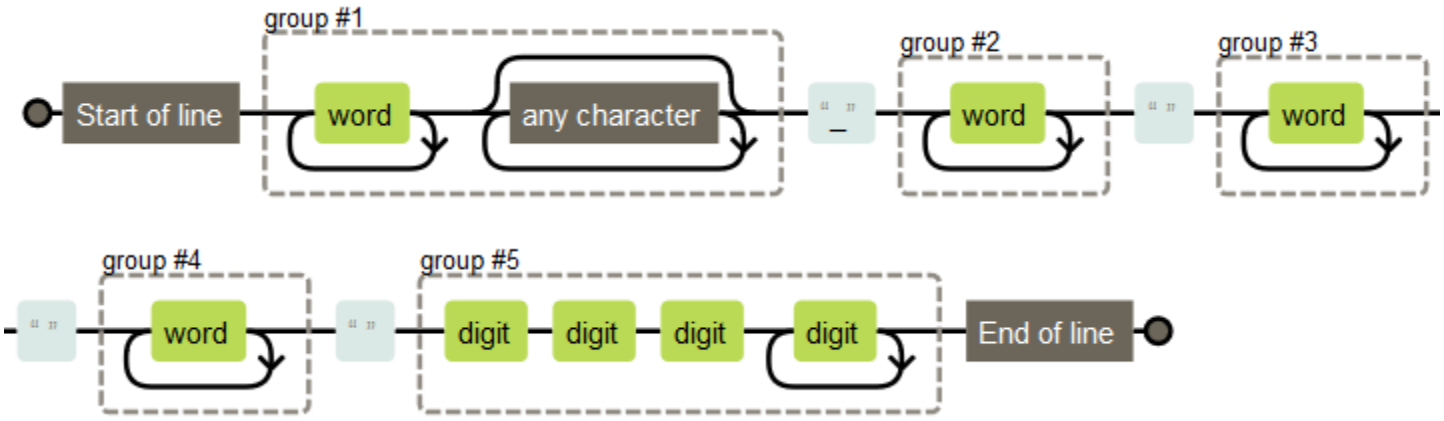
RegEx Equivalent:

Match: ^(\w+.*)_(\w+) (\w+) (\w+) (\d\d\d\d+)$
Replace: \2 \3 \4 \5_\1

Name	New Name
ABR_An empirical study of blah_E...	Smith et al 2015_ABR_An empirical study of blah_Evidence from the public sector.pdf
ABR_An empirical study of blah_S...	Smith et al 2015_ABR_An empirical study of blah.pdf



Expanded:



Sample:

```
.....................................................
Match 1: ABR_An empirical study of blah_Smith et al 2015
Group 1: ABR_An empirical study of blah
Group 2: Smith
Group 3: et
Group 4: al
Group 5: 2015
```

JavaScript

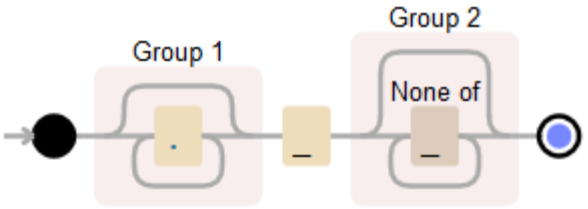
Move everything after name to beginning of filename and add underscore cont.

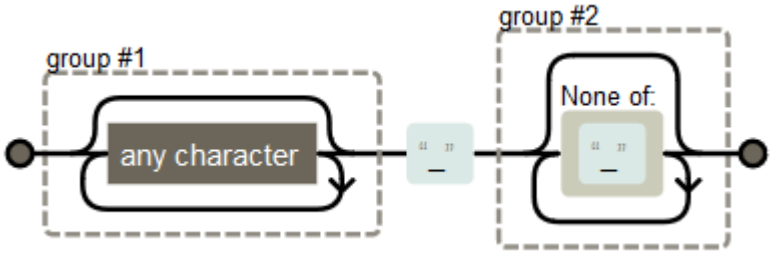
RegEx Equivalent: cont.

Or …

Match: (.*)_([^_]*)
Replace: \2 \1

Name	New Name
ABR_An empirical study of blah_E...	Smith et al 2015_ABR_An empirical study of blah_Evidence from the public sector.pdf
ABR_An empirical study of blah_S...	Smith et al 2015_ABR_An empirical study of blah.pdf





Sample:

```
Match 1: ABR_An empirical study of blah_Smith et al 2015
Group 1: ABR_An empirical study of blah
Group 2: Smith et al 2015
```

Contributor Analysis:

1. Capture Group 2 is anything not having an underscore.
2. It is preceded by an underscore which is preceded by anything (including underscore).
3. Hence, Capture Group 2 is the string after the last underscore.

Forum Contributors: Admin, RobsterUK, HNS

JavaScript

Replace specific characters after a specific position (replace hyphens with dots at section following date)

2012-01-22 14-46-26.JPG to 2012-01-22 14.46.26.JPG

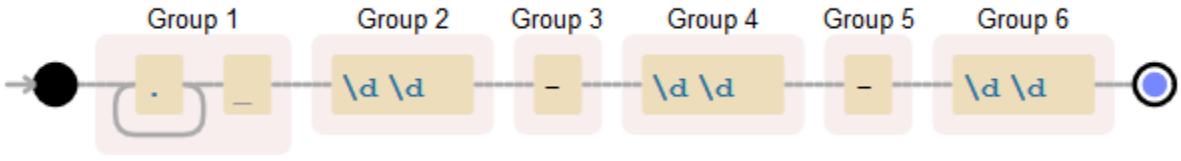
```
var s = name;
var index = 10;
var r = s.substr(index);
r = r.replace(/-/g, '.');

newName = s.substr(0, index) + r;
```

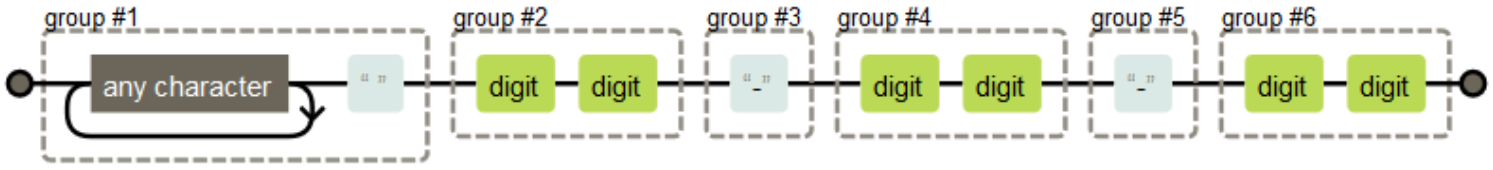
Name	New Name
2012-01-22 14-46-26.JPG	2012-01-22 14.46.26.JPG

In RegEx:

Match: (.+)(\d\d)(-)(\d\d)(-)(\d\d)
Replace: \1\2.\4.\6

Name	New Name
2012-01-22 14-46-26.JPG	2012-01-22 14.46.26.JPG





Sample:

```
................................
Match 1: 2012-01-22 14-46-26
Group 1: 2012-01-22
Group 2: 14
Group 3: -
Group 4: 46
Group 5: -
Group 6: 26
```

JavaScript

Replace specific characters after a specific position (replace hyphens with dots at section following date) cont.

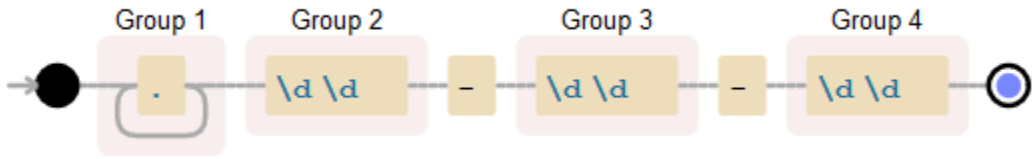
2012-01-22 14-46-26.JPG to 2012-01-22 14.46.26.JPG

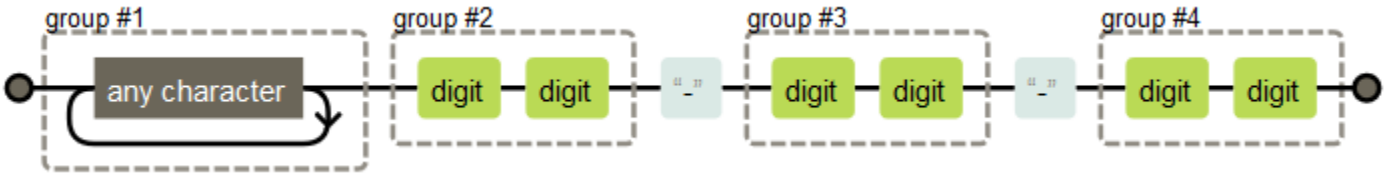
In RegEx: cont.

Or ….

Match: (.+)(\d\d)-(\d\d)-(\d\d)
Replace: \1\2.\3.\4

Name	New Name
2012-01-22 14-46-26.JPG	2012-01-22 14.46.26.JPG





Sample:

```
.............................
Match 1: 2012-01-22 14-46-26
Group 1: 2012-01-22
Group 2: 14
Group 3: 46
Group 4: 26
```

Tim Notes:

1. Both RegEx's are the same. The only difference is that the first one captures the <hyphen> as Capture Group 3, therefore has to use backreference of \4, whereas the second one doesn't and uses the backreference \3.

Forum Contributors: Admin, therube, RobsterUK

JavaScript

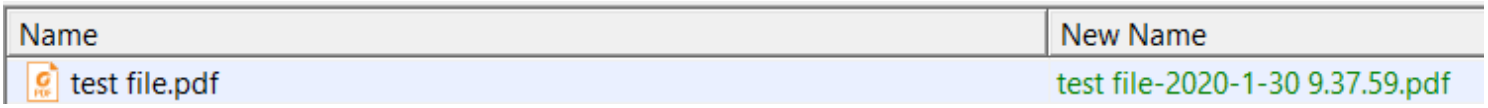
Add today's date and time to filename

Add today's date (as yyyy-mm-dd) followed by time (as hh.mm) to filename:

```
var today = new Date();
var date = today.getFullYear()+'-'+(today.getMonth()+1)+'-'+today.getDate();
var time = today.getHours() + ":" + today.getMinutes() + ":" + today.getSeconds();
var dateTime = date +' '+time;

newName = name + '-' + dateTime;
```

Tim Notes:

1. Had to switch in the dot for the colon because colons are illegal characters for a filename:

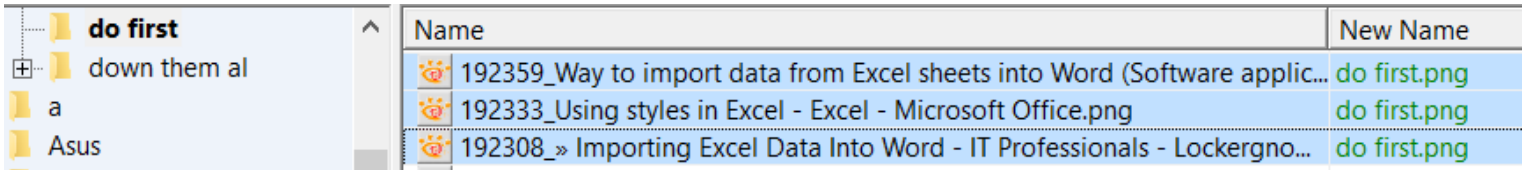


Forum Contributor: Admin

JavaScript

Rename Filename to match the Folder Name

```
var myString = object("folder");
var pathElements = myString.split('\\');
var lastFolder = pathElements[pathElements.length - 2];

newName = lastFolder;
```



Tim Notes:

1. Doesn't append, just replaces.

Forum Contributor: Admin

JavaScript

Rename part of file name from: dd-mm-yyyy to: yyyy-mm-dd

03-01-2012_U0-52dc.pdf to 2012-01-03_U0-52dc.pdf

Observations:

1. Convert [dd-mm-yyyy] to [yyyy-mm-dd]

```
var sep = "-";
var datestr = name.substring(0, 10);
var rest    = name.substring(10);
var parts = datestr.split(sep);

if (parts.length > 2) {
  newName = parts[2] + sep + parts[1] + sep + parts[0] + rest;
}
```

Name	New Name
03-01-2012_U0-52dc.pdf	2012-01-03_U0-52dc.pdf

In case someone needs to organize files with different date formats:

yyyy-d-m yyyy-dd-m yyyy-d-mm yyyy-dd-mm
to
yyyy_mm_dd

```
// declare
var sep = "_";
var years = "";
var days = "";
var months = "";
var rest = "";


// yyyy-dd-mm-
if (name.substring(7,8) === "_" && name.substring(10,11) === "_") {
  years = name.substring(0, 4);
  days = name.substring(5, 7);
  months = name.substring(8, 10);
  rest = name.substring(11,400);

  newName=  years + sep + months + sep + days + sep + rest;

}
```

JavaScript

Rename part of file name from: dd-mm-yyyy to: yyyy-mm-dd cont.

```
// yyyy-d-m-
else if (name.substring(6,7) === "_" && name.substring(8,9) === "_") {
  years = name.substring(0, 4);
  days = name.substring(5, 6);
  months = name.substring(7, 8);
  rest = name.substring(9,400);

  newName=  years + sep + "0" + months + sep + "0" + days + sep + rest;

}

// yyyy-dd-m-
else if (name.substring(7,8) === "_") {

  years = name.substring(0, 4);
  days = name.substring(5, 7);
  months = name.substring(8, 9);
  rest = name.substring(10,400);

  newName=  years + sep + "0" + months + sep + days + sep + rest;

}

// yyyy-d-mm-
else if (name.substring(6,7) === "_") {
  years = name.substring(0, 4);
  days = name.substring(5, 6);
  months = name.substring(7, 9);
  rest = name.substring(10,400);

  newName=  years + sep +  months + sep + "0" + days + sep + rest;

}
```

JavaScript

Rename part of file name from: dd-mm-yyyy to: yyyy-mm-dd cont.

Example using YYYY_D_M 2000_5_7_L1-52dc.pdf

Name	New Name
2000_5_7_L1-52dc.pdf	2000_07_05_L1-52dc.pdf

Example using YYYY_DD_M 2000_23_7_L1-52dc.pdf

Name	New Name
2000_23_7_L1-52dc.pdf	2000_07_23_L1-52dc.pdf

Example using YYYY_D_MM 2000_8_12_L1-52dc.pdf

Name	New Name
2000_8_12_L1-52dc.pdf	2000_12_08_L1-52dc.pdf

Example using YYYY_DD_MM 2000_19_03_L1-52dc.pdf

Name	New Name
2000_19_03_L1-52dc.pdf	2000_03_19_L1-52dc.pdf

Tim Notes:

1. Pattern is: <date>_ - must have underscore after the date to connect to the remainder of the string.

    Doesn't matter what is at the end of the string past the date:

Name	New Name
2000_19_03_hhddjj.pdf	2000_03_19_hhddjj.pdf

Forum Contributors: Admin, matejmeglic

JavaScript

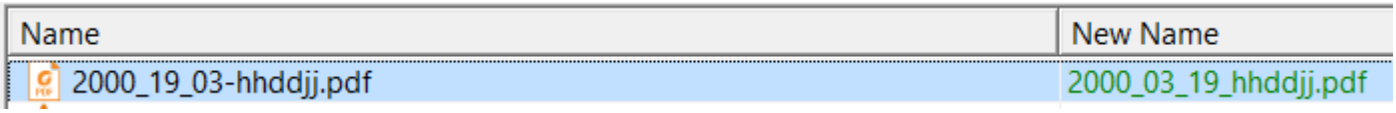
Rename part of file name from: dd-mm-yyyy to: yyyy-mm-dd cont.

Tim Contribution –

If you do not want to use the underscore following the date, then change this line:

```
if (name.substring(7,8) === "_" && name.substring(10,11) === "-") {
```

This changed it to <hyphen>. Example 2000_19_03-hhddjj.pdf

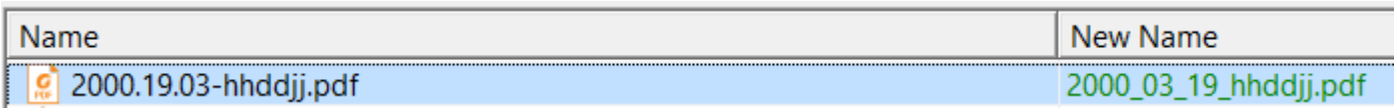


This could easily have been changed to a <space>. Example 2000_19_03 hhddjj.pdf

If you change the date format to be matched from using other than the underscore, change this line:

```
if (name.substring(7,8) === "." && name.substring(10,11) === "-") {
```

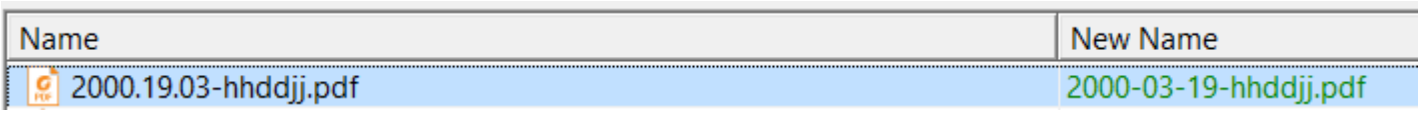
This changed the filenames to be matched using dots. Example 2000.19.03-hhddjj.pdf



If you want to change the New Name date format to using a separator other than the underscore, change this line:

```
var sep = "-";
```

This changed the separator to using a <hyphen>. Example 2000.19.03-hhddjj.pdf



JavaScript

Rename part of file name from: dd-mm-yyyy to: yyyy-mm-dd cont.

Tim Contribution – cont.

If you have filenames that do not have any separation between the date and the rest of the string..

Then Change this line:

from:

```
if (name.substring(7,8) === "." && name.substring(10,11) === "_") {
```

to this:

```
if (name.substring(7,8) === ".") {
```

Example 2000.19.03hhddjj.pdf

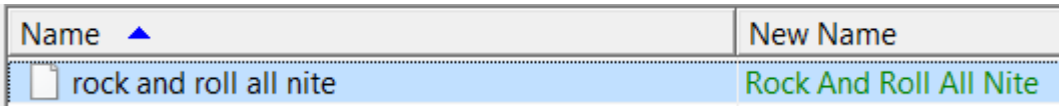
Name	New Name
2000.19.03hhddjj.pdf	2000-03-19-hddjj.pdf

JavaScript

Capitalize first letter of each word

rock and roll all nite to Rock And Roll All Nite

```
function toTitleCase(str)
{
 return str.replace(/\w\S*/g, function(txt){return txt.charAt(0).toUpperCase() + txt.substr(1).toLowerCase();});
}

newName = toTitleCase(name)
```



Forum Contributor: Admin

JavaScript

Add random string of letters and numbers to end of a filename

This appends 5 random alpha/numeric characters to the end of a filename.

```
function makeid() {
  var text = "";
  var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

for (var i = 0; i < 5; i++)
    text += possible.charAt(Math.floor(Math.random() * possible.length));

return text;

}

newName = name + makeid();
```

Name	New Name
oldbookoldbook.jpg	oldbookoldbookxa1rC.jpg
OApatchOApatch.png	OApatchOApatch3P7I5.png
541-200-6_web_ad541-200-6_web_ad.jpg	541-200-6_web_ad541-200-6_web_adJWcCP.jpg
2013-02-31 name.pdf	2013-02-31 nameRR68k.pdf

Forum Contributor: Admin

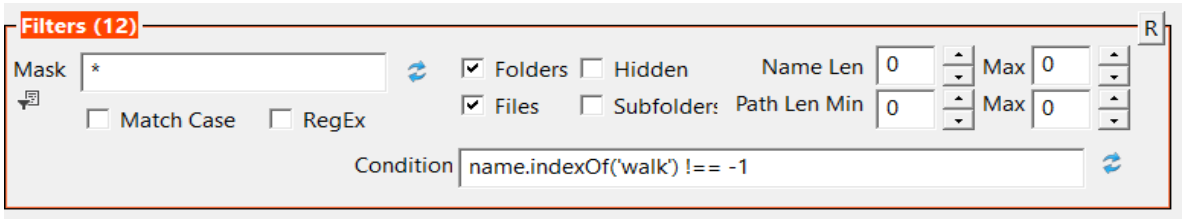
JavaScript

JavaScript Condition Filter all Files that contain name, 'walk'

'Section #12: Filters' -

Set Condition to:

```
name.indexOf('walk') !== -1
```



Click on Filter Refresh button (blue button) to the right of the Condition data field.

Name	New Name
oldbookoldwalkbook.jpg	oldbookoldwalkbook.jpg
2004-walk-21 name	2004-walk-21 name

Forum Contributor: Admin

JavaScript

Rename filenames with random string of letters and numbers

```
function makeid(size)
{
  var text = "";
  var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

  for( var i=0; i < size; i++ )
    text += possible.charAt(Math.floor(Math.random() * possible.length));

return text;
}
newName = makeid(15);
```

Name	New Name
File 1.pdf	AQIIUFSMFWKHtHB.pdf
File 3.pdf	fCjgyf82wZyr93n.pdf
File 4.pdf	IPwJaM7Tycy73qQ.pdf
File 10.pdf	wR685Q6jfP5mh2W.pdf

Contributor Notes:

1. '15' in 'makeid(15)', is the required name size.

Variation:

This appends a random set of three numeric digits to existing filenames*, to photos, for instance:

```
 function makeid(size)
{
  var text = "";
  var possible = "0123456789";

for( var i=0; i < size; i++ )
  text += possible.charAt(Math.floor(Math.random() * possible.length));

return text;
}
newName = name + '_' + makeid(3);
```

Name	New Name
File 1.pdf	File 1_854.pdf
File 3.pdf	File 3_072.pdf
File 4.pdf	File 4_859.pdf
File 10.pdf	File 10_449.pdf

JavaScript

Rename filenames with random string of letters and numbers cont.

Contributor Notes:

1. If you change this code yourself, <spaces> before and after elements like name, +, '_', are required. Omitting them throws up errors.

2. If you would like to append, i.e., four numeric digits rather than three, change 'makeid(3)' in the bottom line to 'makeid(4)'

Forum Contributor: Admin, Ardwych

JavaScript

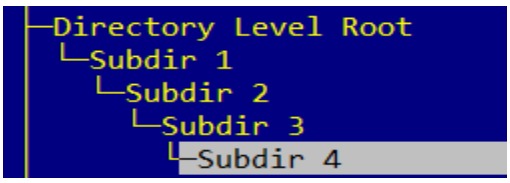
Append Folder Name Levels and Rearrange them

A Brief review is necessary.

'Section #9: Append Folder Names' –

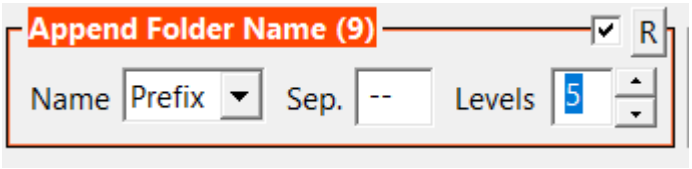
Allows you to append a filename with a directory (folder). Not just the current directory but directory levels above.

A file, i.e., Desserts Of Vitality.pdf, is located in Subdir 4 directory located in the following directory structure:

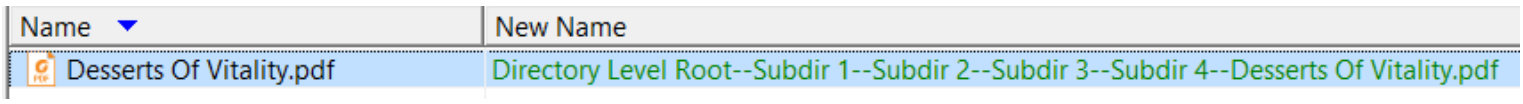


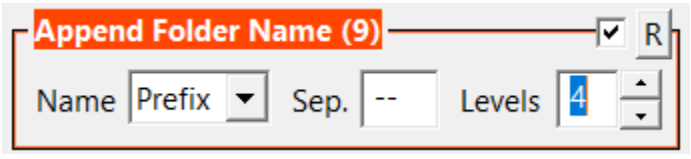
There are 5 directory levels. 'Directory Level Root' is considered 5 levels up from the current directory, 'Subdir 4'.

Using the Append Folder Function set at 'Prefix' using a separator of '--' , here is 'New Name' given the various 'Level' settings:

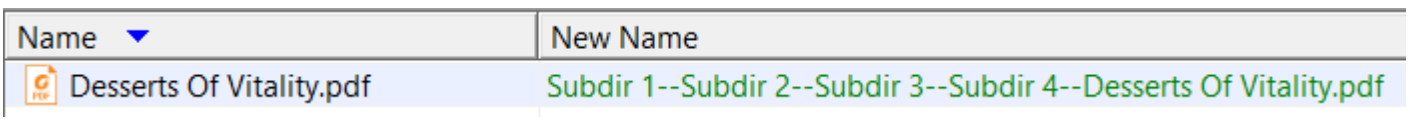


Directory 5 levels up from current directory





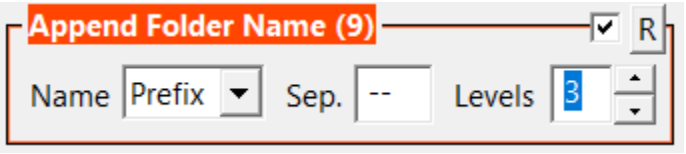
Directory 4 levels up from current directory



JavaScript

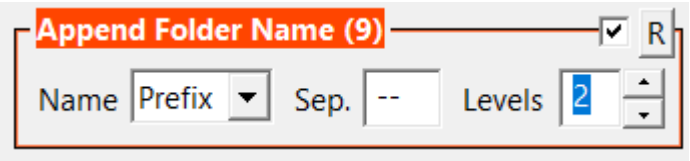
Append Folder Name Levels and Rearrange them cont.

Using the Append Folder Function set at 'Prefix' using a separator of '--' , here is 'New Name' given the various 'Level' settings: cont.



Directory 3 levels up from current directory

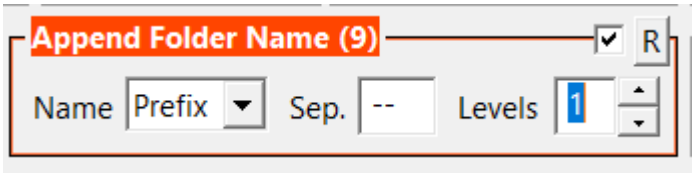
Name	New Name
Desserts Of Vitality.pdf	Subdir 2--Subdir 3--Subdir 4--Desserts Of Vitality.pdf



Directory 2 levels up from current directory

Name	New Name
Desserts Of Vitality.pdf	Subdir 3--Subdir 4--Desserts Of Vitality.pdf

Current directory level



Name	New Name
Desserts Of Vitality.pdf	Subdir 4--Desserts Of Vitality.pdf

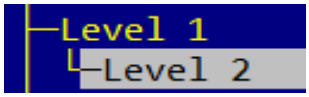
Tim Notes:

1. Negative numbers for 'Level' not allowed.

JavaScript

Append Folder Name Levels and Rearrange them cont.

Now that you understand, let's continue with the request:



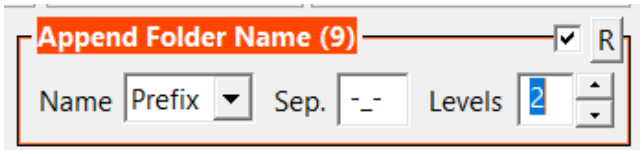
Using a directory structure (at left):
A file, 'Filename', is in the current directory, 'Level 2'

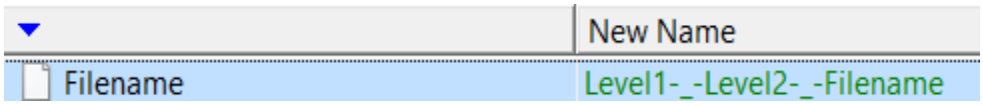
Filename to: Level2Level1Filename

Two separate runs:

First run -

1. Use 'Section #9: Append Folder Name' to rename as 'Level1-_-Level2-_-Filename'





JavaScript

Append Folder Name Levels and Rearrange them cont.

Filename to: Level2Level1Filename

Second run –

Match: (.*)-_-(.*)-_-(.*)
Replace: \2\1\3

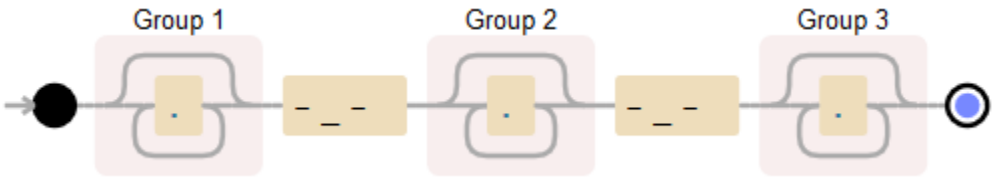
	New Name
Level1-_-Level2-_-Filename	Level2Level1Filename

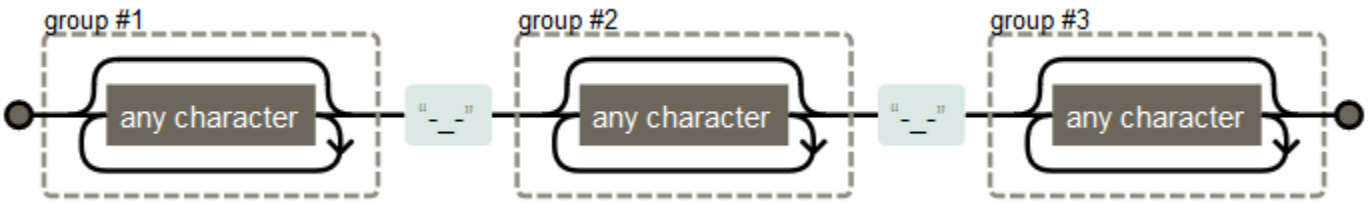
Where:

\1 = Level1 \2 = Level2 \3 = Filename

Sample:

```
Match 1: Level1-_-Level2-_-Filename
Group 1: Level1
Group 2: Level2
Group 3: Filename
```





Forum Contributor: KenP

Tim Notes:

1. The Contributor's RegEx does not directly address the original request.

2. The Contributor got to the New Name in a round-about fashion by first renaming the filename in 'Section #9: Append Folder Name' to a compatible structure that would allow the Contributor's RegEx to match Nothing wrong with that, but I think it is important for people who are learning, to see the RegEx developed based on the request at hand.

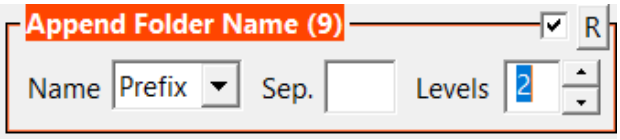
JavaScript

Append Folder Name Levels and Rearrange them cont.

Tim Contribution –

First run -

1. Use 'Section #9: Append Folder Name' to rename as 'Level1Level2Filename'



	New Name
Filename	Level1Level2Filename

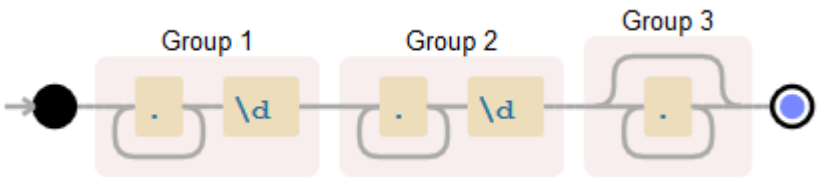
Second run -

This RegEx does not rely on delimiters between the Words to match.

Match: (.+?\d)(.+?\d)(.*)
Replace: \2\1\3

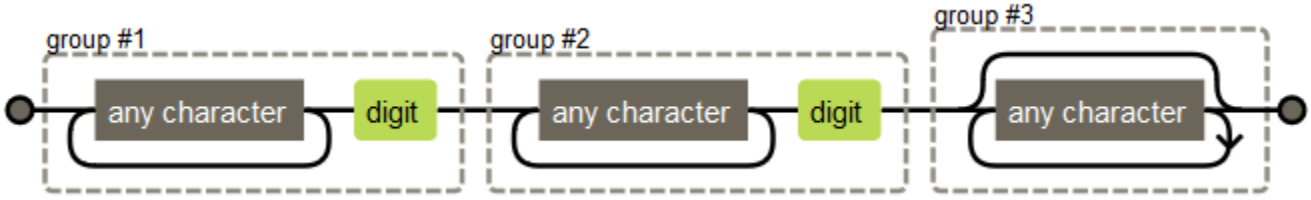
Name	New Name
Level1Level2Filename	Level2Level1Filename

Where:
\1 = Level1 \2 = Level2 \3 = Filename



Sample:

```
Match 1: Level1-_-Level2-_-Filename
Group 1: Level1
Group 2: -_-Level2
Group 3: -_-Filename
```

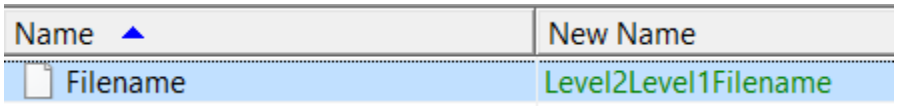


JavaScript

Append Folder Name Levels and Rearrange them cont.

JavaScript Equivalent:

```
var subfolders = object("folder").split("\\");
subfolders.reverse();
newName = subfolders[1] + subfolders[2] + name;
```



Forum Contributors: Admin

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot

<space>File 1 (example of space as prefix).pdf	to	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	to	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	to	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	to	File 10 (example of dot as suffix)..pdf

```
newName = name;

// removes <space> if first char
 var char2find = " ";
 var index = newName.indexOf(char2find);
 if (index == 0) newName = newName.substr(1);

 // removes . if first char
 char2find = ".";
 var index = newName.indexOf(char2find);
 if (index == 0) newName = newName.substr(1);
```

Name	New Name
File 1 (example of space as prefix).pdf	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	File 1 (example of dot as prefix).pdf
File 10 (example of dot as suffix)..pdf	File 10 (example of dot as suffix)..pdf
File 10 (example of space as suffix) .pdf	File 10 (example of space as suffix) .pdf

Tim Contribution –

1. The above does not address if a dot or <space> is found at the end of the filename (suffix).

This does:

For dot as suffix: File 10 (example of dot as suffix)..pdf

```
newName = name;

var char2find = ".";
var index = name.lastIndexOf(char2find);
newName = name.substr(index+1) + name.substr(0,index);
```

Name	New Name
File 10 (example of dot as suffix)..pdf	File 10 (example of dot as suffix).pdf

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

Tim Contribution – cont.

For <space> as suffix: File 10 (example of space as suffix) .pdf

```
newName = name;

var char2find = " ";
var index = name.lastIndexOf(char2find);
newName = name.substr(index+1) + name.substr(0,index);
```

Name	New Name
File 10 (example of space as suffix) .pdf	File 10 (example of space as suffix).pdf

Added to the original script (Thanks to Admin for his assist on this):

```
newName = name;

// removes <space> if first char
var char2find = " ";
var index = newName.indexOf(char2find);
if (index == 0) newName = newName.substr(1);

// removes . if first char
char2find = ".";
var index = newName.indexOf(char2find);
if (index == 0) newName = newName.substr(1);

// removes . if last char
var char2find = ".";
var index = name.lastIndexOf(char2find);
if (name.endsWith(".")) newName = name.substring(0, name.length - 1);

// removes <space> if last char
var char2find = " ";
var index = name.lastIndexOf(char2find);
if (name.endsWith(" ")) newName = name.substring(0, name.length - 1);
```

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

Tim Contribution – cont.

Name	New Name
File 1 (example of space as prefix).pdf	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	File 10 (example of dot as suffix).pdf

Tim Notes:

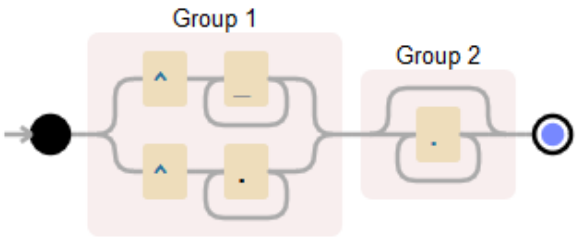
1. Requires sugar.js library (Special Menu – JavaScript Libraries – Include sugar.js). If a function, method or property is used in the script that originates from an outside library, then the library must be included as part of the script. This indicates to the JavaScript compiler to include the library at the time of compiling the script, otherwise an error will be generated.

Posting continues:

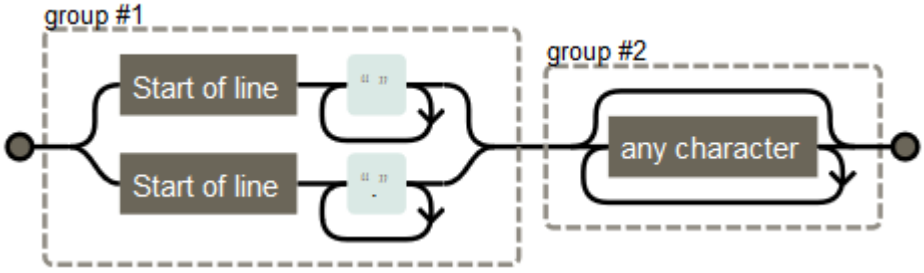
RegEx Equivalent:

Match: (^ +|^\.+)(.*)
Replace: \2

Name	New Name
File 1 (example of space as prefix).pdf	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	File 1 (example of dot as prefix).pdf
File 10 (example of dot as suffix)..pdf	File 10 (example of dot as suffix)..pdf
File 10 (example of space as suffix) .pdf	File 10 (example of space as suffix) .pdf




```
Match 1:  File 1 (example of space as prefix).pdf
Group 1:
Group 2:  File 1 (example of space as prefix).pdf
```



JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

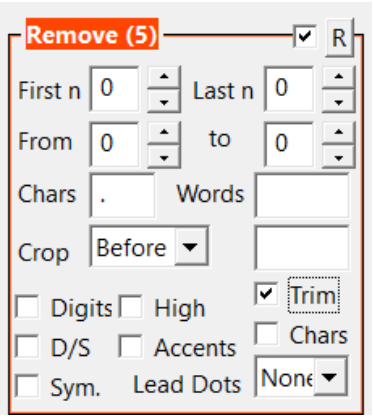
RegEx Equivalent: cont.

Match: (^ +|^\.+)(.*)

Tim Notes:

1. Again, this only address the prefix not the suffix.

2. Using 'Section #5: Remove' in conjunction with the RegEx will correct this:



Trim: Enable
Chars: <dot>

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

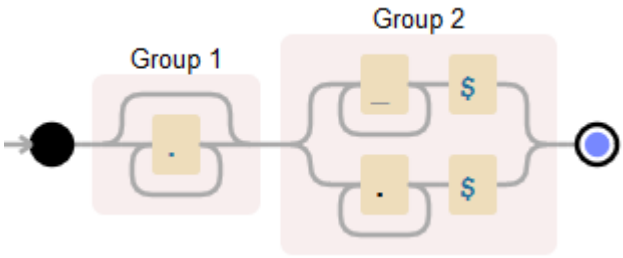
<space>File 1 (example of space as prefix).pdf	to	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	to	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	to	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	to	File 10 (example of dot as suffix)..pdf

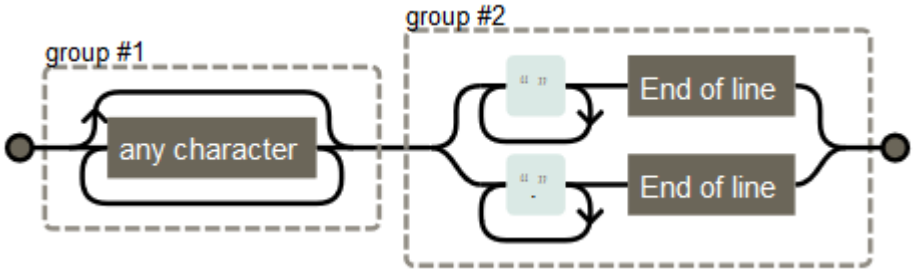
RegEx Equivalent: cont.

The Contributor has recently updated this RegEx (Jan 31, 2020) and is included below.

Match: (.*?)(+$|\.+$)
Replace: \1

Name	New Name
File 10 (example of space as suffix) .pdf	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	File 10 (example of dot as suffix).pdf
File 1 (example of space as prefix).pdf	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	.File 1 (example of dot as prefix).pdf





Sample:

```
                                              Start Length
Match 1: File 10 (example of space as suffix)     0     37
Group 1: File 10 (example of space as suffix)     0     36
Group 2:                                         36      1
```

This will remove a dot or <space> at the end of the filename.

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

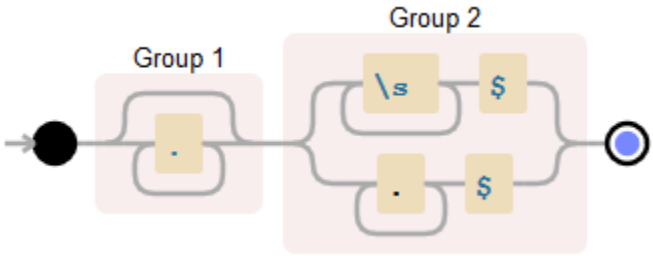
<space>File 1 (example of space as prefix).pdf	to	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	to	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	to	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	to	File 10 (example of dot as suffix)..pdf

RegEx Equivalent: cont.

Another variation of the RegEx:

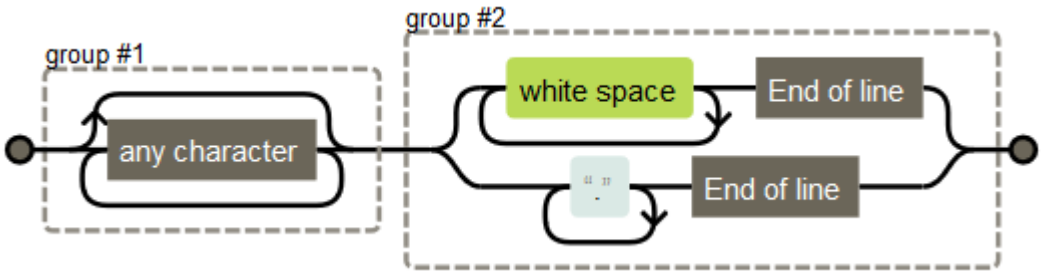
Match: (.*?)(\s+$|\.+$)
Replace: \1

Name	New Name
File 10 (example of space as suffix) .pdf	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	File 10 (example of dot as suffix).pdf
File 1 (example of space as prefix).pdf	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	.File 1 (example of dot as prefix).pdf



Tim Notes:

1. Still though, both of these only deal with one issue. In this case it is either a <space> or dot at the suffix. Prefix was already discussed by the Contributor's first RegEx. See 'My Contribution' for RegEx that can handle both dots or <spaces> at prefix and suffix.



Sample:

```
                                          Start  Length
Match 1: File 10 (example of space as suffix)     0      37
Group 1: File 10 (example of space as suffix)     0      36
Group 2:                                         36       1
```

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

<space>File 1 (example of space as prefix).pdf	to	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	to	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	to	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	to	File 10 (example of dot as suffix)..pdf

RegEx Equivalent: cont.

Another variation of the RegEx: cont.

You can also specify any beginning characters to remove:

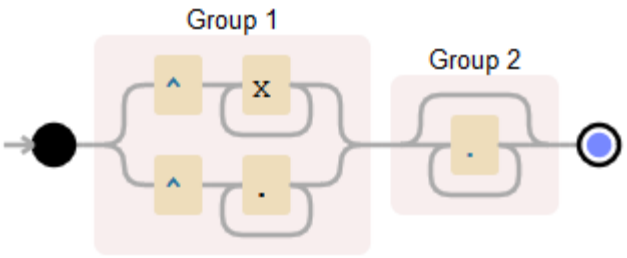
XXXabc123.txt
….abc123.txt

Match: (^X+|^\.+)(.*) Matches either 'X' or '.' as prefix
Replace: \2

Name	New Name
XXXabc123.txt	abc123.txt

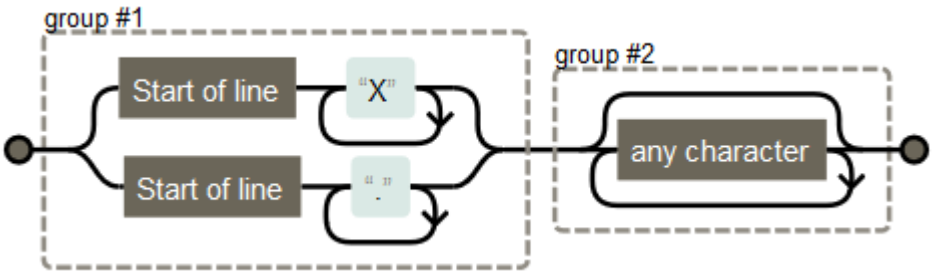
also matches …

Name	New Name
....abc123.txt	abc123.txt



Sample:

```
.....................
Match 1: XXXabc123
Group 1: XXX
Group 2: abc123
```



JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

<space>File 1 (example of space as prefix).pdf	to	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	to	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	to	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	to	File 10 (example of dot as suffix)..pdf

RegEx Equivalent: cont.

Another variation of the RegEx: cont.

or ending characters…

abc123XXX.txt
abc123…..txt

Match: (.*?)(X+$|\.+$) Matches either 'X' or '.' as suffix
Replace: \1

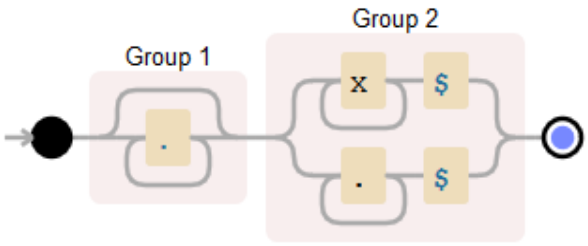
Name	New Name
abc123XXX.txt	abc123.txt

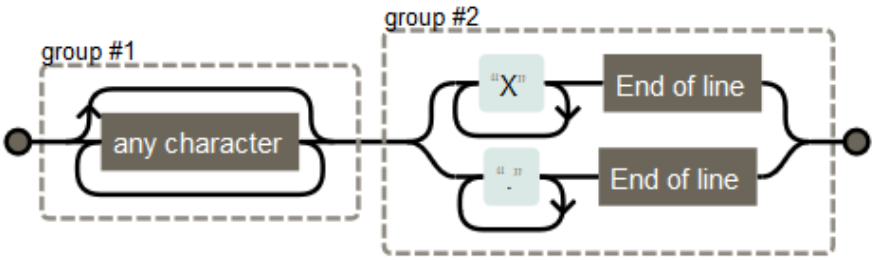
Sample:

```
....................
Match 1: abc123XXX
Group 1: abc123
Group 2: XXX
```

also matches …

Name	New Name
abc123.....txt	abc123.txt





But not both prefix and suffix.

Tim Notes:

1. The use of the Alternate '|' could easily be adapted to select between more than two characters to match, i.e.,

(^\.+|^\s+|^\X)

2. The ^ is the anchor meaning 'beginning of filename' (Beginning of Line or BOL) used here it matches the prefix.
3. the $ is the anchor meaning 'end of filename' (End of Line or EOL) used here it matches suffix.

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

.File 1 (example of dot as prefix and suffix)...pdf to File 1 (example of dot as prefix and suffix).pdf

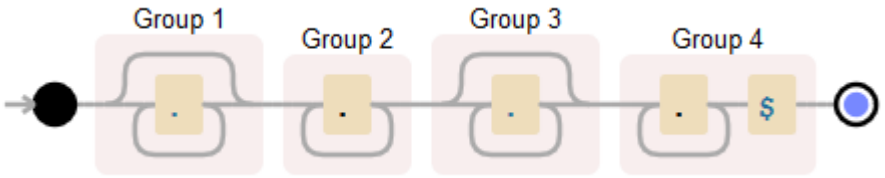
RegEx Equivalent: cont.

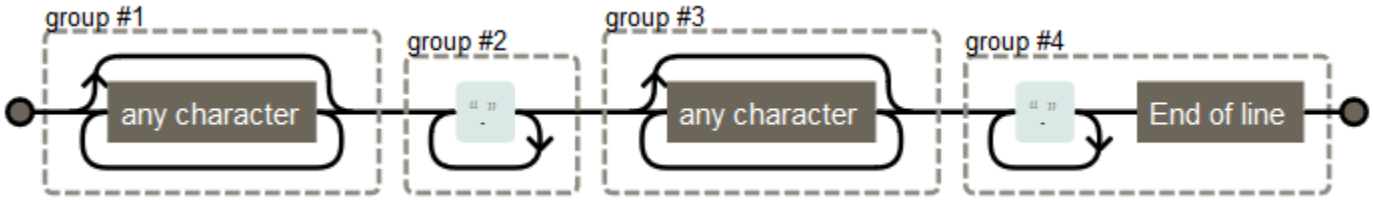
Tim Contribution –

Remove multiple dots at prefix *and* suffix but won't match against a filename with only dots at prefix *or* suffix:

Match: (.*?)(\.+)(.*?)(\.+$)
Replace: \3

Name	New Name
....test....txt	test.txt
.File 1 (example of dot as prefix and suffix)..pdf	File 1 (example of dot as prefix and suffix).pdf





Sample:

	Start	Length
Match 1:	**.File 1 (example of dot as prefix and suffix)..**	0	47
Group 1:		0	0
Group 2:	.	0	1
Group 3:	File 1 (example of dot as prefix and suffix)	1	44
Group 4:	..	45	2

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

<space> File 1 (example of space as prefix and suffix)<space>.pdf
to
File 1 (example of space as prefix and suffix).pdf

<space> <space><space> test <space> <space> <space>.txt
to
test.txt

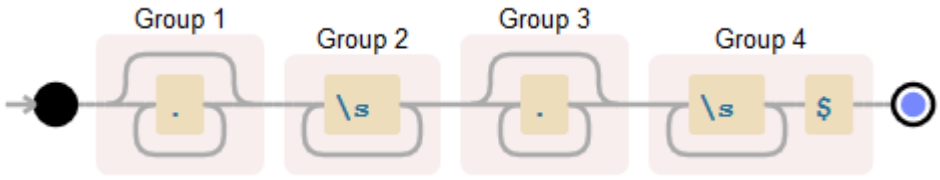
RegEx Equivalent: cont.

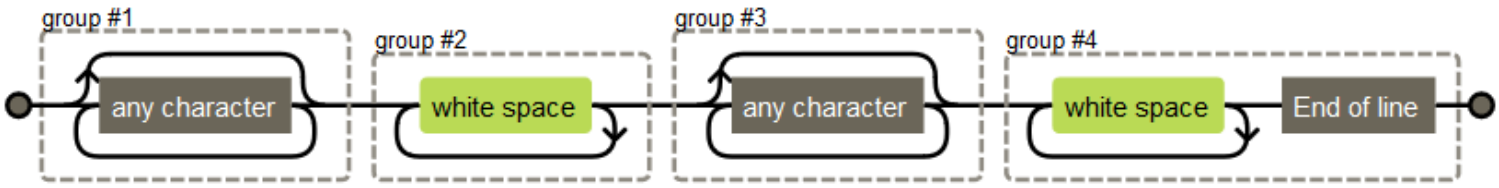
Tim Contribution – cont.

Remove multiple <spaces> at prefix *and* suffix but won't match against a filename with only <spaces> at prefix *or* suffix:

Match: (.*?)(\s+)(.*?)(\s+$)
Replace: \3

Name	New Name
test .txt	test.txt
File 1 (example of space as prefix and suffix) .pdf	File 1 (example of space as prefix and suffix).pdf





Sample:

```
                                                    Start  Length
Match 1:  File 1 (example of space as prefix and suffix)      0      48
Group 1:                                                      0       0
Group 2:                                                      0       1
Group 3: File 1 (example of space as prefix and suffix)       1      46
Group 4:                                                     47       1
```

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

.File 1 (example of dot as prefix and suffix)...pdf		
<space>File 1 (example of space as prefix).pdf	to	File 1 (example of space as prefix).pdf
.File 1 (example of dot as prefix).pdf	to	File 1 (example of dot as prefix).pdf
File 10 (example of space as suffix) .pdf	to	File 10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	to	File 10 (example of dot as suffix)..pdf
<space> <space><space> test.pdf	to	test.pdf
….test.pdf	to	test.pdf
test….pdf	to	test.pdf
test <space><space><space>.pdf	to	test.pdf

<space> File 1 (example of space as prefix and suffix)<space>.pdf
to
File 1 (example of space as prefix and suffix).pdf

RegEx Equivalent: cont.

Tim Contribution – cont.

Removes all dots or <spaces> at prefix *and* suffix regardless if at one or both ends.

Match: (.*?)(\.+|\s+)(.*)(\.+|\s+)(.*)
Replace: \1\3\5

Name	New Name
Test....pdf	Test.pdf
test .pdf	test.pdf
....test.pdf	test.pdf
test.pdf	test.pdf
File 10 (example of space as suffix) .pdf	File10 (example of space as suffix).pdf
File 10 (example of dot as suffix)..pdf	File10 (example of dot as suffix).pdf
.File 1 (example of dot as prefix).pdf	File 1 (example of dot asprefix).pdf
.File 1 (example of dot as prefix and suffix)..pdf	File 1 (example of dot as prefix and suffix).pdf
File 1 (example of space as prefix).pdf	File 1 (example of space asprefix).pdf
File 1 (example of space as prefix and suffix) .pdf	File 1 (example of space as prefix and suffix).pdf

JavaScript

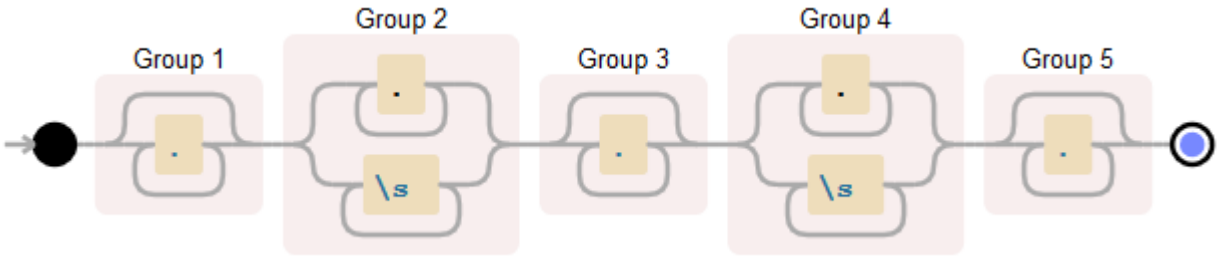
Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

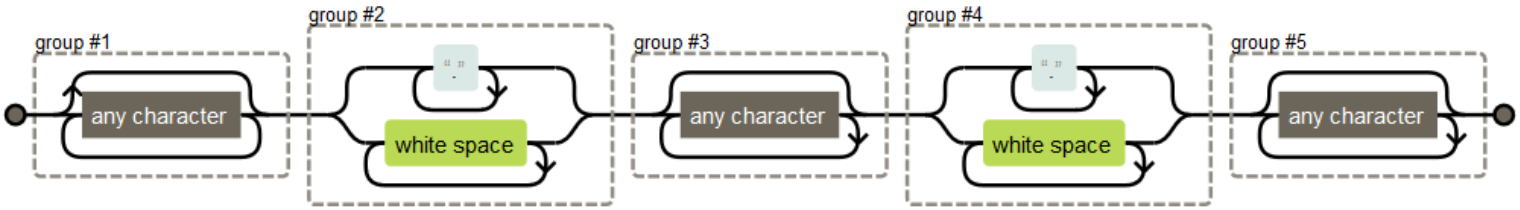
RegEx Equivalent: cont.

Tim Contribution – cont.

Removes all dots or <spaces> at prefix *and* suffix regardless if at one or both ends. cont.

Match: (.*?)(\.+|\s+)(.*)(\.+|\s+)(.*)





Sample:

	Start	Length
Match 1:	0	48
Group 1:	0	0
Group 2:	0	1
Group 3:	1	46
Group 4:	47	1
Group 5:	48	0

Sample: <spaces> at suffix

	Start	Length
Match 1: test	0	8
Group 1: test	0	4
Group 2:	4	3
Group 3:	7	0
Group 4:	7	1
Group 5:	8	0

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

RegEx Equivalent: cont.

Tim Contribution – cont.

Removes all dots or <spaces> at prefix *and* suffix regardless if at one or both ends. cont.

Match: (.*?)(\.+|\s+)(.*)(\.+|\s+)(.*)

But can not handle mixed, meaning <spaces> at one end and dots at the other and vice versa. This is because the OR will always find the <space> first.

i.e.,

Removes <space> at prefix but not the dots at the suffix.

Name	New Name
abc123....txt	abc123...txt

Changing it to this will work on mixed but will mess up others so be selective when choosing files.

Match: (.*?)(\.+|\s+)(.+?)(\.+|\s+$)(.*)
Replace: \1\3\5

Name	New Name
....mixed test .pdf	mixed test.pdf
abc123....txt	abc123.txt

Sample:

	Start	Length
Match 1:	0	19
Group 1:	0	0
Group 2:	0	5
Group 3:	5	10
Group 4:	15	4
Group 5:	19	0

Sample:

	Start	Length
Match 1:	0	13
Group 1:	0	0
Group 2:	0	4
Group 3:	4	6
Group 4:	10	3
Group 5:	13	0

JavaScript

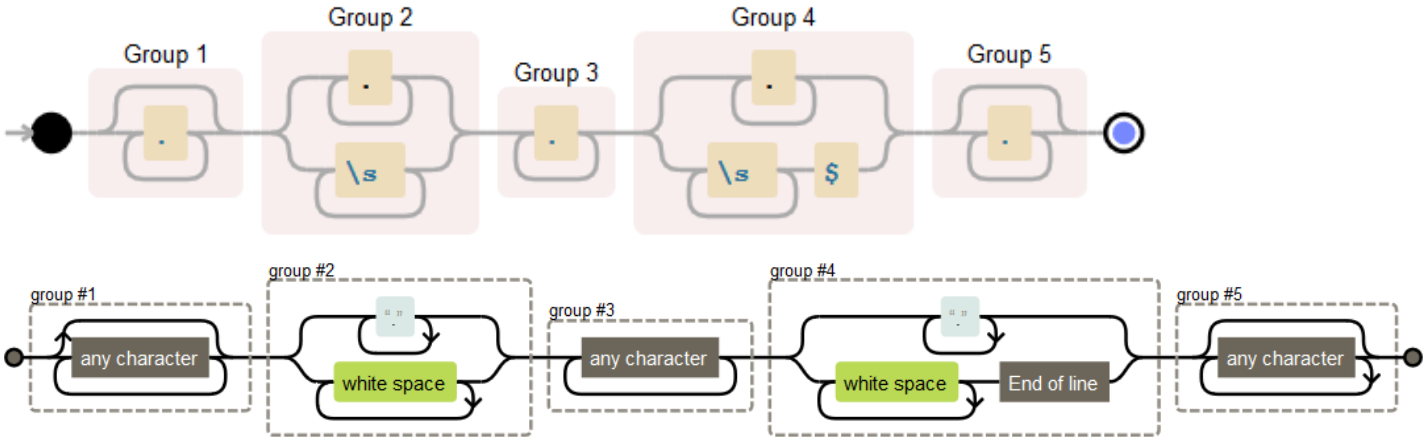
Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

RegEx Equivalent: cont.

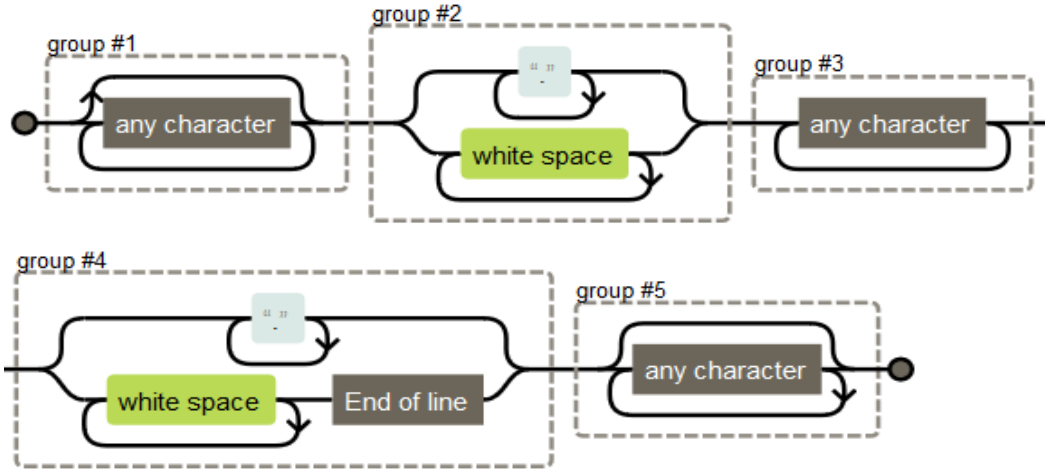
Tim Contribution – cont.

Changing it to this will work on mixed but will mess up others so be selective when choosing files. cont.

Match: (.*?)(\.+|\s+)(.+?)(\.+|\s+$)(.*)



Expanded:



Sample:

```
                                                    Start  Length
Match 1: .File 1 (example of dot as prefix and suffix)..      0      47
Group 1:                                                      0       0
Group 2: .                                                    0       1
Group 3: File 1 (example of dot as prefix and suffix)         1      44
Group 4: ..                                                  45       2
Group 5:                                                     47       0
```

JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

Posting Continues:

Tim Notes:

1. This contributor made an alternative to mine that is simpler – as said in the post, 'The beauty of RegEx':

Removes any combination of Dots-or-Spaces from beginning or end of filename:

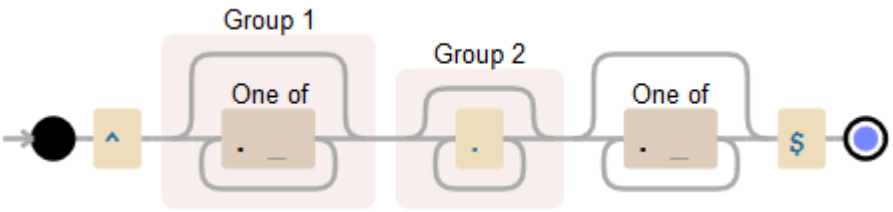
begin .. mid .. end .. .. .. to begin .. mid .. end

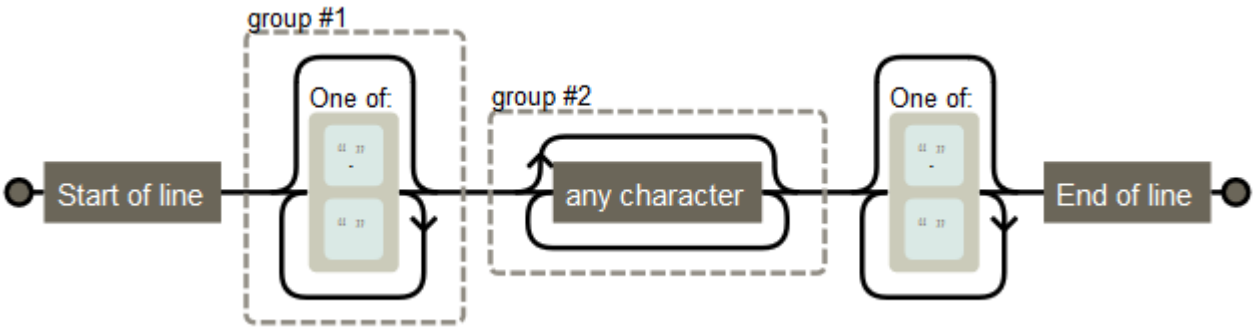
.. .. begin .. mid .. end .. .. to begin .. mid .. end

... . begin .. mid .. end to begin .. mid .. end

Match: ^([.]*)(.*?)[.]*$
Replace: \2

Name	New Name
begin .. mid .. end	begin .. mid .. end
.. .. begin .. mid .. end	begin .. mid .. end
... . begin .. mid .. end	begin .. mid .. end





JavaScript

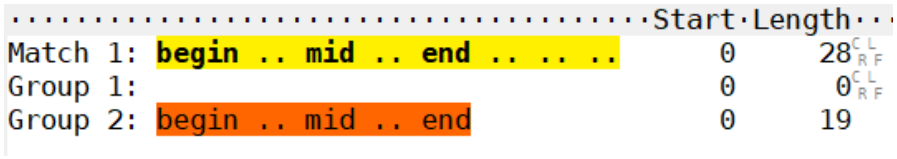
Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

Posting Continues: cont.

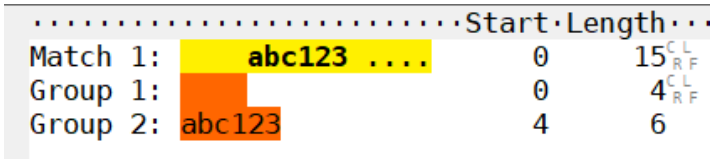
Removes any combination of Dots-or-Spaces from beginning or end of filename: cont.

Match: ^([.]*)(.*?)[.]*$

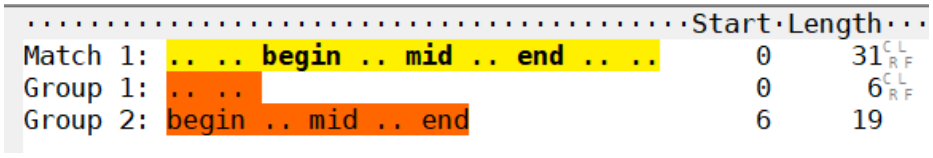
Sample:



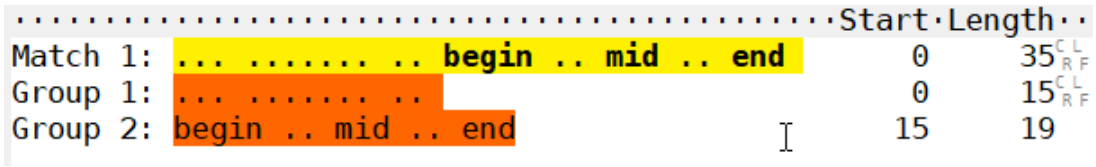
Sample:



Sample:



Sample:



JavaScript

Remove If the prefix or suffix character of a Filename is either a space or a dot cont.

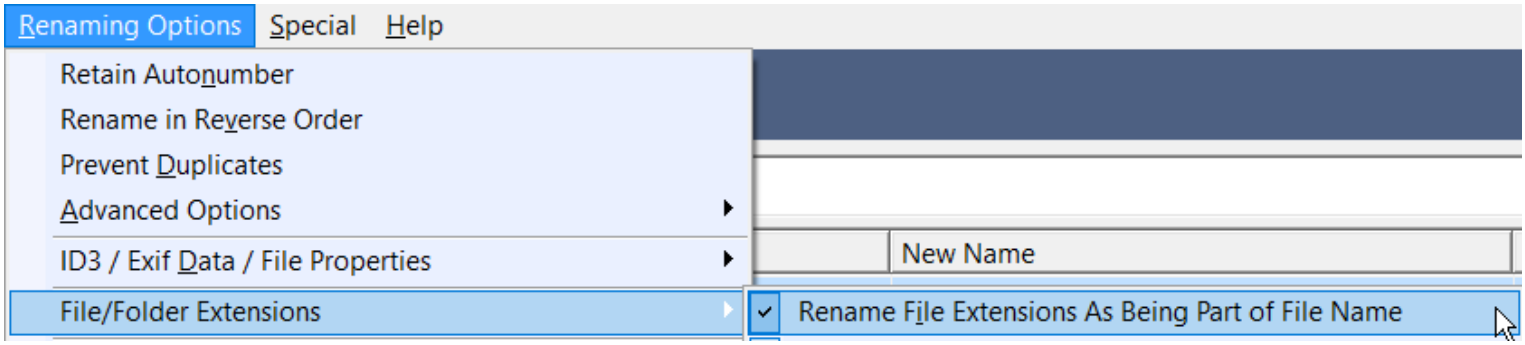
Posting Continues: cont.

Removes any combination of Dots-or-Spaces from beginning or end of filename: cont.

Match: ^([.]*)(.*?)[.]*$

Tim Notes:

1. These samples used dots in the filename. Set 'Rename File Extensions As Being Part of File Name' under File/Folder Extensions sub menu of the Renaming Options Menu.



2. Not sure why BRU sees the second sample as a folder in the New Name photo on the previous page. These are all files.

Forum Contributors: Admin, therube, bru

JavaScript

Shorten Music Directory names by Removing Artist and leaving only Album Names

Folder names

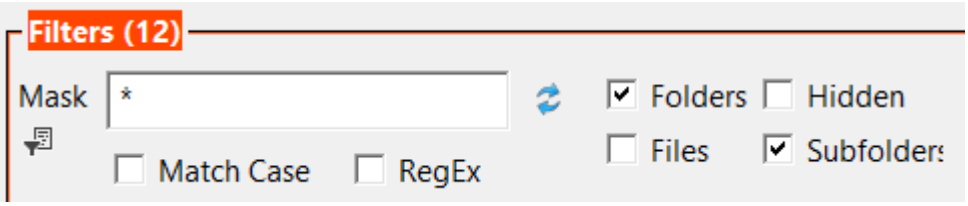
\Music\ZZ TOP\ZZ TOP GREATEST HITS to C:\MUSIC\ZZ TOP\GREATEST HITS

Observations:

1. Rename the sub folders by removing the 'ZZ TOP' from the beginning of all subfolders, leaving just the album name.
2. Remove the initial part of a subfolder name if that matches the parent folder.

Three steps:

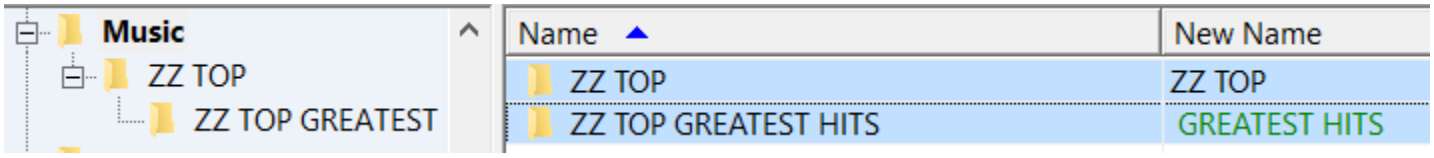
1. Use 'Section #12: Filter' to only select to view Folders and activate Subfolders (recursion).



2. Select *all* files in the Content Pane (the File List Window).

3. JavaScript:

```
var path = object('folder');
var subfolders = path.split("\\");
var lastsub = (subfolders.length > 1) ? subfolders[subfolders.length - 2] : "";
newName = name.replace(lastsub, "");

newName = newName.replace(" - ", "");
```



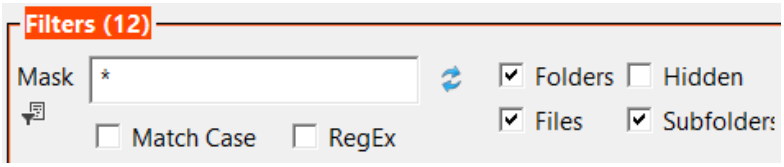
Forum Contributor: Admin

JavaScript

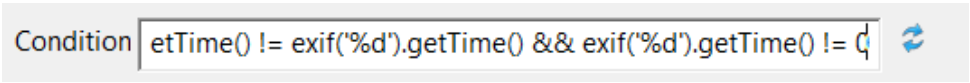
JavaScript Condition Filter files with modified date not matching EXIF date Taken Timestamp

1. Select a folder in the folder tree
2. Include subfolders if needed (recursion):

3. 'Section #12: Filters' -



3. Enter the JavaScript condition in the data field provided in 'Section #12: Filters':




```
object('modified').getTime() != exif('%d').getTime() && exif('%d').getTime() != 0
```

d) Click on the Refresh file list button (blue button to the right of the 'Condition' data entry field).

This will show files with Windows 'modified' timestamp *not matching* EXIF date taken timestamp.

Before:

Name	New Name	Type	Size	Created	Modified	Accessed	Taken (Original)
DSCN0253.JPG	DSCN0253.JPG	JPG File	5 MB	6/4/2011 2:43:10 PM	6/4/2011 2:43:12 PM	8/7/2017 1:00:00 AM	6/4/2011 1:43:11 PM
DSCN0252.JPG	DSCN0252.JPG	JPG File	5 MB	6/4/2011 2:43:10 PM	6/4/2011 2:43:10 PM	8/7/2017 1:00:00 AM	6/4/2011 1:43:10 PM
DSCN0251.JPG	DSCN0251.JPG	JPG File	5 MB	6/4/2011 2:43:08 PM	6/4/2011 2:43:10 PM	8/7/2017 1:00:00 AM	6/4/2011 1:43:09 PM
DSCN0250.JPG	DSCN0250.JPG	JPG File	5 MB	6/4/2011 2:15:16 PM	6/4/2011 2:15:18 PM	8/7/2017 1:00:00 AM	6/4/2011 1:15:17 PM
DSCN0249.JPG	DSCN0249.JPG	JPG File	5 MB	6/4/2011 2:14:00 PM	6/4/2011 2:14:02 PM	8/7/2017 1:00:00 AM	6/4/2011 1:14:02 PM
DSCN0248.JPG	DSCN0248.JPG	JPG File	5 MB	6/4/2011 2:13:22 PM	6/4/2011 2:13:22 PM	8/7/2017 1:00:00 AM	6/4/2011 1:13:22 PM
DSCN0247.JPG	DSCN0247.JPG	JPG File	5 MB	6/4/2011 2:13:06 PM	6/4/2011 2:13:06 PM	8/7/2017 1:00:00 AM	6/4/2011 1:13:07 PM
DSCN0246.JPG	DSCN0246.JPG	JPG File	5 MB	6/4/2011 1:02:22 PM	6/4/2011 1:02:22 PM	8/7/2017 1:00:00 AM	6/4/2011 12:02:22 PM

After:

Name	New Name	Type	Size	Created	Modified	Accessed	Taken (Original)
DSCN0218.JPG	DSCN0218.JPG	JPG File	5 MB	1/28/2011 7:37:48 PM	1/28/2011 7:37:48 PM	8/7/2017 1:00:00 AM	1/28/2011 7:37:49 PM
DSCN0215.JPG	DSCN0215.JPG	JPG File	5 MB	1/28/2011 7:28:34 PM	1/28/2011 7:28:36 PM	8/7/2017 1:00:00 AM	1/28/2011 7:28:35 PM
DSCN0214.JPG	DSCN0214.JPG	JPG File	5 MB	1/28/2011 7:28:04 PM	1/28/2011 7:28:04 PM	8/7/2017 1:00:00 AM	1/28/2011 7:28:05 PM
DSCN0207.JPG	DSCN0207.JPG	JPG File	5 MB	1/28/2011 7:17:44 PM	1/28/2011 7:17:44 PM	8/7/2017 1:00:00 AM	1/28/2011 7:17:45 PM
DSCN0200.JPG	DSCN0200.JPG	JPG File	5 MB	12/25/2010 2:03:10 PM	12/25/2010 2:03:12 PM	8/7/2017 1:00:00 AM	12/25/2010 2:03:11 PM
DSCN0199.JPG	DSCN0199.JPG	JPG File	5 MB	12/18/2010 3:31:18 PM	12/18/2010 3:31:20 PM	8/7/2017 1:00:00 AM	12/18/2010 3:31:19 PM
DSCN0198.JPG	DSCN0198.JPG	JPG File	5 MB	12/18/2010 3:30:56 PM	12/18/2010 3:30:56 PM	8/7/2017 1:00:00 AM	12/18/2010 3:30:57 PM
DSCN0197.JPG	DSCN0197.JPG	JPG File	5 MB	12/18/2010 3:24:20 PM	12/18/2010 3:24:20 PM	8/7/2017 1:00:00 AM	12/18/2010 3:24:21 PM
DSCN0195.JPG	DSCN0195.JPG	JPG File	5 MB	12/18/2010 3:16:02 PM	12/18/2010 3:16:04 PM	8/7/2017 1:00:00 AM	12/18/2010 3:16:03 PM

Prerequisites:

1. Enable extraction of EXIF data through 'Renaming Options Menu - Extract Exif Data (Photos)'

Forum Contributor: Admin

JavaScript

Replace Prefix '18' with '20' in string

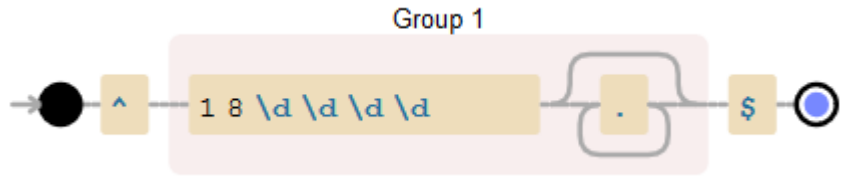
181234anything else.pdf to 201234anything else.pdf
185678another test.pdf to 205678another test.pdf

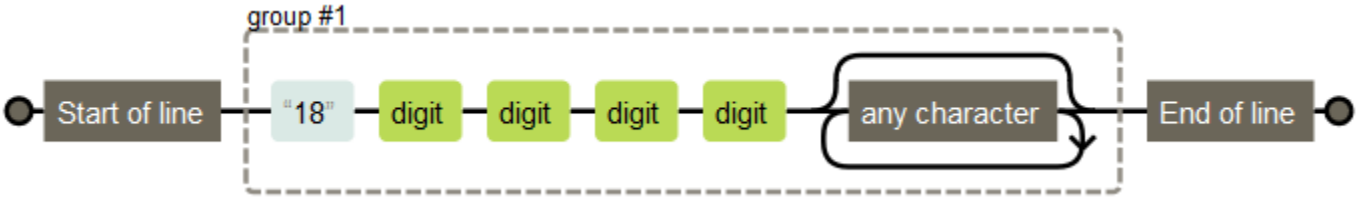
Observations:

1. Match all files that start with 18, followed by 4 numeric digits, followed by anything else.
2. Prefix those files with, 20.

Match: ^(18\d\d\d\d.*)$
Replace: 20\1

Name	New Name
181234anything else.pdf	20181234anything else.pdf
185678another test.pdf	20185678another test.pdf





Sample:

```
.............................
Match 1: 181234anything else
Group 1: 181234anything else
```

JavaScript

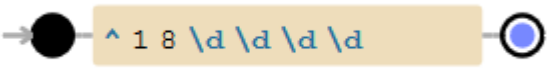
Replace Prefix '18' with '20' in string cont.

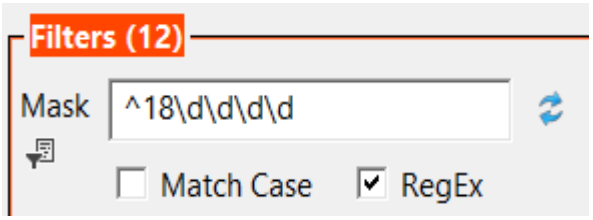
181234anything else.pdf to 201234anything else.pdf
185678another test.pdf to 205678another test.pdf

Alternatively, also in one run:

Pre-select, by various means, only files that start with 18

'Section #12: Filters' –

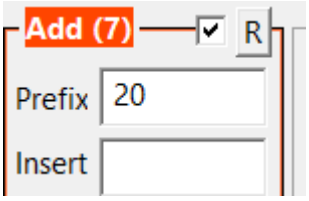




Mask: ^18\d\d\d\d
RegEx: Enabled

Name	New Name
181234anything else.pdf	181234anything else.pdf
185678another test.pdf	185678another test.pdf

2. 'Section #7: Add' –



Prefix: 20

Name	New Name
181234anything else.pdf	20181234anything else.pdf
185678another test.pdf	20185678another test.pdf

JavaScript

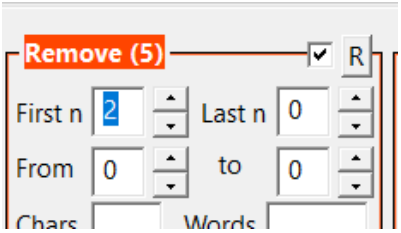
Replace Prefix '18' with '20' in string cont.

181234anything else.pdf to 201234anything else.pdf
185678another test.pdf to 205678another test.pdf

Tim Contribution –

If you wanted to replace the current prefix '18', use this:

3. 'Section #5: Remove' –



First n: 2

Removes the '18' from the filename. Because BRU runs Remove before Add, this won't affect the '20' prefix.

Name	New Name
181234anything else.pdf	201234anything else.pdf
185678another test.pdf	205678another test.pdf

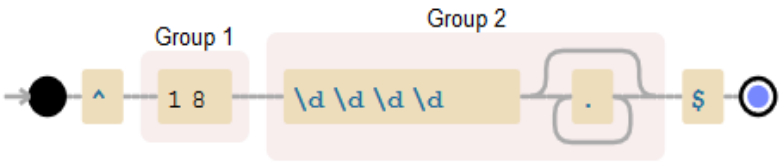
… and change the RegEx to:

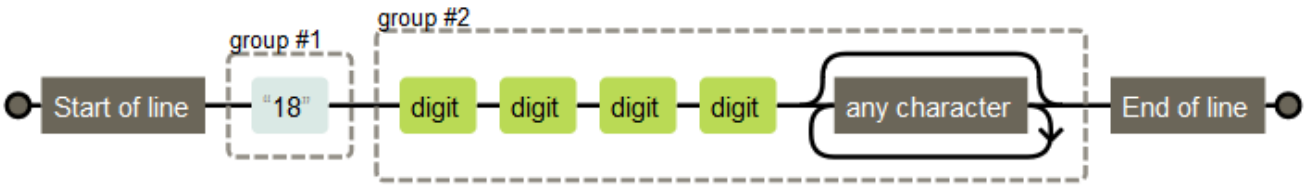
Match: ^(18)(\d\d\d\d.*)$
Replace: 20\2

Name	New Name
181234anything else.pdf	201234anything else.pdf
185678another test.pdf	205678another test.pdf

Sample:

```
Match 1: 181234anything else
Group 1: 18
Group 2: 1234anything else
```





JavaScript

Replace Prefix '18' with '20' in string cont.

181234anything else.pdf	to	201234anything else.pdf
185678another test.pdf	to	205678another test.pdf

Posting Continues:

JavaScript equivalent:

```
if (name.substring(0,2)== '18') newName = '20' + name;
```

Name	New Name
181234anything else.pdf	20181234anything else.pdf
185678another test.pdf	20185678another test.pdf

Forum Contributors: therube, Admin

JavaScript

Removing . (dot) from beginning of files

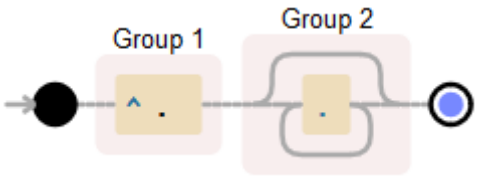
.rock and roll all nite.txt to rock and roll all nite.txt

Observations:

1. Remove first dot at beginning at file (Beginning of Line or BOL) (position 0)
2. Possibly also replace with underscore '_'

Match: (^\.)(.*)
Replace: \2

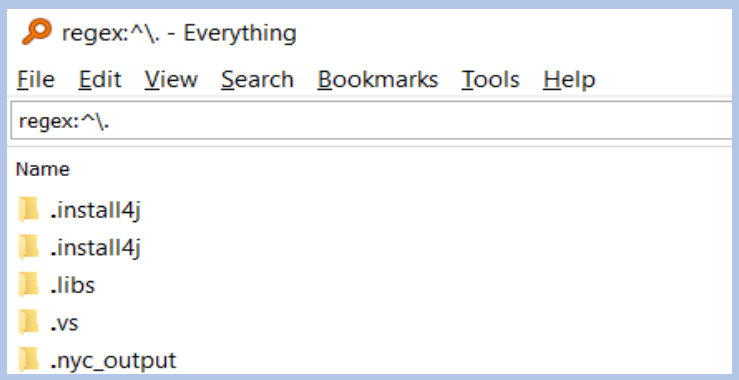
Name	New Name
.rock and roll all nite.txt	rock and roll all nite.txt



Sample:

```
Match 1: .rock and roll all nite
Group 1: .
Group 2: rock and roll all nite
```

Use the program, 'Everything', to search only for filenames that begin with dot - regex: ^\.



JavaScript

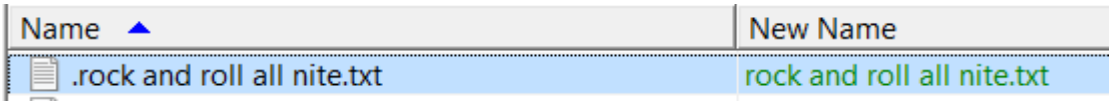
Removing . (dot) from beginning of files cont.

.rock and roll all nite.txt to rock and roll all nite.txt

JavaScript Equivalent:

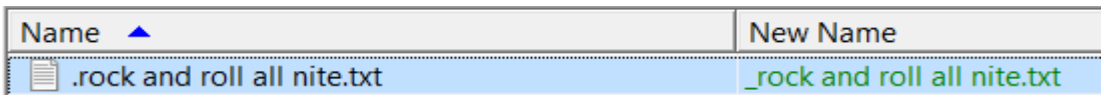
To remove the . in the first position:

```
if (name.charAt(0) == ".") newName = name.substring(1);
```



Or to replace with _ :

```
if (name.charAt(0) == ".") newName = "_" + name.substring(1);
```



Forum Contributors: therube, Admin

JavaScript

Move only renamed files to a new specified folder

Samples:

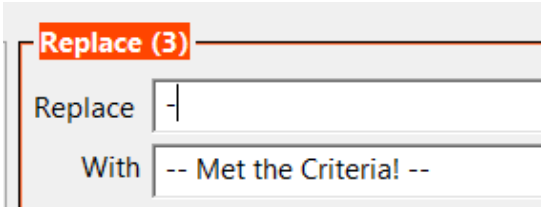
a 11.txt
Anymus_192638322 - Inkyz.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-Mac.mp3
AWOLNATION - Run (B52 Remix)_217279053 - B52.mp3
b&B_296170807 - psupac.mp3
foo bar_Disk2.xyz
Vacation.1980
Vacation.1982
[Word1-Word2] 2010.ext
[Word] 2006.ext
Word1-Word2.com - 2009 r.ext
a 1.txt
a 12.txt
a 2.txt
a 3.txt

Observations:

1. Move only the successfully renamed files to another folder.

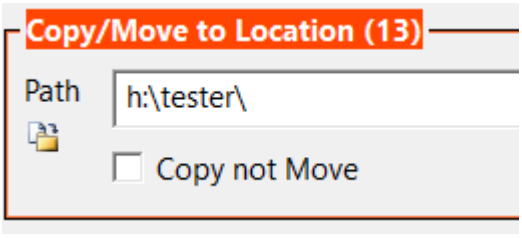
I will use a simple criteria to illustrate the methods:

1. 'Section #3: Replace'



Replace: <hyphen>
With: < text.>

2. 'Section #13: Move to Location' - specify your new folder.



Path: h:\tester Moves renamed files to specified directory

Tim Notes:

1. If directory does not exist, it will be created.

JavaScript

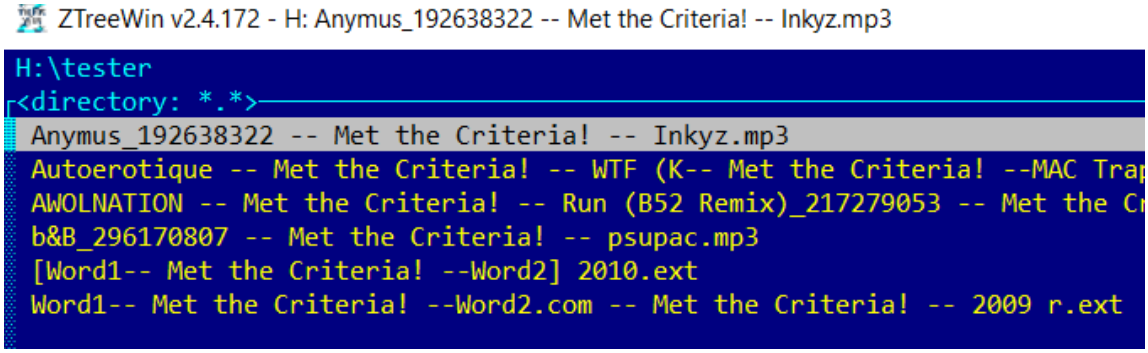
Move only renamed files to a new specified folder cont.

Method #1:

JavaScript:

```
if (name == origName) newLocation = object("folder");
```

This will copy ONLY the renamed files to the new location.



JavaScript

Move only renamed files to a new specified folder cont.

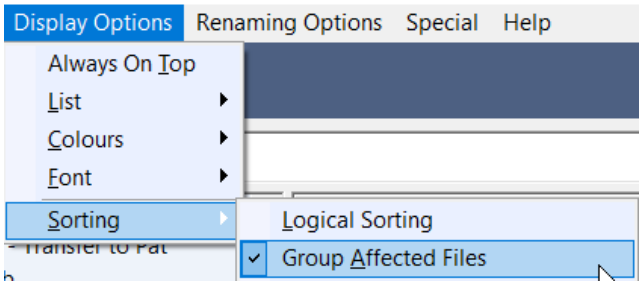
Method #2:

1. In the Content Pane, use Select All (Ctrl + A) to select all of the files.

Name	New Name
Word1-Word2.com - 2009 r.ext	Word1-- Met the Criteria! --Word2.com -- Met the Criteria! -- 2009 r.ext
Vacation.1982	Vacation.1982
Vacation.1980	Vacation.1980
foo bar_Disk2.xyz	foo bar_Disk2.xyz
b&B_296170807 - psupac.mp3	b&B_296170807 -- Met the Criteria! -- psupac.mp3
AWOLNATION - Run (B52 Remix)_217279053 - B...	AWOLNATION -- Met the Criteria! -- Run (B52 Remix)_217279053 -- Met the Criteria! -- B52.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 ...	Autoerotique -- Met the Criteria! -- WTF (K-- Met the Criteria! --MAC Trap VIP)_96732707 -- Met t
Anymus_192638322 - Inkyz.mp3	Anymus_192638322 -- Met the Criteria! -- Inkyz.mp3
a 3.txt	a 3.txt
a 2.txt	a 2.txt
a 12.txt	a 12.txt
a 11.txt	a 11.txt
a 1.txt	a 1.txt
[Word] 2006.ext	[Word] 2006.ext
[Word1-Word2] 2010.ext	[Word1-- Met the Criteria! --Word2] 2010.ext
Vacation.1987	Vacation.1987
test b	test b
test a	test a

2. The files that meet the set criteria will appear in green in New Name column.

3. Re-order the data set to display all matched files to the top of the file list.
   a. Select 'Group Affected Files' from the Sorting sub-menu of the Display Options Menu.



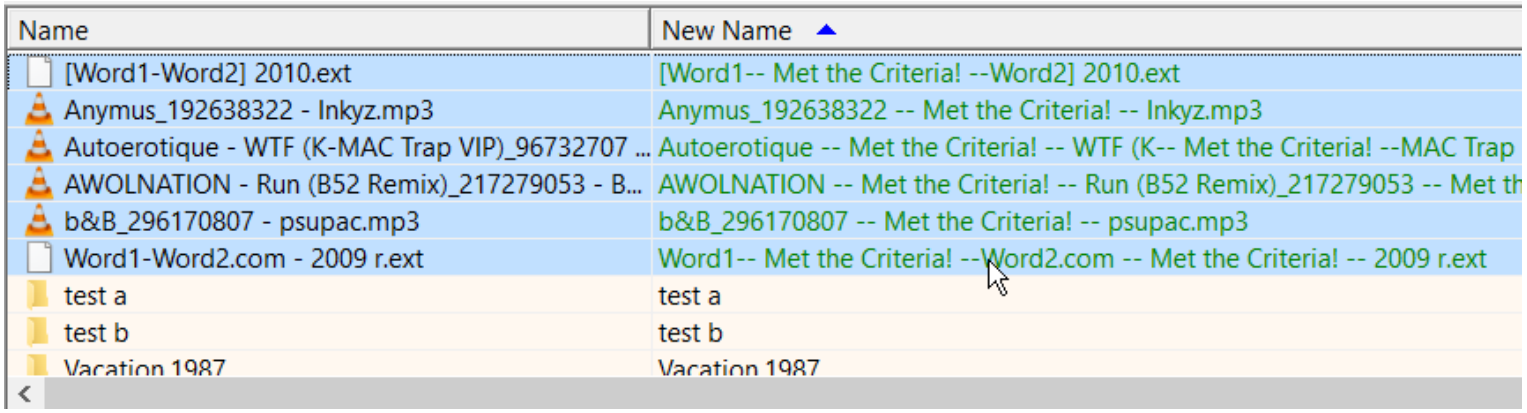
Name	New Name
[Word1-Word2] 2010.ext	[Word1-- Met the Criteria! --Word2] 2010.ext
Anymus_192638322 - Inkyz.mp3	Anymus_192638322 -- Met the Criteria! -- Inkyz.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 ...	Autoerotique -- Met the Criteria! -- WTF (K-- Met the Criteria! --MAC Trap VIP)
AWOLNATION - Run (B52 Remix)_217279053 - B...	AWOLNATION -- Met the Criteria! -- Run (B52 Remix)_217279053 -- Met the C
b&B_296170807 - psupac.mp3	b&B_296170807 -- Met the Criteria! -- psupac.mp3
Word1-Word2.com - 2009 r.ext	Word1-- Met the Criteria! --Word2.com -- Met the Criteria! -- 2009 r.ext
test a	test a
test b	test b
Vacation.1987	Vacation.1987

JavaScript

Move only renamed files to a new specified folder cont.

Method #2: cont.

4. Select Only those files that will be matched.
   a. Click the last green one, then press and hold shift, then click the top one (Consecutive Selection).



5. Click 'Rename'.

Forum Contributors: Admin, hannah141

JavaScript

Replace Filenames with Values based on Nested Loops

Samples - 110 files –

all different names. Samples not shown.

Observations:

1. Remove original file name.
2. Follow a pattern of renaming the files starting at 0x0
3. Begin the count over for every eleven files 0-10
4. Increment = 1 using a nested loop

Tim Notes:

Pseudocode

```
I = 0                          Outer Loop counter set Initial Value
N = 0                          Inner Loop counter set Initial Value
For I = I to 10                Start Outer Loop
    For N = N to 10            Start Inner Loop
      rename filename_IxN      Perform Rename
      N= N+1                   Increment Inner Loop Counter
    Next N                     Do Next Inner Loop Iteration
    N= 0                       After Inner Loop Finshes, Reset counter
    I = I + 1                  Increment Outer Loop Counter
Next I                         Do Next Outer Loop Interation
```

First batch	Second batch	Third batch	…	Last batch
0x1 to 0x10	1x0 to 1x10	2x0 to 2x10		10x0 to 10x10

so .. the Requester is using a sample group of 110 files in all. (11 x 10), each having a .bmp extension.

5. Files reside all in a single directory.

Admin's script:

```
newName = (~~(counter / 10)+1) + "x" + counter % 10;
```

Tim Notes:

1. The '~~' truncates (~ is bitwise 'Not' – two of these together perform a bitwise complement) the counter expression to an integer (removes the decimal portion so 1.1x1 becomes 1x1).

2. Extension doesn't matter in these formulas and in my examples not considered.

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Here is the JavaScript output:

Name	New Name
Word1-Word2.com - 2009 r.ext	1x1.ext
Windows 7 Quick Reference Card.pdf	1x2.pdf
Welcome.doc	1x3.doc
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	1x4
Slide of Mt.Rushmore 738.jpg	1x5.jpg
Receipt_php.txt	1x6.txt
Receipt_php.mht	1x7.mht
Receipt_php.htm	1x8.htm
prefs.js	1x9.js
nc_025213.pdf	2x0.pdf
nc_017733.pdf	2x1.pdf
Microsoft.doc	2x2.doc
localstore.rdf	2x3.rdf
LinkClick.pdf	2x4.pdf
Hassle.doc	2x5.doc
Getting Started Guide.doc	2x6.doc
Generalities.pdf	2x7.pdf
DSCN0001.JPG	2x8.JPG
dad teach.doc	2x9.doc
dad Internet.doc	3x0.doc
CAN-FirstLast.docx	3x1.docx
CAN-First Last.docx	3x2.docx
CAN - FirstLast.docx	3x3.docx
b&B_296170807 - psupac.mp3	3x4.mp3
b&B_296170807 - psupac.mp3	3x5.mp3
AWOLNATION - Run (B52 Remix)_217279053 - B52.m...	3x6.mp3
AWOLNATION - Run (B52 Remix)_217279053 - B52.m...	3x7.mp3
Autoerotique - WTF (K-MAC Trap VIP)_96732707 - K-...	3x8.mp3
Anymus_192638322 - Inkyz.mp3	3x9.mp3
AckleyDustin.txt	4x0.txt
[Word] 2006.ext	4x1.ext
[Word1-Word2] 2010.ext	4x2.ext
3052503155104.Bourjois_FDTAirMat_01_3.png	4x3.png
1863_22_10_2019.txt	4x4.txt

Forum Contributor: Admin

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Tim Contribution –

1. Not what Poster wanted.
   a. Start at zero and range to 10 so this requires a range of 11 (0 – 10)
      1. This JavaScript only has a range of 10 (0-9)
      2. Counter is a constant in BRU whose starting value 1 cannot be changed, so the range is now (1-10)

Analysis:

1. counter / 10 Controls the outer loop. It is the first number in 1x1 and will count from 1x1 to infinity.
2. counter %10 Controls the inner loop. It is the second number in 1x1 and will count from 1x1 to 1x0 (0 = 10). Why 10 = 0 is explained below.

counter /10		counter % 10	
1 / 10 = 0.1 ~ 0	6 / 10 = 0.6 ~ 0	1 mod 10 = 0.1 ~ 1	6 mod 10 = 0.6 ~ 6
2 / 10 = 0.2 ~ 0	7 / 10 = 0.7 ~ 0	2 mod 10 = 0.2 ~ 2	7 mod 10 = 0.7 ~ 7
3 / 10 = 0.3 ~ 0	8 / 10 = 0.8 ~ 0	3 mod 10 = 0.3 ~3	8 mod 10 = 0.8 ~ 8
4 / 10 = 0.4 ~ 0	9 / 10 = 0.9 ~ 0	4 mod 10 = 0.4 ~ 4	9 mod 10 = 0.9 ~ 9
5 / 10 = 0.5 ~ 0	10 / 10 = 1.0 ~ 1	5 mod 10 = 0.5 ~ 5	10 mod 10 = 1.0 ~ 0

where (in JavaScript):

/ = division % = modulo division

i.e.,

4 mod 10 is, 4 divided by 10, leaving a remainder of 4 from the 0.4 result so the Modulo is 4

.. in the 2x0 example, the zero portion representing the inner loop value is actually 10, but 10 mod 10 leaves a modulo of zero thus it becomes 2x0 instead.

3. +1 adds 1 to the value of (counter /10) so 0 + 1 = 1. This is why the count starts at 1 and not 0 for the outer string.

Tim Notes:

1. However the Poster wanted the value to start at zero and not 1
2. Removing the '+1' will get the zero value for the outer loop.

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Tim Contribution – cont.

Here is my clunky fix:

```
part1 = (~~(counter / 11));
part2 = ((counter % 11)-1);
if (part2 < 0){
(part2 = 10);
(part1 = (part1-1));
}

newName = (part1) + "x" + (part2)
```

Name	New Name
RSCN0728.JPG	0x0.JPG
RSCN0514.JPG	0x1.JPG
FSCN0797.JPG	0x2.JPG
FSCN0517.JPG	0x3.JPG
FSCN0407.JPG	0x4.JPG
FSCN0006.JPG	0x5.JPG
DSCN0838.JPG	0x6.JPG
DSCN0837.JPG	0x7.JPG
DSCN0836.JPG	0x8.JPG
DSCN0835.JPG	0x9.JPG
DSCN0834.JPG	0x10.JPG
DSCN0833.JPG	1x0.JPG
DSCN0832.JPG	1x1.JPG
DSCN0831.JPG	1x2.JPG
DSCN0830.JPG	1x3.JPG
DSCN0829.JPG	1x4.JPG
DSCN0828.JPG	1x5.JPG
DSCN0827.JPG	1x6.JPG
DSCN0826.JPG	1x7.JPG
DSCN0825.JPG	1x8.JPG
DSCN0824.JPG	1x9.JPG
DSCN0823.JPG	1x10.JPG
DSCN0822.JPG	2x0.JPG
DSCN0821.JPG	2x1.JPG
DSCN0820.JPG	2x2.JPG
DSCN0819.JPG	2x3.JPG
DSCN0818.JPG	2x4.JPG
DSCN0817.JPG	2x5.JPG
DSCN0816.JPG	2x6.JPG
DSCN0815.JPG	2x7.JPG
DSCN0814.JPG	2x8.JPG
DSCN0813.JPG	2x9.JPG
DSCN0812.JPG	2x10.JPG
DSCN0811.JPG	3x0.JPG

Analysis:

1. part 1 — Outer loop using same script as original except changed range to 11 so it will be 0-10 and removed +1 because that would start at 1 and not 0.

2. part 2 — inner loop, again using 11 BUT -1 because counter starts at 1 and I want to start at zero.

3. If — When the index reaches -1 which it will do on the 11th iteration, when in part 2 the counter = 11 (mod 11= 0-1 = -1), it sets part 2 to 10 and decrements part1.

4. NewName — Puts togther the string.

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Tim Contribution – cont.

To fully understand this, you have to see how I got to *here*. Not to give you every trial and error step, but when I got to this in the process, things started to come together (I am not very good at JavaScript, understand).

```
part1 = (~~(counter / 11));
part2 = ((counter % 11) -1);
if (part2 < 1) part2 = 0;

newName = (part1) + "x" + (part2)
```

The key is the If statement. By this time the values I have for the Outer and Inner loops are correct. It is, however, the If statement that is the problem. When 11 mod 11 = 0, then the value of 11 becomes 0, just as in the Admin's script when 10 mod 10 = 0, it became x0 instead of x10 (the outer loop was also wrong changing over to 2x10 when it should have been 1x10). I did not want this.

The If statement attempts to cover for this by 'catching' when the Inner Loop counter (part2) reaches -1, which it will do on the 11th iteration. When the counter = 11 and mod 11= 0 minus 1 = -1, it resets part 2's value to zero.

This results in:

counter	part 2 value	calculation	counter	part 2 value	calculation
10	9	10 mod 11 = 9	5	4	5 mod 11 = 4
11	0	11 mod 11 = 0 -1 = -1	6	5	6 mod 11 = 5
		If < 1 part2 = 0	7	6	7 mod 11 = 6
			8	7	8 mod 11 = 7
1	0	1 mod 11 = 0 -1 = -1	9	8	9 mod 11 = 8
		if < 1 part2 = 0	10	9	10 mod 11 = 9
			11	0	11 mod 11 = 0 -1 = -1
2	1	2 mod 11 = 1			If < 1 part2 = 0
3	2	3 mod 11 = 2			
4	3	4 mod 11 = 3	1	0	1 mod 11 = 0 -1 = -1
					If < 1 part2 = 0

See the two zero's in part2's value? The first is when counter = 11, and the second is when the counter = 1.

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Tim Contribution – cont.

This can clearly be seen here:

Name	New Name
RSCN0728.JPG	0x0.JPG
RSCN0514.JPG	0x1.JPG
FSCN0797.JPG	0x2.JPG
FSCN0517.JPG	0x3.JPG
FSCN0407.JPG	0x4.JPG
FSCN0006.JPG	0x5.JPG
DSCN0838.JPG	0x6.JPG
DSCN0837.JPG	0x7.JPG
DSCN0836.JPG	0x8.JPG
DSCN0835.JPG	0x9.JPG
DSCN0834.JPG	1x0.JPG
DSCN0833.JPG	1x0.JPG
DSCN0832.JPG	1x1.JPG
DSCN0831.JPG	1x2.JPG
DSCN0830.JPG	1x3.JPG
DSCN0829.JPG	1x4.JPG
DSCN0828.JPG	1x5.JPG
DSCN0827.JPG	1x6.JPG
DSCN0826.JPG	1x7.JPG
DSCN0825.JPG	1x8.JPG
DSCN0824.JPG	1x9.JPG
DSCN0823.JPG	2x0.JPG
DSCN0822.JPG	2x0.JPG
DSCN0821.JPG	2x1.JPG

This affects both the inner loop (part2) as well as the outer loop (part1).

When the Outer Loop Counter (part1) = 12, part 1 = 1:
(counter / 11) or (12 /11 = 1.09 ~ 1)

This results in 1x0.

But since the previous value for newName was 0x9, the next value for newName should be 0x10 not 1x0. This again was the same problem exhibited in the Admin's script.

I've already explained Part2's value of 0 in the 1x0 string.

So both of these problems needed to be fixed.

My solution was clunky but it works:

```
if (part2 < 0){
(part2 = 10);
(part1 = (part1-1));
}
```

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Tim Contribution – cont.

When the Inner Loop Counter of part2 = 11 and the -1 value triggers the If statement, it sets two values, not just the one:

Sets part2 value to 10 instead of 0	This takes care of the 'x0 instead of x10' problem with the Inner loop.
Sets part1 value to part1 -1	So if part 1 = 1, sets it to 0. This takes care of the '1x instead of 0x' problem with the Outer loop.

This results in 0x10 instead of 1x10

On the next iteration, the part1 value is:

part 1 counter = 13 (part1 = ((counter / 11) = 1.18 ~ 1)

resulting in 1x0

so now it is correct:

0x10
1x0
.
.

DSCN0837.JPG	0x7.JPG
DSCN0836.JPG	0x8.JPG
DSCN0835.JPG	0x9.JPG
DSCN0834.JPG	0x10.JPG
DSCN0833.JPG	1x0.JPG
DSCN0832.JPG	1x1.JPG
DSCN0831.JPG	1x2.JPG

.
.

DSCN0824.JPG	1x9.JPG
DSCN0823.JPG	1x10.JPG
DSCN0822.JPG	2x0.JPG
DSCN0821.JPG	2x1.JPG
DSCN0820.JPG	2x2.JPG
DSCN0819.JPG	2x3.JPG

JavaScript

Replace Filenames with Values based on Nested Loops cont.

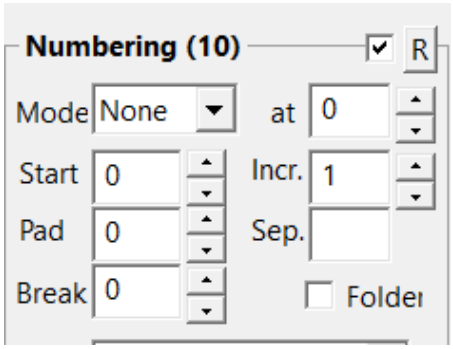
Tim Contribution – cont.

Since the time I wrote the first script, I have a better solution:

1. JavaScript:

```
newName = (~~(object("autonumber") / 11)) + "x" + (object ("autonumber")) % 11;
```

2. 'Section #10: Numbering' -



Start: 0
Mode: None
Increment: value not used.

Analysis:

1. object ("autonumber") — By using the Autonumber object I am no longer limited by the constraints of using the constant, 'counter', namely starting value = 1. The Autonumber object takes its starting value from the 'Start' field of 'Section #10: Numbering'. This is my new 'counter' not to be confused with the BRU JavaScript constant, 'counter'

   Because we use a starting value of zero, there is no need for the clunky If statement.

2. object ("autonumber") /11 — Defines the Outer Loop. Divides the counter value by 11 (already explained previously).
3. object ("autonumber") %11 — Defines the Inner Loop. Takes the Modulo of the counter value (previously explained).

4. 'Section #10: Numbering' — Generally, with Mode set to 'None,' this inactivates the criteria, and for all intents and purposes, it is inactive.

   To be used here, activation is not required. Just have to set the 'Start' to the value you want to transfer to 'object("autonumber")'. The JavaScript autonumber object will read this value.

Tim Notes:

2. The increment value is not used. You have to provide for that in the script.
3. For more information refer to, 'JavaScript BRU Constants and Variables', and 'JavaScript BRU Utility Functions' charts in Volume 1 Bulk Rename Utility Operations Manual', which can also be found in the BRU manual.

JavaScript

Replace Filenames with Values based on Nested Loops cont.

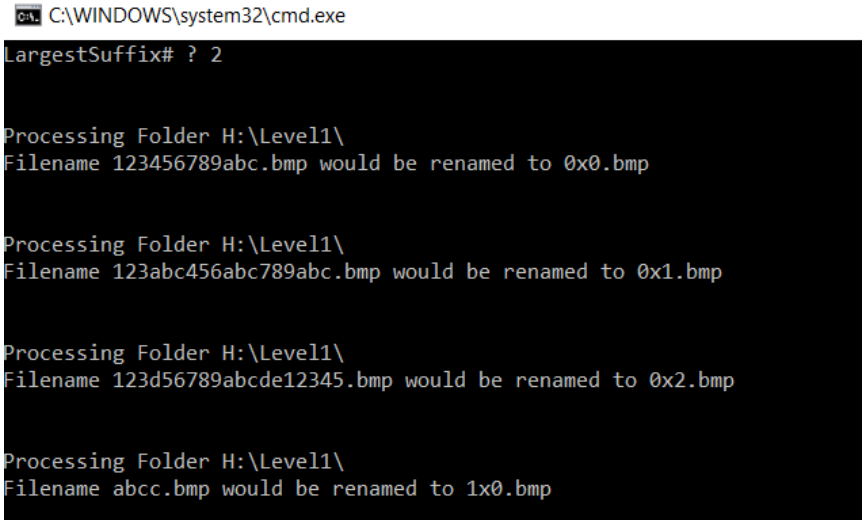
Post Continues -

Here's a batch file using brc32 that prompts the user to enter a LargestSuffix#:

%% =	variables are represented by a double percent sign in a batch file
BRC Specific - &	allow 2 commands on one line
!name!	The exclamations mean the variable will be expanded (see 'SetLocal EnableDelayedExpansion')
<space> =?!	- Character Substitution. Substitute the question mark for any <space> characters in !name!.
&&	- In a batch file, && runs the second command on the line when the first command comes back successfully
Set /a	– set to a numeric
Set /p	– sets the value of a variable to a line of input entered by the user (will request user input)

SetLocal EnableDelayedExpansion- Expand variables from within the batch file so they will display current values. Otherwise they do not and only display initial assigned values. Expand means to show the updated values as they change. The format for an expanded variable in the batch file is !<variable>! and this is in place of the usual %<variable>%.

```
@echo off
Set/p max=LargestSuffix# ?
Set/a pre=0, count=0, max="max"
SetLocal EnableDelayedExpansion
For /f "delims=" %%A IN ('dir/b/a-d *.bmp') DO ((Set name=%%A)&(Set name=!name: =?!)
If !count! leq !max! (brc32 /Pattern:!name! /FixedName:!pre!x!count! /Execute
Set/a count="count+1") ELSE ((Set count=0)&&(Set/a pre="pre+1")))
pause>nul
```



You can remove /Execute to preview what the renames would be.
For instance, if the user were to enter 2:

AnyName.bmp --> 0x0.bmp
AnyName.bmp --> 0x1.bmp
AnyName.bmp --> 0x2.bmp
AnyName.bmp --> 1x0.bmp
AnyName.bmp --> 1x1.bmp
AnyName.bmp --> 1x2.bmp
AnyName.bmp --> 2x0.bmp

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Breaks down to:

Define variables

Set /p max=LargestSuffix# ? - the '/p' requests user input. The 'LargestSuffix#?' is a message of instruction presented to the user. The value entered is assigned into the variable, 'max'.

Set /a pre=0 - used for New Name
count=0 - set counter to zero
max="max" – maximum iterations determined by LargestSuffix#

SetLocal EnableDelayedExpansion

For /f "delims=" %%A IN ('dir/b/a-d *.bmp') - Read one filename into %A. /f = specify it is for filenames. %A = holds filename.

DO ((Set name=%%A)&(Set name=!name:<space>=?!) - Sets a variable name to the filename represented by %A. This is done to allow substitution, since variables in the 'For' statement do not. The reason Substitution is required is for the conversion of <spaces>. This sets the variable 'name' to = 'name' with an added substitution of using a question mark for any <spaces> in the filename.

Why?

BRC uses the <space> as a file delimiter for processing several files in succession. For example if you had:

Brc64 /Pattern:"any particular name"

BRC would match against 3 different filespecs:

any
particular
name

Therefore, the Substitute is used in the batch file to take any <spaces> and convert them into a character that BRC will not see as a filename delimiter. In this case, the question mark is used. So using the same example, the filename becomes:

any?particular?name

Set Substitution Syntax: %<variable name> : <what to substitute> =<substitution>%

In batch file using Expanded Delay the syntax is: !<variable name>: <what to substitute> =<substitution>!

Tim Notes:

1. & Process NextCommand on SameLine
2. && Process NextCommand if PrevCommand suceeded
3. || Process NextCommand if PrevCommand failed

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Breaks down to: cont.

```
If
!count! leq !max!                                   - if Count < max then
        (brc32 /Pattern:!name! /FixedName:!pre!x!count!   - Process filename.
                                                      Fixed will Remove name.
                                                      New name = pre (current value = zero)   = 0
                                                      x (literal)                             = x
                                                      count – current value i.e., 1           = 1

                                                      i.e., New Name = 0x1

        Set /a count="count+1")                       - increment count
ELSE                                                 - to reach 'Else', count = max
((Set count=0)&&(Set /a pre="pre+1")))               This is two separate statements using && to run
                                                     one and then when it is successful, run the other.
                                                     - reset count
                                                     - increment pre
```

Contributor Notes:

1. Here are some modifications:
   a. Add this right after the '@echo off' statement to make it easier to change FolderPaths:
      cd/d "YourFolderPath"

   b. Add this right after the current If statement:[3]
      If !pre! equ !max! If !count! equ !max! goto done

      This will halt renaming upon reaching the user's entered max.

   c. Also enter this right before the 'pause >nul':
      :done

Comparison Operators

Operator	Description
EQU	equal to
NEQ	not equal to
LSS	less than
LEQ	less than or equal to
GTR	greater than
GEQ	greater than or equal to
NOT	will negate a condition

Additional String Comparison Operators

==	String comparison for equality
Not ==	String comparison for inequity

Tim Notes:

1. As you can see, BRC is not just used with the RegEx feature. This example shows that the values evaluated for (pre + 'x' + count) are provided as the New Name to 'Section #2: Name' using a Name value = Fixed.

2. To reflect the Poster's original request, use 10 for the value of max.

3. The Contributor's notes mention adding an additional If statement, but this is not needed I find, because the 'Else' clause takes care of handling the max.

4. Removing the /Execute provides a preview of files to be renamed otherwise it will execute immediately. Restore it back to run the BRC.

Forum Contributor: bru

JavaScript

Replace Filenames with Values based on Nested Loops cont.

Here is another JavaScript that will also work:

```
newName=(~~(counter/11.0001))+"x"+(counter-1) % 11
```

Forum Contributor: RegexNinja

A Word about ~~

~ JavaScript Bitwise Not Operator

~~ JavaScript Bitwise Complement. The Result of two bitwise Not operators together on a positive number, is the conversion that is done before the first operator is applied, i.e. converting the value to a signed 32-bit integer number.

It removes everything after the decimal point because the bitwise operators implicitly convert their operands to signed 32-bit integers. This works whether the operands are (floating-point) numbers or strings, and the result is a number.

Bitwise Operators –

Bitwise operators treat their operands as a sequence of 32 bits (Zeros and ones- Binary or Base 2), rather than as decimal, hexadecimal, or octal numbers. For example, the decimal number nine has a binary representation of 1001. Bitwise operators perform their operations on such binary representations, but they return standard JavaScript numerical values.

The following table summarizes JavaScript's bitwise operators:

Operator	Usage	Description
Bitwise AND	a & b	Returns a 1 in each bit position for which the corresponding bits of both operands are 1s.
Bitwise OR	a | b	Returns a 1 in each bit position for which the corresponding bits of either or both operands are 1s.
Bitwise XOR	a ^ b	Returns a 1 in each bit position for which the corresponding bits of either but not both operands are 1s.
Bitwise NOT	~ a	Inverts the bits of its operand.
Left shift	a << b	Shifts a in binary representation b (< 32) bits to the left, shifting in 0s from the right.
Sign-propagating right shift	a >> b	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off.
Zero-fill right shift	a >>> b	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off, and shifting in 0s from the left.

Let's discuss the Bitwise Not operator.

JavaScript

A Word about ~~ cont.

Signed 32-bit integers

The operands of all bitwise operators are converted to signed 32-bit integers in two's complement format, except for zero-fill right shift which results in an unsigned 32-bit integer.

Two's complement format means that a number's negative counterpart (e.g. 5 vs. -5) is all the number's bits inverted (this is the 'bitwise NOT' of the number, or 'ones' complement' of the number) plus one or -(x + 1). Positive numbers do not require these steps.

~ Ones' Complement: -(N+1)

Examples -

~ 0 = -(0+1) = -1
~ 5 = -(5+1) = -6
~-7 = -(-7+1) = 6

~~ Twos' Complement: -(-(N+1)+1)

Examples -

~~ 5 = -(-(5+1)+1) = 5
~~ -3 = -(-(-3+1)+1) = -3

For example, the following encodes the integer 314 in two's complement signed 32-bit Binary Base 2:

00000000000000000000000100111010 ⟸ Original integer value 314

The following encodes the integer -314 in two's complement signed 32-bit Binary Base 2:

11111111111111111111111011000101 ⟸ The following encodes ~314 (~ the ones' complement of 314)
11111111111111111111111011000110 ⟸ The following encodes -314, (~~ the two's complement of 314)

The two's complement guarantees that the left-most bit is 0 when the number is positive and 1 when the number is negative. Thus, this left-most bit is called the 'sign' bit. An integer that has the sign bit is called a 'Signed Integer'. The remaining 31 bits besides the sign bit are used to represent the integer.

Therefore, the maximum 32-bit integer that can be represented is:

(2^31 - 1) = 2147483647

.. while the minimum integer is:

-(2^31) = -2147483648

For integers that fall outside the 32-bit signed integer range, the most significant bits are discarded until the integer falls within range. What this does is convert the integer to a 32-bit signed integer.

Summary:

The numbers -2147483648 and 2147483647 are the minimum and the maximum integers representable through a 32-bit signed number.

JavaScript

A Word about ~~ cont.

Signed 32-bit integers cont.

Here are the 32-bit sequence representations of some important numbers:

0 =>	00000000000000000000000000000000
	The number 0 is the integer that is composed completely of 0 bits.
-1 =>	11111111111111111111111111111111
	The number -1 is the integer that is composed completely of 1 bits.
2147483647 (base 10) =	01111111111111111111111111111111 (base 2)
	The number 2147483647 (hexadecimal representation: 0x7fffffff) is the integer that is composed completely of 1 bits, except the first (left-most) one. This is the signed bit indicating the number is positive.
-2147483648 (base 10) =	10000000000000000000000000000000 (base 2)
	The number -2147483648 (hexadecimal representation: -0x80000000) is the integer that is composed completely of 0 bits, except the first (left-most) one. This is the signed bit indicating the number is negative.

From the above representations, it is evident that:

~ 0 =>	-1
~-1 =>	0
~ 2147483647 =>	-2147483648
~-2147483648 =>	2147483647

Tim Notes:

1. It is used as a faster substitute for Math.floor() but it is more like Math.trunc() because it converts a decimal value (removes the decimal portion by truncating) to an integer and it is prone to errors on larger numbers. Not recommended for general use.

2. This differs from Math.floor in that negative numbers are rounded toward zero, not away from it. The floor function, always rounds down to the next lower integer, regardless of whether the number is positive or negative.

Source Material this section: Guffa, PleaseStand, CroMagnon, Stackoverflow.com, //blog.logrocket.com, //developer.mozilla.org

JavaScript

Remove the second repetition of filename from the end

541-200-6_web_ad541-200-6_web_ad.jpg	to	541-200-6_web_ad.jpg
OApatchOApatch.png	to	OApatch.png
oldbookoldbook.jpg	to	oldbook.jpg

Observations:

1. Take the length of the filename, divide by 2.
2. Cut the filename at answer+1.

```
newName = name.substr(name.length/2);
```

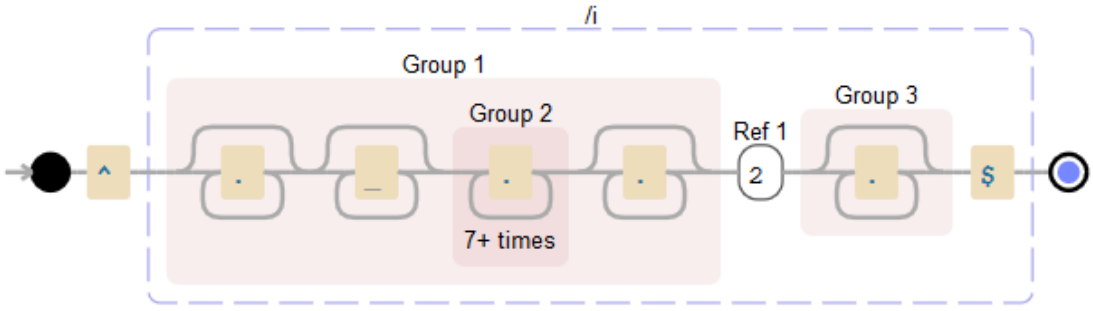
Name	New Name
oldbookoldbook.jpg	oldbook.jpg
OApatchOApatch.png	OApatch.png
541-200-6_web_ad541-200-6_web_ad.jpg	541-200-6_web_ad.jpg

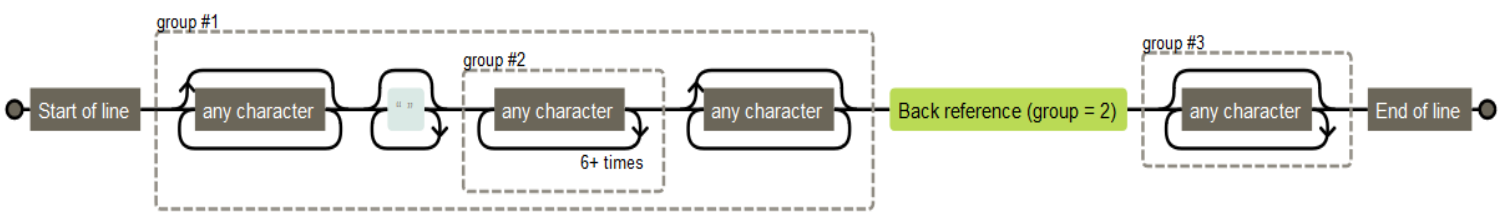
RegEx Equivalent:

1. It removes the 1st-repeating word/phrase of 8-or-more characters (including <spaces>) anywhere in filename.
2. It's case-insensitive.

Match: ^(?i)(.*? *(.{7,}).*?)\2(.*)$
Replace: \1\3

	New Name
oldbookoldbook.jpg	oldbook.jpg
OApatchOApatch.png	OApatch.png
541-200-6_web_ad541-200-6_web_ad.jpg	541-200-6_web_ad.jpg





JavaScript

Remove the second repetition of filename from the end cont.

RegEx Equivalent: cont.

Tim Notes:

1. For second Visualization, had to remove the 'Case Insensitive' '(?i)' because it was not supported by the program that produces the diagram. Everything else is diagrammed correctly.

2. For the first group of samples, I had to change the minimum value to 7 in order to match because i.e.,

OApatchOApatch.png	OApatch is seven characters long before it repeats.
oldbookoldbook.jpg	oldbook is seven characters long before it repeats
541-200-6_web_ad541-200-6_web_ad.jpg	541-200-6_web_ad is fifteen characters long before it repeats

Thus minimum = 7

New sample base:

one <space> is always preserved if matching phrases mid-name like 3rd sample

PDF for dummies ... PDF for Dummies is all you need	to	PDF for dummies ... is all you need
PDF for Dummies… Pdf for dummies.. Another one	to	PDF for Dummies.. Another one
AAA any repeated phrase BBB Any Repeated Phrase CCC	to	AAA any repeated text BBB CCC
AAASomeVeryLongWordBBBsomeverylongwordCCC	to	AAASomeVeryLongWordBBBCCC

Match: ^(?i)(.*? *(.{8,}).*?)\2(.*)$
Replace: \1\3

	New Name
PDF for Dummies... Pdf for dummies.. Another one	PDF for Dummies... .. Another one
AAA any repeated phrase BBB Any Repeated Phrase CCC	AAA any repeated phrase BBB CCC
PDF for dummies ... PDF for Dummies is all you need	PDF for dummies ... is all you need
AAASomeVeryLongWordBBBsomeverylongwordCCC	AAASomeVeryLongWordBBBCCC

Sample: Minimum 7

```
.........................Start·Length····
Match 1: OApatchOApatch      0      14
Group 1: OApatch             0       7
Group 2: OApatch             0       7
Group 3:                    14       0
```

Sample: Minimum 8

```
.....................................................
Match 1: PDF for dummies ... PDF for Dummies is all you need
Group 1: PDF for dummies ...
Group 2: PDF for dummies
Group 3: is all you need
```

Tim Notes:

1. Same RegEx. Just changed out the 7 for an 8 in the '(.{8})' so no change in Visualizations noteworthy.

Forum Contributor: Admin, bru

JavaScript

Remove the second repetition of filename from the end cont.

RegEx Equivalent: cont.

Using: AAA any repeated phrase BBB Any Repeated Phrase CCC

Match: ^(?i)(.*? *(.{8,}).*?)\2(.*)$

```
Match 1: AAA any repeated phrase BBB Any Repeated Phrase CCC
Group 1: AAA any repeated phrase BBB
Group 2: any repeated phrase
Group 3: CCC
```

Where:

\1 = The first part of the string up until the repeated segment
AAA any repeated phrase BBB <space>

\2 = the repeated segment
any repeated phrase <space>

\3 = any text past the repeated segment that is not repeated
CCC

Breaks down (Case Insensitive switch not shown):

	Capture Group 1			Capture Group 3	
		Capture Group 2			
anchor					anchor
^	(.*? <space> *	(.{8,})	.*?)	\2	(.*) $
	AAA <space>	any repeated phrase <space>	BBB	any repeated phrase <space>	CCC

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2. (?i)	Case Insensitive	
3. .	Match against any character	Capture Group 1 = A
4. *	Make it Greedy	Capture Group 1 = <entire string>
5. ?	But not too Greedy	Capture Group 1 = BOL
6. <space>	Match against a <space> character Matches against the <space> after 'AAA'	Capture Group 1 = AAA <space>
7. *	Make it Greedy	Capture Group 1 = <entire string>
8. .	Match against any character already at EOL	Capture Group 1 = Unchanged Capture Group 2 = null
9. {8,}	{8 characters Minimum, Maximum} – Maximum not specified so 8 characters or more Already at EOL	All Capture Groups = Unchanged
10. .	Match against any character Backtracks. Matches against the last 'C' at the end of the string	Changes Capture Group 2 = AAA any repeated phrase BBB Any Repeated <space> Phrase CC Capture Group 1 = Unchanged

JavaScript

Remove the second repetition of filename from the end cont.

RegEx Equivalent: cont.

Using: AAA any repeated phrase BBB Any Repeated Phrase CCC

Match: ^(?i)(.*? *(.{8,}).*?)\2(.*)$

Tim Analysis: Cont.

11. *	Make it Greedy	Capture Group 1, 2 = <entire string>
12. ?	But not too Greedy Current position = EOL	All Capture Groups = Unchanged
13. \2	References back reference \2 (Capture Group 2) = 'Any Repeated Phrase <space>. (Not Captured)	Changes Capture Group2 = any repeated phrase <space> Changes Capture Group 1 = AAA any repeated phrase BBB <space>

Tim Notes:

This is where it gets interesting. Up until now, the current position is EOL, but in step 13, by specifying Capture Group 2, it begins to backtrack as expected, but in doing so, it reevaluates all of the previous Capture Groups, again as expected. But to me, the interesting point to be made here is that Capture Group 2 had the value changed to the <entire string> in step 11, and now that it is backtracking, Capture Group 2 is also being reevaluated.

The RegEx Engine backtracks to BOL, and begins to move forward. The first subexpression of Capture Group 1 restores back to the 'AAA <space>. Capture Group 2, nested inside Capture Group 1, reevaluates to 'any repeated phrase <space>' (again moving forward) and the second sub-expression of Capture Group 1 evaluates to the 'BBB <space>' giving a value of, 'AAA any repeated phrase BBB <space> to Capture Group 1 because of the nesting.

The \2 back reference, also begins evaluating. The value of Capture Group 2 = 'any repeated phrase'. Because it is Case Insensitive, moving forward, it matches at the second repetition, 'Any Repeated Phrase <space>'. It is Not Captured. This is the repeated phrase that will not be included in New Name.

14. .*	Match against the remainder of the string a. This will include any text past the repeated segment that is not repeated.	Capture Group 3 = CCC
15. $	Must be at EOL (End Of Line)	

Here is Regex Buddy analysis:

Match: ^(?i)(.*? *(.{8,}).*?)\2(.*)$

```
.............................................................
Match 1: AAA any repeated phrase BBB Any Repeated Phrase CCC
Group 1: AAA any repeated phrase BBB
Group 2: any repeated phrase
Group 3: CCC
```

JavaScript

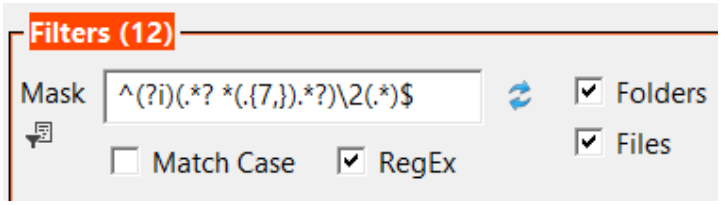
Remove the second repetition of filename from the end cont.

The contributor adds this caveat:

Warning:

1. If you decide to lower the 8-char-minimum with many files, recommend to:

   a. 'Section #12: Filters' –

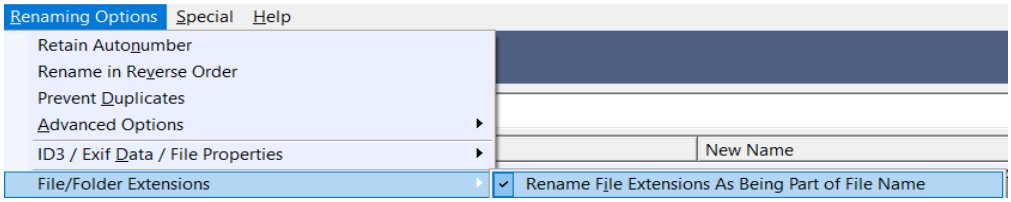


1. Mask: ^(?i)(.*? *(.{7,}).*?)\2(.*)$
2. RegEx: Enable

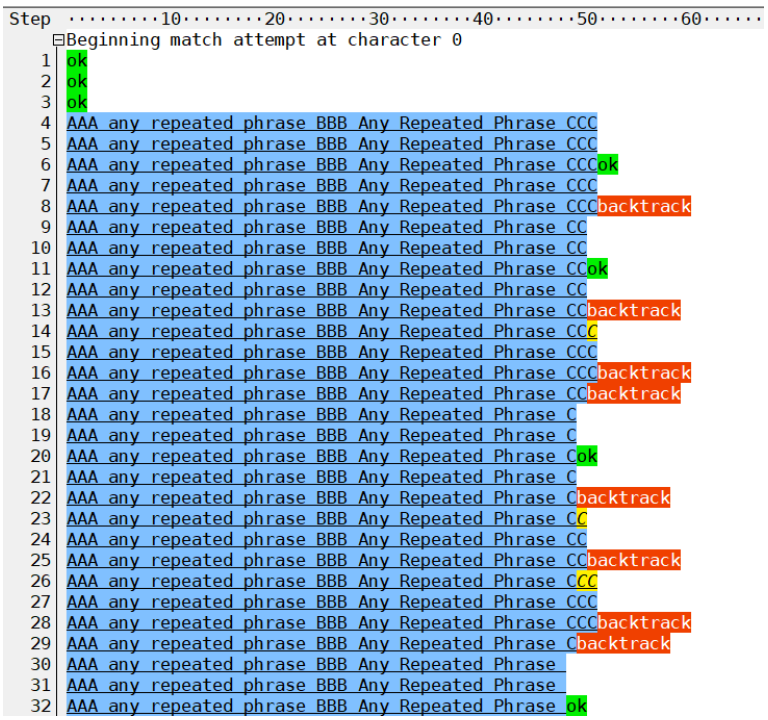
This will filter out the file list to only display those files that have repeating segments to avoid selecting files that would inadvertently be renamed.

   b. Click the blue Filter Refresh Button to the right of the Mask data field to refresh the Content Pane.

2. Since dot characters are in the samples, don't forget to enable, 'Rename File Extension As Being Part of File Name':



Forum Contributor: bru



Nice technique, but because of the almost constant backtracking will be slow processing a lot of files.

JavaScript

Change to Title case from 40th character

sample:

Change to Title case from 40th character to the end of file name
to
Change to Title case from 40th character To The End Of File Name

String =
Internet Explorer 11 will only open if I right click and then select Run As Administrator from windows server 2012.png

Observations:

1. To replace <space> (Any Lowercase) with <space> (Uppercase) beginning at the filename's 40th-character:

JavaScript:

```
newName=name.substr(0,40)+name.substr(40).replace(/( [a-z])/g, function(up){return up.toUpperCase()})
```

Name:
Internet Explorer 11 will only open if I right click and then select Run As Administrator from windows server 2012.png

New Name:
Internet Explorer 11 will only open if I Right Click And Then Select Run As Administrator From Windows Server 2012.png

BRC: Command line:

brc32 /MinLen:41 /Insert:"|:40" /ReplaceCI:" |: | " /CutFirstN:39 /ChangeCase:T /PasteAtStart /ReplaceCI:"|:" /TidyDS /Execute

```
Processing Folder H:\Level1\
Filename Internet Explorer 11 will only open if I right click and then select Run As Administrator from windows server 2
012.png would be renamed to Internet Explorer 11 will only open if I Right Click And Then Select Run As Administrator Fr
om Windows Server 2012.png
Filename SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv requires no changes

H:\Level1>
```

JavaScript

Change to Title case from 40th character cont.

Contributor Notes:

Does the same thing as the JavaScript but in a different way, in this order:

1. Insert | as the 40th-character in the string.

2. IF '|' was inserted after <space>, then add another <space> after '|'
   <space> <pipe character> to <space> <pipe character> <space>

3. Cut the first 39-characters of the string.

4. Apply Title case-conversion (against the remaining string which is made up of the 40th-character to EOL)

5. Paste the first 39-characters back into filename at the start of the string, thus restoring the original content of the string together with the modified portion.

6. Remove the ' |' character (had to be removed anyway, otherwise rename would fail because this character is invalid in filenames).

7. Convert any double spaces back to a single space (possibly created during step 2).

JavaScript

Change to Title case from 40th character cont.

Understanding BRC

BRC (Bulk Rename Command) whether it is in the form of a batch file or used as a command line tool, utilizes the functionality that is also available through the GUI (Graphic User Interface) version of the program, called Bulk Rename Utility or BRU for short.

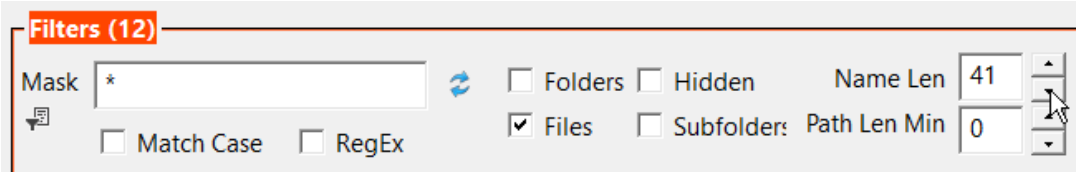
brc32 /MinLen:41 /Insert:"|:40" /ReplaceCI:" |: | " /CutFirstN:39 /ChangeCase:T /PasteAtStart /ReplaceCI:"|:" /TidyDS /Execute

I will take apart the BRC above and show you how it relates to the BRU program. It should be 'generally' stated that because BRC has adavantages over BRU in several ways, that even if you were to 'set' all of the functions I will demonstrate, it may be highly unlikely you will get the same results as the BRC. This is on a case by case basis, so let's see if the same results *can* be obtained.

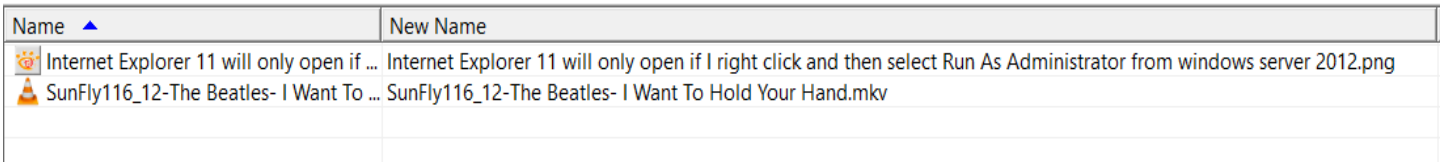
1. MinLen:41 - sets Name Len to 41

Definition: /MINLEN - Include only those files whose name is nnn characters or more

This is the 'Section #12: Filters'



Mask: *
Name Len: 41



JavaScript

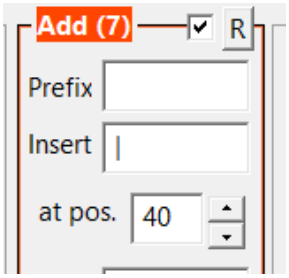
Change to Title case from 40th character cont.

Understanding BRC cont.

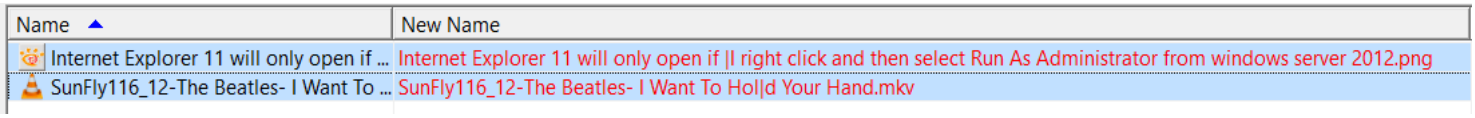
2. Insert:"|:40" - Insert <pipe character> at position 40

Definition: /INSERT:Text:Pos - Insert text at the specific position in the name. If the position is before the start (or after the end) of the name then it will be treated as a prefix or suffix accordingly

This is the 'Section #7:Add'



Insert: |
at pos: 40



Tim Notes:

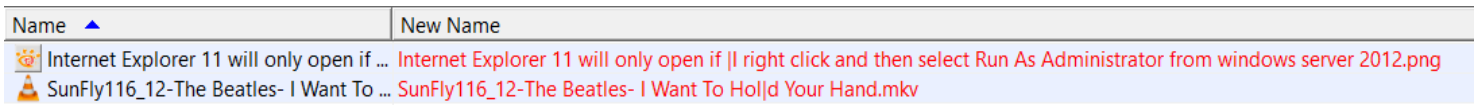
1. Why the Red and not Green? Red = Invalid – there is something wrong with the name even though it wants to match. In this case it is the <pipe character>. It is an illegal character in a filename.

3. ReplaceCI:" |: | " - Replace <space> <pipe character> with <space> <pipe character> <space>

Definition: /REPLACECI:<Replace:With> - Perform case-insensitive (the 'CI') search and replace

This is 'Section #3: Replace'

Replace: <space> |
With: <space> | <space>



JavaScript

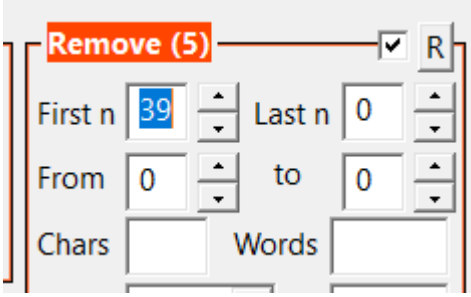
Change to Title case from 40th character cont.

Understanding BRC cont.

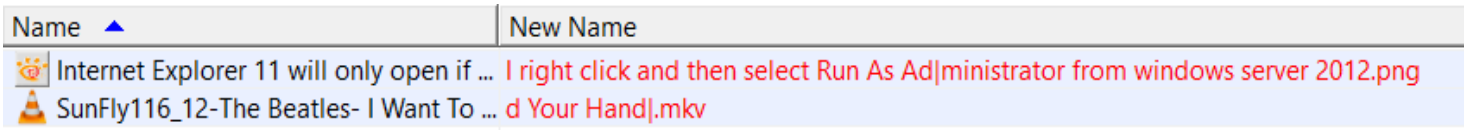
4. CutFirstN:39 - Remove first 39 characters and copy to temporary buffer for later retrieval

Definition: /CUTFIRSTN:n - Remove the first n characters from the name and store them into a buffer for later paste.

This is 'Section #5: Remove'



First n: 39

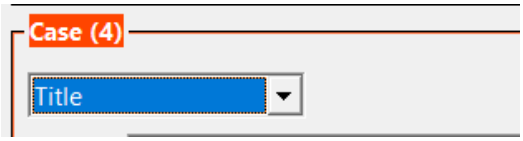


5. ChangeCase:T - Change case to Title case

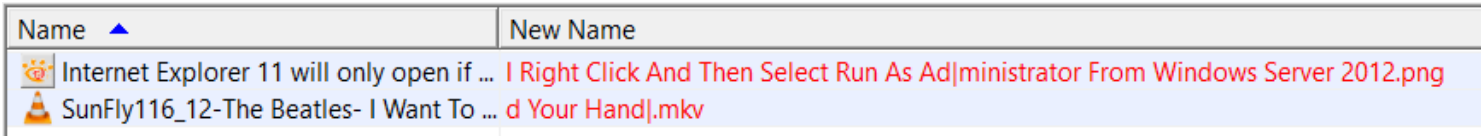
Definition: /CHANGECASE:Mode - Change the case of the filename. Mode is as follows:

L -Change to Lower case
U -Change to Upper case
T -Change to Title case
S -Change to Sentence case

This is 'Section #4: Case'



Case: Title



JavaScript

Change to Title case from 40th character cont.

Understanding BRC cont.

Tim Notes:

1. At this point I realize that the renaming using the GUI has to be done in more than one run. The first run would be performed now before step 6, because step 6 will require the 'Section #7: Add' function again.

2. Problem is that <pipe character> is still in place which prevents the renaming process.
   a. Work-around – would be to use a character that would be acceptable in place of the <pipe character> back in step 2 since it is only used as a marker within the string, I will use an exclamation point or something similar that is not too common but still a valid character so that I can continue.

Name	New Name
Internet Explorer 11 will only open if ...	I Right Click And Then Select Run As Ad!ministrator From Windows Server 2012.png
SunFly116_12-The Beatles- I Want To ...	d Your Hand!.mkv

3. Definite advantage of BRC over BRU is that multiple operations can be performed in one step.

4. Another advantage, and this is significant, is that BRC is NOT restricted by the Order of Evaluation. For example, in BRU you could never have 'Section #7: Add' performed prior to 'Section #3: Replace' as it does here.

After Renaming:

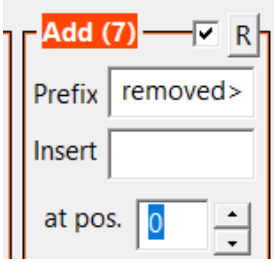
Name	New Name
I Right Click And Then Select Run As Ad!ministra...	I Right Click And Then Select Run As Ad!ministrator From Windows Server 2012.png

Run # 2

6. PasteAtStart - Paste back the 39 characters that were removed in 'Section #5: Remove'

Definition: /PASTEATSTART - Paste the buffer (created by a previous cut/copy command) to the start.

This is the 'Section #7: Add' using the <removed> tag.



Prefix: <removed>
at pos: 0

JavaScript

Change to Title case from 40th character cont.

Understanding BRC cont.

Tim Notes:

1. Problem – the buffer data is gone once you rename. Using the <removed> tag it would seem, has to be done within the same run as 'Section #5: Remove'. This is therefore impossible to do using the GUI.

3. Another problem is that BRU *is* restricted to the Order of Evaluation, unlike BRC apparently, and therefore 'Section #4: Case' cannot be performed after 'Section #7: Add'.

2. From here, realizing that BRU cannot provide the same outcome as the BRC script, I will continue to demonstrate just the BRU functions referenced by the script rather than trying to apply them to their conclusion.

7. ReplaceCI:"|:" Replace the <pipe character> with null

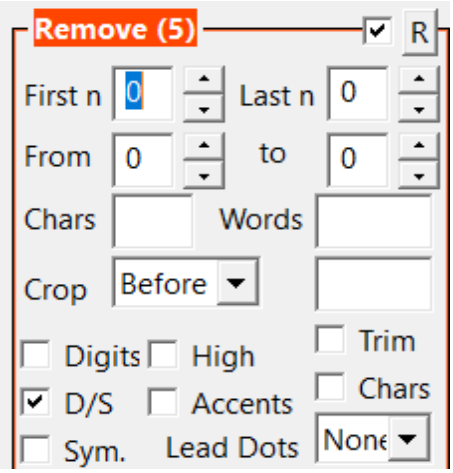
This is the 'Section #3: Replace'

Replace: |
With: ""

8. TidyDS - Remove any possible <double spaces> that may have inadvertently occurred in Step 2.

Definition: /TIDYDS - This will replace all occurrences of two <spaces> with a <single space>

This is the 'Section #5: Remove'



D/S: Enable

Forum Contributor: RegexNinja

JavaScript

Renumber in Groups of Five

Observations:

1. Add a prefix to a series of files but in groups of five.
2. The first five would be prefixed with "01", the next five with "02"

```
newName='0'+Math.ceil(counter/5)+'--'+name
```

Name	New Name
iPhone by Apple	01--iPhone by
1	01--1
10	01--10
100867	01--100867
106	01--106
5	02--5
8259438 some text	02--8259438 some
ABC-123-45	02--ABC-123-45
ABC_DEF_GHI-123-45	02--ABC_DEF_GHI-123-45
Bahá'í Muñoz	02--Bahá'í
CoffeeAndTea	03--CoffeeAndTea
InThe4HrsISpentResearchingThisIC...	03--InThe4HrsISpentResearchingThisICouldHaveTypedThemOut..
Longer_File_name (2016_01_30 07_...	03--Longer_File_name (2016_01_30 07_16_32
My File v.1.2.345	03--My File
my.File2.v1.2.345	03--my.File2.v1.2.345
nc_017733	04--nc_017733
ThisIsReally&Annoying$&.whateve...	04--ThisIsReally&Annoying$&.whatever.pdf
xxx xxx-xxx.txt	04--xxx
xxx_xxxx-xxxxx-xxx	04--xxx_xxxx-xxxxx-xxx

Forum Contributor: RegexNinja

JavaScript

Remove (e##) from File Name

RandomName1(e25)RandomName2.txt	to	RandomName1RandomName2.txt
(e85)RandomName1.txt	to	RandomName1.txt
(e86)RandomName2.txt	to	RandomName2.txt

Observations:

1. Remove the " (e ##)', where ## are two random numbers
2. Can appear anywhere in string.
3. Can appear multiple times in string.

```
newName=name.replace(/\(e\d{2}\)/g, '')
```

	New Name
RandomName1(e25)RandomName2.txt	RandomName1RandomName2.txt
(e85)RandomName1.txt	RandomName1.txt
(e86)RandomName2.txt	RandomName2.txt

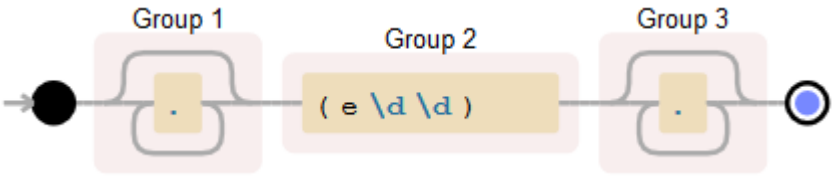
Forum Contributors: therube, RegexNinja

JavaScript

Remove (e##) from File Name cont.

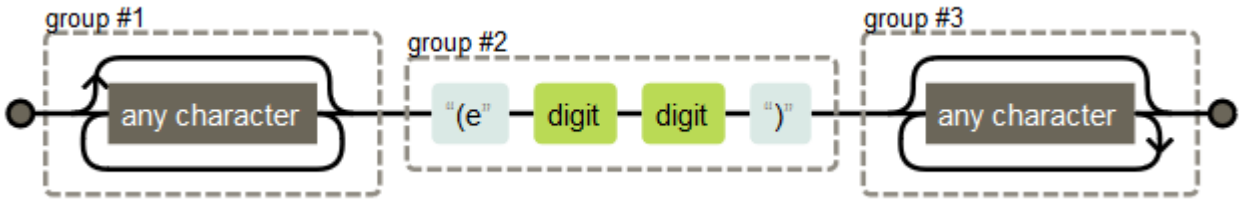
RandomName1(e25)RandomName2.txt	to	RandomName1RandomName2.txt
(e85)RandomName1.txt	to	RandomName1.txt
(e86)RandomName2.txt	to	RandomName2.txt

RegEx Equivalent



Match: (.*?)(\(e\d\d\))(.*)
Replace: \1\3

	New Name
RandomName1(e25)RandomName2.txt	RandomName1RandomName2.txt
(e85)RandomName1.txt	RandomName1.txt
(e86)RandomName2.txt	RandomName2.txt



Samples

```
                          Start  Length
Match 1: (e85)RandomName1     0      16
Group 1:                      0       0
Group 2: (e85)                0       5
Group 3: RandomName1          5      11
```

```
Match 1: RandomName1(e25)RandomName2
Group 1: RandomName1
Group 2: (e25)
Group 3: RandomName2
```

Using: (e85)RandomName1.txt

Where:

\1 = null \2 = (e85) \3 = RandomName1

Tim Analysis:

1..	Match against any character	Capture Group 1 = (
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy	Capture Group 1 = BOL
4. \(	Match against literal left parenthesis	Capture Group 2 = (
		Capture Group 1 = null (remains at BOL)
5. e	Match against string character 'e'	Capture Group 2 = (e
6. \d\d	Match against 2 numeric digits	Capture Group 2 = (e85
7. \)	Match against literal right parenthesis	Capture Group 2 = (e85)
8. .*	Match against remaining string	Capture Group 3 = RandomName1

JavaScript

Add Incrementing Counter within String

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

Tim Contribution –

```
function test() {                          // Create a function called 'test' that when run, would:
  a=name.split("_")                        // Set a = 'str' split into array (using _ as delimiter)
  a[4]="AZ"+("00000"+counter).slice(-4)    // Set a's 5th-element to: AZ(0000+counter) cropped to Last 4 …
                                           //  .. Characters
  newName=a.join("_")                      // Set NewName =  a's elements rejoined with _ as their separator
 return newName}

test(name);
```

Name	New Name
John_q_Adams_01-01-1900_null_null_null	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

Modified code based on information from several parties including Michael, RegexNinja

JavaScript

Add Incrementing Counter within String cont.

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

A discussion using .toLowerCase() and .toUpperCase() with arrays

Someone had suggested this:

```
var str = name;

function test(str) {
  a = str.split("_");
  b = a[4].toUpperCase();
  c = a[5].toUpperCase();
  if (b === c) {
    a[4] = "AZ" + ("00000" + (counter)).slice(-5);
    newName = a.join("_");
  return newName;
  } else if (b.substr("-4", 4) === c) {
    a[5] = "null";
    newName = a.join("_");
  return newName;
  } else {
    newName = a.join("_");
  return newName;
  }
}

test(str);
```

This uses the methods, .toUppercase() and .toLowerCase() on the array, 'a'.

Which failed.

The simple answer is that these functions are string methods and won't work with arrays unless:

1. The array is a string (text) array. The elements of the array are therefore sub-strings.
2. These sub-strings are extracted and then converted using the functions, and placed back inside a new array.

This example used split() which performs a similar function to map() (for numerals only). It breaks the array into its substrings based on the value used to indicate where to perform the 'split' (the delimiter). In this example it was split wherever an underscore character appeared in the string.

JavaScript

Add Incrementing Counter within String cont.

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

Additional Samples added:

john 1990 abc.abc.abc.txt def 2010 try.xyz Index Headers for book.txt again 1978 456.xyz

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

So you *can* case-convert sub-strings in an array. All elements in an array created by split() are converted to strings, even against names like 1234, but the problem became when string methods like .toLowerCase were applied against 'undefined' elements in an array.

What do I mean by this?

What this implies is, if you have string that does not include the underscore character, then this creates an array with one element, a[0], because a[0] will always contain the full string if split() failed to match against the underscore.

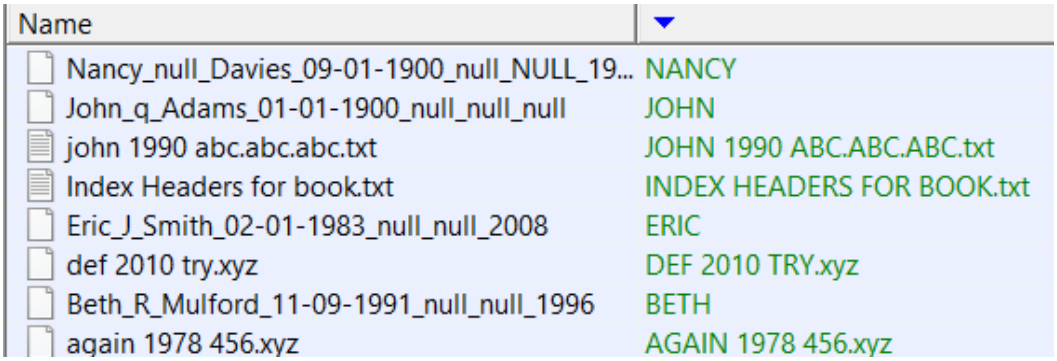
In the previous script,

'str' is the filename converted to an array.
The array is then 'split' into a new array, 'a'
If str does not contain the underscore character, testing for individual elements a[4] and a[5] will fail because only a[0] contains a value.

To see the first element of the array, 'a', enter this in the JavaScript code entry form.

```
a=name.split("_")
newName=a[0].toUpperCase()
```

Results:



JavaScript

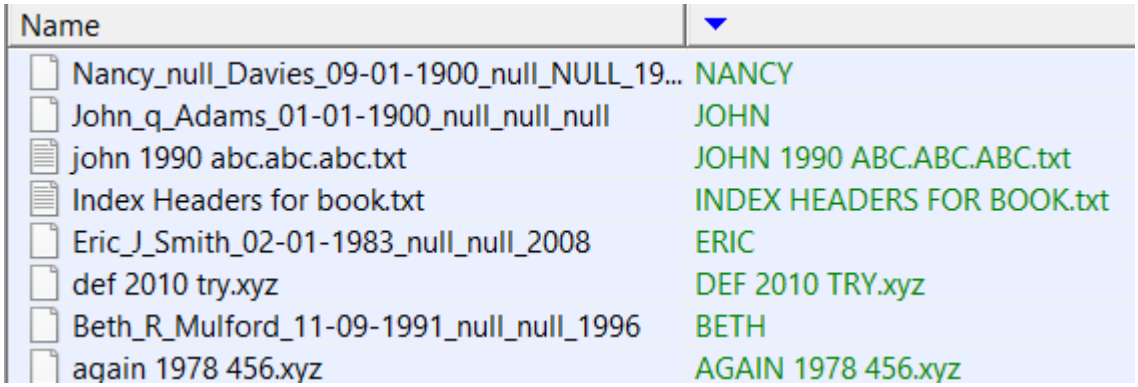
Add Incrementing Counter within String cont.

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

Additional Samples added:

john 1990 abc.abc.abc.txt def 2010 try.xyz Index Headers for book.txt again 1978 456.xyz

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.



The following strings in this sample group have the underscore character and will split:

John_q_Adams_01-01-1900_null_null_null	a[0] = JOHN
Rick_null_Jones_02-01-1900_null_null_2000	a[0] = RICK
Eric_J_Smith_02-01-1983_null_null_2008	a[0] = ERIC
Beth_R_Mulford_11-09-1991_null_null_1996	a[0] = BETH
Nancy_null_Davies_09-01-1900_null_NULL_1963	a[0] = NANCY

The following strings in this sample group do not have an underscore character and will not split:

john 1990 abc.abc.abc.txt	a[0] = JOHN 1990 ABC.ABC.ABC.txt
def 2010 try.xyz	a[0] = DEF 2010 TRY.xyz
Index Headers for book.txt	a[0] = INDEX HEADERS FOR BOOK.txt
again 1978 456.xyz	a[0] = AGAIN 1978 456.xyz

Tim Notes:

1. Notice that, regardless, the function .toUpperCase did in fact work even if it was just for a[0].

JavaScript

Add Incrementing Counter within String cont.

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

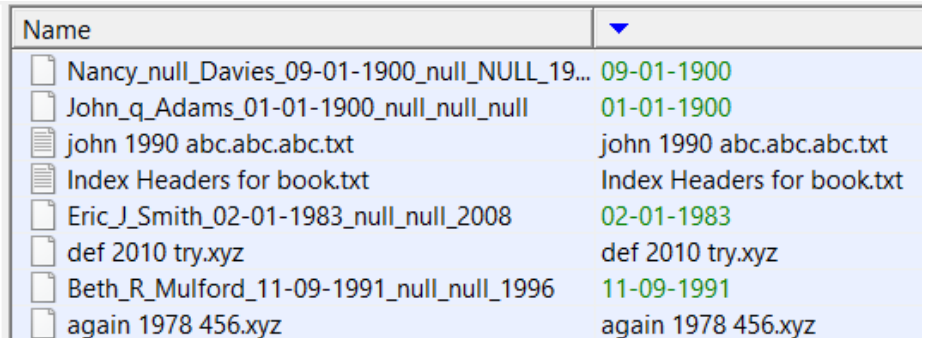
Additional Samples added:

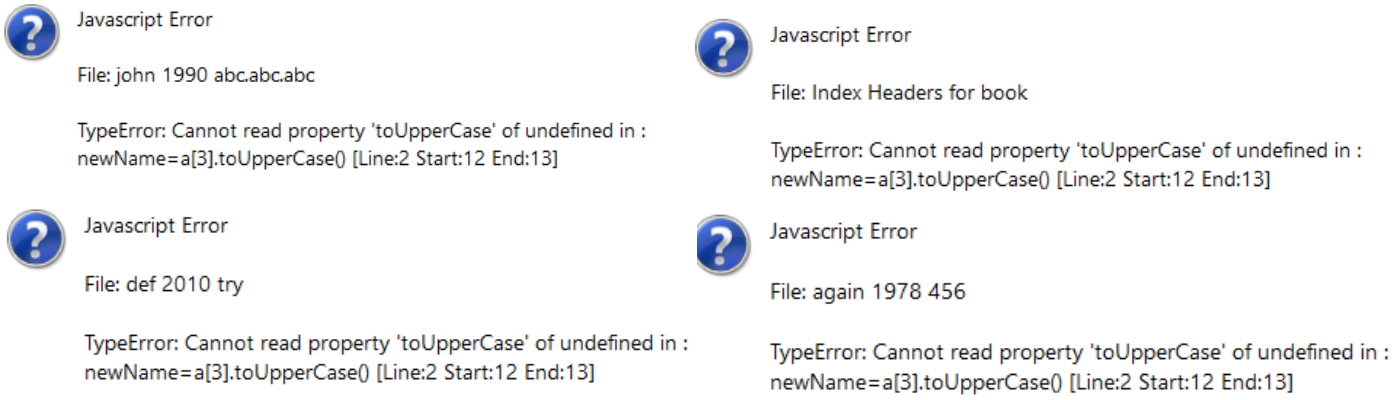
john 1990 abc.abc.abc.txt def 2010 try.xyz Index Headers for book.txt again 1978 456.xyz

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

Now use a[3] instead

```
a=name.split("_")
newName=a[3].toUpperCase()
```





The Errors are a result of when .toUpperCase has no string element value to convert. In each of these strings there are no underscore characters, therefore only a[0] has a value. When a[3] is tested, it has a value of 'undefined' and fails.

JavaScript

Add Incrementing Counter within String cont.

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

Additional Samples added:

john 1990 abc.abc.abc.txt def 2010 try.xyz Index Headers for book.txt again 1978 456.xyz

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

Removing .toUpperCase() from the script removes the error. You can, however, still reference non-existent element-numbers. These elements will contain the value, 'undefined', ie., a[3] = 'undefined' when str did not have a fourth sub-string value.

If you do a comparison of one against another:

```
if (a[4]===a[5])
```

You are in a sense comparing:

```
('undefined'==='undefined') …
```

.. which evaluates to true.

With this in mind, You can eliminate errors by pre-checking for the 'undefined' value of an array subscript:

```
if (typeof a[4] !== 'undefined')
```

and using this script:

```
function test() {                                   // Create a function called 'test' that when run, would:
  a=name.split("_")                                 // Set a = OrigName split into array (using _ as delimiter)
  if (typeof a[4] !== 'undefined') {                // IF array has a 5th-element, then:
  if (typeof a[5] !== 'undefined') {                //  IF array has a 6th-element, then:
  if (a[4].toLowerCase()===a[5].toLowerCase()) {    // IF 5th-element(lowercased) = 6th-element(lowercased),
                                                    // ..  then:
  a[4]="AZ"+("0000"+counter).slice(-4)              // Replace 5th-element with AZ(0000+counter), cropped
                                                    // ..  to Last 4 Characters
  newName=a.join("_")                               // Set NewName = a's elements rejoined with _ as their
                                                    // .. separator
return newName}}}}                                  // Process NewName
test(name)                                          // Run the function test against the variable name
```

The script uses .toLowercase() instead of .toUppercase it should be noted, but this makes no difference in the discussion.

JavaScript

Add Incrementing Counter within String cont.

John_q_Adams_01-01-1900_null_null_null	to	John_q_Adams_01-01-1900_AZ0001_null_null
Rick_null_Jones_02-01-1900_null_null_2000	to	Rick_null_Jones_02-01-1900_AZ0002_null_2000
Eric_J_Smith_02-01-1983_null_null_2008	to	Eric_J_Smith_02-01-1983_AZ0003_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	to	Beth_R_Mulford_11-09-1991_AZ0004_null_1996
Nancy_null_Davies_09-01-1900_null_NULL_1963	to	Nancy_null_Davies_09-01-1900_AZ0005_NULL_1963

Additional Samples added:

john 1990 abc.abc.abc.txt def 2010 try.xyz Index Headers for book.txt again 1978 456.xyz

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

There is a problem though. Because this script doesn't prevent these filenames from being evaluated and incrementing counter's value, the AZ value will not be in sequential order:

Name	
Nancy_null_Davies_09-01-1900_null_NULL_19...	Nancy_null_Davies_09-01-1900_AZ0001_NULL_1963
John_q_Adams_01-01-1900_null_null_null	John_q_Adams_01-01-1900_AZ0002_null_null
john 1990 abc.abc.abc.txt	john 1990 abc.abc.abc.txt
Index Headers for book.txt	Index Headers for book.txt
Eric_J_Smith_02-01-1983_null_null_2008	Eric_J_Smith_02-01-1983_AZ0005_null_2008
def 2010 try.xyz	def 2010 try.xyz
Beth_R_Mulford_11-09-1991_null_null_1996	Beth_R_Mulford_11-09-1991_AZ0007_null_1996
again 1978 456.xyz	again 1978 456.xyz

Here you can see that using the comparison against the 'undefined' element prevented those file names that did not have underscore characters in the string from participating in the renaming process. They have, however, incremented counter's value so you end up with:

Nancy_null_Davies_09-01-1900_AZ0001_NULL_1963	Counter = 1
John_q_Adams_01-01-1900_AZ0002_null_null	Counter = 2
john 1990 abc.abc.abc.txt	Counter = 3
Index Headers for book.txt	Counter = 4
Eric_J_Smith_02-01-1983_AZ0005_null_2008	Counter = 5
def 2010 try.xyz	Counter = 6
Beth_R_Mulford_11-09-1991_AZ0007_null_1996	Counter = 7
again 1978 456.xyz	Counter = 8

The only way to avoid this is to be more selective in your filenames or to use a filter

JavaScript

Add Incrementing Counter within String cont.

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

In each of these scripts, it is the 5th and 6th element of the array that is tested.

Remember that the array starts at zero so in the scripts, a[4] refers to the fifth element and a[5] refers to the sixth element.

The element is defined as each part of the string that is split into sub-strings by the split() function using the underscore as the delimiter.

This means for example,

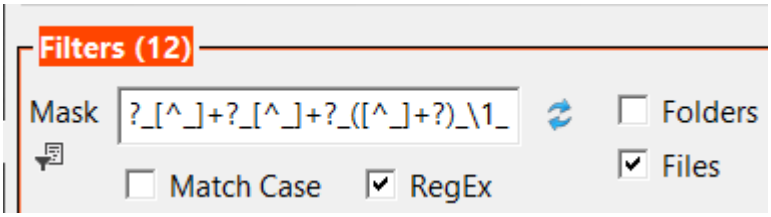
John_q_Adams_01-01-1900_null_null_null

Is split into:

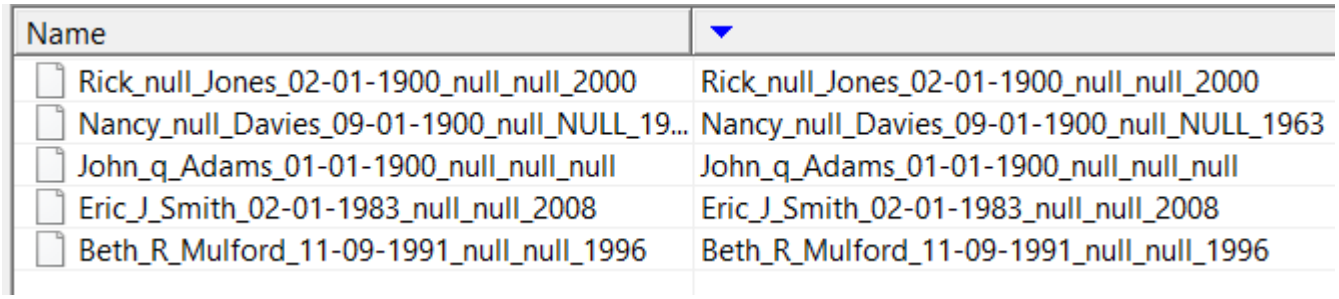
Sub-Strings	1	2	3	4	5	6	7
Array Subscripts							
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
	John	q	Adams	01-01-1900	null	null	null

To qualify under the script, the string must contain a 5th (a[4]) and 6th (a[5]) element.

This filter will do just that. It weeds out only those strings that will qualify for the script and prevent those that do not.



Mask: ^(?i)[^_]+?_[^_]+?_[^_]+?_[^_]+?_([^_]+?)_\1_



Use the Filter Refresh blue button next to Mask or press F5 to see the results.

JavaScript

Add Incrementing Counter within String cont.

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

using: John_q_Adams_01-01-1900_null_null_null

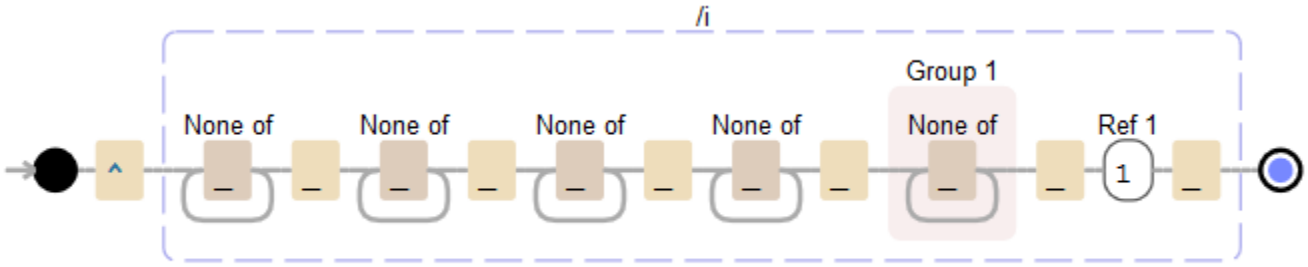
Filter: ^(?i)[^_]+?_[^_]+?_[^_]+?_[^_]+?_([^_]+?)_\1_

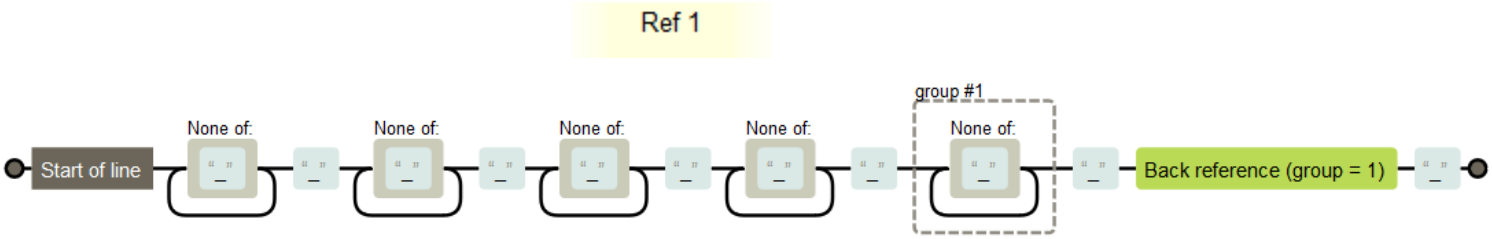
Tim Analysis:

1. (?) Make the filter case insensitive
2. [^_] Match against a Negated Class consisting of an underscore character. Match against anything but an underscore.
3. + Make it Greedy
4. ? But not too Greedy.
5. _ Match against the underscore character.

Repeat this sequence for 5 times to locate strings that have at least five underscore characters. Capture the last group as Capture Group 1.

6. _ Match against the underscore character
7. \1 Use backreference 1 to match against the value of Capture Group 1. Capture group 1 = string value of 'null' (not the null result meaning no value) so looking to match gainst string value 'null'
8. _ Match against the underscore character. This guarantees at least 1-character (besides underscore) in the 5th /6th elements, & keeps the numbering sequential.





Tim Notes:

1. Had to remove case insensitivity for the second visualization program because it is not supported.

JavaScript

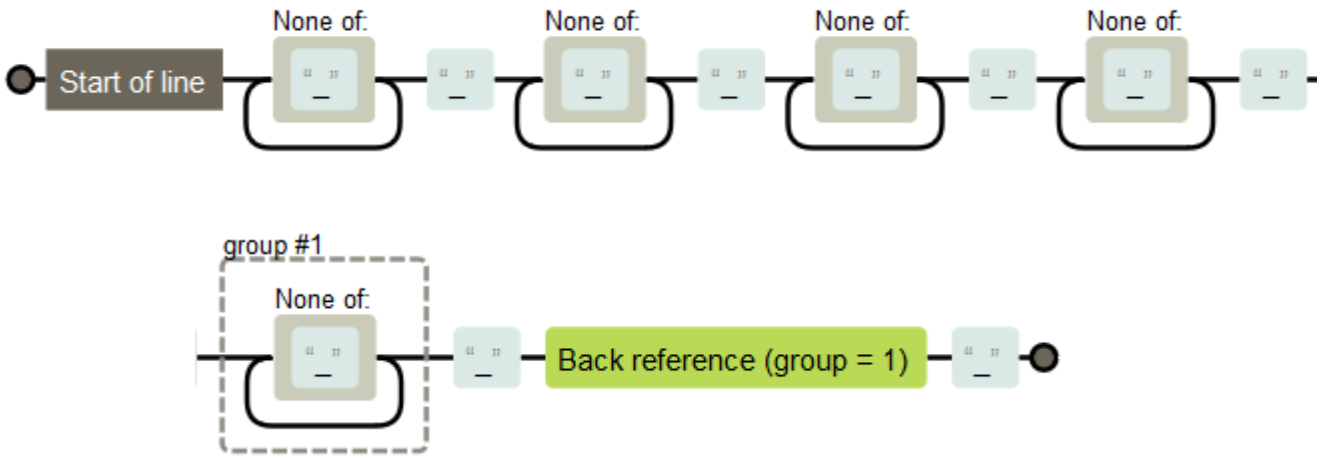
Add Incrementing Counter within String cont.

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

using: John_q_Adams_01-01-1900_null_null_null

Filter: ^(?i)[^_]+?_[^_]+?_[^_]+?_[^_]+?_([^_]+?)_\1_

Expanded:



^(?i)

[^_]+? _ [^_]+? _ [^_]+? _ [^_]+? _ ([^_]+?) _ \1 _

John _ q _ Adams _ 01-01-1900 _ null _ null _ null

Tim Notes:

1. There is no need to test for the 7th sub-string.

JavaScript

Add Incrementing Counter within String cont.

A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

With the filter in place, use this script (ignore any errors if you click Test):

```
function test() {                                    // Create a function called 'test' that when run, would:
  a=name.split("_")                                  // Set a = OrigName split into array (using _ as delimiter)
  if (a[4].toLowerCase()===a[5].toLowerCase()) {     // IF 5th-element(lowercased) = 6th-element(lowercased),
                                                     // …then:
  a[4]="AZ"+("0000"+counter).slice(-4)               // Replace a's 5th-element with AZ(0000+counter),
                                                     // … cropped to Last 4 Characters
  newName=a.join("_")                                // Set NewName = a's elements rejoined with _ as their
                                                     // …separator
return newName}}                                     // Process NewName

test(name)                                           // Run the function test against the variable name
```

Name	
Rick_null_Jones_02-01-1900_null_null_2000	Rick_null_Jones_02-01-1900_AZ0001_null_2000
Nancy_null_Davies_09-01-1900_null_NULL_1963	Nancy_null_Davies_09-01-1900_AZ0002_NULL_1963
John_q_Adams_01-01-1900_null_null_null	John_q_Adams_01-01-1900_AZ0003_null_null
Eric_J_Smith_02-01-1983_null_null_2008	Eric_J_Smith_02-01-1983_AZ0004_null_2008
Beth_R_Mulford_11-09-1991_null_null_1996	Beth_R_Mulford_11-09-1991_AZ0005_null_1996

Forum Contributors: Michael, RegexNinja

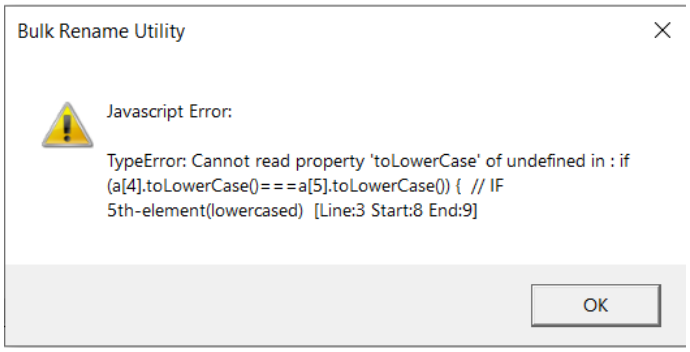
JavaScript

Add Incrementing Counter within String cont.

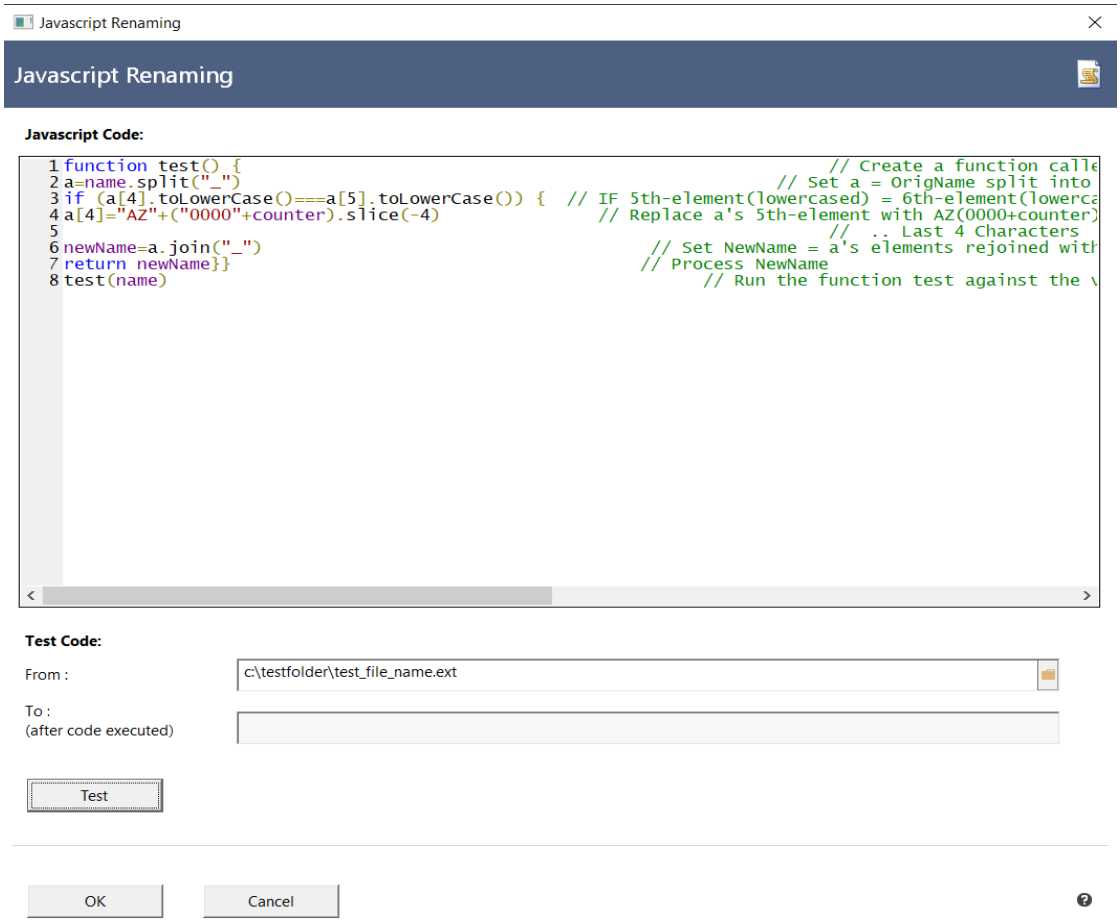
A discussion using .toLowerCase() and .toUpperCase() with arrays cont.

Tim Notes:

1. Test will always come up with an error because the testing is based on the syntax of the JavaScript against known values.
2. .toLowerCase is normally not applied against an array so it provides a type mismatch error. If you click OK it will run fine without errors.



3. This is important. Now you know that you can **run the JavaScript without having to first Test the syntax**.



JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space>

1.a.b.c.d.2.3.4.5.6	to	1.abcd.2.3.4.5.6
12.34.a.b.c.56.78	to	12.34.abc.56.78
a.b.c.d.e.f1.g.h.i	to	abcdef1.ghi
Home.Photostudio.Repack.v16.0	to	HomePhotostudioRepackv16.0
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	to	MicrosoftNETCoreDesktopRuntimev3.1.4_x64
Movavi.Photo.Editor.Repack.v4.4.0	to	MovaviPhotoEditorRepackv4.4.0
NCH.PhotoPad.v6.13	to	NCHPhotoPadv6.13
Nitro.Pro.13.15.1.282.Ent_x64	to	NitroPro.13.15.1.282.Ent_x64

```
newName = name.replace(/([^\d])\.(?![\d])/g, '$1')
```

Name	New Name
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	MicrosoftNETCoreDesktopRuntimev3.1.4_x64
Movavi.Photo.Editor.Repack.v4.4.0	MovaviPhotoEditorRepackv4.4.0
NCH.PhotoPad.v6.13	NCHPhotoPadv6.13
Home.Photostudio.Repack.v16.0	HomePhotostudioRepackv16.0
Nitro.Pro.13.15.1.282.Ent_x64	NitroPro.13.15.1.282.Ent_x64
1.a.b.c.d.2.3.4.5.6	1.abcd.2.3.4.5.6
12.34.a.b.c.56.78	12.34.abc.56.78
a.b.c.d.e.f1.g.h.i	abcdef1.gh.i

If you want <spaces> to replace <dots> use:

```
newName = name.replace(/([^\d])\.(?![\d])/g, '$1 ')
```

Name	New Name
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	Microsoft NET Core Desktop Runtime v3.1.4_x64
Movavi.Photo.Editor.Repack.v4.4.0	Movavi Photo Editor Repack v4.4.0
NCH.PhotoPad.v6.13	NCH PhotoPad v6.13
Home.Photostudio.Repack.v16.0	Home Photostudio Repack v16.0
Nitro.Pro.13.15.1.282.Ent_x64	Nitro Pro.13.15.1.282.Ent_x64
1.a.b.c.d.2.3.4.5.6	1.a b c d.2.3.4.5.6
12.34.a.b.c.56.78	12.34.a b c.56.78
a.b.c.d.e.f1.g.h.i	a b c d e f1.g h.i

See Tim Analysis for more information on this.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Tim Analysis:

If you want the <space> character as indicated in the request, just a simple change:

```
newName = name.replace(/([^\d])\.(?![\d])/g, '$1 ')
```

Tim Notes:

1. <space> was added after $1
2. The $1 is another way of stating \1 for the Replacement String.
3. The /g is the global switch meaning match everything in the string - unlike BRU RegEx which is limited to the first match only.
4. The JavaScript is basically running a RegEx using a global switch. The result of which is replaced (the replace() method) in the original filename passed to the program via the 'name' constant (applies onlty to BRU JavaScript) and assigned to a new string, newName used for the Renaming.

This would look like this in RegEx:

Match: ([^\d])\.(?![\d])
Replace: \1

Of course this wouldn't work in RegEx (no global switch) - I am just illustrating what the JavaScript is doing.

1. It matches against any character that is not a numeric digit.
   a. The [^\d] is a Negated class meaning don't match against a numeric digit - represented by the \d
2. … until it hits a <dot>
   a. The \.
3. Then it looks ahead to see that there are *not* any numeric digits after the <dot>.
   a. The (?![\d]) is a negative lookahead
4. Because of the global switch (/g) , it continues to try and match against this pattern until the string is exhausted and no more matches can be made.
5. There is only one Capture Group –

   Capture Group 1 = ([^\d])

   The other is called a 'Non-Marking Group' and does not capture.

6. So it matches letters with dots and ignore numeric digits. The dot is removed in a sense because it is not captured.
   a. The \. is the dot and it is not in parentheses so it is not captured.

7. You end up with a collection of letters together and numeric digits with preserved dots. The <space> is added in the Replacement String '$1 <space>'

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Here it replaces ANY string-of-dots that dont have a number in front of them with a <space>

1.a.b.c.d.2.3.4.5.6	to	1.a b c d 2.3.4.5.6
12.34.a.b.c.56.78	to	12.34.a b c 56.78
a.b.c.d.e.f1.g.h.i	to	a b c d e f1.g h i
Home.Photostudio.Repack.v16.0	to	Home Photostudio Repack v16.0
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	to	Microsoft NET Core Desktop Runtime v3.1.4_x64
Movavi.Photo.Editor.Repack.v4.4.0	to	Movavi Photo Editor Repack v4.4.0
NCH.PhotoPad.v6.13	to	NCH PhotoPad v6.13
Nitro.Pro.13.15.1.282.Ent_x64	to	Nitro Pro 13.15.1.282.Ent_x64
Driver.Booster.Repack.7.4.0.730	to	Driver Booster Repack 7.4.0.730

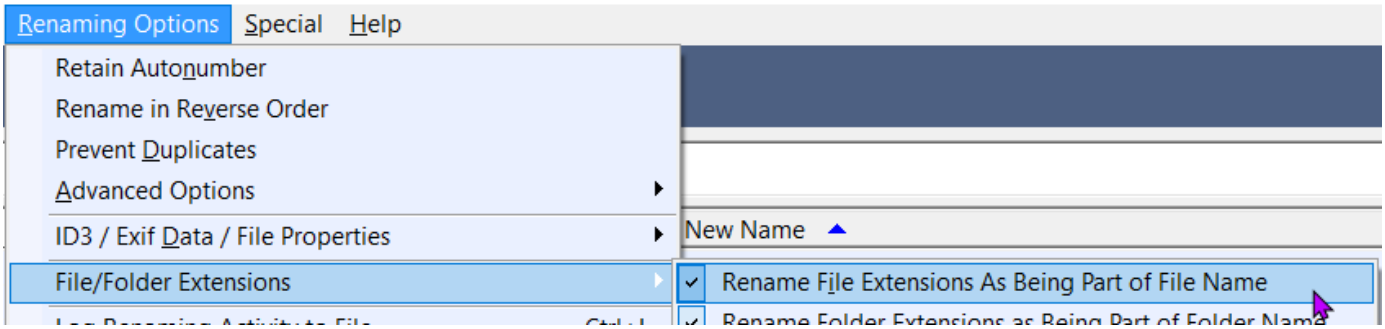
```
newName = name.replace(/([^\d])\.+/g, '$1 ')
```

Name	New Name
1.a.b.c.d.2.3.4.5.6	1.a b c d 2.3.4.5.6
12.34.a.b.c.56.78	12.34.a b c 56.78
a.b.c.d.e.f1.g.h.i	a b c d e f1.g h i
Driver.Booster.Repack.7.4.0.730	Driver Booster Repack 7.4.0.730
Home.Photostudio.Repack.v16.0	Home Photostudio Repack v16.0
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	Microsoft NET Core Desktop Runtime v3.1.4_x64
Movavi.Photo.Editor.Repack.v4.4.0	Movavi Photo Editor Repack v4.4.0
NCH.PhotoPad.v6.13	NCH PhotoPad v6.13
Nitro.Pro.13.15.1.282.Ent_x64	Nitro Pro 13.15.1.282.Ent_x64

Tim Notes:

1. In order to achieve the a b c d e f1.g h i and not a b c d e f1.g h.i

You have to enable 'Rename File Extensions As Being Part of the File Name' from the File/Folder Extensions sub-menu of the Renaming Options Menu (this is also known as 'Ignore File Extension' – named from previous version of BRU).



JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

New Sample:

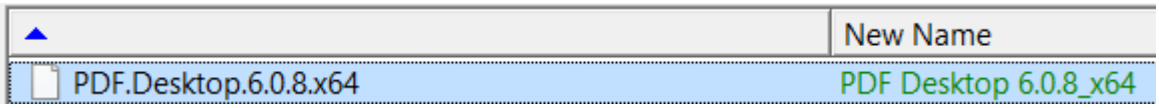
PDF.Desktop.6.0.8.x64 to PDF Desktop 6.0.8_x64

Observation:

1. Remove dots between last number of file version and x64
2. x64/x86 is at the end of a filename
3. Still requires 'Ignore File Extension' enabled – see Tim Notes previous page.

Here's an example to replace it with an underscore:

```
a = name.replace(/([^\d])\.+/g, '$1 ')

newName = a.replace(/^(.*\d)\.(x64|x86)$/,'$1_$2')
```



Tim Notes:

1. This only affects those files that end with either 'x86' or 'x64'

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Additional Samples:

PDF Architect 7.1.14.4969.x64	to	PDF Architect 7.1.14.4969_x64
PDF.Desktop.6.0.8.x64	to	PDF Desktop 6.0.8_x64
Nitro.Pro.13.15.1.282.Ent_x64	to	Nitro Pro 13.15.1.282 Ent_x64
Nuance_PowerPDF_Advanced_3.00.6439_x64	to	?

Further Observations:

1. Remove dots anywhere but not between numbers or in other words remove dots anywhere with the exception of those between numbers of file version.

```
a = name.replace(/^(.*)[-\. ](x64|x86)(.*)$/,'$1_$2$3')
b = a.replace(/([^ \d])\.+/g, '$1 ')

newName = b.replace(/\.+([^ \d])/g, ' $1')
```

	New Name
PDF Architect 7.1.14.4969.x64	PDF Architect 7.1.14.4969_x64
Nuance_PowerPDF_Advanced_3.00.6439_x64	Nuance_PowerPDF_Advanced_3.00.6439_x64
Nitro.Pro.13.15.1.282.Ent_x64	Nitro Pro 13.15.1.282 Ent_x64
PDF.Desktop.6.0.8.x64	PDF Desktop 6.0.8_x64

Does't match against 'Nuance_PowerPDF_Advanced_3.00.6439_x64' because pattern not matched – changed from previous pattern. Doesn't adhere to Observations current or previous. This sample not only has an underscore at the end of the string, but used throughout, where normally the pattern would match against <space> or <dot> characters. Fault is with original Poster who keeps 'moving the Goalpost'.

It's my Goalpost. I'm going home.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

1.a.b.c.d.2.3.4.5.6
Home.Photostudio.Repack.v16.0
Nitro.Pro.13.15.1.282.Ent_x64
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64
12.34.a.b.c.56.78
Movavi.Photo.Editor.Repack.v4.4.0
Driver.Booster.Repack.7.4.0.730
a.b.c.d.e.f1.g.h.i
NCH.PhotoPad.v6.13

Additional samples:

PDF_Architect....7.1.14.4969.x64
20200519022023.x64
12.34.abc.56.78
Nuance_PowerPDF_Advanced_3.00.6439_x64
202005190-x64.1234
20200519022023 x64
PDF.Desktop.6.0.8.x64
firefox.2020-01-08_065639
202005190-x64.1234

This one addresses dots, <spaces> and underscores

```
a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1|$2$3')
b = a.replace(/([^\d])[\.]+/g, '$1 ')
c = b.replace(/[\.]+([^\d])/g, ' $1')
d = c.replace(/_+/g,' ')
e = d.replace(/ +/g, ' ')

newName = e.replace('|','_')
```

Name	
PDF_Architect....7.1.14.4969.x64	PDF Architect 7.1.14.4969_x64
PDF.Desktop.6.0.8.x64	PDF Desktop 6.0.8_x64
PDF Architect 7.1.14.4969.x64	PDF Architect 7.1.14.4969_x64
Nuance_PowerPDF_Advanced_3.00.6439_x64	Nuance PowerPDF Advanced 3.00.6439_x64
Nitro.Pro.13.15.1.282.Ent_x64	Nitro Pro 13.15.1.282 Ent_x64
NCH.PhotoPad.v6.13	NCH PhotoPad v6.13
Movavi.Photo.Editor.Repack.v4.4.0	Movavi Photo Editor Repack v4.4.0
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	Microsoft NET Core Desktop Runtime v3.1.4_x64
Home.Photostudio.Repack.v16.0	Home Photostudio Repack v16.0
firefox.2020-01-08_065639	firefox 2020-01-08 065639
Driver.Booster.Repack.7.4.0.730	Driver Booster Repack 7.4.0.730
a.b.c.d.e.f1.g.h.i	a b c d e f1 g h i
20200519022023.x64	20200519022023_x64
20200519022023 x64	20200519022023_x64
202005190-x64.1234	202005190_x64.1234
12.34.abc.56.78	12.34 abc 56.78
12.34.a.b.c.56.78	12.34 a b c 56.78
1.a.b.c.d.2.3.4.5.6	1 a b c d 2.3.4.5.6

Forum Contributor: RegexNinja

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

1.a.b.c.d.2.3.4.5.6
Home.Photostudio.Repack.v16.0
Nitro.Pro.13.15.1.282.Ent_x64
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64
12.34.a.b.c.56.78
Movavi.Photo.Editor.Repack.v4.4.0
Driver.Booster.Repack.7.4.0.730
a.b.c.d.e.f1.g.h.i
NCH.PhotoPad.v6.13

Additional samples:

PDF_Architect....7.1.14.4969.x64
20200519022023.x64
12.34.abc.56.78
Nuance_PowerPDF_Advanced_3.00.6439_x64
202005190-x64.1234
20200519022023 x64
PDF.Desktop.6.0.8.x64
firefox.2020-01-08_065639
202005190-x64.1234

This one addresses dots, <spaces> and underscores cont.

Here I will 'attempt' to provide some analysis but if you thought it was difficult to analyze some of these BRC examples in the book, it is really difficult to analyze *these*, primarily because the same technique used in the BRC coding is also used here (same Contributor) and his/her technique has become that much more evolved. So bear with me – this is as close as I can get.

First you have to understand the JavaScript Replace method.

Purpose: The replace() method replaces a specified value with another value in a string. The replace() method searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced.

Syntax

string.replace(searchvalue, newvalue)

Where:

searchvalue	Required. The value, or regular expression, that will be replaced by the new value
newvalue	Required. The value to replace the search value with

The regEx is the search value, the new value is the replace string and the string is the constant, name, representing the filename . The new string will be 'a'

Where:
name = BRU constant that holds the value of the string being processed (the filename)

```
a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1|$2$3')
```

Tim Notes:

1. This JavaScript line only affects those files that end with either 'x86' or 'x64'

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent_x64

This one addresses dots, <spaces> and underscores cont.

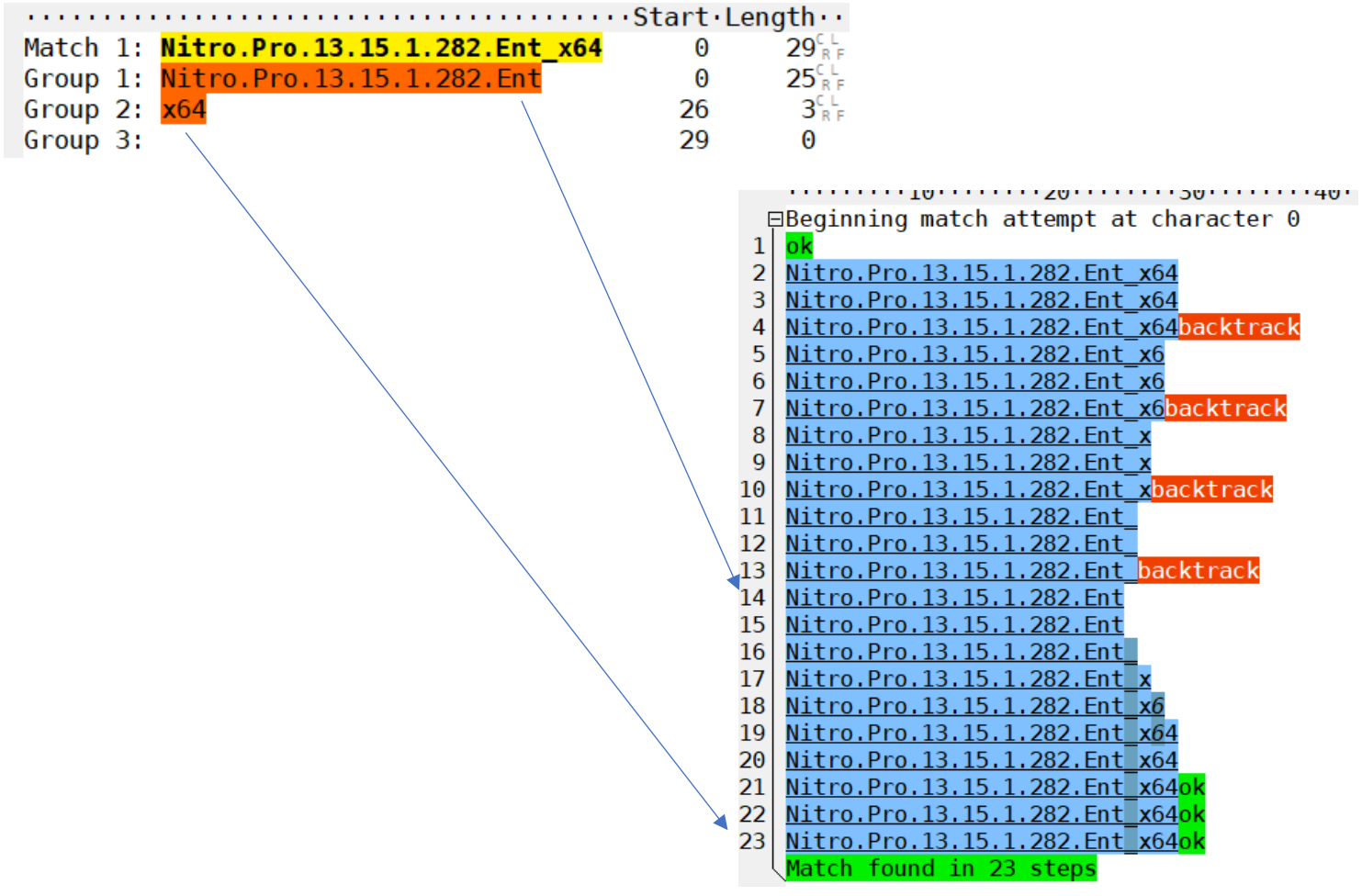
Just to give you an idea that the RegEx used in a JavaScript is really no different than in BRU – with some exceptions – like the support of a Global Switch, here is the Regex Buddy debug output:

1st RegEx:

Match: ^(.*)[-_\.](x64|x86)(.*)$
Replace: \1|\2\3

Nitro.Pro.13.15.1.282.Ent_x64

Match: ^(.*)[-_\.](x64|x86)(.*)$



JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent_x64

This one addresses dots, <spaces> and underscores cont.

You must perform the Rename function so that the next RegEx will run properly.

1st RegEx: cont.

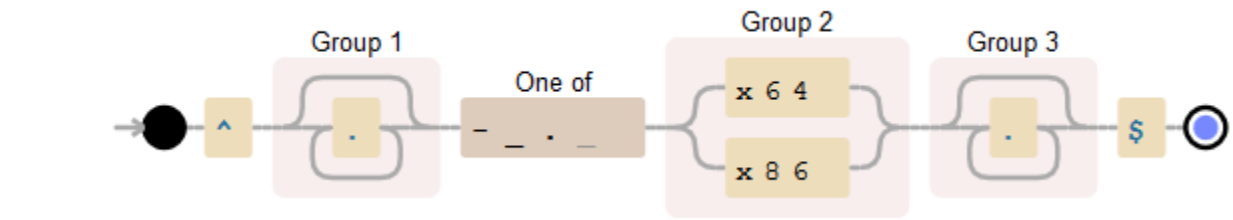
Match: ^(.*)[-_\.](x64|x86)(.*)$
Replace: \1|\2\3

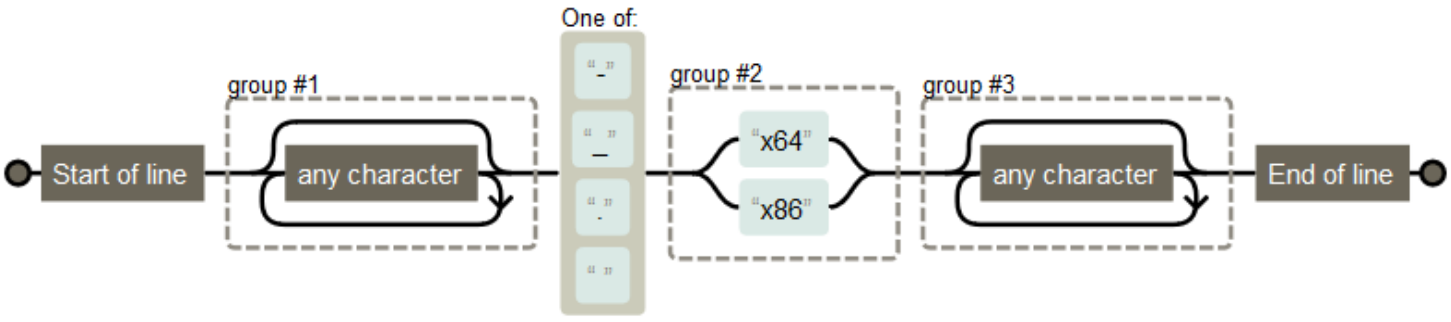
Nitro.Pro.13.15.1.282.Ent | x64

Changed to exclamation point for the purpose of analysis
(BRU can't process the pipe character- illegal character in a Windows Filename).

Replace: \1!\2\3

Nitro.Pro.13.15.1.282.Ent ! x64





JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent_x64

This one addresses dots, <spaces> and underscores cont.

1st RegEx: cont.

Match: ^(.*)[-_\.](x64|x86)(.*)$

Where:

\1 = Nitro.Pro.13.15.1.282.Ent \2 = x64 \3 = null

Tim Analysis:

1. ^	Start at beginning of string (BOL)	
2..	Match against any character	Capture Group 1 = N
3. *	Make it Greedy Current position = EOL	Capture Group 1 = <entire string>
4. [-_\.]	Match against a class consisting of a (Not Captured) a. <hyphen> b. an underscore c. a literal <dot> Backtracks to match against the undrscore after 'Ent'	Changes Capture Group 1 = Nitro.Pro.13.15.1.282.Ent
5. x64 | x86	Alternate. a. Match against string 'x64' = 'x64' = True or b. Match against string 'x86'	Capture Group 2 = x64
6. .*	Match against remaining string	Capture Group 3 = null

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent_x64

This one addresses dots, <spaces> and underscores cont.

1st RegEx: cont.

Match: ^(.*)[-_\.](x64|x86)(.*)$

Where:

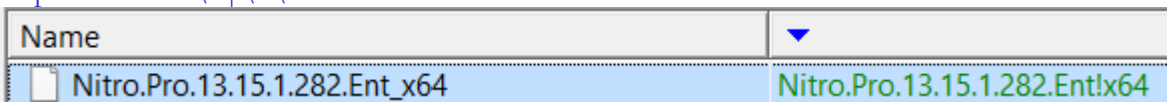
\1 = Nitro.Pro.13.15.1.282.Ent \2 = x64 \3 = null

The Replace string of the RegEx, not the JavaScript replace() looks like this:

```
'$1|$2$3'
```

But in BRU it would look like this:

Replace: \1|\2\3



The whole concept of this RegEx is to isolate the underscore, <hyphen> or literal <dot> using a separator – the 'pipe' character, starting from the right of the string, that replaces that character..

The replace() method is broken down to this:

```
new string = current filename . replace ( <RegEx to perform search> , <new value>)
a = name . replace (/ ^(.*)[-_\. ](x64|x86)(.*)$ / , '$1|$2$3' )
```

The new value is applied to 'name', overwriting any previous value but because the original string doesn't change, this value is written out to 'a' (couldn't change a 'constant' variable anyway).

Result (using original 'pipe' character):

Nitro.Pro.13.15.1.282.Ent_x64 to Nitro.Pro.13.15.1.282.Ent|x64

a = Nitro.Pro.13.15.1.282.Ent|x64

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

2nd RegEx:

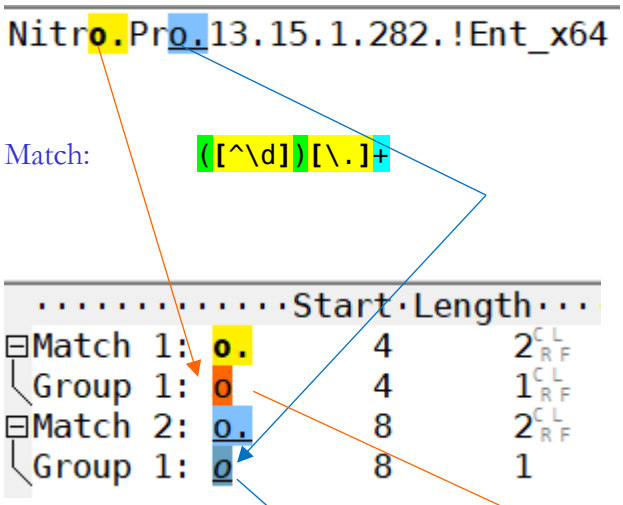
Match: ([^\d])[\.]+
Replace: \1

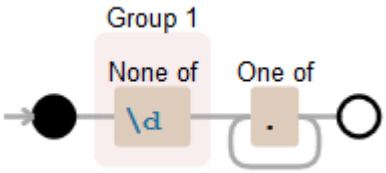
The next line of JavaScript becomes a lot harder:

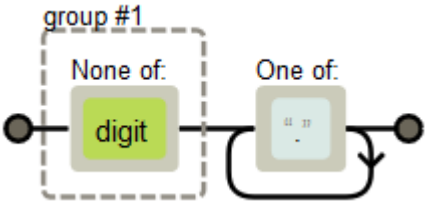
```
b = a.replace(/([^\d])[\.]+/g, '$1 ')
```

This is because of the Global Switch, something that both BRU and BRC do not yet support. The Global Switch allows all matches to take place, not just the one that BRU currently supports.

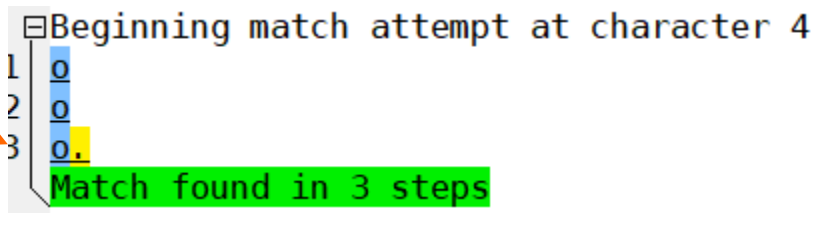
In Regex Buddy (Global is always used) this can be clearly seen -

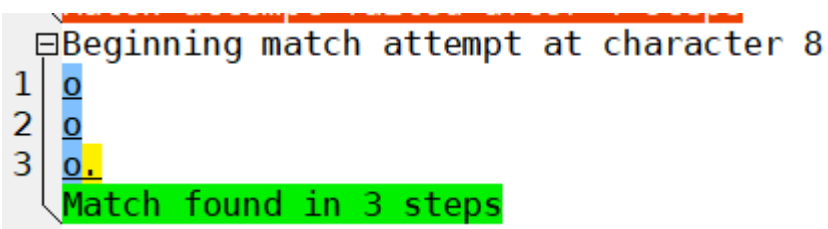






Two matches are performed at one time.





Tim Notes:

1. Neither Visualization program above supports the visualization of the Global Switch.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

2nd RegEx: cont.

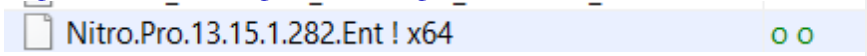
Match: ([^\d])[\.]+
Replace: \1 <space>

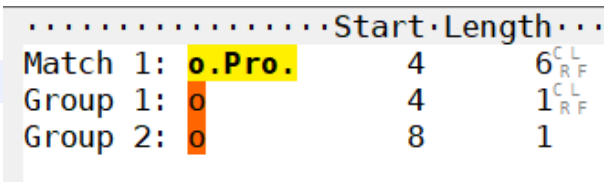
What this next RegEx does is to isolate and replace one or more <dot> characters with a single <space>, unless that <dot> character is between two numeric digits, i.e., '1.282', will not be affected, but will affect '2.Ent'.

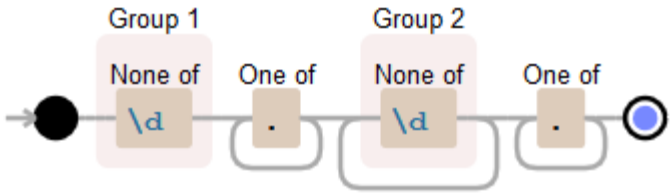
It's hard to visualize because there is no step by step walkthrough in JavaScript, at least that BRU supports anyway, but I will attempt to explain:

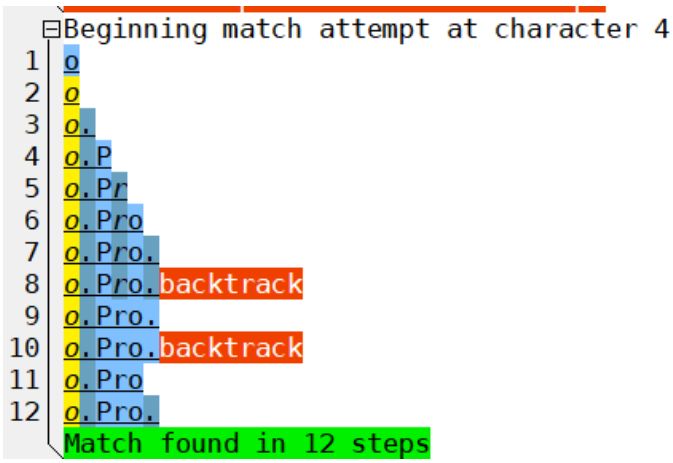
If I take the RegEx above and add to it just a little bit of code so that it will at least do two Capure Groups, you might get a better idea.

Match: ([^\d])[\.]+([^\d])+[\.]+
Replace: \1<space> \2 <space>



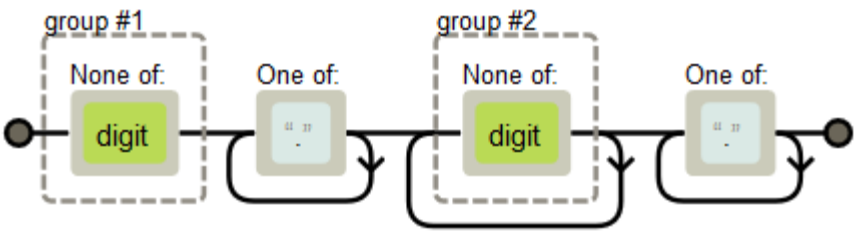






Match: ([^\d])[\.]+([^\d])+[\.]+

Nitro.Pro.13.15.1.282.Ent|x64



JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent|x64

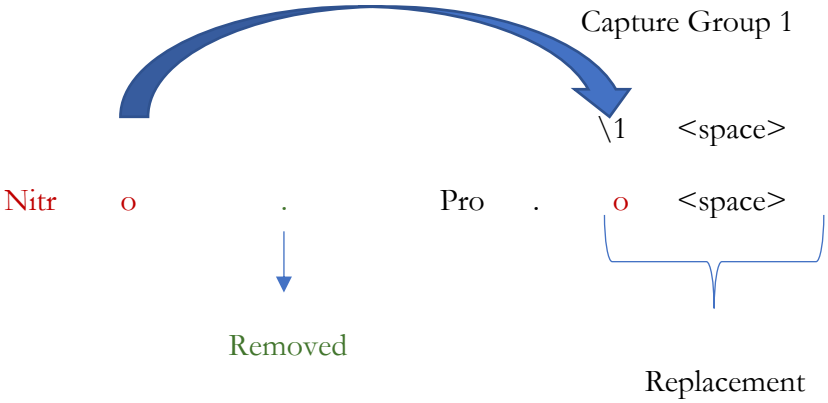
This one addresses dots, <spaces> and underscores cont.

2nd RegEx: cont.

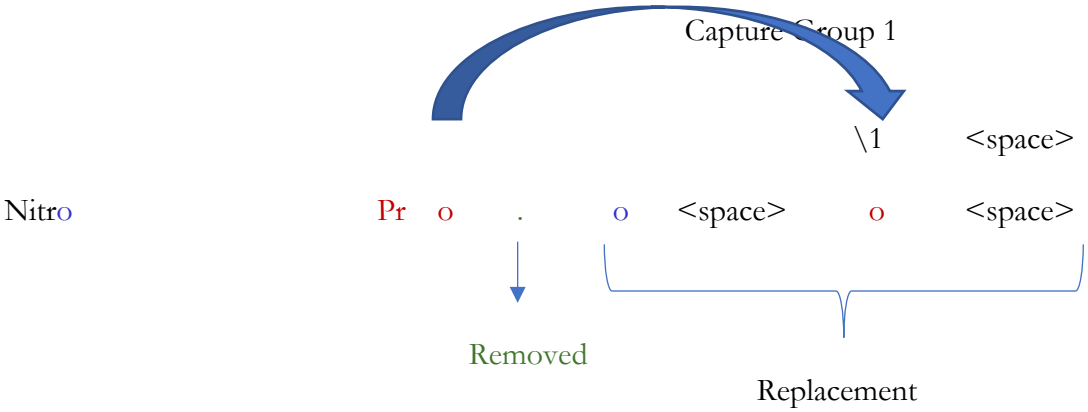
Match: ([^\d])[\.]+([^\d])+[\.]+

What it did is isolate the <dot > character from 'Nitro' – last character preceding the <dot> is the 'o' of 'Nitro' as well as the <dot> character from 'Pro' – last character preceding the <dot> is the 'o' of 'Pro'.

In the Replacment RegEx, because it was global, the [^\d] would first match until it encounters a <dot> [\.]. This would capture the 'o' of the word 'Nitro' and save it as Capture Group 1. This in turn would be added to the Replacement value \1 + <space>. It would look like this:



Continuing to move forward through the string for the next match , the [^\d] would move forward until it encountered a second <dot> [\.]. This time it would capture the 'o' of the word 'Pro' and save it to Capture Group 1. Note that the Global switch allows all matches to be retained so the new match value doesn't replace the current but 'adds' to it. So this in turn would be added to the Replacement value + <space>. It would look like this:



The RegEx Engine moves forward but encounters the numeric '1' of '13' and doesn't match.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

2nd RegEx: cont.

Match: ([^\d])[\.]+([^\d])+[\.]+

The result is that <dot> characters have been removed other than those that appear between two numeric digits. In this sample, this accounts for the removal of two before encountering the numeric digits and failing to match.

```
b = a.replace(/([^\d])[\.]+/g, '$1 ')
```

The replace() method is broken down to this:

```
new string = current filename . replace ( <RegEx to perform search> , <new value>)
b = a . replace (/ ([^\d])[\.]+ /g , '$1 <space>' )
```

The new replacement value is applied to 'a', overwriting any previous value but because the original string doesn't change, this value is written out to 'b'.

Result (using original 'pipe' character):

Nitro.Pro.13.15.1.282.Ent|x64 to Nitro Pro 13.15.1.282.Ent|x64

b = Nitro Pro 13.15.1.282.Ent|x64

Where:

a = the new current filename processed by the first Replace method.
b = the new string comprised of the replaced value from 'a'

Tim Notes:

1. Although 'a' is not a constant, all String Methods do not change the original string, but instead, write out to a new variable.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro.Pro.13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

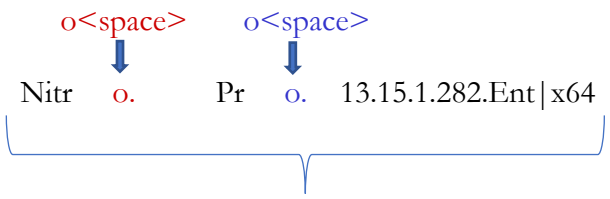
2nd RegEx: cont.

Match: ([^\d])[\.]+([^\d])+[\.]+

Tim Notes:

1. Unlike the first RegEx, where the entire parts of the string were captured and subsequently replaced in the replace() method, this time only the two matches of the 'o', both at different positions within the string are matched. Where BRU RegEx has to match against everything that you want to keep, the JavaScript replace() method will only replace those specific matches without disturbing the rest of the string value.

   In JavaScript, the two 'o's replace those at the same exact positions within the string retaining all else. Furthermore it uses the RegEx Replacement String of \1<space>, so each replacement becomes 'o<space>'



Nitro Pro 13.15.1.282.Ent|x64

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

2nd RegEx: cont.

Match: ([^\d])[\.]+ /g
Replace: \1 <space>

Where:

\1 = oo

Tim Analysis:

1. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit.	Capture Group 1 = N
2. [\.]	Match against a class consisting of a literal <dot> (Not Captured) = <dot>. This is the <dot> following 'Nitro'. Capture Group 1 changes to reflect this by capturing the 'o' of 'Nitro'.	Changes Capture Group 1 to o
3. +	Make it Greedy	Capture Group 1 = Unchanged
4. /g	Set the Global Switch Only available in JavaScript. Matches against all possible matches in string not just the first match which is what BRU RegEx is limited to.	
	What this will do is to add the 'o' of 'Pro' to Capture Group 1 before encountering the '1' of '13' and failing.	Capture Group 1 = oo

Tim Notes:

1. If I wanted to obtain the same result in BRU RegEx – I would use:

Match: ([^\d]+?)([\.]+)([^\d]+?)([\.]+)(.*)
Replace: \1 \3 \5

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

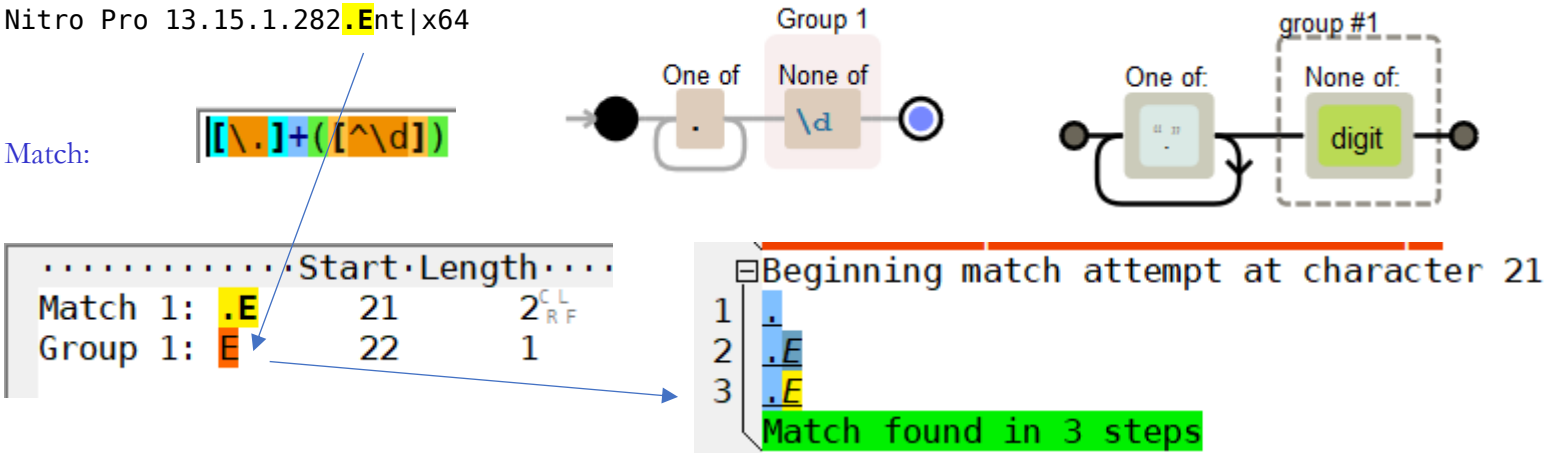
3rd RegEx:

Match: [\.]+([^\d])/g
Replace: <space> /1

This one is similar to the last one. What this next RegEx does is to isolate and replace one or more <dot> characters with a single <space>, unless that <dot> character is between two numeric digits, i.e., '1.282', will not be affected, but will affect '2.Ent'. Since the 2nd RegEx addressed any dots prior to the numeric digits, this moves past those numeric digits and ignores any <dot> characters between them, to match against any <dot> characters that don't.

Once again, the Global Switch is used.

```
c = b.replace(/[\.]+([^\d])/g, ' $1')
```



You can see that one match was made. The 'E' of 'Ent'. Just as before, the <dot> is removed and replaced with the <space> using the Replacement String in the RegEx (not to be confused with the JavaScript replace() method)

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282.Ent|x64

This one addresses dots, <spaces> and underscores cont.

3rd RegEx: cont.

Match: [\.]+([^\d])/g
Replace: <space> /1

The result is that <dot> characters have been removed other than those that appear between two numeric digits. In this sample, this accounts for the removal of one before encountering the EOL and exhausting the string.

```
c = b.replace(/[\.]+([^\d])/g, ' $1')
```

The replace() method is broken down to this:

```
new string = current filename . replace ( <RegEx to perform search> , <new value>)
c = b . replace (/ [\.]+([^\d]) /g , '<space>$1' )
```

The new replacement value is applied to 'b', overwriting any previous value but because the original string doesn't change, this value is written out to 'c'.

Result (using original 'pipe' character):

Nitro Pro 13.15.1.282.Ent|x64 to Nitro Pro 13.15.1.282 Ent|x64

c = Nitro Pro 13.15.1.282 Ent|x64

Where:

b = the new current filename processed by the second Replace method.
c = the new string comprised of the replaced value from 'b'

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

3rd RegEx: cont.

Match /[\.]+([^\d])/g
Replace: <space>\1

Where:

\1 = E

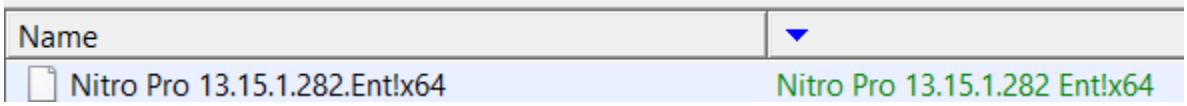
Tim Analysis:

1. [\.]	Match against a class consisting of a literal <dot> (Not Captured) = <dot> after '13'	
2. +	Make it Greedy= No Change to Not Captured	
3. [^\d]	Match against a Negated Class consisting of a numeric digit. Match against anything but a numeric digit. Changes the <dot> to the <dot> after '282'	Capture Group 1 = E
4. /g	Set the Global Switch Only available in JavaScript. Matches against all possible matches in string not just the first match which is what BRU RegEx is limited to. There are no more <dot> characters to match. String exhausted.	

Tim Notes:

1. If I wanted to obtain the same result in BRU RegEx – I would use:

Match: (.*)([\.]+)([^\d]+)(.*)
Replace: \1 \3\4



JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

4th RegEx:

Match: _+/g
Replace: none

This one is different. It relies on the JavaScript method of replace() rather than the RegEx. It does not capture any values, instead the RegEx does a sweeping search and when it encounters an underscore character, the replace() method replaces it with a <space>. It doesn't use a Replacement String in the RegEx to accomplish this, simply the straight syntax of the JavaScript replace() method.

Syntax

```
string . replace ( searchvalue,          newvalue)
  d    = replace (/      _+    /g    ,        ''    )
```

Once again, the Global Switch is used to include any and all matches against the underscore.

```
d = c.replace(/_+/g,' ')
```

No matches were made because this string contains no underscore characters.

Where:

c = the new name processed by the third Replace method.
d = the new string comprised of the replaced value from 'c'

Tim Analysis:

1. _ Match against underscore
2. + Make it Greedy
3. /g Set the Global Switch
   Only available in JavaScript. Matches against all possible matches in string not just the first match which is what BRU RegEx is limited to.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

5th RegEx:

Match: <space> +/g
Replace: none

Just as with the 4th RegEx, it relies on the JavaScript method of replace() rather than the RegEx. It does not capture any values, instead the RegEx does a sweeping search and when it encounters 'multiple' <space> characters, the replace() method replaces it with a 'single' <space>. It doesn't use a Replacement String in the RegEx to accomplish this, simply the straight syntax of the JavaScript replace() method.

Syntax

```
string . replace ( searchvalue, newvalue)
```

```
d = replace (/ <space>+ /g , '' )
```

Once again, the Global Switch is used to include any and all matches against multiple <space> characters.

```
e = d.replace(/ +/g,' ')
```

No matches were made because this string does not contain multiple <space> characters.

Where:

d = the new name processed by the forth Replace method.
e = the new string comprised of the replaced value from 'd'

Tim Analysis:

1. <space> Match against underscore
2. + Make it Greedy
3. /g Set the Global Switch
   Only available in JavaScript. Matches against all possible matches in string not just the first match which is what BRU RegEx is limited to.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

Finally, the Replacement is made of the 'pipe' character with the original underscore character.

Why use the 'pipe' character?

In the last request, the poster changed the pattern to include underscore characters throughout the string, not just at the end. To preserve this last underscore character, it was temporarily removed and replaced with the 'pipe' character so that the RegEx can perform the underscore sweep (4th RegEx). Then when all is said and done, all of the RegEx have been run and the values replaced, the underscore character is restored back. This is what this final part of the JavaScript does.

```
newName = e.replace('|','_')
```

Replace the occurrence of the pipe character with an underscore.

No RegEx – straight find and replace()

The final value is assigned to newName and the Renaming Function is ready.

Where:

e =	the new name processed by the fifth Replace method.
newName =	the new string comprised of the replaced value from 'e'

newName is the value set to rename the filename. Everything up to now has been preliminary and all changes have occurred in memory (this applies to the original JavaScript code – here in this analysis, the exclamation point had to be substituted in order for BRU to process the renaming). and that is why the pipe character, normally an illegal character in a Windows Filename, was allowed.

Tim Notes:

1. This Replacement of the '|' will not affect all samples, only those sample that end with either 'x86' or 'x64'. Otherwise, newName is assigned the last value of 'e' without having to perform the replacement().

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

Applying simulated Breakpoint Analysis -

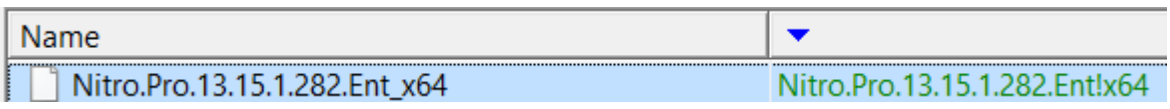
In researching my technique of the previous analysis I found an alternative method of testing the JavaScript's use of RegEx.

```
a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1|$2$3')

b = a.replace(/([^\d])[\.]+/g, '$1 ')

c = b.replace(/[\.]+([^\d])/g, ' $1')

d = c.replace(/_+/g,' ')

e = d.replace(/ +/g, ' ')

newName = e.replace('|','_')
```

Take each RegEx and directly adapt this to NewName, rather than building the final results from each separate ReEx string. The results are the same, and, although you won't get as accurate a picture of what is happening, if you want to save time this method should work. Just as before, you must perform the Rename function so that the next RegEx will run properly. This technique cannot use the | 'pipe' character. Use the exclamation point instead.

1st RegEx: Replace last underscore with | 'pipe' character (represented in these examples using exclamation point)

```
a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1!$2$3')
newName = a
```



Tim Notes:

1. This JavaScript line only affects those files that end with either 'x86' or 'x64'
2. Breakpoint Analysis as described here, has to do with setting a breakpoint in the code and performing an analysis.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

Applying simulated Breakpoint Analysis – cont.

2nd RegEx: Search and replace any <dot> characters not between numeric digits until a numeric digit.

```
b = name.replace(/([^\d])[\.]+/g, '$1 ')
newName = b
```

Name	
Nitro.Pro.13.15.1.282.Ent!x64	Nitro Pro 13.15.1.282.Ent!x64

3rd RegEx: Search and replace any remaining <dot> characters not between numeric digits

```
c = name.replace(/[\.]+([^\d])/g, ' $1')
newName = c
```

Name	
Nitro Pro 13.15.1.282.Ent!x64	Nitro Pro 13.15.1.282 Ent!x64

4th RegEx: Search and replace any underscore characters with a single <space>

```
d = name.replace(/_+/g,' ')
newName = d
```

Name	
Nitro Pro 13.15.1.282 Ent!x64	Nitro Pro 13.15.1.282 Ent!x64

No change because there are no underscores in sample.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

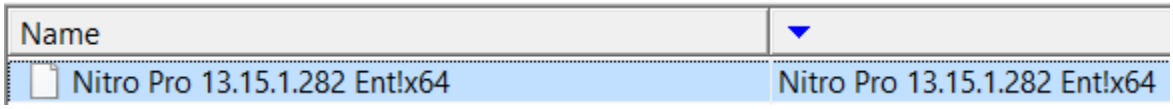
Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

Applying simulated Breakpoint Analysis – cont.

5th RegEx: Search and replace any multiple <space> characters with a single <space>

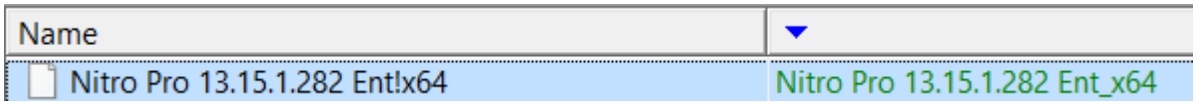
```
e = name.replace(/ +/g, ' ')
newName = e
```



No change because there are not any multiple <spaces> in sample.

Final Rename: Replace the 'pipe character | with the original underscore

```
newName = name.replace('!','_')
```



Replaces the | with the underscore.

In the future, you can use this technique to perform some simulated breakpoint analysis. A breakpoint is essentially a pause that let's you examine values and perform any needed analysis at the 'point of the break'.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

1.a.b.c.d.2.3.4.5.6
Home.Photostudio.Repack.v16.0
Nitro.Pro.13.15.1.282.Ent_x64
PDF_Architect....7.1.14.4969.x64
20200519022023.x64
12.34.abc.56.78
Nuance_PowerPDF_Advanced_3.00.6439_x64
12.34.a.b.c.56.78
Movavi.Photo.Editor.Repack.v4.4.0
Driver.Booster.Repack.7.4.0.730
202005190-x64.1234
20200519022023 x64
PDF.Desktop.6.0.8.x64
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64
a.b.c.d.e.f1.g.h.i
NCH.PhotoPad.v6.13
firefox.2020-01-08_065639
202005190-x64.1234

One last example that I am not going to analyze, but I leave it to you. With all of the knowledge gained, this final example is your graduation exercise if you will. It is similar to what I have just demonstrated so it shouldn't be that difficult to figure out. I will give you a tiny bit of help.

The Contributor adds this:

If you'd rather treat underscore just like dots (keep them when in between numbers), use this instead:

```
a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1|$2$3')
b = a.replace(/([^\d])[\._]+/g, '$1 ')
c = b.replace(/[\._]+([^\d])/g, ' $1')
d = c.replace(/ +/g, ' ')

newName = d.replace(' |','_')
```

Name	
PDF_Architect....7.1.14.4969.x64	PDF Architect 7.1.14.4969_x64
PDF.Desktop.6.0.8.x64	PDF Desktop 6.0.8_x64
PDF Architect 7.1.14.4969.x64	PDF Architect 7.1.14.4969_x64
Nuance_PowerPDF_Advanced_3.00.6439_x64	Nuance PowerPDF Advanced 3.00.6439_x64
Nitro.Pro.13.15.1.282.Ent_x64	Nitro Pro 13.15.1.282 Ent_x64
NCH.PhotoPad.v6.13	NCH PhotoPad v6.13
Movavi.Photo.Editor.Repack.v4.4.0	Movavi Photo Editor Repack v4.4.0
Microsoft.NET.Core.Desktop.Runtime.v3.1.4_x64	Microsoft NET Core Desktop Runtime v3.1.4_x64
Home.Photostudio.Repack.v16.0	Home Photostudio Repack v16.0
firefox.2020-01-08_065639	firefox 2020-01-08_065639
Driver.Booster.Repack.7.4.0.730	Driver Booster Repack 7.4.0.730
a.b.c.d.e.f1.g.h.i	a b c d e f1 g h i
20200519022023.x64	20200519022023_x64
20200519022023 x64	20200519022023_x64
202005190-x64.1234	202005190_x64.1234
12.34.abc.56.78	12.34 abc 56.78
12.34.a.b.c.56.78	12.34 a b c 56.78
1.a.b.c.d.2.3.4.5.6	1 a b c d 2.3.4.5.6

The results are identical to the first extended samples, except that the firefox file keeps the underscore.

JavaScript

Remove <dots> between words only (Not numbers) in File Name and replace it with <space> cont.

Using: Nitro Pro 13.15.1.282 Ent|x64

This one addresses dots, <spaces> and underscores cont.

Comparison against original:

Where:

Purple = No Change Orange = New Changes Blue = Original

```
New:      a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1|$2$3')
Original: a = name.replace(/^(.*)[-_\. ](x64|x86)(.*)$/,'$1|$2$3')

New:      b = a.replace(/([^\d])[\._]+/g, '$1 ')
Original: b = a.replace(/([^\d])[\.]+/g, '$1 ')

New:      c = b.replace(/[\._]+([^\d])/g, ' $1')
Original: c = b.replace(/[\.]+([^\d])/g, ' $1')

New:      d = c.replace(/ +/g, ' ')
Original: d = c.replace(/_+/g,' ')

New:      newName = d.replace('|','_')
Original: newName = d.replace('|','_')
```

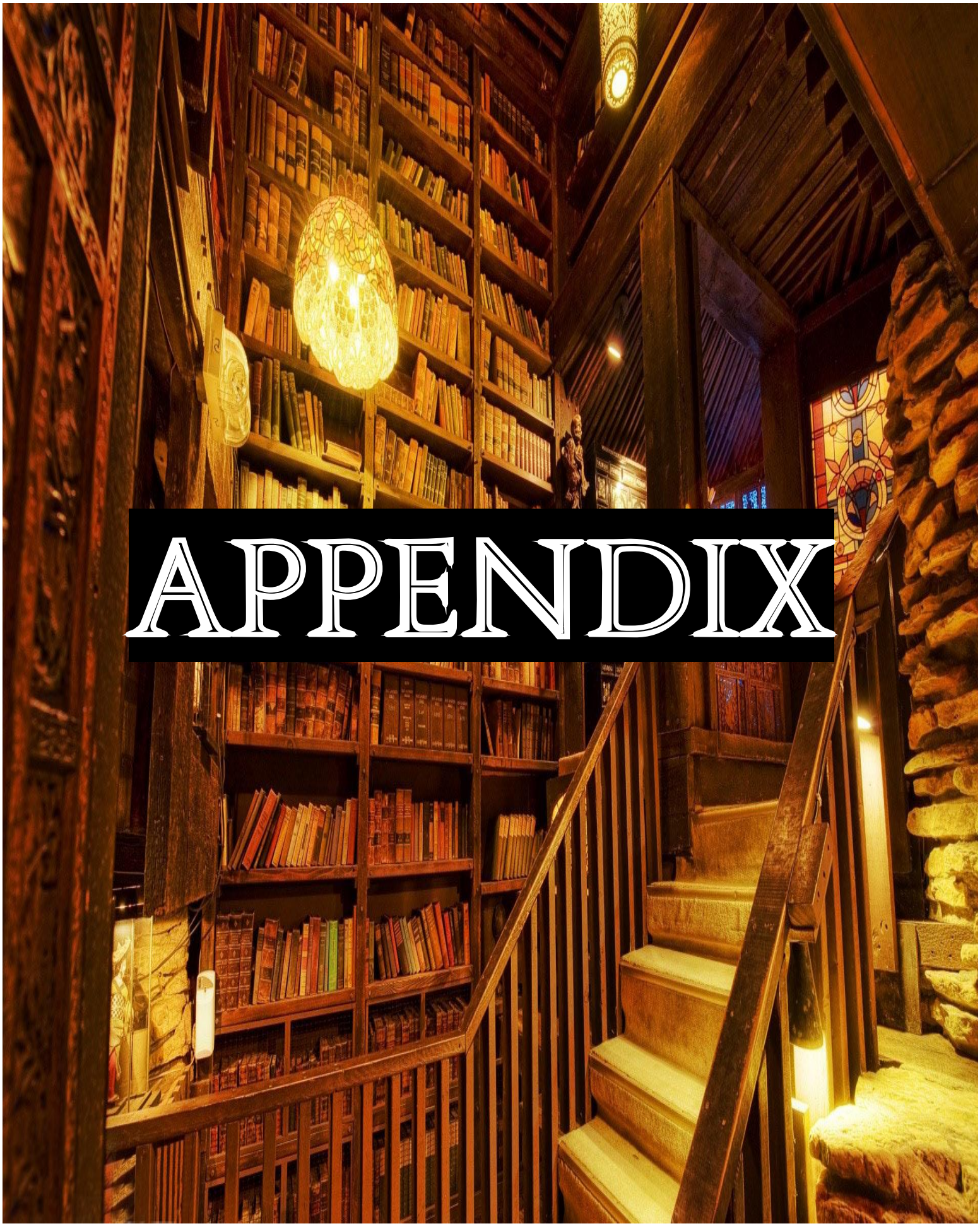
Breaks down:

```
          b = a.replace   (/([^\d])   [\.  _ ]   +/g, '$1 ')
original  b = a.replace   (/([^\d])   [\.    ]   +/g, '$1 ')


          c = b.replace   (/[\.  _  ]   +   ([^\d])/g, ' $1')
original  c = b.replace   (/[\.    ]   +   ([^\d])/g, ' $1')
```

That's it. That's all I'm going to give you. Good luck. I hope you have had fun.

Forum Contributor: RegexNinja

[This Page Intentionally Blank]


APPENDIX

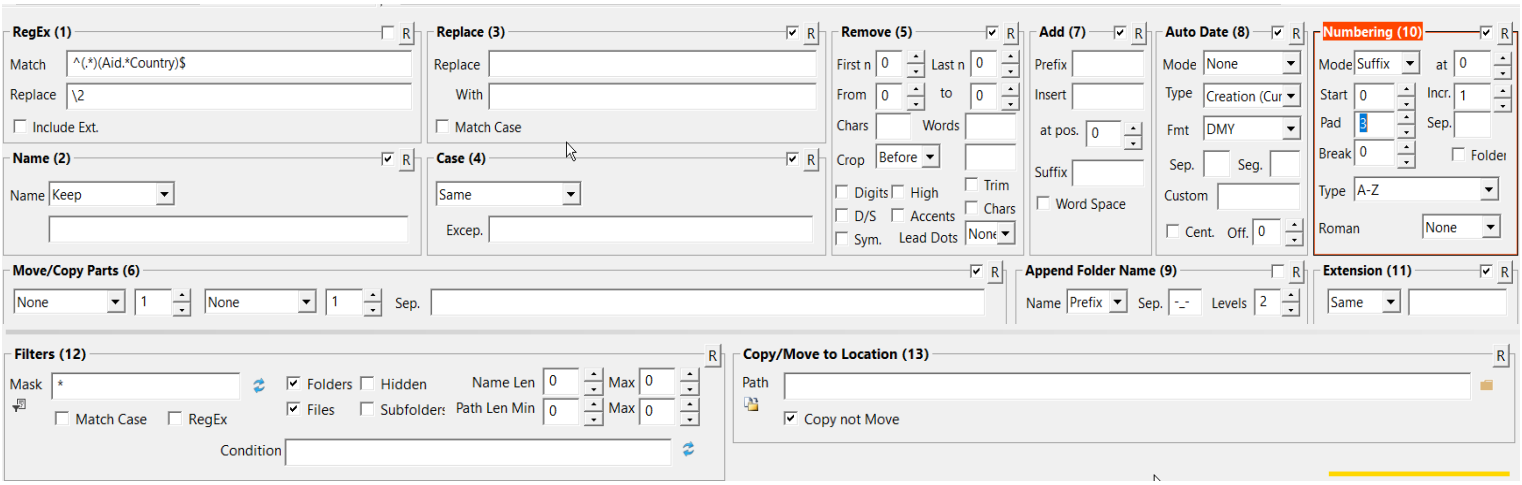
A Word on Order of Evaluation

Appendix

Tim's Tips & Tidbits:

A Word on Order of Evaluation

Although I detailed this in 'Volume I Bulk Rename Utility Operations Manual, pretty extensively as well as a lengthy discussion under the Debug New Name function, I thought I would just add a few clarifications for those users who may still have difficulty comprehending the concept.



The Numbers next to each section in the Interface, that I refer to in my books as 'Section # n: <name>', are representative of the Order of Evaluation – for the most part, – there are exceptions and I refer you to Volume I.

What this means is that if I have, for example, a string that I want to completely rename to 'CAT LAST':

The Quick Brown Fox Jumps Over the Lazy Dog

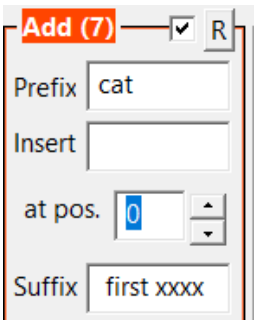
to

CAT LAST

I would like to perform the rename in this order:

1. Remove the filename.

2. Add 'cat' as prefix and add a suffix of '<space>first xxxx'



string = cat first xxxx

Appendix

Tim's Tips & Tidbits:

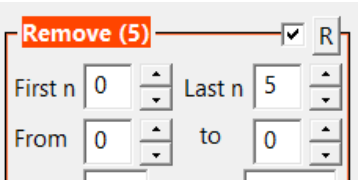
A Word on Order of Evaluation cont.

The Quick Brown Fox Jumps Over the Lazy Dog

to

CAT LAST

3. Remove the last 5 characters of the string
(I changed my mind so sue me)



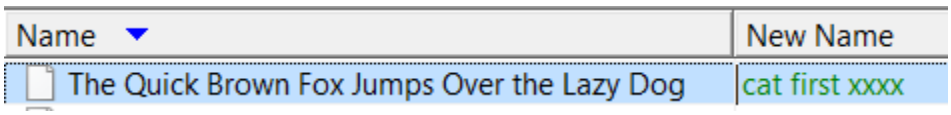
string = cat first

4. Change the string to uppercase
string = CAT FIRST

5. Replace the 'FIRST' with 'LAST'

string = CAT LAST

So what did I end up with?



.. Not CAT LAST … Why?

Because the desired order of Criteria entered did not correlate to the order of evaluation:

step 1: 'Section #2: Name' was performed first. Removed 'name'.	string now = empty "" or null value
step 5: 'Section #3: Replace'	– replace what? – the string is empty
step 4: 'Section #4: Case'	– nothing to change – string still empty
step 3: 'Section #5: Remove'	– nothing to remove – string still empty
step 2: 'Section #7: Add' – Prefix added, Suffix added	string = cat first xxxx

Conclusion:

The Order of Evaluation is not based on the order in which the Criteria is entered. It is the BRU program that predetermines the order of evaluation by the defined sections. You cannot have section 7 evaluate before section 5, for example (exceptions noted). The end result is not based on user desired outcome, but the parameters established by the program. BRC (Bulk Rename Command), it should be noted, is not restricted to the Order of Evaluation.

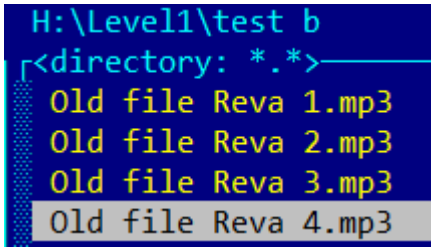
Documenting Tools

Appendix

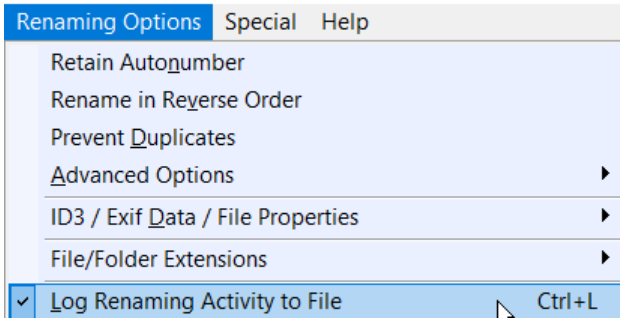
Tim's Tips & Tidbits:

Documenting Tools

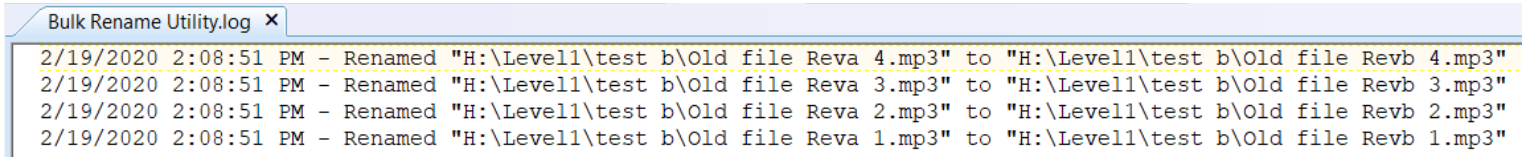
If you have a bunch of files to be renamed and you want to keep a revision log you can:



1. Use the Log Activity option.



Logs activity to Bulk Rename Utility.log located at \Documents\Bulk Rename Utility\



2. Use the Preview Button – but no transcript is generated.

Preview

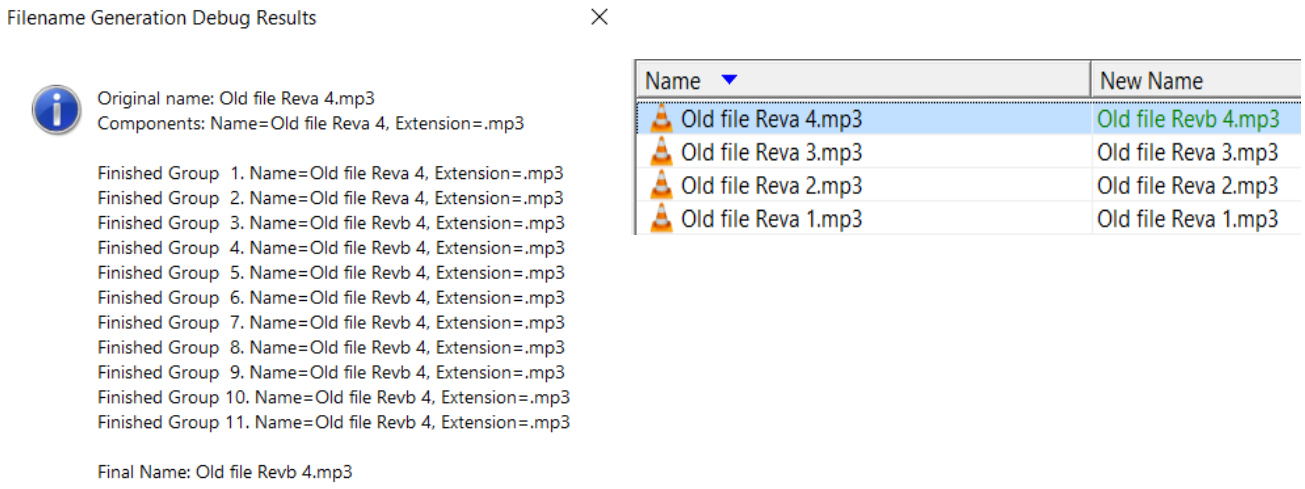
Current Name	New Name
Old file Reva 4.mp3	Old file Revb 4.mp3
Old file Reva 3.mp3	Old file Revb 3.mp3
Old file Reva 2.mp3	Old file Revb 2.mp3
Old file Reva 1.mp3	Old file Revb 1.mp3

Appendix

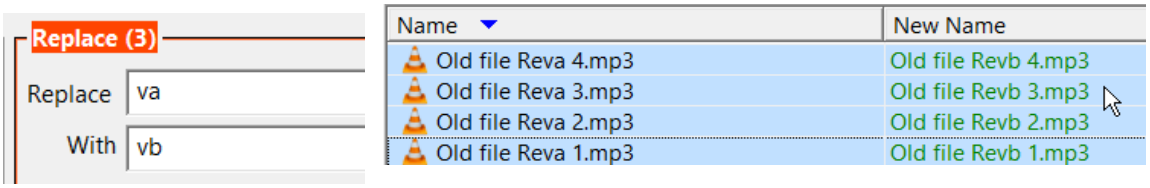
Tim's Tips & Tidbits:

Documenting Tools cont.

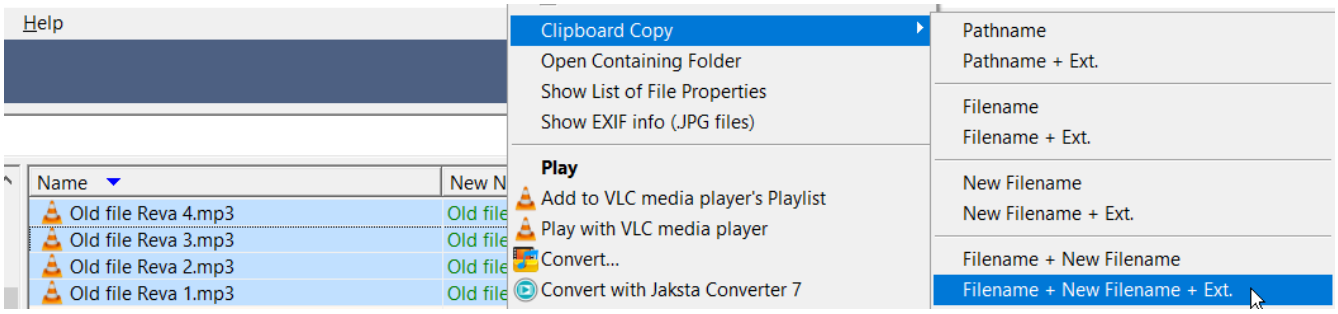
3. You can use Debug New Name – but this does not create any transcript either.



4. Use Clipboard Copy and create an export revision list:
   a. Enter your criteria as normal and select files to be renamed.



   b. Right click on file list in Content Pane and select Clipboard Copy.
   c. Select your option from the submenu, e.g. 'Filename + New Filename + Ext.'
   d. File information is copied to the clipboard.



   e. Paste information where desired.

Old file Reva 4.mp3	Old file Revb 4.mp3
Old file Reva 3.mp3	Old file Revb 3.mp3
Old file Reva 2.mp3	Old file Revb 2.mp3
Old file Reva 1.mp3	Old file Revb 1.mp3

Methods of Creating File Lists For Import Rename-Pairs

Appendix

Tim's Tips & Tidbits:

Methods of Creating File List Structure for BRU's Import Rename-Pairs Function

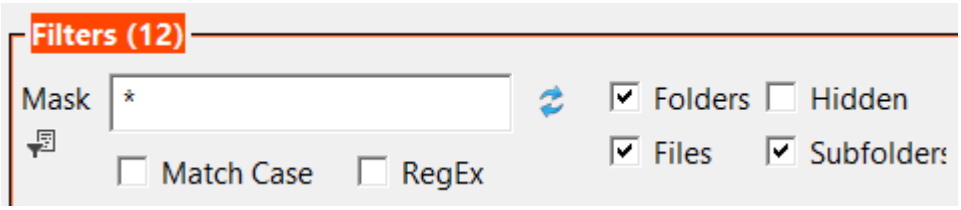
These techniques will only create a list of files from a directory or in some cases create the structure, but they do not add the new names - you have to do this manually through a text editor or word processor.

e.g, oldname|

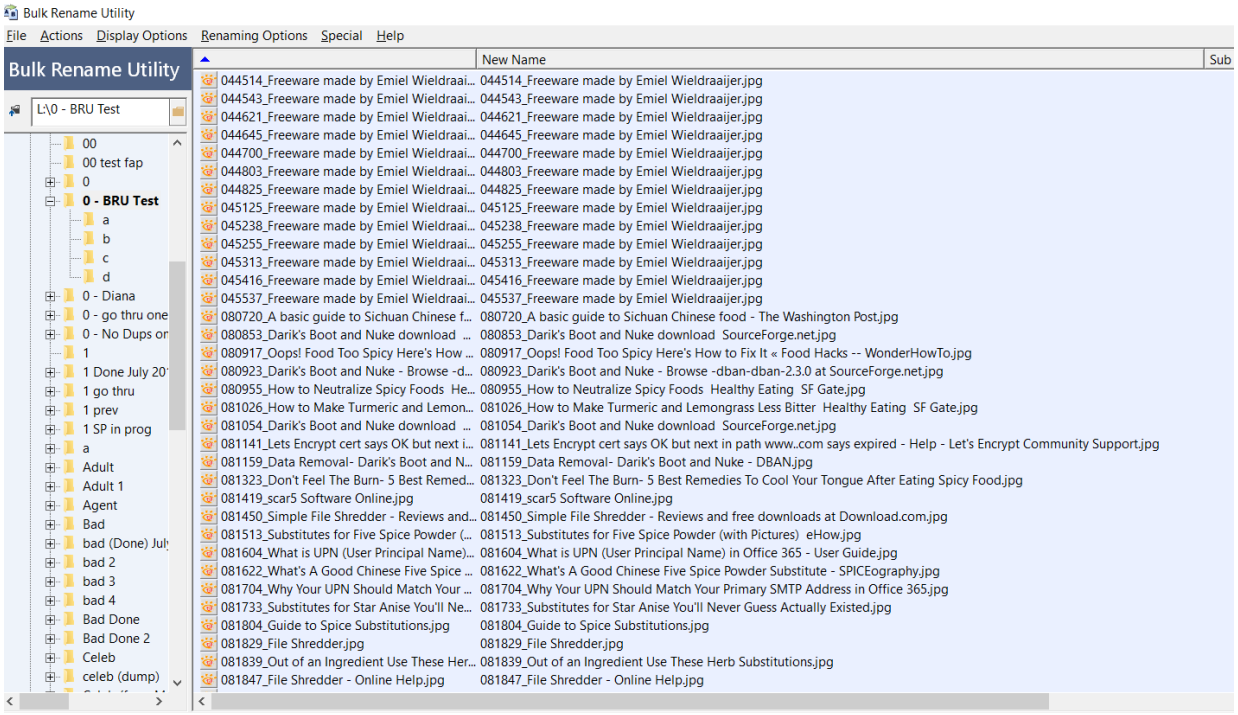
1. The Excel Technique can already be found in 'Volume I Bulk Utility Operations Manual', under the Actions Menu section.

2. You can use BRU itself to create file list for import into rename pair.

   a. In the Content Pane, display the list of files you wish to include as the file list. If needed, you can turn on recursion by enabling the 'Subfolders' option under, 'Section #12: Filters', to include any files in the subdirectories.



   b. Highlight the files in the Content Pane.



   1. You can narrow down your selections manually by selecting which files to include, or use a filter.

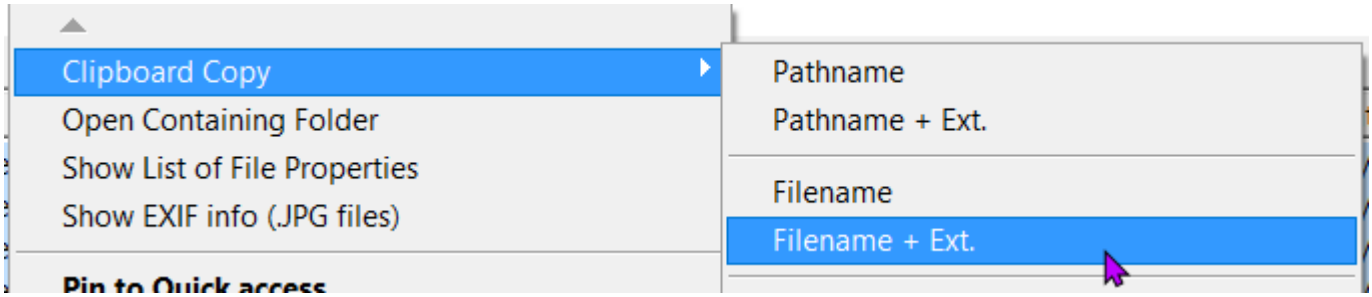
Appendix

Tim's Tips & Tidbits:

Methods of Creating File List Structure for BRU's Import Rename-Pairs Function cont.

2. You can use BRU itself to create file list for import into rename pair. cont.

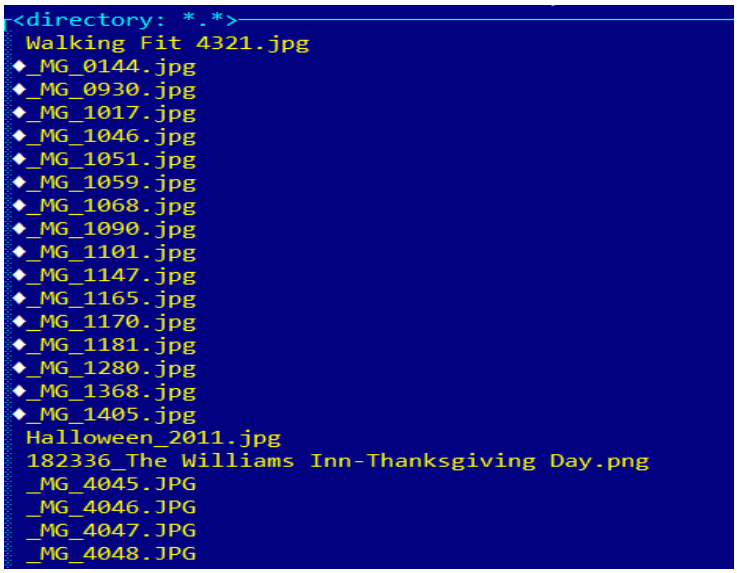
c. Right click on files in BRU and copy to clipboard using Filename + Ext.



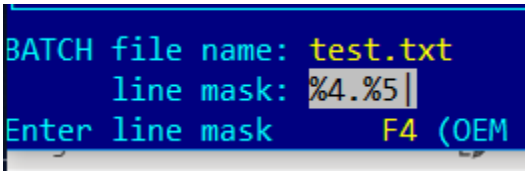
d. Paste the clipboard contents and add the new names using a word processor or text editor.

3. Ztree or other File Manager that has a Create Batch file method for creating the structure for adding new files:

a. Select the directory containing the intended files.
b. Tag only those files you need.



c. Call up Ztree's Batch facility (Ctrl+ B).



d. Give it a name and specify the mask as:

%4 . %5 |

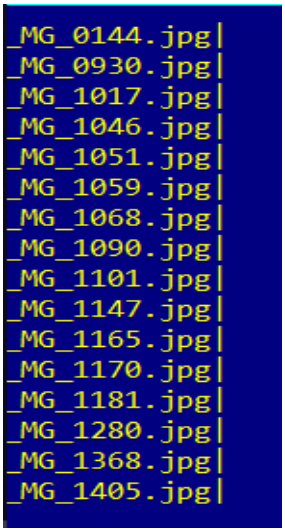
(spaces added only for clarity)
% 4 <dot> %5 <pipe>

Appendix

Tim's Tips & Tidbits:

Methods of Creating File List Structure for BRU's Import Rename-Pairs Function cont.

3. Ztree or other File Manager that has a Create Batch file method for creating the structure for adding new files: cont.



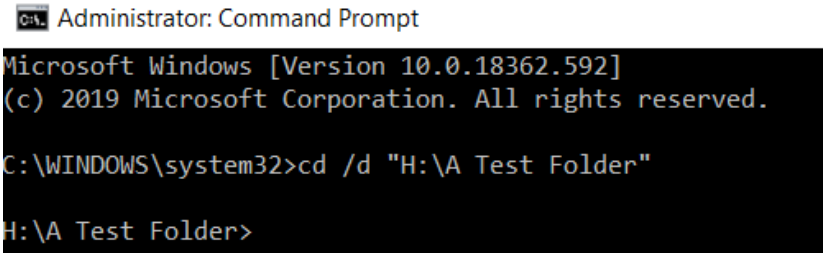
e. A Look inside the file you just created –

f. Now all you have to do is add the new names using a word processor or text editor.

You can be selective about which files are included in the list through the use of the Tag function. Ztree also has filtering available.

4. Utilize DOS to create a list of files from a directory:

a. Open a command Prompt.
b. Change directory to the desired directory containing the files to be included in your list.



c. To add all the files in the current directory:
   1. Type, "Dir /b > _FileList.txt"
d. To add only certain files, i.e., *.Jpg
   1. Type, "Dir /b *.jpg > _JPGs_FileList.txt"

Appendix

Tim's Tips & Tidbits:

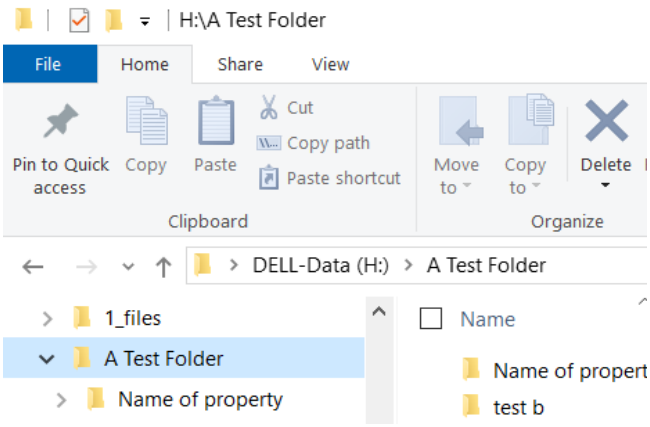
Methods of Creating the Complete File Lists for BRU's Import Rename-Pairs Function

These techniques will create a complete list (old and new names) of files for inclusion into BRU's Import Rename-Pairs Function:

e.g, oldname|newname

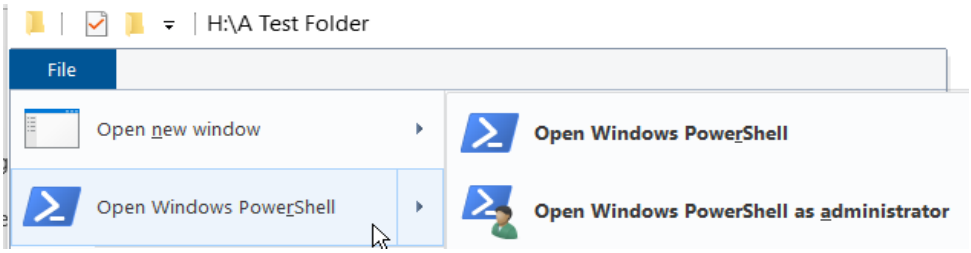
This involves creating two separate lists – oldnames.txt and newnames.txt and using some script language to merge the two files together into the proper format.

1. Utilize PowerShell
   a. Create the oldnames.txt
      1. Use Windows File Explorer and browse to the directory containing the files to be added.

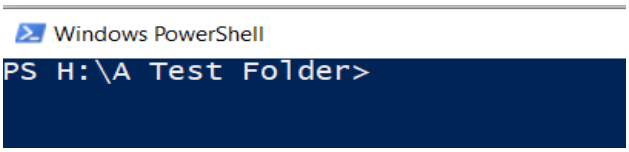


      2. Double click on the folder name to open it.

      3. From File Menu, select 'Open Windows Powershell'



      4. Windows Powershell console opens already at the selected directory.



Appendix

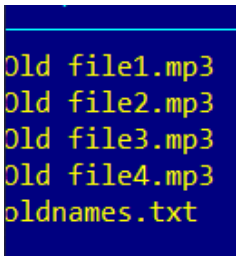
Tim's Tips & Tidbits:

Methods of Creating the Complete File Lists for BRU's Import Rename-Pairs Function

1. Utilize PowerShell cont.
   a. Create the oldnames.txt cont.

      5. Type in:

```
dir | ForEach{if(!$_.PSisContainer){ $_.Name }} >> oldnames.txt
```



Tim Notes:

1. It adds the oldnames.txt and newnames.txt into the file as well. This will have to be edited out. Not sure how to prevent this – I don't know Powerscript.

   b. Create the oldnames.txt
      1. Use Windows File Explorer and browse to the directory containing the files to be added.
      2. Double click on the folder name to open it.
      3. From File Menu, select 'Open Windows Powershell'
      4. Windows Powershell console opens already at the selected directory.
      5. Type in:

```
dir | ForEach{if(!$_.PSisContainer){ $_.Name }} >> newnames.txt
```

   c. Create the Merged List, i.e., List.txt
      1. Place oldnames.txt and newnames.txt in the same directory.
      2. Use Windows File Explorer and browse to the directory containing the files to be added.
      3. Double click on the folder name to open it.
      4. From File Menu, select 'Open Windows Powershell'
      5. Windows Powershell console opens already at the selected directory.
      6. Type in:

```
$old=(gc oldnames.txt)
$new=(gc newnames.txt)
$iteration= $old.count -1
$List=""
0..$iteration | ForEach{ $List += ($old[$_] + "|" + $new[$_] + "`r`n")}
$List
$List > List.txt
```

Appendix

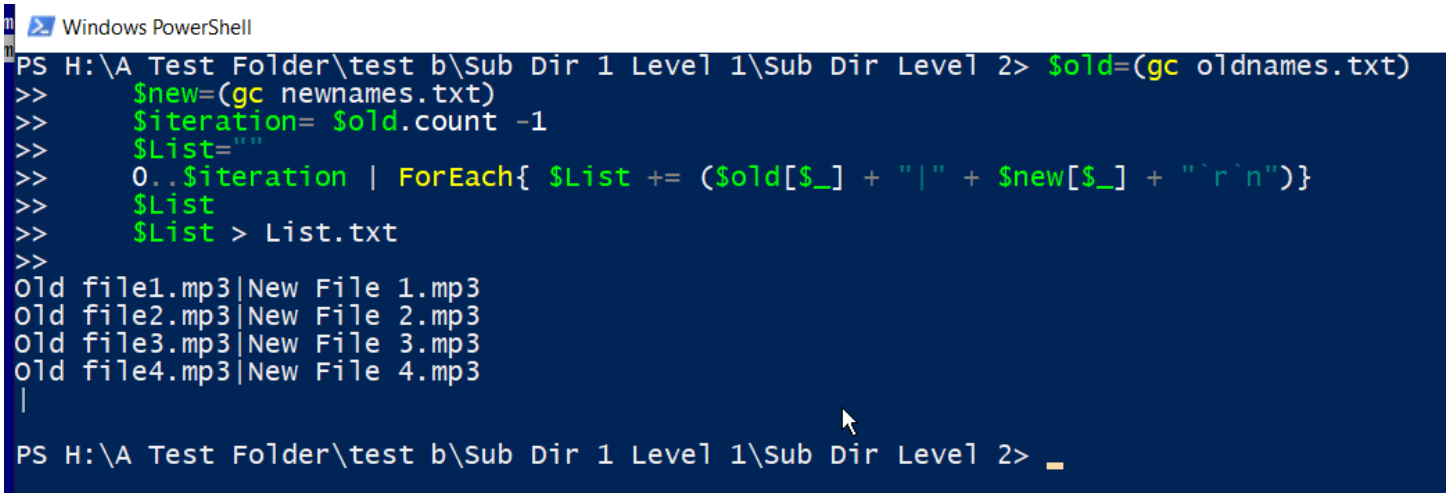
Tim's Tips & Tidbits:

Methods of Creating the Complete File Lists for BRU's Import Rename-Pairs Function cont.

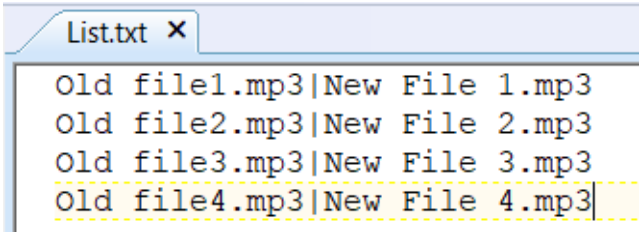
1. Utilize PowerShell cont.

c. Create the Merged List, i.e., List.txt cont.

7. The result after running the script:



8. .. And the file created…



Tim Notes:

1. If Open Windows Powershell is grayed out, you need to doubleclick on the folder to open it – this is an indicator that you have not.

Appendix

Tim's Tips & Tidbits:

Methods of Creating the Complete File Lists for BRU's Import Rename-Pairs Function cont.

2. The AutoHotkey Script
   a. Requires AutoHotkey (https://www.autohotkey.com)
   b. The script follows:

```
;REM AutoHotkey1.0 Script , v0.001, Stefan 2015-06-19
;REM Purpose: Join Two Lists Together (JTLT) for Bulk Rename Utility's "Renaming From A Text File" feature.
;REM Example line: "Origin file name.ext|New Filename.ext"
;REM Needs AHK: http://ahkscript.org/download/1.0/AutoHotkey104805.zip
;=================================================================
#SingleInstance force
OutFile = __BRU_TextFile.txt
;=================================================================

HelpMessage =
(
Merge two list together for BRUs "Renaming From A Text File" feature.

1) First create your list of oldnames.txt and newnames.txt
   a. Use one of the methods in 'Methods of Creating File Lists' of
      the 'Appendix' Section' found in -
      'Volume II Bulk Rename Utility Operations Manual'
   b. Place both files in the same directory.
2) Copy and paste this script to a text file with a AHK extension
   and place this is the same directory as oldnames.txt and newnames.txt
3) Execute the script by opening it – AutoHotkey will run the script.
4) After hitting any key to continue after reading this:
   a. File Open dialog – select the file that contains the files to be renamed
   b. File Open dialog – select the file that contains the new filenames
5. Generates a new file,  __BRU_TextFile.txt:

        old name1.ext|new name1.ext
           old name2.ext|new name2.ext
                 ...
)
MsgBox,,JTLT Help message,%HelpMessage%e%
;=================================================================
```

Appendix

Tim's Tips & Tidbits:

Methods of Creating the Complete File Lists for BRU's Import Rename-Pairs Function cont.

2. The AutoHotkey Script cont.

```
FileSelectFile, ListNo1,,%A_WorkingDir%,Select list-file with origin names
  If ListNo1 !=
  {
     SplitPath, ListNo1,,sLastUsedFolder
     FileSelectFile, ListNo2,,%sLastUsedFolder%,Select list-file with wanted names
     If ListNo2=
        GoSub, ErrorMessage
  }
  ;=====================================================================
  Loop,Read,%ListNo1%
  {
   iZEILE++
   FileReadLine,sCurrLineFile1,%ListNo1%,%iZEILE%
   FileReadLine,sCurrLineFile2,%ListNo2%,%iZEILE%
   outArray .= sCurrLineFile1 "|" sCurrLineFile2 "`r`n"
  }

  If outArray=
     Return
  FileDelete,            %OutFile%
  FileAppend, %outArray%, %OutFile%
  DoneMessage =
  (
  File list created.

  )
  MsgBox,,JTLT All Done!,%DoneMessage%

  Run,%OutFile%
  ExitApp
  ;=====================================================================
  ErrorMessage:
  ErrorMessage = ERROR: Please select two files, one with origin names, one with wanted names.
  ErrorMessage = %ErrorMessage%`n`nRun Script again!`n`nHelp:`n`n%HelpMessage%
  MsgBox,,JTLT ERROR !!!,%ErrorMessage%
  ExitApp
  ;=====================================================================
```

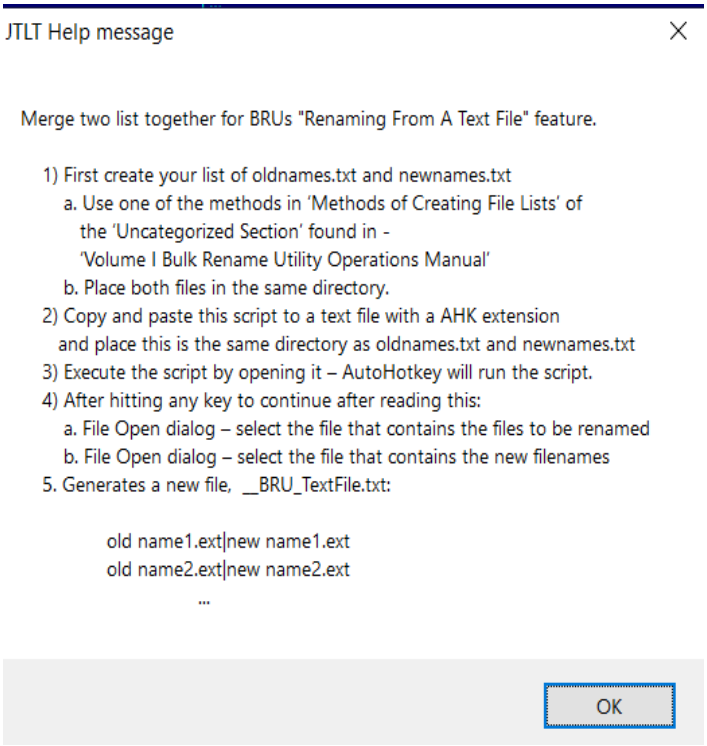
Appendix

Tim's Tips & Tidbits:

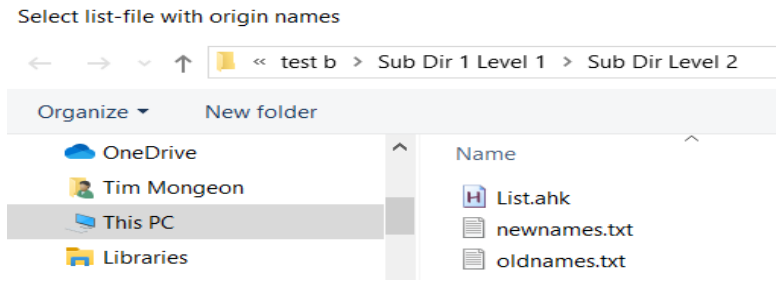
Methods of Creating the Complete File Lists for BRU's Import Rename-Pairs Function cont.

2. The AutoHotkey Script cont.

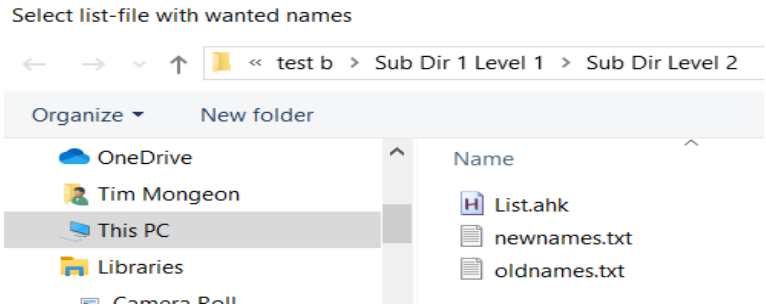
c. Here is a program run:

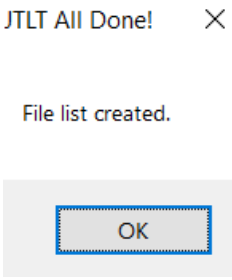


Select the oldnames.txt file

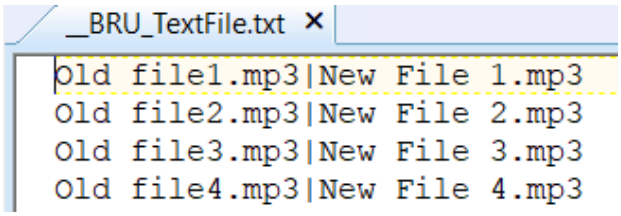


Select the newname.txt file





Here is what the finished list looks like:



Tim Notes:

1. I modified the help message to make it more concise and referenced the reader to this volume to provide additional methods for creating the initial file lists.
2. Message format looks like it is a little off center in one area, oldname1.exe – but it must remain this way to get the message lined up correctly during the run.

Forum Contributor: Stefan

Accessing a Network Drive

Appendix

Accessing a Network Drive

BRU does not support UNC Paths:

(Universal Naming Convention) A standard for identifying servers, printers and other resources in a network.

1. A UNC path uses double slashes or backslashes to precede the name of the computer.
2. The path (disk and directories) within the computer are separated with a single slash or backslash.

i.e.,

\\servername\path

3. These names consist of three parts: a host device name, a share name, and an optional file path.

a. Host-Name - can consist of either a network name string set by an administrator and maintained by a network naming service like DNS or WINS, or by an IP address.
   1. These hostnames normally refer to either a Windows PC or a Windows-compatible printer.

b. Share-Name - references a label created by an administrator or, in some cases, within the operating system.
   1. In most versions of Microsoft Windows, the built-in share name admin$ refers to the root directory of the operating system installation—usually C:\Windows but sometimes C:\\WINDOWS or C:\\WINNT.
   2. UNC paths do not include Windows driver letters, only a label that may reference a particular drive.

c. File_Path - references a local subdirectory beneath the share section. This part of the path is optional.
   1. When no file_path is specified, the UNC path simply points to the top-level folder of the share.
   2. The file_path must be absolute. Relative paths are not allowed.

Notes:

1. If you're going to access your local computer (or any computer) using UNC, you'll need to setup a share. If you haven't already setup a share, you could use the default administrative shares.

   Example:

   \\localhost\c$\my_dir

   ... accesses a folder called "my_dir" via UNC on your C: drive. By default all the hard drives on your machine are shared with hidden shares like c$, d$, etc.

Now you know what UNC is, let's see the work-around:

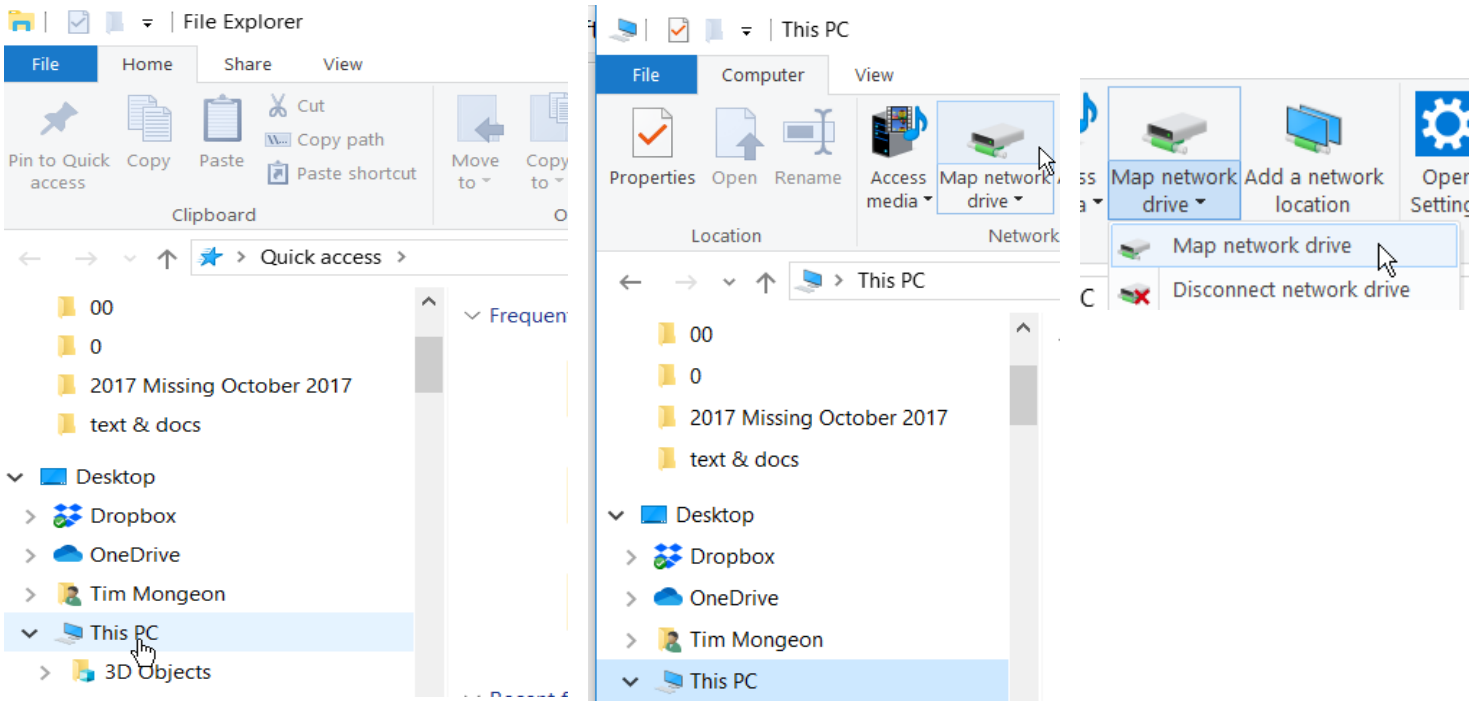
Instead of UNC, you can map a network drive. This means to tell Windows to treat the network drive as if it were a local drive. Mapped drives are assigned drive letters. Once a drive is mapped, you can access it like any other drive and BRU will be able to access the drive and the files residing on that drive.

Appendix

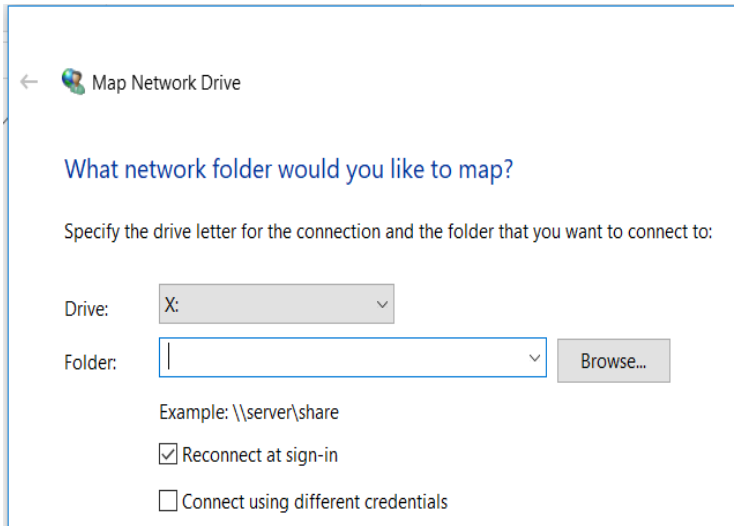
Accessing a Network Drive cont.

Mapping a Network Drive

1. Open File Explorer from the taskbar or the Start menu, or press the Windows logo key + E.
2. Select This PC from the left pane. Then, on the Computer tab, select Map network drive.



3. In the **Drive** list, select a drive letter. (Any available letter will do.)



4. In the Folder box, type the path of the folder or computer, or select Browse to find the folder or computer.
   a. To connect every time you log on to your PC, select the Reconnect at sign-in check box.

5. Select Finish.

Troubleshooting Notes:

1. If you can't connect to a network drive or folder, the computer you're trying to connect to might be turned off, or you might not have the correct permissions.

2. Mapping works under a user context, so if you run BRU "As Admin", you are at that point running as a different user and BRU won't see the drive.

Appendix

Accessing a Network Drive cont.

Freeze or Slow Response when Accessing Mapped Network Drive

The slow response appears to be related to querying the Size and Date fields on a network drive.

Tim Notes:

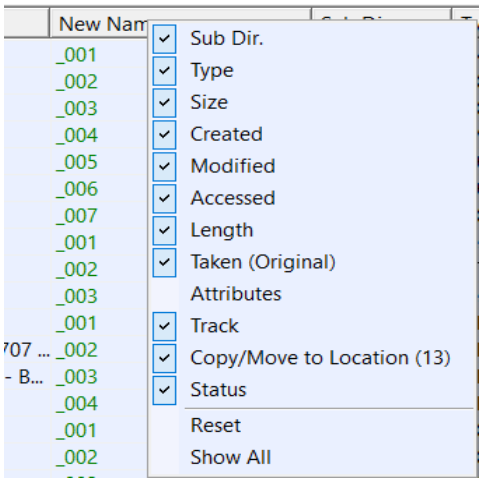
1. When a folder on a drive is selected from the Navigation Pane (the Tree Hierarchy), BRU scans the folder including any files in that folder (and if recursion is set, any files in all the subdirectories that fall beneath). This typically takes no time depending on your settings (For example if 'ID3 /EXIF Data/ File Properties' from Renaming Options Menu are set, this can take a while as the program scans all of the metadata).

   Other data, such as column settings for displaying 'Created', 'Modified' and 'Accessed' dates and 'File Size' for instance, are also collected and displayed. But for some reason, when it comes to collecting the data for these, the scan reduces to a crawl. So slow, in fact, it will appear the BRU program is frozen. Displaying Icons can also be an issue. Wait it out, and after what seems like a very long time, the scan will finish and BRU will become responsive again. The next time this happens, look at the status bar. It will indicate that a scan is in progress.

2. For more information, refer to Volume I –
   a. ID3 / Exif Data/ File Properties
   b. Speeding Up the Program found in the Appendix
   c. Recursive Scans, under Considerations at the beginning of the volume
   d. Notation #1 under 'Section #12: Filters'

To work around this issue:

1. Go to a local drive and select a folder that contains files.
2. Right Click on the header row and turn off the Size and Date fields.



3. Navigate to a network drive and the file names will appear quickly.

Appendix

Accessing a Network Drive cont.

Mapped drives not showing in BRU

It has to do with your local permissions versus your mapped drive permissions.

If you are mapping a drive with:

1. Active Directory account
   or
2. An account on another windows machine on your network

.. and you use Bulk Rename with "Run As Administrator"

.. you are actually telling the Bulk Rename executable to use an account that most likely doesn't have permissions to access the mapped drive's network location, even if it's a local administrative credential.

Solution:

1. The account you are logging into your computer with is a member of the Local Administrators group, i.e. MSHOME.
2. The account has permissions to read and write to the network location of your mapped drive.
3. Do not run 'Bulk Rename Utility' as Administrator.

Tim Notes:

1. This is a known issue with Microsoft, unfixed of course. The Explanation is as follows:

When UAC is enabled, the system creates two logon sessions at user logon. Both logon sessions are linked to one another. One session represents the user during an elevated session, and the other session where you run under least user rights.

When drive mappings are created, the system creates symbolic link objects ("DosDevices") that associate the drive letters to the UNC paths. These objects are specific for a logon session and are not shared between logon sessions.

Meaning that when UAC is enabled. and it is, on default – you will see it pop up as:

> User Account Control
> Do you want to allow this app from an unknown Publisher…

This creates You as two separate users – one that has Elevated privilege and one that doesn't. Mapped drive access requires unelevated privileges. If you're at an elevated level, you are seen as a separate user who doesn't have access.

Of course none of their work-arounds work. So bottom line, it doesn't matter if it's BRU or some other program, mapped drives are unavailable if you use an elevated command, 'Run as Administrator'.

Forum Contributors: Admin, jgbrown54, jcolton

Source This section: Wiliam Leara Stack Overflow, Lifewire, PC-Mag Encyclopedia, suppport.microsoft.com

Renaming with Dates Prior to January 1, 1971

Appendix

Renaming with Dates Prior to January 1, 1971

1. BRU does not support setting timestamps older than 'January 1, 1971'. It has been this way since the beginning when Jim Willsher created the program. This affects 'Section #14: Change (Timetamps)' and 'Section #8: Auto Date'. He explained that it defaults to the Epoch. Epoch is a system for describing a point in time. Originally Epoch time was set to 00:00:00 UTC on 01/01/1971. This was later changed to 01/01/1970.

2. The column data displayed in the Content pane will not display a date older than 'December 31, 1969' for any file with a 'Creation', 'Modified' or 'Accessed' date prior. This affects not only the column data but again, 'Section #8: Auto Date'. It is presumed that Jim may have changed the code at two different times resulting in one feature having the 1971 Epoch limitation and the other having the 1970 Epoch limitation.

3. Auto Date has another quirk when it comes to this. This will be explained below.

As a workaround, the following Poster has a better than nothing solution.

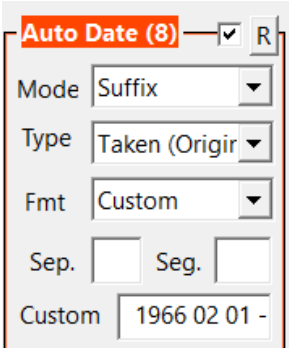
DSCN0001.JPG with a Date Taken of February 2, 1966 11:59:59 AM

Observations:

1. Need to add the 'Date Taken' [Exif metadata 'Taken (Original)'] to the suffix of the filenames.
2. Limitations noted above make the normal procedures of BRU impossible for the task.

Work-around:

1. 'Section #8: Auto Date' –



Mode: Suffix
Type: Taken (Original)
Fmt: Custom

Use a Custom of, i.e.,

1966 02 01 - %H.%M.%S
for batches of files with a date of February 2nd 1966

Name	New Name	Type	Size	Created	Modified	Accessed	Taken (Original)
DSCN0001.JPG	DSCN0001 1966 02 01 - 23.59.59.JPG	JPG File	2 MB	12/31/1969 7:00:00 PM	2/4/2020 2:36:23 PM	2/4/2020 2:36:23 PM	1/1/1971 11:59:59 PM

Name	New Name	Taken (Original)
DSCN0001.JPG	DSCN0001 1966 02 01 - 23.59.59.JPG	1/1/1971 11:59:59 PM

Important -

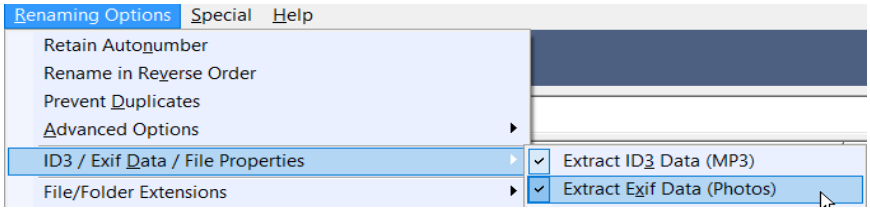
1. Originally, Contributor used 'Creation (Curr.)' as the Type. But this wouldn't work because of the limitation previously discussed (the 'Created' date displays as December 31, 1969 even though the date is actually February 2nd 1966 – see notation #2 above). But because the Poster wanted to use the 'Taken (Original)' date in the original request, this was used in place of the 'Creation (Curr)' and did work as specified.

Appendix

Renaming with Dates Prior to January 1, 1971 cont.

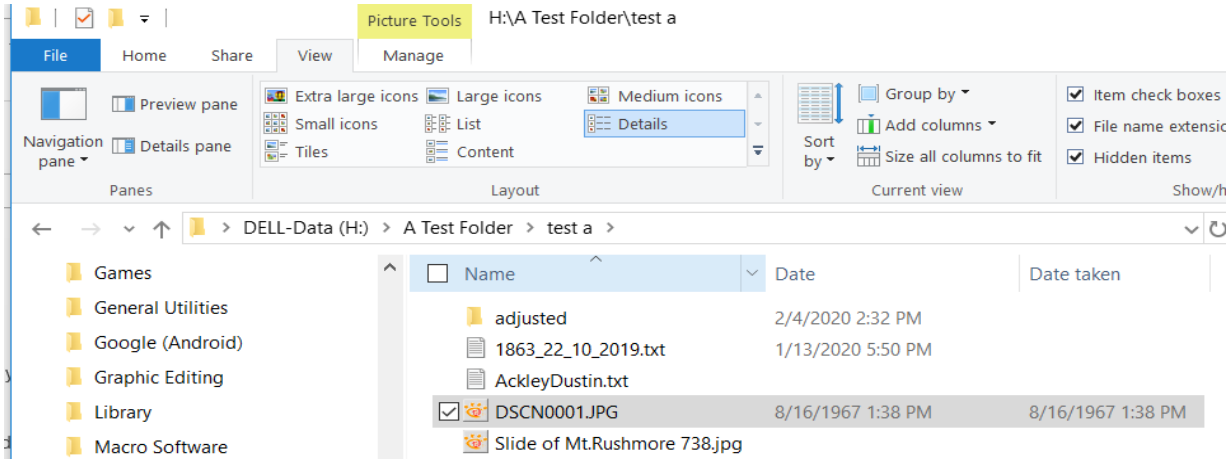
Tim Notes:

1. The work-around is to place the actual Date Taken into the text of the 'Custom' format string.
2. The 'Hours:', 'Minutes:' and 'Seconds' are accurate. This is extracted from the actual Exif data of 'Taken (Original)' and displayed in the 'Taken (Original)' column.
3. The 'Taken (Original)' date displayed in the column is not accurate and represents the oldest timestamp BRU is capable of listing, January 1, 1971.
4. The above work-around requires files that have Exif metadata and a value for 'Taken (Original)'.
5. Requires that 'Extract ID3 Data (Photos)' be set from the ID3 /Exif Data / File Properties sub menu of the Renaming Options Menu.



Procedure:

1. Use Windows Explorer to display the 'Date Taken' of the files.



2. Manually group these in batches by their date. Best to probably place them in separate directories.
   a. Make a note of the batch dates. These will be used in the 'Custom Format' date.
   b. This procedure is meant for renaming many files. The illustrated example above is just using one for simplicity.

3. Run the 'Section #8: Auto Date' using the ,'Type: Taken (Original)' and change the format string for each batch using the file date you recorded earlier in step 2a.

4. Rename the files and move to the next directory to process the next group of files.

5. Return to step 3 and repeat for each batch.

Forum Contributors: JST200

Appendix

Renaming with Dates Prior to January 1, 1971 cont.

Tim Notes:

1. The only real downside of this method is that this requires multiple runs for each file set with different dates.
2. This procedure is the way (and the only way) to go without using JavaScript.

Tim Contribution –

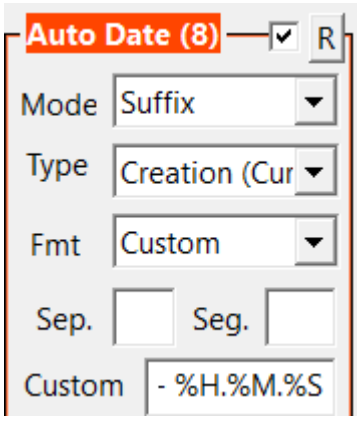
samples:

[Word1-Word2] 2010.ext	with a timestamp of February 2, 1966 8:51:14 AM
Slide of Mt.Rushmore 738.jpg	with a timestamp of December 31, 1969 7:00:00 PM
AckleyDustin.txt	with a timestamp of January 1, 1971 9:26:49 AM

Observations:

1. To use the above method on files that do not have Exif data and therefore do not have any value for 'Taken (Original)'.

1. 'Section #8: Auto Date' –



Mode: Suffix
Type: Creation (Curr.)
Fmt: Custom

Use a Custom of, i.e.,

1966 02 01 - %H.%M.%S
for batches of files with a date of February 2nd 1966

Name	New Name	Type	Size	Created
[Word1-Word2] 2010.ext	[Word1-Word2] 2010.ext	EXT File	6 KB	12/31/1969 7:00:00 PM

Didn't work.

The information was correct but these settings will not work. This is because BRU will not recognize any date before January 1, 1971. Therefore specifying the creation date will not match, even if the value is one day before:

Name	New Name	Type	Size	Created
Slide of Mt.Rushmore 738.jpg	Slide of Mt.Rushmore 738.jpg	JPG File	6 KB	12/31/1969 7:00:00 PM

Appendix

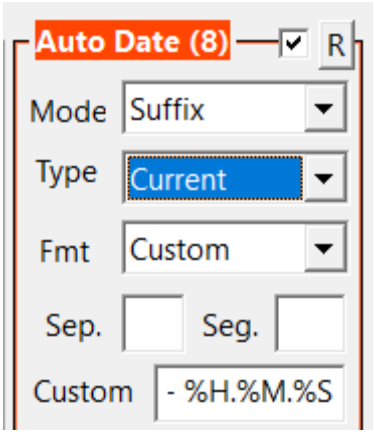
Renaming with Dates Prior to January 1, 1971 cont.

Also remember that although the test sample's actual creation date is February 2nd 1966, the column can only display a value of December 31, 1969 while 'Section #8: Auto Date' will only support files with a date of January 1, 1971 and higher. This will work on the AckeyDustin.txt file because the actual timestamp of the file is January 1, 1971:

Name	New Name	Type	Size	Created
AckleyDustin.txt	AckleyDustin 1966 02 01 - 09.26.49.txt	Text Document	6 KB	1/1/1971 9:26:49 AM

Here is the solution:

Since the date selected in 'Type' is not really being appended because the actual date which is to be appended is provided in a text form in the Custom formatting, use the, 'Type: Current', instead. 'Current' is recognized because it matches the current date always. No other Type listed will work.



Mode: Suffix
Type: Current
Fmt: Custom

Use a Custom of, i.e.,

1966 02 01 - %H.%M.%S
for batches of files with a date of February 2nd 1966

Now it matches against the file no matter it's original creation date. The only problem is that the 'hours: minutes: seconds:' that are extracted will not be accurate to the original timestamp because these do reflect the current timestamp.

This is the trade-off.

i.e., A file that had an original Timestamp of February 1, 1966, correctly displays in New Name -

Name	New Name	Type	Size	Created
[Word1-Word2] 2010.ext	[Word1-Word2] 2010 1966 02 01 - 09.33.54.ext	EXT File	6 KB	12/31/1969 7:00:00 PM

.. the hours, minutes and seconds represented by 09.33.54 are inaccurate. These reflect the timestamp of the current date – Today, and not the file's original timestamp. Therefore the %H,%M, and %S should not be used in the Custom formatting with this method, unless you add this to the custom text string itself.

1966 02 01 - 08.51.14 AM

But again, if you have a bunch of files to rename using that date, then it wouldn't make sense to have them all using the same minutes, hours and seconds. There would be no distinction between the timestamps.

Working with Folders (Directories)

Appendix

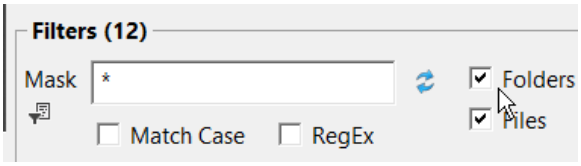
Tim's Tips & Tidbits:

Working with Folders (Directories)

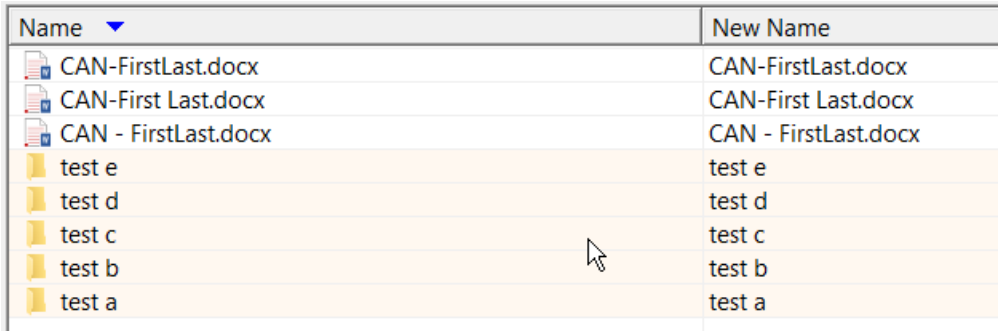
Although most of the examples presented here use files as the samples, BRU treats files and folders almost the same. To the File System (FAT32, exFAT, UDFS and NFTS), a directory is simply a file with a special attribute designating it as a directory. It can be named and renamed following the same naming conventions as a regular file. This means that if you need to work with folders or perform renaming operations on them, you can use BRU almost the same as you would with filenames. BRU also provides additional options that are specific to Folders.

1. To Work with Folders you have several options:

   a. 'Section #12: Filters'

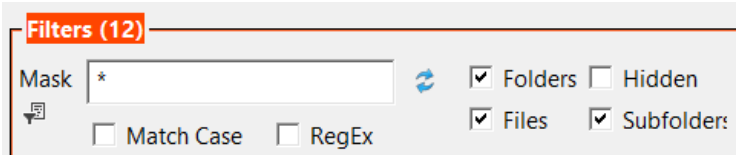


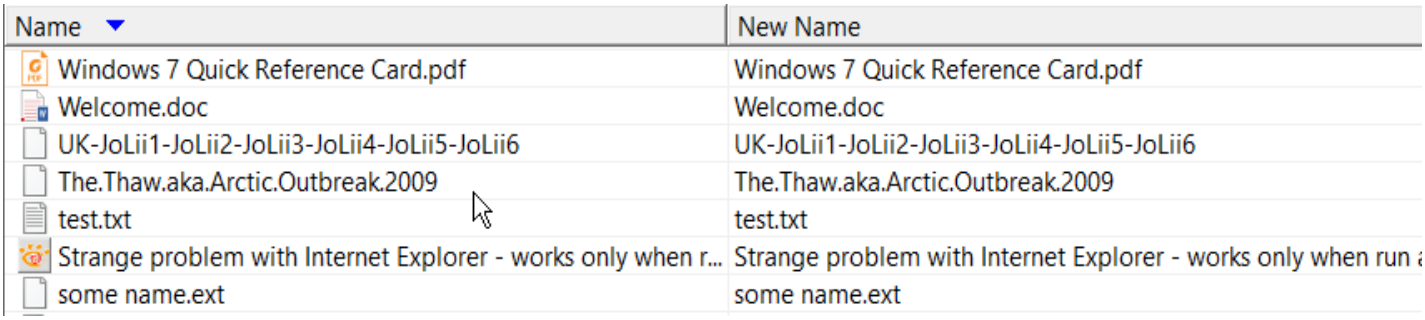
1. To see Folders in the Content Pane, enable 'Folders option'



Once enabled, you can see and select the file/folders to be renamed:

   b. If you want to work with both the folders and the files contained herein, enable the 'Subfolders' option. This turns on recursion.





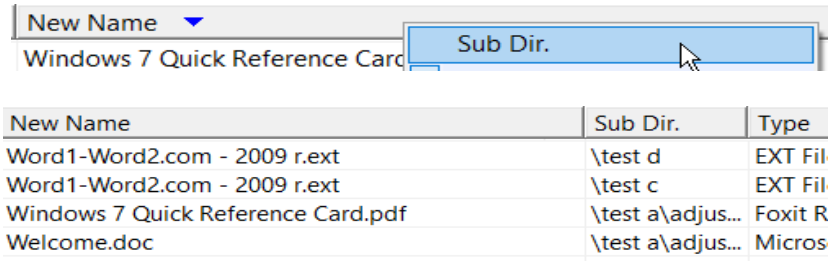
Source Material this section: https://docs.microsoft.com/en-us/windows/win32/fileio/naming-a-file?redirectedfrom=MSDN#maxpath

Appendix

Tim's Tips & Tidbits:

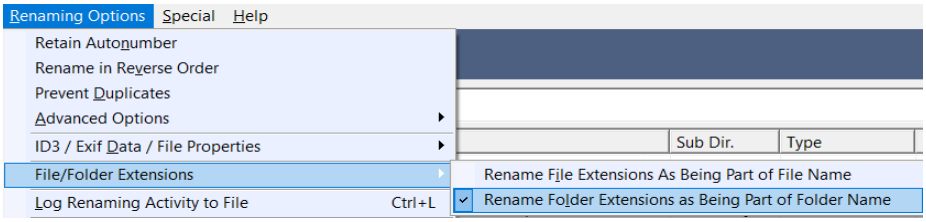
Working with Folders (Directories) cont.

c. Once recursion is on, you can add the 'Sub. Dir' column to the Content Pane. Do this by right clicking on the Title Bar and select, 'Sub.Dir'

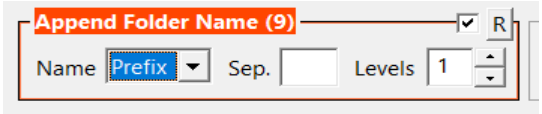


Alternatively, you can enable the Sub. Dir column from the 'Select Columns' option under the List submenu of the Display Options menu.

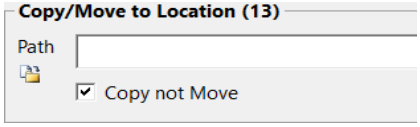
d. If a folder name has dot characters then it is recommended to enable the 'Rename Folder Extensions as Being Part of Folder Name' option from the File/Folder Extensions sub menu of the Renaming Options Menu. Fortunately, this option is enabled by default.



e. 'Section #9: Append Folder Name' allows you to add the current folder name and both subsequent and preceding folder levels from the current directory level.



f. 'Section #13: Copy/Move to Location'. This gets a lot of people confused. You cannot move folders containing files to a different location. It is mainly used as a destination for files *after* they have been renamed. Files can be copied or moved to a different location – *not* the directories themselves. It can, however create empty directories thereby creating a mirror directory structure (with no files). If you specify a directory and have 'Subfolders' enabled (recursion), then select the folder in the Content Pane and give it a location here, it will re-create the empty directory structure.



For further information, please refer to the appropriate sections in 'Volume I Bulk Rename Utility Operations Manual'

Long File Paths

Appendix

Tim's Tips & Tidbits:

Long File Paths

It has been a long 'Pain in the Butt' limitation of Windows not to support filepaths longer than 260 characters. This includes the folder path as well as the filename and extension. In today's modern methods of naming files and directories, this can easily become a problem. There have been some work-arounds over the years, and I will detail some of them, but Microsoft finally got around to addressing this very troublesome oversight, at least in part.

Beginning with Windows 10 version 1607, Windows can support long file paths in 'common win32 and directory functions' (Win32 and Windows Store Applications for example)– but you must enable this – it is not on by default.

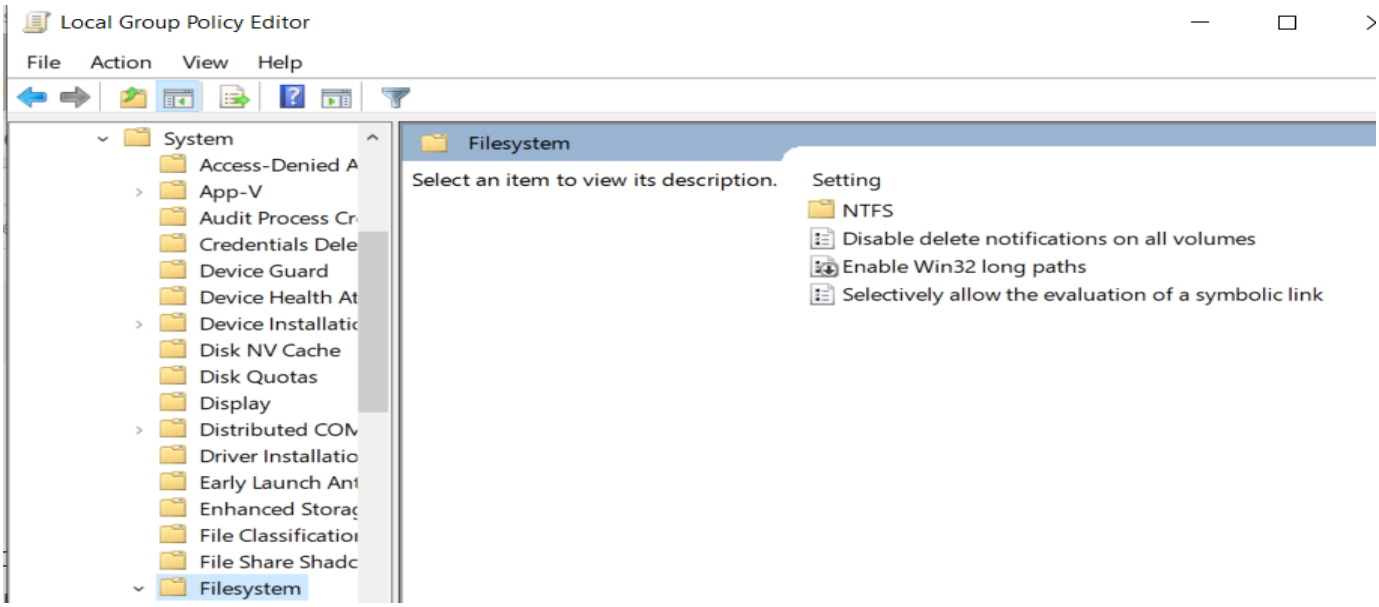
In Windows Home, you can set a registry key:

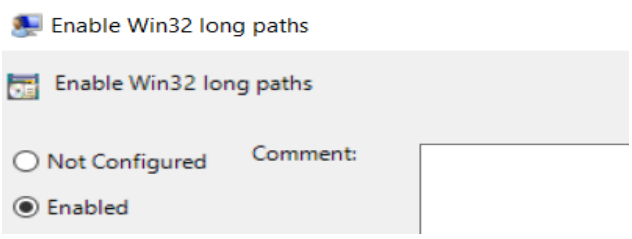
HKLM\SYSTEM\CurrentControlSet\Control\FileSystem LongPathsEnabled

Type: REG_DWORD set to '1'

If the key doesn't exist, you will need to create it. If you don't know how to do this or you don't understand how to use the Registry, then don't bother. If you don't know what you are doing, you don't belong poking around in the Windows Registry where you can mess up your system if you are not careful.

In Windows Pro, you enable the option through Group Policy:





Double Click on the 'Enable Win32 long paths' entry and click on Enable then click 'OK' to save.

Appendix

Tim's Tips & Tidbits:

Long File Paths cont.

The problem is that I have found, it still has limitations especially with the long file paths I routinely use on my system. Another limitation is only 3rd party programs that support long file names will benefit from having this setting.

First the historical work-arounds:

Unicode and UNC (Universal Naming Convention) –

You can designate a filepath and bypass all filepath limitations (32,767 characters) by using a syntax of:

Unicode:

\\?\D:\very long path

UNC:

\\?\UNC\server\share

Where:

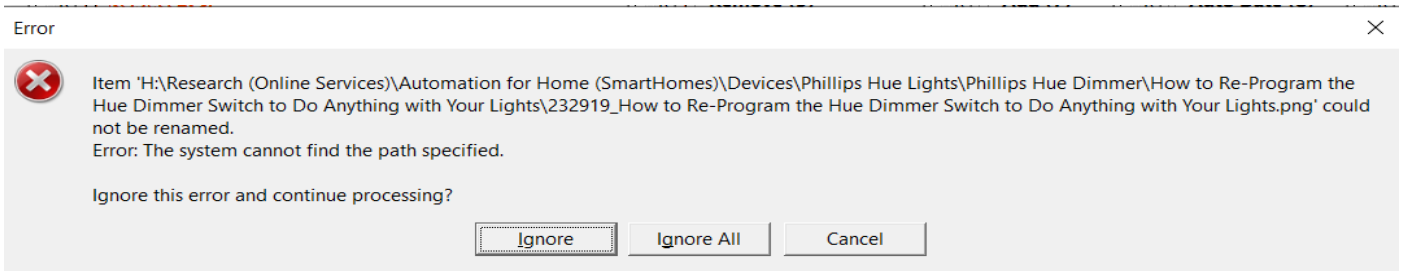
server	the name of the computer
share	is the name of the shared folder.

This sends the request straight to the File System completely bypassing the OS.

Notes:

1. These prefixes are not used as part of the path itself.
2. Path must be absolute (full paths) not relative (current directory path). Relative paths will have the filepath limitation applied. BRU at the present time does not display the **full** pathname.
3. Forward slashes to represent path separators are not allowed.
4. Use of a period to represent the current directory, or double dots to represent the parent directory are not allowed.
5. **BRU doesn't support UNC or UNICODE File path designations at this time.**
6. BRU **doesn't support Long File Paths**. A previous edition of 'Volume I Bulk Rename Utility Operations Manual' misstated that it does. This has been corrected in later revisions.

   Although BRU can display filenames with paths that exceed 260 characters, an error will be generated if you try to rename the file:



Appendix

Tim's Tips & Tidbits:

Long File Paths cont.

Using Substituted Drives:

1. Substituting a drive typically thought of in terms of network drives, can be used to map a path to a location on a drive and assign a drive letter. Access to that location is treated by Windows as accessing a separate drive. In doing so, any path for accessing that location is replaced by that drive designation. However, paths leading from that designation are counted toward the accumulated total of the path length.

i.e.,

File name:
232919_How to Re-Program the Hue Dimmer Switch to Do Anything with Your Lights.png

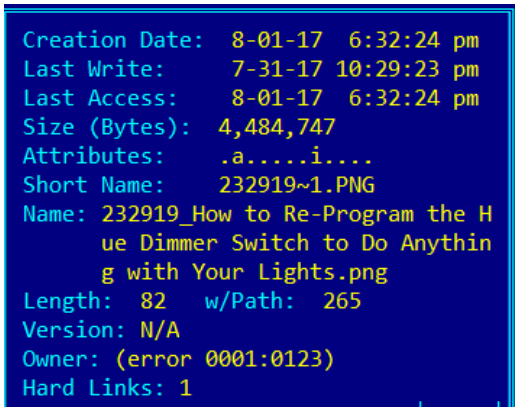
Folder Path:

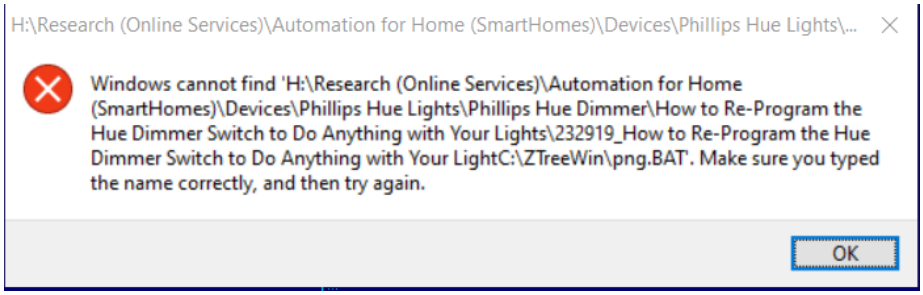
H:\Research (Online Services)\Automation for Home (SmartHomes)\Devices\Phillips Hue Lights\Phillips Hue Dimmer\How to Re-Program the Hue Dimmer Switch to Do Anything with Your Lights\

Full Path:

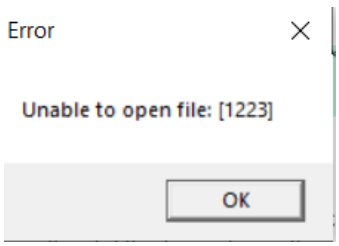
H:\Research (Online Services)\Automation for Home (SmartHomes)\Devices\Phillips Hue Lights\Phillips Hue Dimmer\How to Re-Program the Hue Dimmer Switch to Do Anything with Your Lights\ 232919_How to Re-Program the Hue Dimmer Switch to Do Anything with Your Lights.png

If I try to access, I get an error. The stats on the file show a full path length of 265 characters, exceeding the 260 limit:





Tim Notes:



1. Windows Explorer can open the file because it supports long file paths (or it may just send the 'Short Name' - unknown). The example above is to illustrate the error one may encounter with programs that do not, or if your OS is not configured for Long File Path support.

2. The exact error messages you get may vary depending on the program used to access.

Appendix

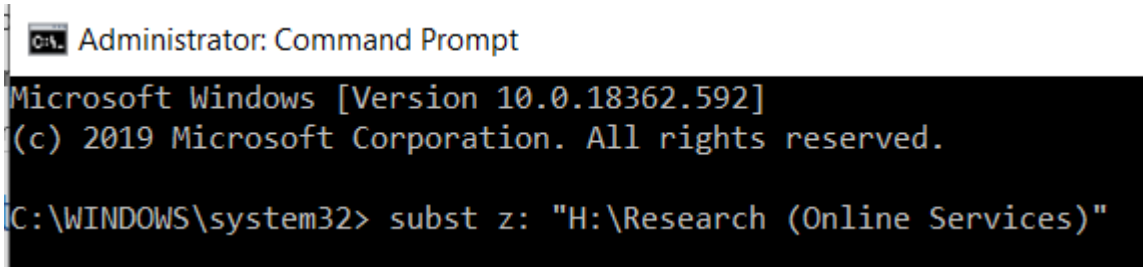
Tim's Tips & Tidbits:

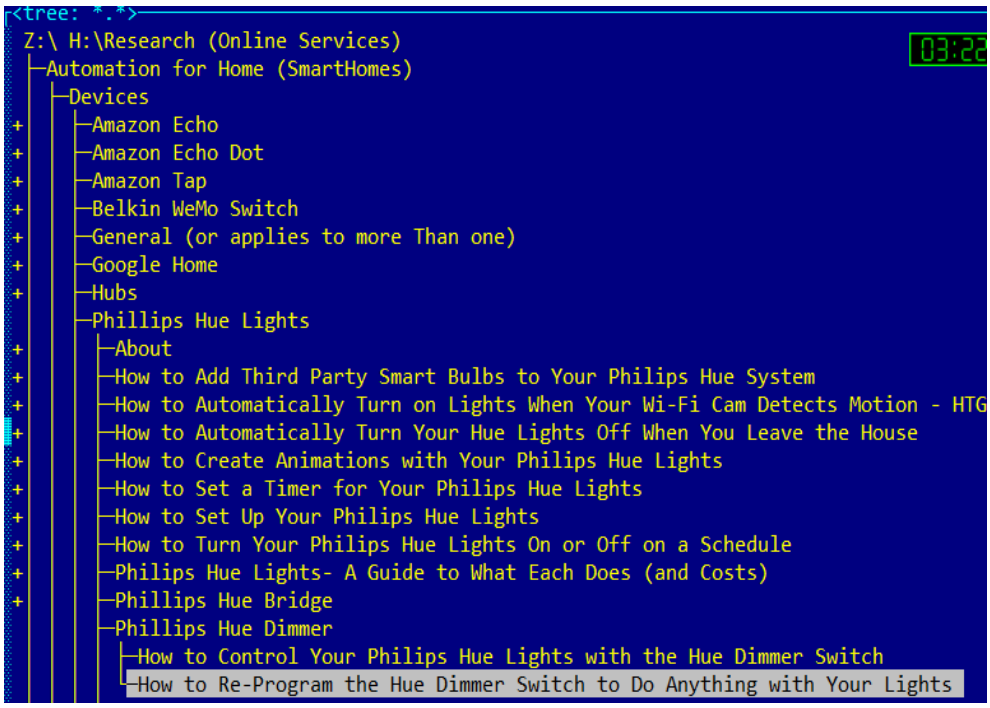
Long File Paths cont.

Using Substituted Drives: cont.

Using the Substitute command from an administrator Command prompt:

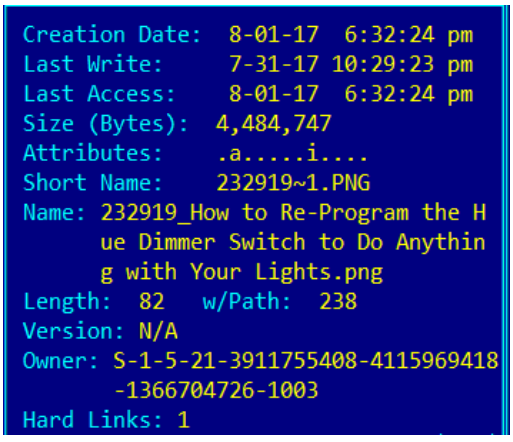
subst Z: "H:\Research (Online Services)"





You can see from this File Manager that Windows now sees the substituted path as Drive Z:

The stats on the path show a length of 238 characters so the file is accessible.



To detach the Z: drive use the following command from an Adminstrative Prompt:

Subst z: /d

Tim Notes:

1. BRU supports any method that assigns a drive as a substitution for a file path

Appendix

Tim's Tips & Tidbits:

Long File Paths cont.

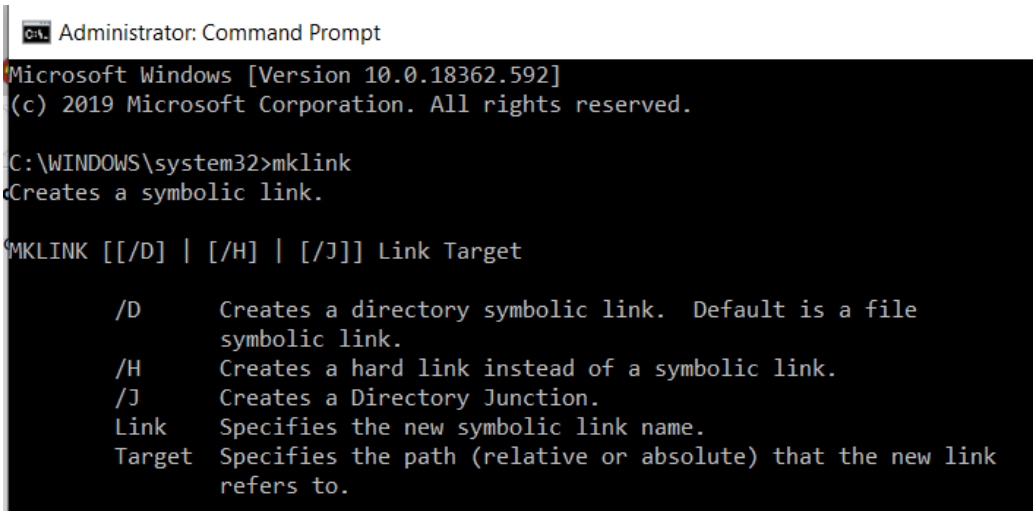
Symbolic Links (Hard and Soft Links)

Similar to the Subst command are symbolic links—also known as symlinks—that point to a file or folder on your system. There are two kind of symbolic links: Hard and soft.

Soft symbolic links work similarly to a standard shortcut. When you open a soft link to a folder, you will be redirected to the folder where the files are stored.

Hard links makes it appear as though the file or folder actually exists at the location of the symbolic link, and your applications won't know any better. That makes hard symbolic links more useful in most situations.

The links are created using the mklink command:



Without any extra options, mklink creates a symbolic link to a file.

mklink Link Target

Soft Link:

Use /D when you want to create a soft link pointing to a directory:

mklink /D Link Target

Appendix

Tim's Tips & Tidbits:

Long File Paths cont.

Symbolic Links (Hard and Soft Links) cont.

Hard Links:

Use /H when you want to create a hard link pointing to a file:

mklink /H Link Target

Use /J to create a hard link pointing to a directory, also known as a directory junction:

mklink /J Link Target

i.e.,

mklink /J "C:\Link To Folder" "C:\Users\Name\Original Folder"

Deleting Symbolic Links:

1. To delete a symbolic link to a file (created with /H):
   a. The normal del key from Windows Explorer or another File Manager or the command del <linkName> from a command prompt will delete the link without affecting the target file.
2. To delete a symbotic link to a directory (created with /D):
   a. It works just like an empty folder. You need to use rmdir from a command prompt or use Delete from Windows Explorer or other File Manager to remove it. The target directory will be unaffected.

i.e.,

H:\Research (Online Services)

* On another drive, or the same drive, H:, create a junction directory to that folder:

mklink /J "H:\research – Test Junction" "H:\Research (Online Services)"

* The command prompt will show that the folder on E is a "JUNCTION" when you do a directory listing.
* To remove the junction directory on E, type:

rmdir "H:\Research (Online Services)"

IMPORTANT

- Make sure you have the correct drive letter.
- Make sure you are removing the Junction and NOT the actual file/folder

Appendix

Tim's Tips & Tidbits:

Long File Paths cont.

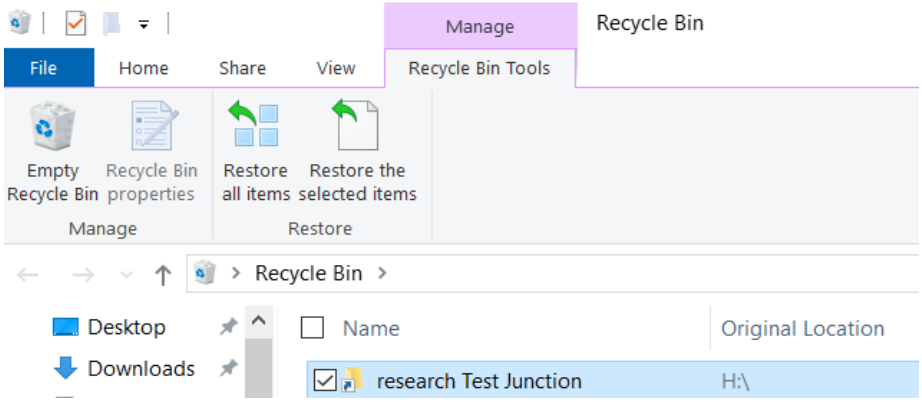
Symbolic Links (Hard and Soft Links) cont.

Notes:

1. If you use the 'del' command from the Command Prompt on the link or junction, then it works just like on any other folder –
   a. Prompt to confirm whether you want to delete all the directory's files (but not subdirectories).
   b. The target directory's files will be deleted if you answer "y".
      1. You can avoid the prompt if you use del /q.

```
C:\Windows\System32>del /?
Deletes one or more files.

DEL [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names
ERASE [/P] [/F] [/S] [/Q] [/A[[:]attributes]] names

  names         Specifies a list of one or more files or directories.
                Wildcards may be used to delete multiple files. If a
                directory is specified, all files within the directory
                will be deleted.

  /P            Prompts for confirmation before deleting each file.
  /F            Force deleting of read-only files.
  /S            Delete specified files from all subdirectories.
  /Q            Quiet mode, do not ask if ok to delete on global wildcard
  /A            Selects files to delete based on attributes
  attributes    R  Read-only files            S  System files
                H  Hidden files               A  Files ready for archiving
                I  Not content indexed Files  L  Reparse Points
                O  Offline files              -  Prefix meaning not

If Command Extensions are enabled DEL and ERASE change as follows:

The display semantics of the /S switch are reversed in that it shows
you only the files that are deleted, not the ones it could not find.
```

2. In Windows Explorer, if you delete the link or junction, only that link or junction is removed; the target folder is unaffected.
   a. This also applies to the Recycle Bin. When moving the link or junction, you are prompted to confirm that you want to move the "folder", but actually, only the link or junction will be put in the Recycle Bin, not the target folder.



… and can be restored is need be.

Appendix

Tim's Tips & Tidbits:

Long File Paths cont.

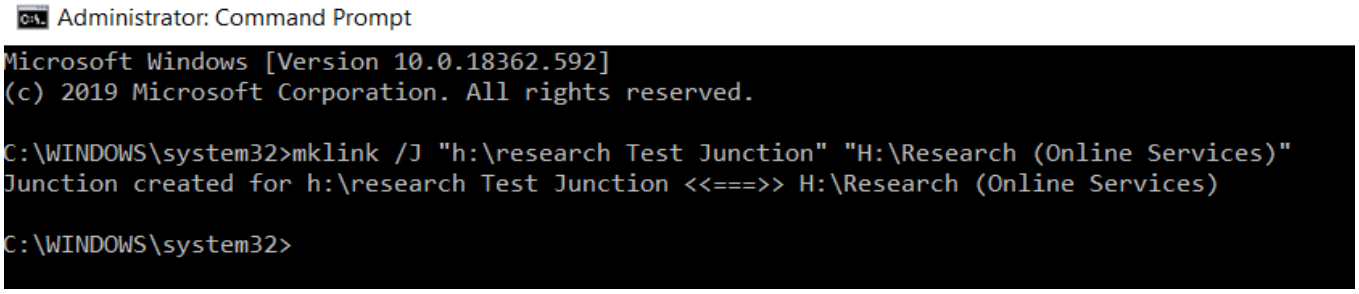
Symbolic Links (Hard and Soft Links) cont.

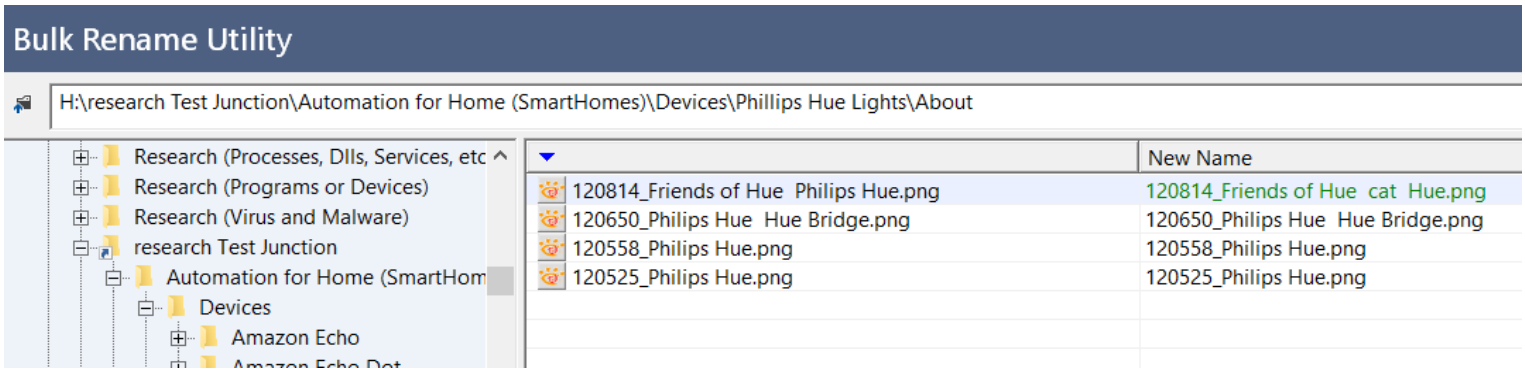
3. If you DELETE the ACTUAL target file or folder by accident or otherwise, any symbolic links or junctions pointing to it will REMAIN, but unusable until the target is recreated.
   a. Any hard links to deleted files, or to files in deleted directories, will be unaffected.

4. There remain arguments over whether Junctions are considered Symbolic links or not – I don't get into this discussion.

Notes:

1. BRU supports the use of Links or Junctions





Subst Vs Symbolic Links

1. The Subst command creates a virtual drive letter for that path.
2. Links are pointers to a location and are not virtual drives.

Source This Section: answers.microsoft.com/en-us/windows/forum/windows_vista-files/how-to-remove-a-symbolic-link/e171f76a-a62b-4604-8777-3439492bf1f7,
www.howtogeek.com/howto/16226/complete-guide-to-symbolic-links-symlinks-on-windows-or-linux/
www.sevenforums.com/tutorials/278262-mklink-create-use-links-windows.html

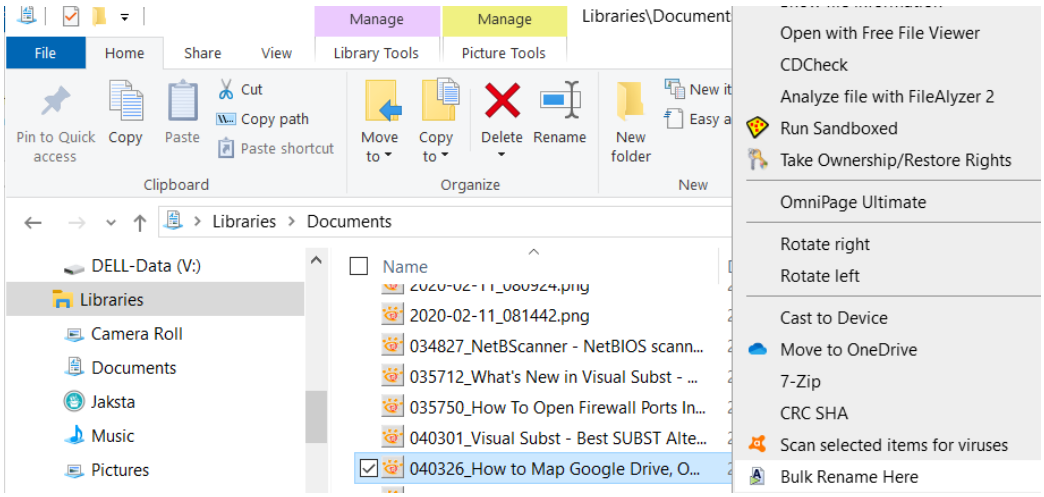
Privileges
Administrator/User Access

Appendix

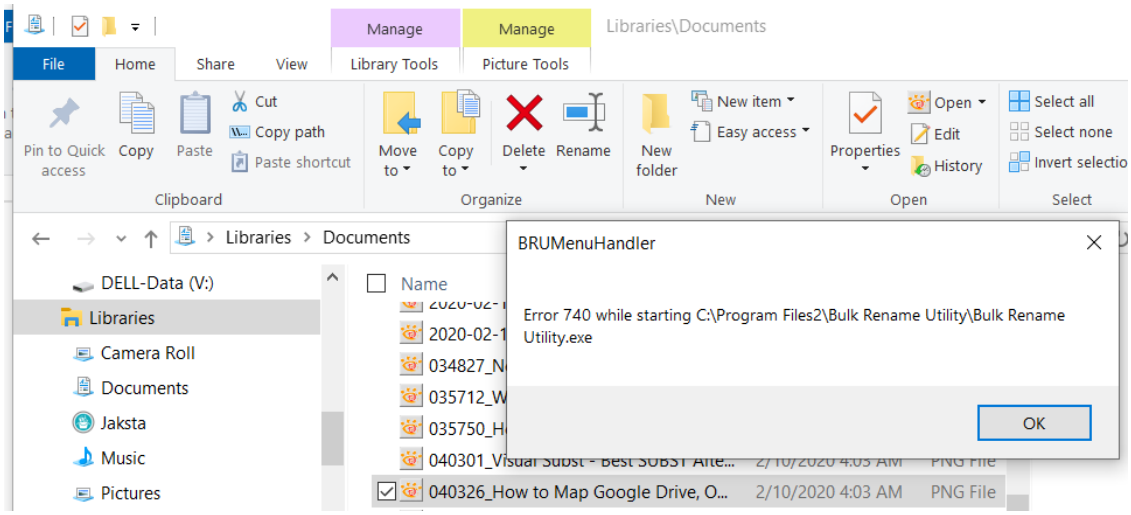
Tim's Tips & Tidbits:

Error 740 with 'Bulk Rename Here' from Windows Explorer

While selecting 'Bulk Rename Here' from Windows Explorer Context menu (right click on file)…



Generates an Error 707.
This means that the Requested Operation Requires Elevation.



Reason:

1. You have Bulk Rename Utility configured to run as Administrator. This may have been done in order to facilitate access to files and directories that might otherwise require special permissions access.

   Windows allows access under two privilege levels – Elevated and Non-Elevated with Non-Elevated or more commonly referred to as 'User Access' being the default. The Elevated Access or more commonly referred to as 'Administrator Access' requires specific permissions. These permissions are controlled by UAC (User Account Control).

Appendix

Tim's Tips & Tidbits:

Error 740 with 'Bulk Rename Here' from Windows Explorer cont.

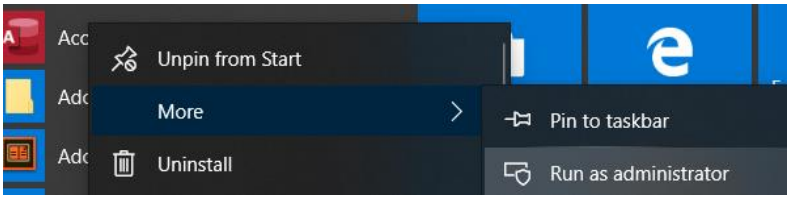
Administrator Access can be granted from different sources:

a. Windows itself
– when in Explorer, you try to access or peform some other operation, Windows may bring up a dialog allowing you access if you authorize it (provided that you have Administrator rights on the current system).

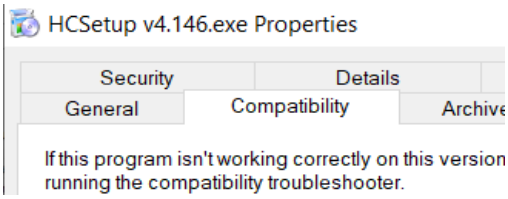
b. Within a Program (naturally – no option)
– the program has administrator rights built into it – so when it starts, it automatically brings up the UAC dialog.

c. By setting the program to optionally start with Administrator rights within the program itself.
– Many programs through their options or preferences provide the ability to run in Administrator Mode.

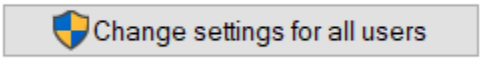
d. In the Start Menu
– If you right click on an item in Windows Start Menu and select 'More', you have the option to run that program with Elevated Privilege.



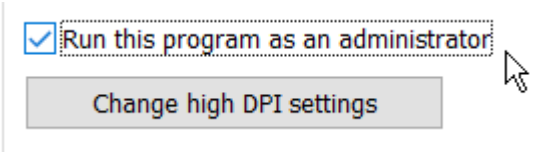
e. By configuring the executable to run Elevated
– Right click on the executable, and bring up the 'Properties' and select the 'Compatibility' tab.



In the 'Compatibility' tab, click on 'Change Settings for All Users'



From 'Compatibility for All Users' select 'Run This Program as an Administrator.

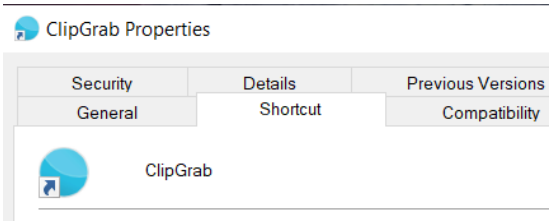


Appendix

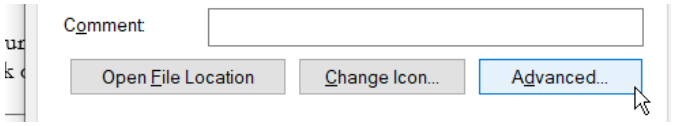
Tim's Tips & Tidbits:

Error 740 with 'Bulk Rename Here' from Windows Explorer cont.

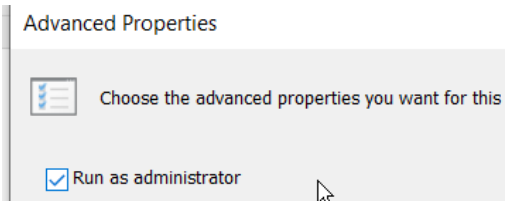
f. By configuring the shortcut to the executable to run Elevated.
– Right click on the shortcut icon bring up the 'Properties' and select the 'Shortcut' tab.



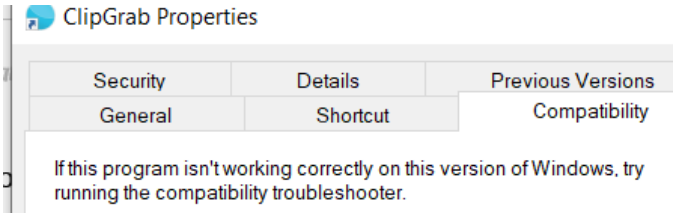
Select 'Advanced'



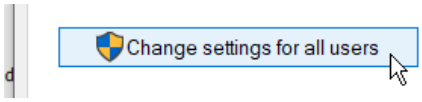
Select 'Run as Administrator'



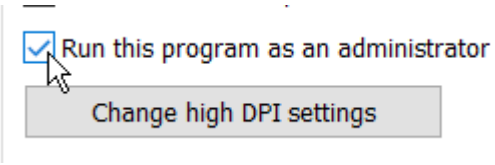
Click Ok to go back and select 'Compatibility' tab.



Select 'Change Settings for All Users'



Select 'Run this Program as Administrator'



The problem is that Windows views the privileges as two separate users – one with elevated privileges and one without. Access to programs, files, etc. is controlled by these privileges and this in turn is controlled through the UAC (User Account Control). Many functions of Windows only require normal User Access and this is where the conflict comes into play.

The Context menus of Windows Explorer, for example, require only User Access. When you try to run a program that is elevated (has Administrator rights), an error is generated – Error 704. Windows is indicating that YOU do not have the elevated privilege to access this program. If BRU is set to run Elevated, and you try to access BRU through the 'Bulk Rename Here' context menu – you get the error (or something similar).

Solution – Re-Configure BRU to NOT run with Administrator Access using one or more of the methods outlined.

Appendix

Tim's Tips & Tidbits:

Remove 'Bulk Rename Here' from Windows Explorer Context Menu

1. Open Administrator Command Prompt:

2. regsvr32 /u "c:\Program Files\Bulk Rename Utility\BRUHere.dll"

Tim Notes:

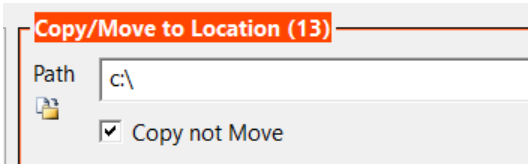
1. Substitute your path to the .dll file if different. Place the path in quotations if there are any <spaces> in the path name.

Appendix

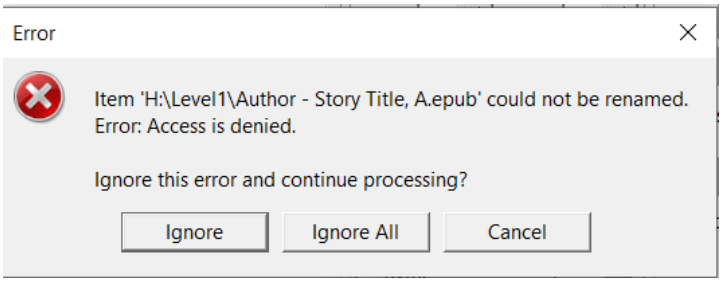
Tim's Tips & Tidbits:

Copy/Move Files to the C:\ Root Directory

When using the 'Section #13: Copy/ Move to Location', if you enter into the path, the C:\ root directory



generates this error:



Access is Denied.

Windows does not allow Copying, deleting any moving files into the root directory of C: unless the program you are using has Elevated privileges – Run As Administrator. By default, BRU runs as a Normal User (without elevated privileges). To copy/move files into the C: root directory, restart BRU with Administrator privileges.

Tim Notes:

1. For further information, refer to, 'Error 740 with 'Bulk Rename Here' from Windows Explorer', in this section.

A Word on Pad Alpha

Appendix

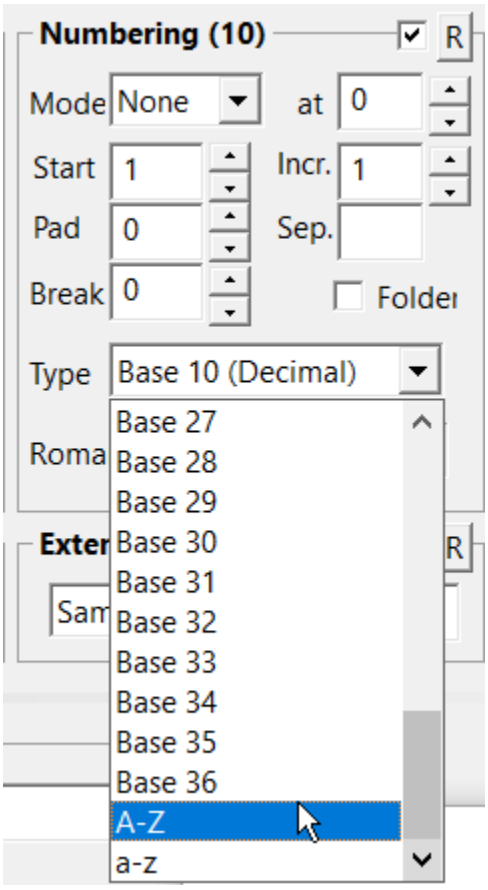
A Word on Pad Alpha

From my research on the forums this seems to be complicated for users to understand. One reason is because it doesn't work as Jim Willsher, the original author of BRU and BRC intended. It has an issue (aka 'bug').

Alpha is available through the 'Section #10: Numbering' -

There are two – 'A-Z', representing uppercase, and 'a-z', representing lowercase (no option for mixed).

Using:



Mode: Suffix
Start: 1
Incr: 1
Pad: 0

From Start –

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -B.jpg
wq - dr 00005 3700014_3_o -C.jpg
wq - dr 00006 3700014_4_o -D.jpg
..
wq - dr 00028 3700054_11_o -Z.jpg
wq - dr 00029 3700054_12_o -AA.jpg
wq - dr 00030 3700054_13_o -AB.jpg
wq - dr 00031 3700054_14_o -AC.jpg
..
wq - dr 00054 3656446_6_o -AZ.jpg
wq - dr 00055 3656446_7_o -BA.jpg
wq - dr 00056 3656446_8_o -BB.jpg
wq - dr 00057 3656446_9_o -BC.jpg
wq - dr 00058 3656446_10_o -BD.jpg
wq - dr 00059 3656446_11_o -BE.jpg

This will continue on to what I refer to as 'Alpha Infinity' …. aa - az, aaa - aaz, aaaa - aaaz, etc.

I will use a Separator of '<space> <hyphen> <space' for the remainder of these examples.

Tim Notes:

1. As of BRU v3.320, this problem remains unresolved. Use the work-arounds as detailed in the following pages.
2. Pad Alpha was originally coined by a user/Contributor. Sounded kind of SciFi so I kept it.

Appendix

A Word on Pad Alpha cont.

Increasing the Start just moves it to the next alpha character:

Start = 2 Pad = 0 Increment = 1 First file = wq - dr 00003 3700014_1_o -B.jpg

Increasing the Increment value will skip over the alpha characters much the same as it does if these were numerical digits:

Start = 1 Pad = 0 Increment = 2

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -C.jpg
wq - dr 00005 3700014_3_o -E.jpg
wq - dr 00006 3700014_4_o -G.jpg
wq - dr 00007 3700014_5_o -I.jpg

Start = 1 Pad = 0 Increment = 3

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -D.jpg
wq - dr 00005 3700014_3_o -G.jpg
wq - dr 00006 3700014_4_o -J.jpg
wq - dr 00007 3700014_5_o -M.jpg

It is the Pad function that has the issue with **Start = 0**. Here is the problem, and the work-around.

Most users expect that with Start at Zero, the first file would not be renamed and subsequent files would.

Start = 0 Pad = 0 Increment = 1

wq - dr 00003 3700014_1_o -.jpg
wq - dr 00004 3700014_2_o -A.jpg
wq - dr 00005 3700014_3_o -B.jpg
wq - dr 00006 3700014_4_o -C.jpg
wq - dr 00007 3700014_5_o -D.jpg

This is the expected behavior. The first file would not be renamed (except that the Separator was used so it does show up in New Name).

Watch, though, what happens when Pad is increased.

Pad = 1

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -A.jpg
wq - dr 00005 3700014_3_o -B.jpg
wq - dr 00006 3700014_4_o -C.jpg
wq - dr 00007 3700014_5_o -D.jpg

Pad = 2

wq - dr 00003 3700014_1_o -AZ.jpg
wq - dr 00004 3700014_2_o -AA.jpg
wq - dr 00005 3700014_3_o -AB.jpg

Pad = 3

wq - dr 00003 3700014_1_o -AZZ.jpg
wq - dr 00004 3700014_2_o -AAA.jpg
wq - dr 00005 3700014_3_o -AAB.jpg
wq - dr 00006 3700014_4_o -AAC.jpg

Pad = 4

wq - dr 00003 3700014_1_o -AZZZ.jpg
wq - dr 00004 3700014_2_o -AAAA.jpg
wq - dr 00005 3700014_3_o -AAAB.jpg
wq - dr 00006 3700014_4_o -AAAC.jpg

etc.

Appendix

A Word on Pad Alpha cont.

The first file is messed up everytime.
So why use Pad at all?

When Pad is set at zero, the alpha padding does work ranging from A-Z to infinity. But you don't always want to start at 'A'. Padding is a 'shortcut' path to getting to a certain place in the alpha sequence.

For example:

Pad 3 = Start file as filenameAAA.ext

But with Pad (Alpha) at any value other than zero, this won't work as expected. Instead use a Start value of one or more.

Here is how to use Pad Alpha with the issue – what works and what doesn't.

Start = 0 Samples

Start = 0 Pad = 0 Increment = 1

wq - dr 00003 3700014_1_o -.jpg
wq - dr 00004 3700014_2_o -A.jpg
wq - dr 00005 3700014_3_o -B.jpg
wq - dr 00006 3700014_4_o -C.jpg

Start = 0 Pad = 1 Increment = 1

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -A.jpg
wq - dr 00005 3700014_3_o -B.jpg
wq - dr 00006 3700014_4_o -C.jpg

Start = 0 Pad = 2 Increment = 1

wq - dr 00003 3700014_1_o -AZ.jpg
wq - dr 00004 3700014_2_o -AA.jpg
wq - dr 00005 3700014_3_o -AB.jpg
wq - dr 00006 3700014_4_o -AC.jpg

Start = 0 Pad = 3 Increment = 1

wq - dr 00003 3700014_1_o -AZZ.jpg
wq - dr 00004 3700014_2_o -AAA.jpg
wq - dr 00005 3700014_3_o -AAB.jpg
wq - dr 00006 3700014_4_o -AAC.jpg

As Already discussed, the above doesn't work

Appendix

A Word on Pad Alpha cont.

Start = 1 Samples

Start = 1 Pad = 0 Increment = 1

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -B.jpg
wq - dr 00005 3700014_3_o -C.jpg
wq - dr 00006 3700014_4_o -D.jpg

Start = 1 Pad = 1 Increment = 1

wq - dr 00003 3700014_1_o -A.jpg
wq - dr 00004 3700014_2_o -B.jpg
wq - dr 00005 3700014_3_o -C.jpg
wq - dr 00006 3700014_4_o -D.jpg

Start = 1 Pad = 2 Increment = 1

wq - dr 00003 3700014_1_o -AA.jpg
wq - dr 00004 3700014_2_o -AB.jpg
wq - dr 00005 3700014_3_o -AC.jpg
wq - dr 00006 3700014_4_o -AD.jpg

Start = 1 Pad = 3 Increment = 1

wq - dr 00003 3700014_1_o -AAA.jpg
wq - dr 00004 3700014_2_o -AAB.jpg
wq - dr 00005 3700014_3_o -AAC.jpg
wq - dr 00006 3700014_4_o -AAD.jpg

Interesting to note that there is no difference between Pad = 0 and Pad = 1 with Start = 1

Start = 2 Samples

Start = 2 Pad = 0 Increment = 1

wq - dr 00003 3700014_1_o -B.jpg
wq - dr 00004 3700014_2_o -C.jpg
wq - dr 00005 3700014_3_o -D.jpg
wq - dr 00006 3700014_4_o -E.jpg

Start = 2 Pad = 1 Increment = 1

wq - dr 00003 3700014_1_o -B.jpg
wq - dr 00004 3700014_2_o -C.jpg
wq - dr 00005 3700014_3_o -D.jpg
wq - dr 00006 3700014_4_o -E.jpg

Start = 2 Pad = 2 Increment = 1

wq - dr 00003 3700014_1_o -AB.jpg
wq - dr 00004 3700014_2_o -AC.jpg
wq - dr 00005 3700014_3_o -AD.jpg
wq - dr 00006 3700014_4_o -AE.jpg

Start = 2 Pad = 3 Increment = 1

wq - dr 00003 3700014_1_o -AAB.jpg
wq - dr 00004 3700014_2_o -AAC.jpg
wq - dr 00005 3700014_3_o -AAD.jpg
wq - dr 00006 3700014_4_o -AAE.jpg

Again, there is no difference between Pad = 0 and Pad = 1

Appendix

A Word on Pad Alpha cont.

Start = 3 Samples

Start = 3 Pad = 1 Increment = 1

wq - dr 00003 3700014_1_o -C.jpg
wq - dr 00004 3700014_2_o -D.jpg
wq - dr 00005 3700014_3_o -E.jpg
wq - dr 00006 3700014_4_o -F.jpg

Start = 3 Pad = 2 Increment = 1

wq - dr 00003 3700014_1_o -AC.jpg
wq - dr 00004 3700014_2_o -AD.jpg
wq - dr 00005 3700014_3_o -AE.jpg
wq - dr 00006 3700014_4_o -AF.jpg

Start = 3 Pad = 3 Increment = 1

wq - dr 00003 3700014_1_o -AAC.jpg
wq - dr 00004 3700014_2_o -AAD.jpg
wq - dr 00005 3700014_3_o -AAE.jpg
wq - dr 00006 3700014_4_o -AAF.jpg

Start = 3 Pad = 4 Increment = 1

wq - dr 00003 3700014_1_o -AAAC.jpg
wq - dr 00004 3700014_2_o -AAAD.jpg
wq - dr 00005 3700014_3_o -AAAE.jpg
wq - dr 00006 3700014_4_o -AAAF.jpg

Established there is no difference between Pad = 0 and Pad = 1 so I started with Pad = 1

So there you have it – how to Use Pad Alpha under 'Section #10: Numbering'

Summary:

1. Don't bother with Start = 0. Doesn't work.
2. Use Pad = 1 with Start = 1 or greater (Pad = 0 not required)

Selection Issue in Content Pane using BRU

Appendix

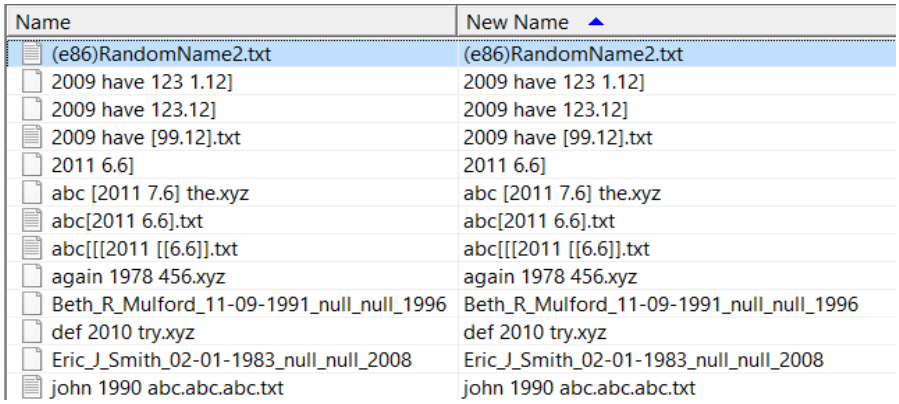
Selection Issue in Content Pane using BRU

There is an ongoing issue with selecting files in the Content Pane. This is not the issue with Drag and Drop that has already been discussed in both Volumes. This has to do with the selection and deselection of entries using either the Shift-Down Arrow Consecutive method or the Ctrl-Mouse Click for individual Non-Consecutive selection.

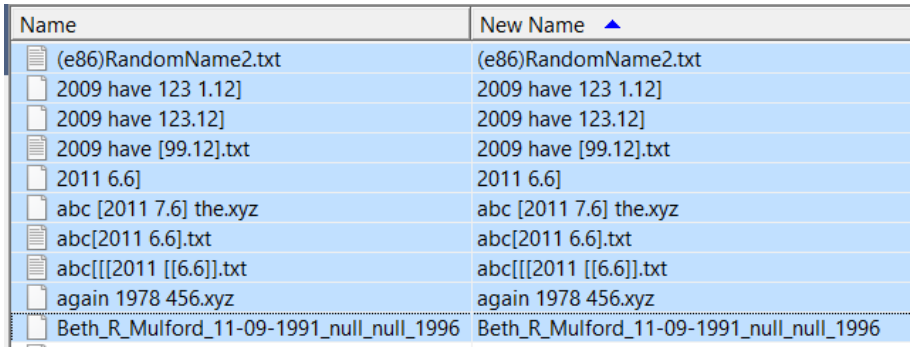
Without any Criteria Selected

Using the Shift + Down/Up Arrow for Consecutive Selection

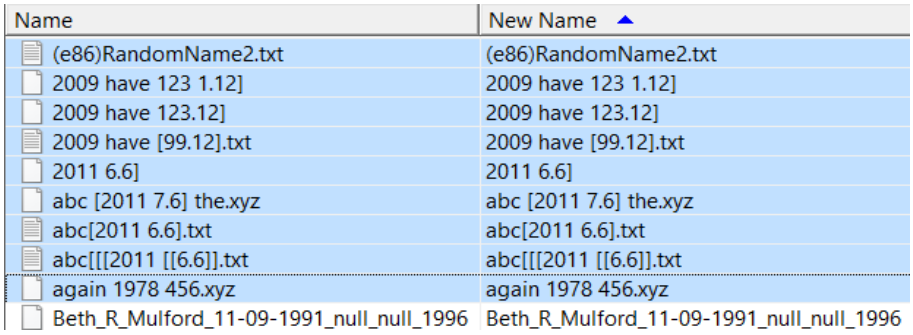
Panel a: Before selection



Panel b: Using Shift + Down Arrow for Consecutive selection



Panel c: Using the Shift + Up arrow to remove highlight from last entry



No problem.

Appendix

Selection Issue in Content Pane using BRU

With Criteria Selected - New Name is active:

Using the Shift + Down/Up Arrow for Consecutive Selection

Panel a: Using Shift + Down Arrow for Consecutive selection

Name	New Name
(e86)RandomName2.txt	RandomName2.txt___AZ0001
2009 have 123 1.12]	2009 have 123 1.12]___AZ0002
2009 have 123.12]	2009 have 123.12]___AZ0003
2009 have [99.12].txt	2009 have [99.12].txt___AZ0004
2011 6.6]	2011 6.6]___AZ0005
abc [2011 7.6] the.xyz	abc [2011 7.6] the.xyz___AZ0006
abc[2011 6.6].txt	abc[2011 6.6].txt___AZ0007
abc[[[2011 [[6.6]].txt	abc[[[2011 [[6.6]].txt___AZ0008
again 1978 456.xyz	again 1978 456.xyz___AZ0009
Beth_R_Mulford_11-09-1991_null_null_1996	Beth_R_Mulford_11-09-1991_AZ0010_null_1996

Panel b: Using the Shift + Up arrow to remove last entry from selction.

Name	New Name
(e86)RandomName2.txt	RandomName2.txt___AZ0001
2009 have 123 1.12]	2009 have 123 1.12]___AZ0002
2009 have 123.12]	2009 have 123.12]___AZ0003
2009 have [99.12].txt	2009 have [99.12].txt___AZ0004
2011 6.6]	2011 6.6]___AZ0005
abc [2011 7.6] the.xyz	abc [2011 7.6] the.xyz___AZ0006
abc[2011 6.6].txt	abc[2011 6.6].txt___AZ0007
abc[[[2011 [[6.6]].txt	abc[[[2011 [[6.6]].txt___AZ0008
again 1978 456.xyz	again 1978 456.xyz
Beth_R_Mulford_11-09-1991_null_null_1996	Beth_R_Mulford_11-09-1991_null_null_1996

Problems detected:

#1. Last entry deselected as expected.
#2. The next entry above this **has also been deselected** even though the blue highlight remains.

Let's examine this problem further.

Tim Notes:

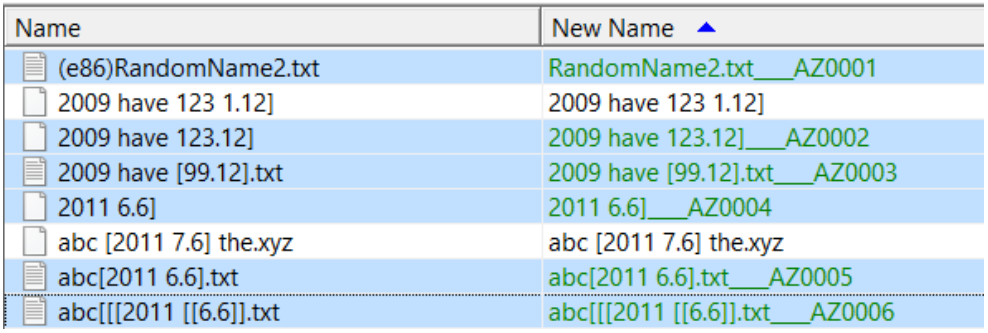
1. This has been resolved in BRU v3.320 but I leave the original text for those that are running older versions.

Appendix

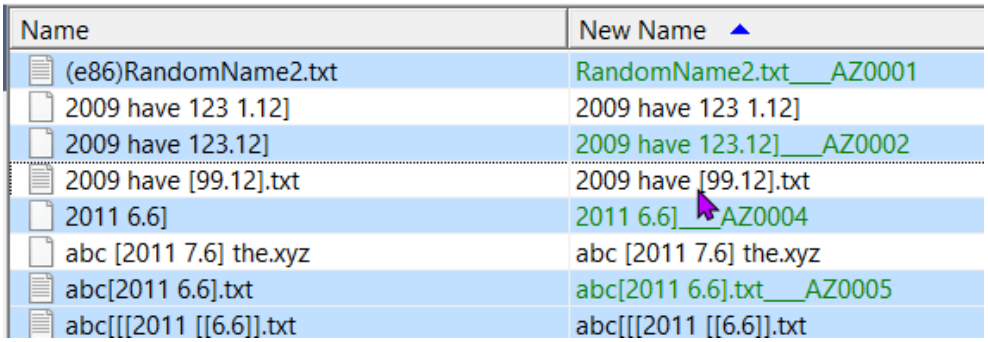
Selection Issue in Content Pane using BRU

With Criteria Selected - New Name is active: cont.

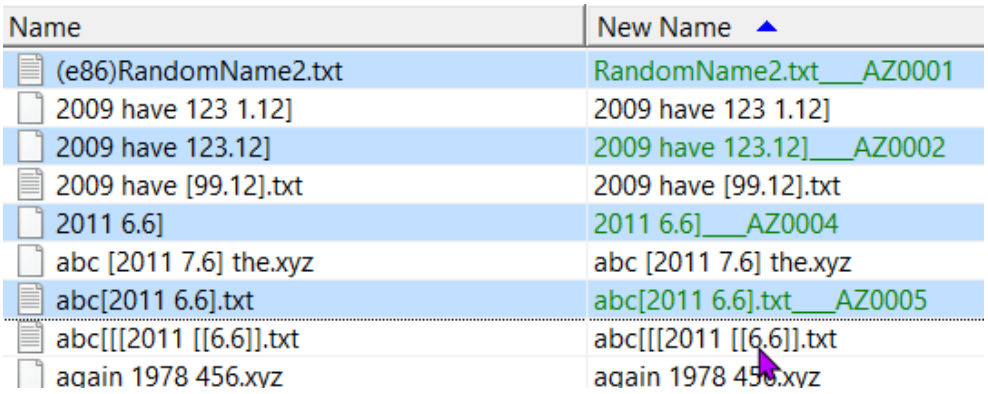
Using Ctrl + Mouse for Non-Consecutive Selection



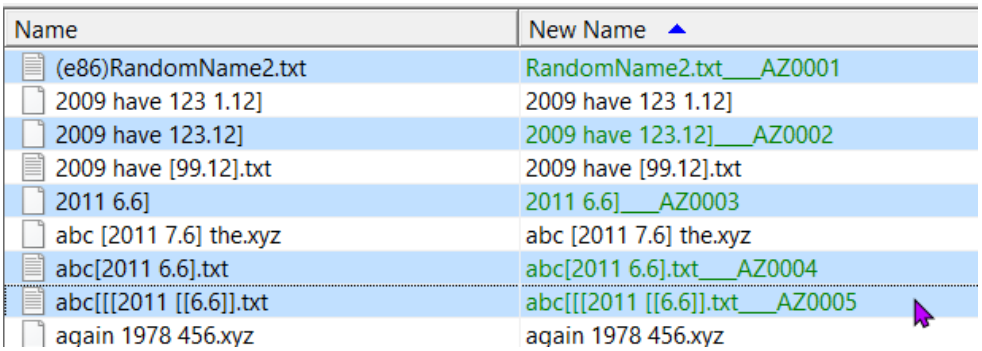
Panel a:
Files have been selected individually using Ctrl + Mouse Click



Panel b:
Deselecting the file '2009 have [99.12].txt' has also resulted in deselecting 'abc[[[2011 [[66.6]].txt' even though it remains highlighted. Same problem as before.



Panel c:
Clicking on the deselected entry to re-establish the selection only removed the highlight. The file remains unselected.



Panel d:
Clicking on it a second time will select the file again. Other files unaffected this time.

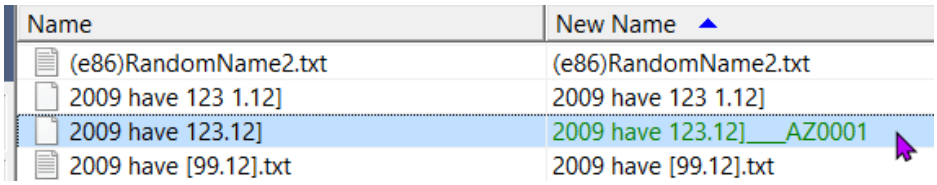
Appendix

Selection Issue in Content Pane using BRU

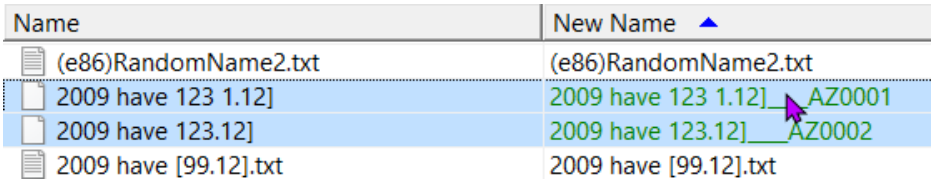
With Criteria Selected - New Name is active: cont.

Using Ctrl + Mouse for Non-Consecutive Selection cont.

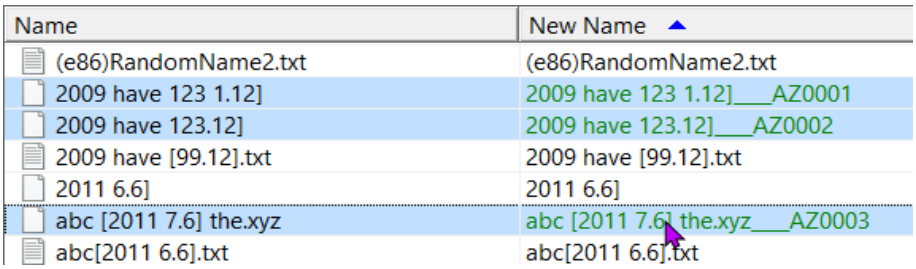
Another example: this time I will track each order of selection



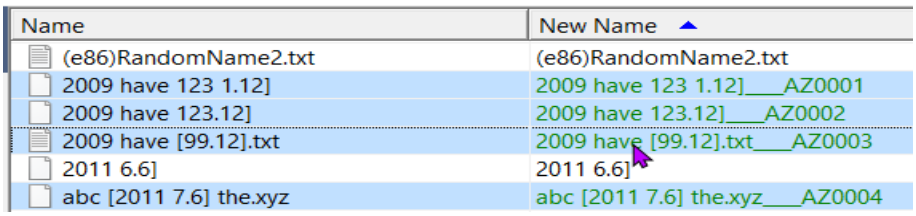
Selection #1: 2009 have 123.12]



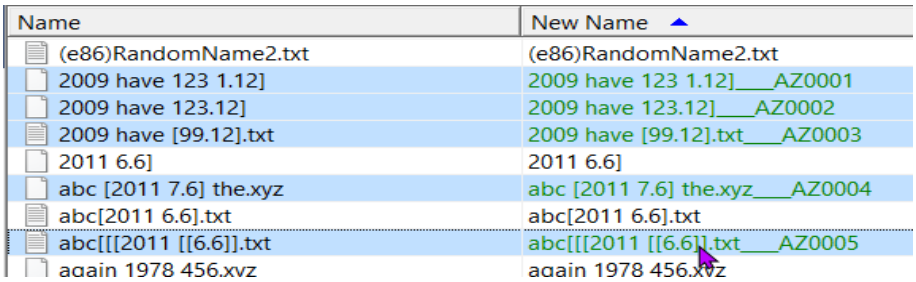
Selection #2: 2009 have 123 1.12]



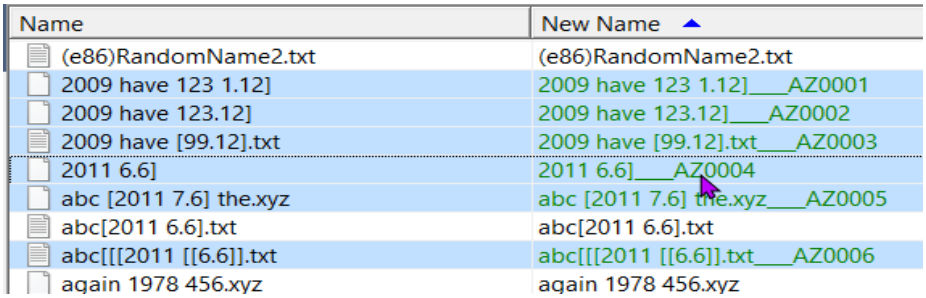
Selecton #3: abc [2011 7.6] the.xyz



Selection #4: 2009 have [99.12].txt



Selection #5: abc[[[2011 [[6.6]].txt



Selection #6: abc[2011 6.6].txt

Appendix

Selection Issue in Content Pane using BRU

With Criteria Selected - New Name is active: cont.

Using Ctrl + Mouse for Non-Consecutive Selection cont.

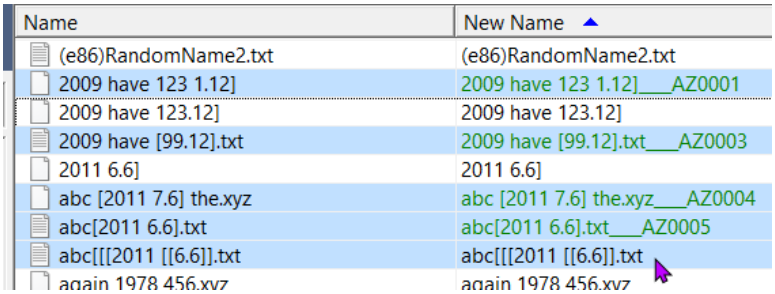
Now I will deselect one entry that was NOT the last entry selected.

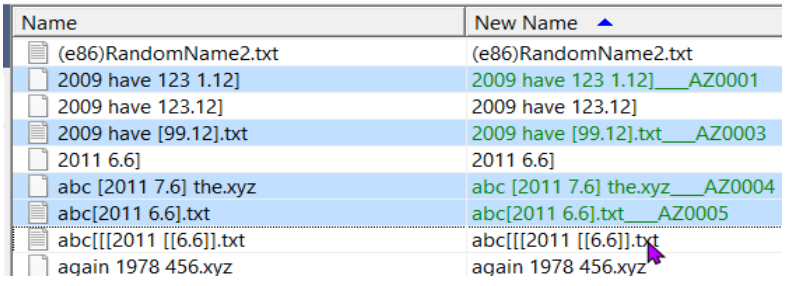
Name	New Name
(e86)RandomName2.txt	(e86)RandomName2.txt
2009 have 123 1.12]	2009 have 123 1.12]___AZ0001
2009 have 123.12]	2009 have 123.12]
2009 have [99.12].txt	2009 have [99.12].txt___AZ0003
2011 6.6]	2011 6.6]
abc [2011 7.6] the.xyz	abc [2011 7.6] the.xyz___AZ0004
abc[2011 6.6].txt	abc[2011 6.6].txt___AZ0005
abc[[[2011 [[6.6]].txt	abc[[[2011 [[6.6]].txt
again 1978 456.xyz	again 1978 456.xyz

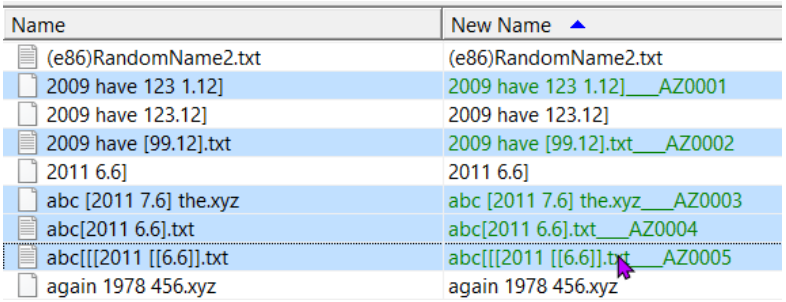
It is as I suspected. The file that is inadvertently deselected is the LAST file selected. When I deselected the file, '2009 have 123.12]' , it also deselected the file, 'abc[2011 6.6].txt'. This is the same behavior exhibited when using the Shift Down-Arrow/ Up arrow selection but this was not as obvious because that always used consecutive selection and the Ctrl + Mouse Click method employs Non-Consecutive Selection.

Conclusion:

Watch for this when manually selecting files. The work-around is to simply select the file again by using Ctrl-Mouse to click on it TWICE. The first time will deselect it the rest of the way – remove the highlight, and the second time will select it again.







Regular Expressions (RegEx) Manual Volume II

RegEx Manual Volume II

Also refer to page 88, 'A Note on Replacement Strings**'** –

Literal Text Match

A literal text match 'is' case sensitive and matches anywhere in the string.

Example: Search for the string 'email'.

Match: (email)
Replace: \1 <space>

e-mail	e-mail
Email	Email
femail	email
G EMAILings	G EMAILings
many emails	email
many_emails	many_emails

Matches any occurrence of email lowercase.

Tim Notes:

1. Searches are case sensitive but refer to Volume I for an option to make a search case insensitive.

The <dot> Metacharacter (aka Wildcard)

The ' . ' dot character can match against any single character – and I stress *any*.

It is used in many instances in RegEx most commonly together with a Greedy or Non-Greedy (Lazy) metacharacter as in..

.* or .? or .+

Example: Use a <dot> metacharacter to search for any one character before the string "ubject".

Match: (.ubject)
Replace: \1<space>

Fubject	Fubject
Subject	Subject
subjective	subject
subjects	subject
subtext	subtext
subversive	subversive

Matches against any character in a filename that contains the string 'ubject', so it matches the 'F' in 'Fubject', the 'S' in 'Subject', the 's' in 'subjective' and the 's' in 'subjects'. Because a Capture Group is used to contain the expression, New Name reflects the matched character as well as the string, 'ubject'.

Tim Notes:

1. This document is a supplement to Volume I. It does not supplant or reproduce information found there. Please refer to Volume I for additional information.

RegEx Manual Volume II

The <dot> Metacharacter (aka Wildcard) cont.

If the Match was changed to:

Match: (.ubject.*)
Replace: \1<space>

Fubject	Fubject
Subject	Subject
subjective	subjective
subjects	subjects
subtext	subtext
subversive	subversive

.. the New Name would reflect the entire string, i.e., 'subjective' and 'subjects'. This is because adding .* at the end indicates to match against the remainder of the string to EOL (End of Line).

Example: Use three dots "..." to search for any three characters within a string.

Match: (t...s)
Replace: \1<space>

t-4-s	t-4-s
t1234s	t1234s
t345s	t345s
tEENs	tEENs
trees	trees
twentys	twentys

The filename must begin with a 't', have three characters following – any three characters, followed by an 's'. This matches against 't '-4- 's' and 't' 345 's' as well as 't' een 's' and 't' ree 's' , but 'twentys' doesn't match the pattern nor does 't1234s' because although it starts with a 't' and finishes with an 's', there are 4 characters in between, '1234' not three.

Example: Use several wildcards to match characters throughout a string.

Match: (.a.a.a)
Replace: \1<space>

3a4a5a	3a4a5a
aaa	aaa
aaaa	aaaa
alabama	labama
banana	banana
Canada	Canada
Mata Hari	Mata Hari
Gama Rays	Gama Rays

The filename must contain three 'a' characters to match - but more than this - it establishes a pattern - and that is what RegEx is about - patterns. For the RegEx to work, the expression must follow an established pattern. The pattern is established by the format of the filenames to be renamed.

If the RegEx is too restrictive it may be limited to a specific pattern that may not match against all of the filenames you wish to include. If it is too broad - what I refer to as universal, it may match against filenames you do not want to include. In that case, you can either revise the RegEx to be more restrictive to the pattern or be more selective when you choose which files to rename in the Content Pane (the File List Window) or use a filter that limits the selection of files.

RegEx Manual Volume II

The <dot> Metacharacter (aka Wildcard) cont.

The pattern here is.. <any character> 'a' <any character> 'a' <any character> 'a'

In this example, it matches against '3' a '4' a '5' a as well as 'alabama', 'banana' and Canada. It does not match against 'Mata Hari' or 'Gama Rays' because although both filenames contain three 'a' characters, they do not match the established pattern.

Alabama is interesting because it is made up of:

a l a b a m a

The RegEx matches against the first a, - any character represented by the <dot> but the next character expected is another 'a'. Instead the next character is an 'l' so the evaluation continues and the RegEx engine moves forward through the string.

It then matches against the 'l' starting the match process again, and next it matches against the 'a', the 'b', the 'm' and the final 'a'. The expression is satisfied, the Evaluation of the expression = TRUE, and the result of the expression appears in the New Name column as 'labama', cutting off the first character because it didn't match.

A Word on Tokens

This word in RegEx gets thrown around a lot. I find it a bit confusing myself. In most respects it refers to Capture Groups – Capture Group is a token represented by a backreference. In other respects, some refer to it as the elements or sub-expressions that make up the expression within a capture group. While still others may refer to the individual metacharacters as tokens. Perhaps it is meant to be a generic name, perhaps not. There are also references to tokens within a string:

i.e.,

A1B1C1

where A1 and B1 and C1 are three separate tokens.

Go ahead and look up Tokens on the internet under RegEx –

Because of this, I have deliberately not used the word token but tried to be more explicit to what I am referencing at the time. – an element, a metacharacter or a Capture Group, or a specific part of a string that makes up either the filename in question or a string/sub-string/sub-expression within the RegEx expression.

So, I'll leave 'Tokens' to the Subway Transit System at least for now.

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy (Non-Greedy)

Although the RegEx Manual found in the Appendix of Volume I is very good if I do say so myself, it is not perfect. I wrote it at a time when I was first learning the language. Since then I have become more adept (How you ask? By reading through both of these volumes and studying the examples and performing the analysis of how they work.), but I still yield my knowledge to others. This book displays that. Below are some RegEx concepts that I will attempt to better explain for new users and introduce some new ones.

A Word About * Vs +

Both are Greedy – meaning they will match against as many characters as they can. The difference is how they match.

*

Matches Zero or more instances of the preceding expression (repeats). If you use * it will still match if there are no instances of it. Eats everything in sight.

For example if you have .* it will continue to match (repeat) each character until it reaches either EOL (the entire string) or another expression follows it.

+

Matches one or more instances of the preceding pattern (repeats). If you use a + there must be at least one instance of the pattern.

? Quantifier

Repeats the Greedy Quantifier once or not at all. Also called 'Non-Greedy' because it takes the Greedy operator and makes it 'Lazy', instead of repeating it more than once, it repeats it once and once only or not at all. This means that typically A Greedy Operator such as * will match as many as possible whether or not a match to the expression exists. The +, also Greedy will also match as many as possible as long as there is at least one match to the expression. By applying the ? Quantifier, it limits the * and the + to once or not all. Use of this, most always results in backtracking.

Tim Notes:

1. + Shorthand for {1,}
2. * Shorthand for {,}
3. ? Shorthand for {0,1} (made optional)

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

The ' ? ' Explained

?

Greedy.
Makes the preceding expression optional Greedy because the optional item is included in the match if possible.

Example:

string = ab
abc

Match: (abc?)
Replace: \1 <space>

Name	New Name
ab	ab
abc	abc

??

Lazy.
Makes the preceding expression optional. Lazy because the optional item is excluded in the match if possible.

Example:

string = ab
abc

Match: (abc??)
Replace: \1 <space>

Name	New Name
ab	ab
abc	ab

RegEx Manual Volume II

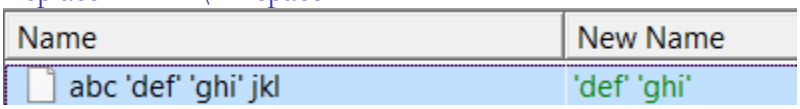
Additional RegEx Information on Greedy and Lazy cont.

The ‘ ? ‘ Explained cont.

*? Lazy.
Makes the Greedy Quantifier Lazy. The Greedy Quantifier will match zero or more times. Making it Lazy will allow it to ‘skip’ over the first match

String = abc 'def' 'ghi' jkl

Match: ('.*')
Replace: \1 <space>



Tim Analysis:

1. ‘	Matches against left single apostrophe. This is the first apostrophe preceding ‘def’	Capture Group 1 = ‘
2. .	Match against any character	Capture Group 1 = ‘d
3. *	Make it Greedy. RegEx Engine is backtracking from here since it has reached EOL.	Capture Group 1 = ‘def’ ‘ghi’ jkl
4. ’	Matches against right single apostrophe Backtracks to match against the right apostrophe of ‘ghi’.	Capture Group 1 = ‘def’ ‘ghi’

Summary:

The Greedy Quantifier matched against all possible matches in the string.

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

The ‘ ? ‘ Explained cont.

*? Lazy. cont.

String = abc 'def' 'ghi' jkl

Match: ('.*?')
Replace: \1

Name	New Name
abc 'def' 'ghi' jkl	'def'

Tim Analysis:

1. ‘	Matches against left single apostrophe. This is the first apostrophe preceding ‘def’	Capture Group 1 = ‘
2. .	Match against any character	Capture Group 1 = ‘d
3. *	Make it Greedy. RegEx Engine is backtracking from here since it has reached EOL.	Capture Group 1 = ‘def’ ‘ghi’ jkl
4. ?	But not too Greedy Backtracks back to the apostrophe before ‘def’	Capture Group 1 = ‘
5. ’	Matches against right single apostrophe The RegEx Engine moves forward to match against the right single apostrophe of ‘def’	Capture Group 1 = ‘def’

Summary:

The Lazy Quantifier backtracking, ‘skipped’ over the first match, the ‘ghi’ in the string, and matched against the second match, ‘def’ in the string.

RegEx Manual Volume II

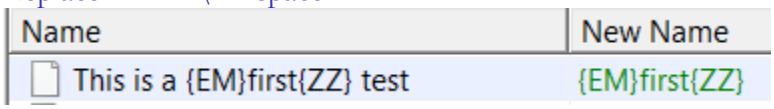
Additional RegEx Information on Greedy and Lazy cont.

The ' ? ' Explained cont.

+? Lazy.
Makes the Greedy Quantifier Lazy. The Greedy Quantifier will match once or more times. Making it Lazy will allow it to 'skip' over the first match

String = This is a {EM}first{ZZ} test

Match: ({.+})
Replace: \1 <space>



Tim Analysis:

1. {	Matches against the first left curly brace of '{EM}'	Capture Group 1 = {
2. .	Match against any character	Capture Group 1 = {E
3. +	Make it Greedy	Capture Group 1 = {EM}first{ZZ} test
	Current position = EOL (End of Line)	
4. }	Matches against the right curly brace	Capture Group 1 = {EM}first{ZZ}

Summary:

By Making the <dot> Greedy, it matched one or more times. So it matched from the first left curly brace to the end of the string. Adding the match against the right curly brace, it backtracked and matched against the first right curly brace it encountered, which is the right curly brace of '{ZZ}', that caused Capture Group 1 to give up the '<space> test' characters.

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

The ' ? ' Explained cont.

+? Lazy. cont.

String = This is a {EM}first{ZZ} test

Match: ({.+?})
Replace: \1 <space>

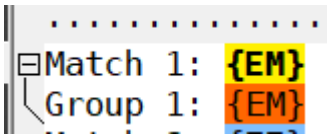
Name	New Name
This is a {EM}first{ZZ} test	{EM}

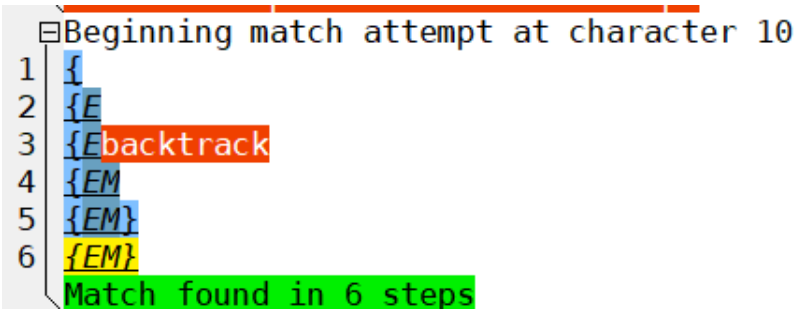
Tim Analysis:

1. {	Matches against the first left curly brace of '{EM}'	Capture Group 1 = {
2. .	Match against any character	Capture Group 1 = {E
3. +	Make it Greedy	Capture Group 1 = {EM}first{ZZ} test
	Current position = EOL (End of Line)	
4. ?	But not too Greedy RegEx Engine is backtracking.	Capture Group 1 = {E
5. }	Matches against the right curly brace RegEx Engine is moving forward.	Capture Group 1 = {EM}

Summary:

By Making the <dot> Greedy, it matched one or more times. So it matched from the first left curly brace to the end of the string. By making the Quantifier Lazy, it made the last match optional, and backtracked to the match of the <dot> against the 'E' of '{EM]'. Adding the match against the right curly brace, it moved forward to match against the right curly brace of '{EM]'.





RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

The ' ? ' Explained cont.

Tim Notes:

1. In both of these examples, the RegEx Engine backtracked. Here is an alternative to obtaining the same outcome but with no backtracking.

String = This is a {EM}first{ZZ} test

Match: ({[^}]+})
Replace: \1 <space>

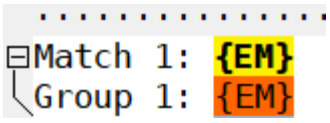
Name	New Name
This is a {EM}first{ZZ} test	{EM}

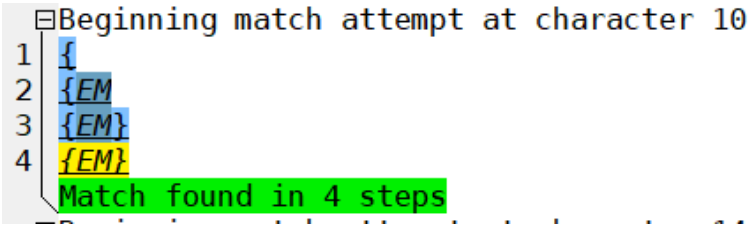
No backtracking using a Negated Class.

Tim Analysis:

1. {	Matches against the first left curly brace of '{EM}'	Capture Group 1 = {
2. [^}]	Match against a Negated Class consisting of a right curly brace.	Capture Group 1 = {E
3. +	Make it Greedy	Capture Group 1 = {EM
4. }	Match against the right curly brace	Capture Group 1 = {EM}

No Backtracking.





Source Material this section: regular-expressions.info

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

The ' ? ' Explained cont.

More Examples -

String = Calendars packing slip 09-29-11.pdf

Replace: \1 <space>

Match: (pa)

Name	New Name
Calendars packing slip 09-29-11.pdf	pa

Match: (pa.)

Name	New Name
Calendars packing slip 09-29-11.pdf	pac

Match: (pa?)

Name	New Name
Calendars packing slip 09-29-11.pdf	pa

Match: (pa??)

Name	New Name
Calendars packing slip 09-29-11.pdf	p

Match: (pa.*)

Name	New Name
Calendars packing slip 09-29-11.pdf	packing slip 09-29-11.pdf

Match: (pa.*?)

Name	New Name
Calendars packing slip 09-29-11.pdf	pa

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

The ' ? ' Explained cont.

More Examples – cont.

String = Calendars packing slip 09-29-11.pdf

Replace: \1 <space>

Match: (pa.+)

Name	New Name
Calendars packing slip 09-29-11.pdf	packing slip 09-29-11.pdf

Match: (pa.+?)

Name	New Name
Calendars packing slip 09-29-11.pdf	pac

Tim Analysis of the last example:

1. pa	Matches against 'pa' of 'packing'	Capture Group 1 = pa
2. .	Match against any character	Capture Group 1 = pac
3. +	Make it Greedy	Capture Group 1 = packing slip 09-29-11
	Current position = EOL	
4. ?	But not too Greedy	Capture Group 1 = pac
	Backtracks back to the 'c' of 'pac'	

Summary:

The ' ? ' made the previous match (the ' + ') optional so it backtracked to the last match of the <dot>.

Tim Notes:

1. This discussion concerning the ' ? ' Quantifier has been about applying it to a Greedy Quantifier like * or +, making it lazy, but when applied against an expression it makes that expression optional, in that it will run Zero or 1 time {0,1}.

   For example, if a sub-expression doesn't match, the entire match is removed as if the sub-expression never existed because to the RegEx Engine, the sub-expression was never evaluated. This may result in a No Value or null for a non-match, or it may result in an Invalid designation for that Capture Group because it did not participate in the Match (this can be identified in Regex Buddy easily), or it may result in a Zero Occurrence Match also resulting in a null value. It all depends on how the RegEx is coded.

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

More Examples of Greed and Lazy:

string = ab

Match: (ab.*)

= TRUE = ab

Even though there are no other characters following 'ab', the * makes the <dot> character Greedy so it matches up to EOL.

Match: (ab.+)

= FALSE

There are no other characters following 'ab'. The <dot> character has nothing to match up against and the Quantifier makes it Lazy so it won't even match against EOL.

String =aba

Match: (ab.+)

= TRUE = aba

There is a character that follows after 'ab' and even though the Quantifier makes it Lazy, the <dot> repeats once and matches the second 'a' in 'aba'.

Another Example: Use "+" to match the preceding "5" at least once.

Match: (bob5+@foo\.com)
Replace: \1<space>

bob5555@foo.com	bob5555@foo.com
bob5@foo.com	bob5@foo.com
bob65555@foo.com	bob65555@foo.com
bob@foo.com	bob@foo.com

Doesn't match bob6* or bob@* because it requires bob5 to match.

Tim Notes:

1. The last example requires enabling, 'Rename File Extensions as Part of the File Name' from the File/Folder Extensions sub-menu of the Renaming Options Menu, because the RegEx involved the file extension.

2. The previous examples are just to illustrate the points. In BRU these TRUE matches would consume the entire string so there would be nothing to rename and New Name would display no changes. In the last example, I used \1 <space> to allow the successful matches to display New Name.

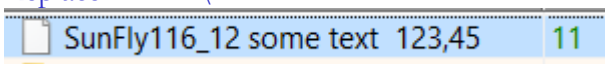
RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

Using BRU for real application of these:

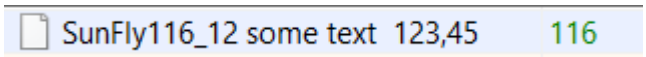
String = SunFly116_12 some text 123,45

Match: (\d\d)
Replace: \1



\d\d Match against two numeric digits in sequence.

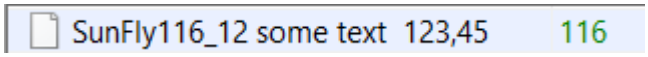
Match: (\d\d*)



\d\d* Make it Greedy – matches next two numeric digits, but because the next character is an underscore, stops matching. Remember that it repeats the last expression \d , accordingly '_' is not a numeric digit. The RegEx engine evaluates each individual expression so although we are looking to match against two numeric digits together they are evaluated as:

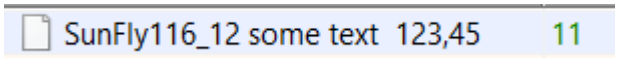
1) match against one numeric digit = TRUE
2) match against one numeric digit = TRUE

Match: (\d\d+)



\d\d+ Make it Greedy – matches next two numeric digits, but because the next character is an underscore, stops matching.

Match: (\d\d+?)



\d\d+? But not too Greedy - uses the ? Quantifier to limit the repetitions. The Quantifier restricts the expression from repeating once or not at all. It makes the Greedy operator 'Lazy' doing the least amount of work as possible. As a result the repetition is zero (optional), consequently only the first two values of \d are evaluated = '11' and the RegEx engine backtracks.

Tim Notes:

1. Using the ? Lazy Quantifier can result in Zero Occurrence Matches = null value.

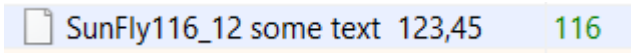
RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

Using BRU for real application of these: cont.

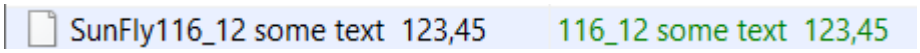
String = SunFly116_12 some text 123,45

Match: (\d\d.)



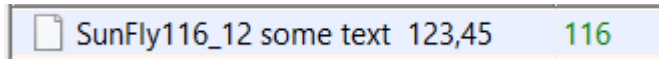
\d\d. Match against two numeric digits in sequence followed by any character (the dot). Matches the '11' followed by the '6'. The dot represents any character so the '6' could have been anything not just a numeric digit.

Match: (\d\d.*)



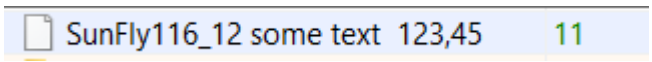
\d\d.* Match against two numeric digits in sequence followed by any characters to EOL (.*)

Match: (\d\d.?)



\d\d.? Match against two numeric digits in sequence followed by a Lazy Quantifier. The Quantifier repeats the last expression once or not at all. Repeats once so the '6' is matched.

Match: (\d\d.*?)



\d\d.*? Match against two numeric digits in sequence followed by a Lazy Quantifier. The Quantifier repeats the last expression once or not at all. The last expression is the <dot> and repeats zero (optional) so only the first two values of \d are evaluated = '11' and the RegEx engine backtracks.

Source Material this section: Azri Jamil, Stack Overflow, codeacademy.com, web.mit.edu/gnu/doc/html/regex_3.html

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

A Short Discussion on Capture Groups 'Giving Up' Characters - Greedy

Some refer to the RegEx Engine 'giving up characters', some refer to it as 'giving back when needed'. Myself, I call it 'reevaluating', but whatever you decide to call it, it refers to when the RegEx can't match a pattern because some Greedy Quantifier has taken too many characters. When this happens, the engine backtracks to the expression that caused the problem and forces it to 'give up', give back' or 'reevaluate' previously matched characters. It does this either in chunks, in the case of a matched subpattern that matched against several characters, or individually, in the case of singularly matched characters.

Example: Using a Greedy Quantifier

String = A tasty apple

Match: (.*)(apple)

Where: \1 = A tasty \2 = apple

Tim Analysis:

1. .	Match against any character	Capture Group 1 = A
2. *	Make it Greedy	Capture Group 1 = <entire string>

Okay so what does step 2 actually do?

a. It greedily matches every single character in the string reaching EOL.
b. Current position = EOL (End of Line)
c. Capture Group 1 = 'A tasty apple'

3. a p p l e
   a. Match against 'a' = EOL = False.
   b. The engine has advanced in the RegEx to the next character 'a', which fails to match EOL as there are no characters left in the string.
   c. The engine backtracks back to the .*, which gives up the 'e' in 'apple'. Capture Group 1 = A tasty appl
   d. The engine begins to backtrack in the string and test each character against the 'a'… each time it fails, Capture Group 1 'gives up' a character.

1. Match against	'a' = e = false		Capture Group 1 = A tasty app
2. Match against	'a' = l = false		Capture Group 1 = A tasty ap
3. Match against	'a' = p = false		Capture Group 1 = A tasty a
4. Match against	'a' = p = false		Capture Group 1 = A tasty
5. Match against	'a'	= True	Capture Group 2 = a
a. Engine moves forward.			
6. Match against	'p'	= True	Capture Group 2 = ap
7. Match against	'p'	= True	Capture Group 2 = app
8. Match against	'l'	= True	Capture Group 2 = appl
9. Match against	'e'	= True	Capture Group 2 = apple

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

A Short Discussion on Capture Groups 'Expanding' Characters - Lazy

With a lazy quantifier (aka Non-Greedy, Reluctant), the engine starts out by matching as few of the characters as the quantifier allows. But if so few characters have been matched that the rest of the pattern can not match, the engine backtracks to the expression that caused the problem makes it add ('expand') characters to the match. After matching each new character or expression, the engine tries once again to match the rest of the pattern.

Example: Using a Lazy Quantifier

String = Two_apples

Match: (.*?)(apples)

Where: \1 = Two_ \2 = apples

Tim Analysis:

1. .	Match against any character	Capture Group 1 = T
2. *	Make it Greedy	Capture Group 1 = <entire string>
3. ?	But not too Greedy (Lazy)	Capture Group 1 = null

Okay so what does step 2-3 actually do?

a. It greedily matches every single character in the string reaching EOL.
b. Current position = EOL (End of Line)
c. Capture Group 1 = 'Two_apples'
d. Match as few characters as possible
e. Capture Group 1 = Zero Occurrence Match = null
Current position = 'T'

4. a p p l e s
a. Match against 'a' = 'T' = False
b. The engine backtracks back to the .*?, which 'expands' to add the 'T' Capture Group 1 = T
c. Match against 'a' = 'T' = False
d. The engine moves forward in the string and tests each character against the 'a'… each time it Fails, Capture Group 1 'expands' a character.

1. Match against	'a' = w	= false		Capture Group 1 = Tw
2. Match against	'a' = o	= false		Capture Group 1 = Two
3. Match against	'a' = _	= false		Capture Group 1 = Two_
4. Match against		'a'	= True	Capture Group 2 = a
5. Match against		'p'	= True	Capture Group 2 = ap
6. Match against		'p'	= True	Capture Group 2 = app
7. Match against		'l'	= True	Capture Group 2 = appl
8. Match against		'e'	= True	Capture Group 2 = apple
9. Match against		's'	= True	Capture Group 2 = apples

RegEx Manual Volume II

Additional RegEx Information on Greedy and Lazy cont.

A Short Discussion on Capture Groups 'Expanding' Characters – Lazy cont.

Summary:

Groups like (.*) always lessen their match (give up characters), to let the whole expression match (short of Atomic Groups that either match or don't). Likewise, (.+?) will increase its match (add characters) if needed, to let the whole expression match.

This what is meant by giving up or giving back only as much as needed. In both the Greedy and the Lazy examples, one character at a time was either given up or added. Each time this occurs the RegEx engine backtracks in the RegEx costing valuable processing time.

With Thanks to RegexNinja for additional comments.

Source Material this section: Rexegg.com, Stackoverflow.com

RegEx Manual Volume II

Additional RegEx Information on Range Quantifiers

Range Quantifiers specify the minimum number of matches to find and the maximum number of matches to allow.

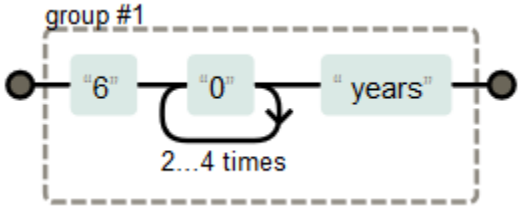
{3,7}	Between 3 to 7 matches
{,10}	Up to 10 matches with no lower limit (i.e. the low limit is 0)
{3,}	At least 3 matches with no upper limit (i.e. the high limit is infinity)
{,}	No upper limit or lower limit for the number of matches i.e., the lower limit is 0 and the upper limit is infinity
{5}	Exactly 5

Example: Match the preceding "0" at least 3 iterations with a maximum of 5 iterations

Match: (60{3,5} years)
Replace: \1<space>

60 years	60 years
600 years	600 years
6000 years	6000 years
6003 years	6003 years
60000 years	60000 years
600000 years	600000 years
6000000 years	6000000 years

Those strings that contained from 3 to 5 zeros matched. Those strings that contained less than 3 zeros did not, as well as those strings that contained more than 5 zeros.

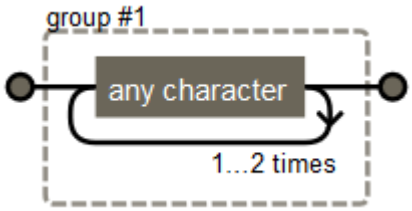


Example: Using the <dot> to match any character sequence two or three characters long

Match: (.{2,3})
Replace: \1<space>

4	4
44	44
404	404
a	a
aaaa	aaa
com	com
w3	w3

Those strings that contained from 2 to 3 characters matched. Those strings that contained less than 2 did not, as well as those strings that contained more than 3.



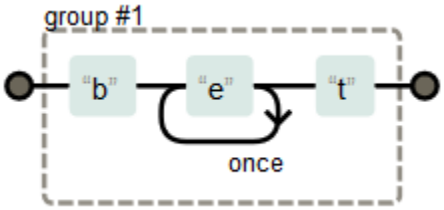
Tim Notes:

1. If I didn't add the <space> in the Replace string, those filenames, Although TRUE, would have nothing to rename since the Match consumed all of the characters.

RegEx Manual Volume II

Additional RegEx Information on Range Quantifiers cont.

Example: Match the preceding "e" exactly twice



Match: (be{2}t)
Replace: \1<space>

beet	beet
bet	bet
beat	beat
eee	eee

Those strings that contained 2 and only 2 'e' characters in succession are matched. Those strings that contained less than 2 did not, as well as those strings that contained more than 2.

In addition, Range Quantifiers are not only used to specify repetitions for the preceding single characters, they can be used with expressions. Parentheses (Capture Groups) are used to enclose the expression.

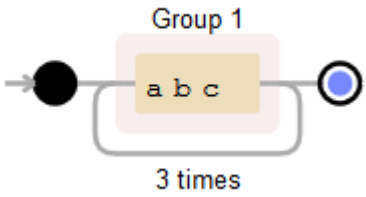
Example: Use a Capture Group and a range quantifier to find instances of the string "abcabcabc"

Match: (abc){3}
Replace: \1<space>

abc	abc
abcabc	abcabc
abcabcabc	abc
abcabcabcabc	abc
abcabcabcabcab	abc
abcabcabcabcabc	abc

Those strings that contained at least 3 instances of the string, 'abc' are matched. Those strings that contained less than 3 did not. Notice that by specifying an expression, strings that contained more than 3 instances are also matched as well as the 'abcabcabcabcab' that contained a partial 'ab' at the end. This string match was TRUE because of the preceding 3 instances of 'abc'.

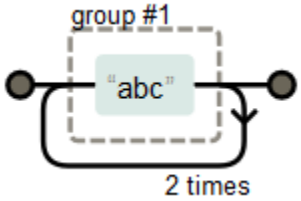
Range Quantifiers can also be used for simplifying numeric digit metacharacter expressions (as well as others) which can be seen throughout this book:



Intstead of:

\d\d\d to match against three numeric digits, this can be shortened to:

\d{3}



or

\d{3,} for three numeric digits or more

etc.

Tim Notes:

1. In this book Range Quantifiers are also referred to as just Quantifiers.

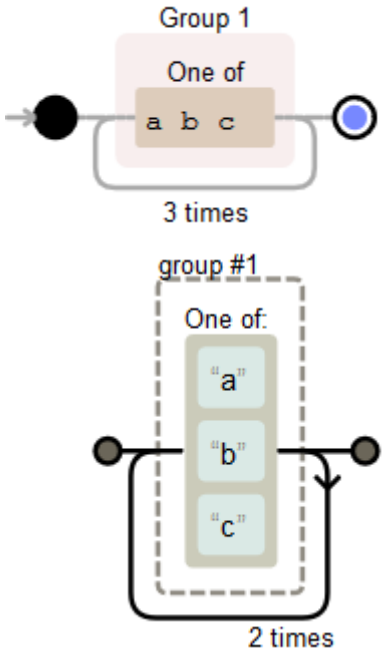
RegEx Manual Volume II

Additional RegEx Information on Range Quantifiers cont.

Example: Match against the third character in a string of a sequence of either an 'a' 'b' or 'c'

Match: ([abc]){3}
Replace: \1 <space>

123456789abc	c
123abc456abc789abc	c
123d56789abcde12345	c
ababc	a
abbcab	b
abc	c
abca	c
abcbba	c
abd	abd
ACABC	ACABC
babxcac	b
bc	bc
bd	bd



Because of the {3} quantifier, the match makes 3 iterations.

Tim Analysis:

i.e., 'abc'

[abc] Match against a single character of a class consisting of 'a' 'b' and 'c'

First test (equivalent to {1}) matches against 'a'
Second test (equivalent to {2}) matches against 'b'
Third test (equivalent to {3}) matches against 'c' = 'c' = True

Didn't match ACABC because case sensitive
Didn't match 'bc' or 'bd' because it hit EOL by third character match.
Didn't match 'abd' because third character didn't match 'a' 'b' or 'c' = d = Fail

RegEx Manual Volume II

Additional RegEx Information on Range Quantifiers cont.

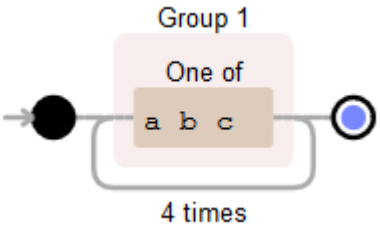
Example: Match against the fourth character in a string of either an 'a' 'b' or 'c'

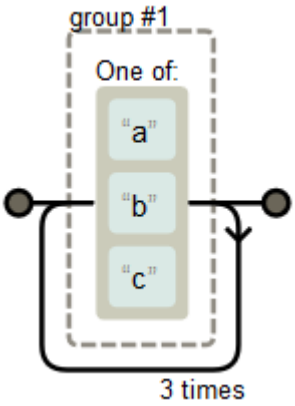
Match: ([abc]){4}
Replace: \1 <space>

123456789abc	123456789abc
123abc456abc789abc	123abc456abc789abc
123d56789abcde12345	123d56789abcde12345
ababc	b
abbcab	c
abc	abc
abca	a
abcbba	b
abd	abd
ACABC	ACABC
babxcac	babxcac
bc	bc
bd	bd

Didn't match '123d56789abcde12345' because the fourth character of the tested sequence 'a' b' or 'c' is 'd' = Fail.

Didn't match '123abc456abc789abc' because in each of the tested sequences of 'a' 'b' and 'c', there is no fourth character match = '4' for the first sequence, '7' for the second sequence and EOL for the third = Fail.

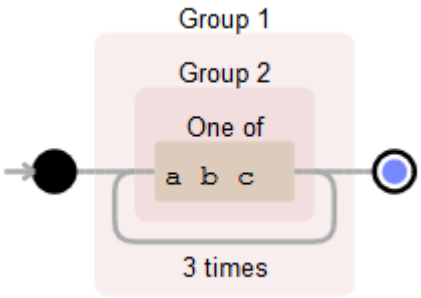


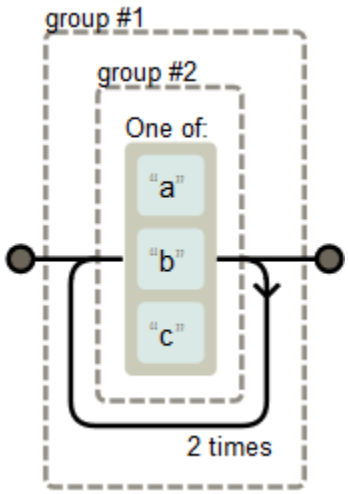


Example: Match against the first 3 letters in sequence made up of 'a' 'b' or 'c'

Match: (([abc]){3})
Replace: \1 <space>

123456789abc	abc
123abc456abc789abc	abc
123d56789abcde12345	abc
ababc	aba
abbcab	abb
abc	abc
abca	abc
abcbba	abc
abd	abd
ACABC	ACABC
babxcac	bab
bc	bc
bd	bd





Enclosing the entire expression creates a Capture Group that captures the entire sequence and not just the 3rd single character of that sequence.

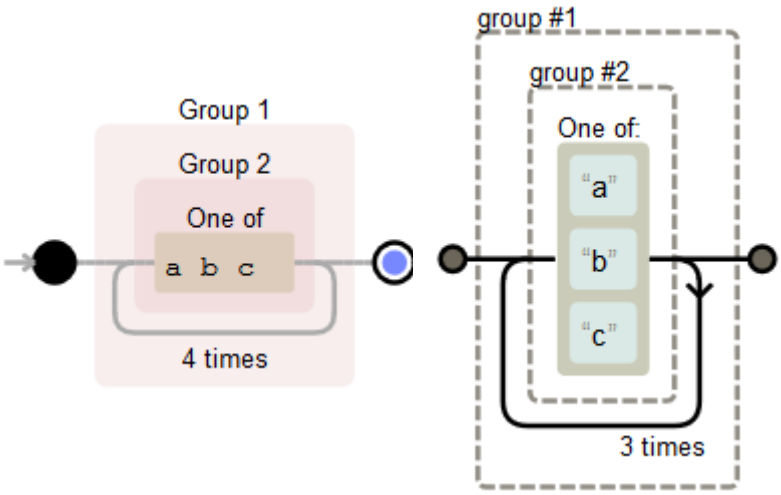
RegEx Manual Volume II

Additional RegEx Information on Range Quantifiers cont.

Example: Match against the first 4 letters in sequence made up of 'a' 'b' or 'c'

Match: (([abc]){4})
Replace: \1 <space>

123456789abc	123456789abc
123abc456abc789abc	123abc456abc789abc
123d56789abcde12345	123d56789abcde12345
ababc	abab
abbcab	abbc
abc	abc
abca	abca
abcbba	abcb
abd	abd
ACABC	ACABC
babxcac	babxcac
bc	bc
bd	bd

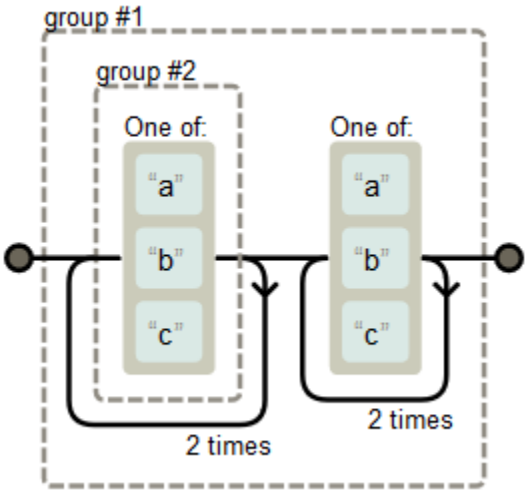


If we want to match two identical sequences of 'a' 'b' or 'c', use this:

Example: Match against the first 6 letters in sequence made up of 'a' 'b' or 'c'

Match: (([abc]){3}[abc]{3})
Replace: \1 <space>

123456789abc	123456789abc
123abc456abc789abc	123abc456abc789abc
123d56789abcde12345	123d56789abcde12345
ababc	ababc
abbcab	abbcab
abc	abc
abca	abca
abcbba	abcbba
abd	abd
ACABC	ACABC
babxcac	babxcac
bc	bc
bd	bd



This time there are only two matches that have 6 characters in sequence.

babxcac didn't match because of the 'x' as the fourth character in the sequence in the first test, and the EOL as the fourth character in the sequence in the second test.

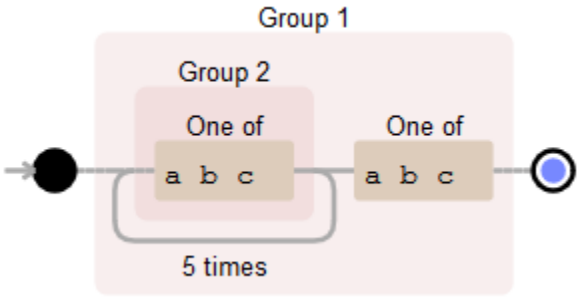
RegEx Manual Volume II

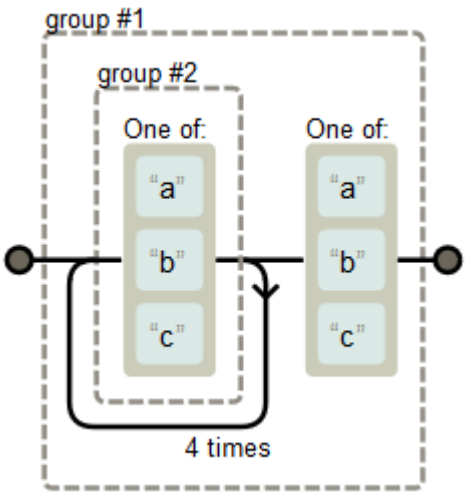
Additional RegEx Information on Range Quantifiers cont.

This is equivalent to:

Match: (([abc]){5}[abc])
Replace: \1 <space>

123456789abc	123456789abc
123abc456abc789abc	123abc456abc789abc
123d56789abcde12345	123d56789abcde12345
ababc	ababc
abbcab	abbcab
abc	abc
abca	abca
abcbba	abcbba
abd	abd
ACABC	ACABC
babxcac	babxcac
bc	bc
bd	bd



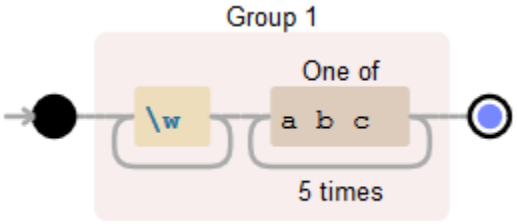


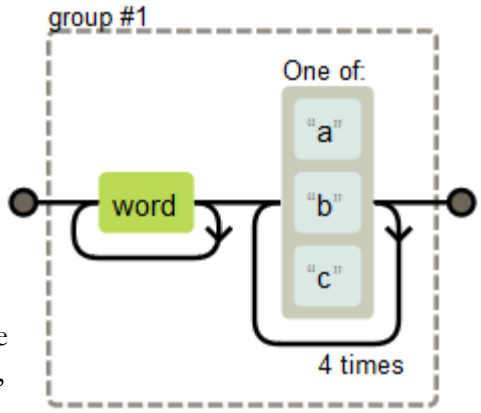
The first expression matches 'a' 'b' 'c' in sequence 5 iterations because of the {5} quantifier. The second expression matches 'a' 'b' or 'c' in sequence once because it has no quantifier, for a total of a 6 character sequence

or::

Match: (\w+[abc]{5})
Replace: \1 <space>

123456789abc	123456789abc
123abc456abc789abc	123abc456abc789abc
123d56789abcde12345	123d56789abcde12345
ababc	ababc
abbcab	abbcab
abc	abc
abca	abca
abcbba	abcbba
abd	abd
ACABC	ACABC
babxcac	babxcac
bc	bc
bd	bd





Match against a Word Character, made Greedy so it matches the entire first word in a string (the samples do not contain more than one word, so the entire filename would be matched), then limit those matches to strings that have only sequences of 'a' 'b' or 'c', and finally repeat that match 5 additional times for a total of a 6 character sequence.

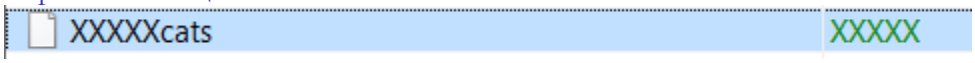
RegEx Manual Volume II

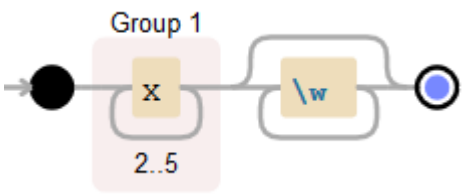
Additional RegEx Information on Range Quantifiers cont.

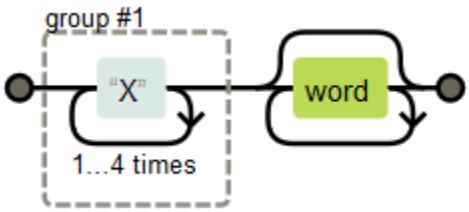
One more Example:

String = XXXXXcats

Match: (X{2,5})\w*
Replace: \1







Tim Analysis:

1. X	Match against the 'X'	Capture Group 1 = X
2. {2,5}	Specified range of 2 or more (up to 5) of the 'X' character followed by…	Capture Group 1 = XXXXX
3. \w	.. Word character (Not Captured) = 'c'	
4. *	Make it Greedy (Not Captured) = 'cats'	

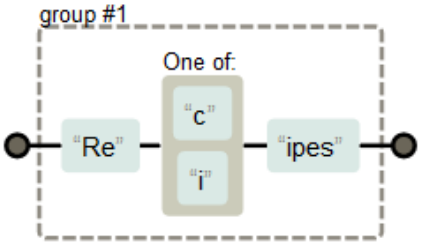
RegEx Manual Volume II

Understanding Character Classes

I already simplifed character classes in Volume I so I don't want to rehash the material here. Instead I will present additional examples.

Example Use a character class to match a 'c' or 'i' in a string

Match: (Re[ci]ipes)
Replace: \1<space>



Name	New Name
Receipt	Receipt
Receive	Receive
Recipe	Recipe
Recipes	Recipes
Reciprocity	Reciprocity
Recuse	Recuse

Match against the string 'Re' followed by a class consisting of a lowercase 'c' or a lowercase 'i' followed by more string characters, 'ipes'. Given the restrictive nature of this, there is only one match – Recipes. The use of the Class is not to indicate the match should be 'ci' but 'c' or 'i' whichever comes first. That is the key – whichever comes first.

If the string did not contain a 'c' but instead was 'Reiipes', a non-sensical word, it would still be a match because it matched against the 'i' in the Class [ci]. In the string, 'Recipes', that contained both a 'c' and an 'i', it matched against the 'c' because that character came before the 'i'.

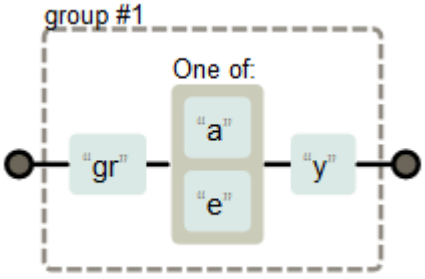
If the string was 'Reciipes', it would not match because although there is a 'c' and an 'i', it can only match against one and not both – so it would match against the 'c' satisfying the Class, but because the next character is an 'i' followed by another 'i', it would not match the next part of the pattern 'Recip'. The expression = FALSE and the RegEx Fails.

Example Use a character class to limit the scope of alternative matches on the words gray and grey

Match: (gr[ae]y)
Replace: \1<space>

graey	graey
gray	gray
grey	grey
gry	gry

Similar to the last example, the pattern established is the string characters 'gr' followed by a character class consisting of either a lowercase 'a' or a lowercase 'e' followed by a 'y'. Matches against the 'a' in 'gray' and the 'e' in 'grey', but does not match 'graey' because, although there is an 'a' that would be matched, it does not match against the pattern.



RegEx Manual Volume II

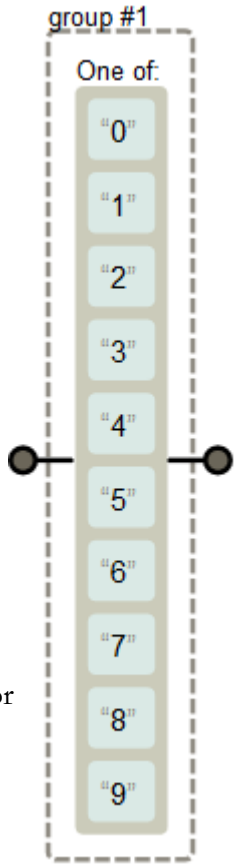
Understanding Character Classes cont.

Example: Use a character class to match any one numeric digit in the list

Match: ([0123456789])
Replace: \1<space>

&	&
0	0
001 text	0
5	5
9	9
F	F
text 001	0
x	x

Matches against any string containing a numeric digit even if there are other characters that make up the string.

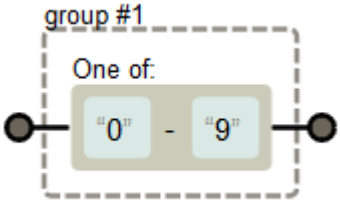


Ranges:

To simplify the previous example, use a <hyphen> "-" within a character class to denote a range for matching any one numeric digit in the list.

Match: ([0-9])
Replace \1<space>

&	&
0	0
001 text	0
5	5
9	9
F	F
text 001	0
x	x

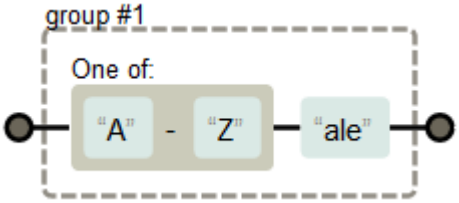


Example: Use a <hyphen> "-" within a character class to denote an alphabetic range for matching various words

Match: ([A-Z]ale)
Replace: \1<space>

Female	Female
kale	kale
Male	Male
Malet	Male
Pale	Pale
Retail	Retail
sale	sale
Salento	Sale

This pattern looks for an uppercase letter (any) followed by the string 'ale'.



RegEx Manual Volume II

Understanding Character Classes cont.

Here is an example taken from an earlier part of the book.

Match: (\d\d)([A-Z]{3,4})(\d{4})
Replace: \3\2\1

01JAN2009	2009JAN01
02JAN2009	2009JAN02
13NOV2013	2013NOV13

Refer to 'Rename DDMMYYYY to YYYYMMDD' for further information on RegEx used.

The pattern is to match against two numeric digits, that could have been expressed as \d{2}, followed by an uppercase letter that is repeated at least three iterations and no more than 4 ({minimum, maximum}), followed by 4 numeric digits.

01	JAN	2009
\d\d	[A-Z] {3,4}	\d{4}

This is called parsing – it takes the string 01JAN2009 and separates it out into its various sections. The values of these sections is held by Capture Groups designated by the enclosed parentheses and referenced using backreferences designated by \1 - \9 for up to 9 Capture Groups. By doing this the filename can then be re-arranged.

The Replacment string is used to re-order the filename:

\1 = 01 \2 = JAN \3 = 2009

\3	\2	\1
2009	JAN	01

Resulting in the New Name = 2009JAN01

Note that a class can contain more than just one type of character type:

[A-Za-z]	matches against either an uppercase letter or a lowercase letter (whichever comes first)
[a-z0-9]	matches against either a lowercase letter or a numeric digit (whichever comes first)
[ace5]	matches against either a lowercase 'a', 'c' or 'e' or a numeric digit '5' (whichever comes first)
[A1e]	matches against an uppercase 'A', or a numeic digit '1' or a lowercase 'e' (whichever comes first)

Literals don't require an Escape backslash metacharacter in a class except for] ^ and - … \] \^ \-

i.e.,

[A.0-9]	Matches against an uppercase 'A', an actual dot character (not a metacharacter), or a numeric digit.
[ ]	Matches against a <space> character.

The Open Bracket ' [' does not require an Escape metacharacter in a class.

RegEx Manual Volume II

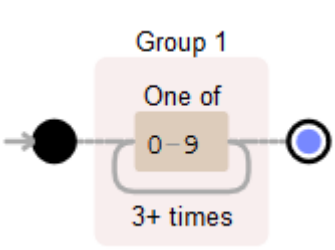
Understanding Character Classes cont.

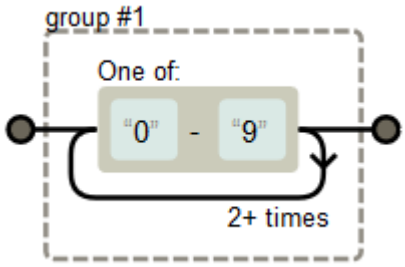
Example: Match any three or more numeric digits listed in the character class.

Match: ([0-9]{3,})
Replace: 1<space>

01JAN2009	2009
02JAN2009	2009
13NOV2013	2013
Text12abc3456Text	3456
000003	000003
42	42
aa123456zz	123456
Female	Female

Match against a numeric digit repeated three or more iterations that appears anywhere in the string.





Negated Classes

A Negated Class is just the opposite of a Class (Positive class). Instead of matching against a found character in the class, it will not match against a character within the class if found in the string. The RegEx engine basically ignores the character and moves on. This is used to prevent or 'negate' matches with unwanted or uneeded characters or undesireable strings. In other words it can be used to match against anything *but* the characters in the Negated Class.

The syntax of a Negated Class is to use a caret inside the class preceding the characters [^]

Example: Prevent a match on any numeric string. Use the "*" Quantifier to match an item zero or more times

Match: ([^0-9]*)
Replace: \1<space>

	New Name
text 001.txt	text .txt
Text12abc3456more.jpg	Text .jpg
000003.docx	.docx
01JAN2009.txt	.txt
02JAN2009.pdf	.pdf
1234text.docx	.docx
13NOV2013.ext	.ext
42.png	.png
aa123456zz.dll	aa .dll
Female.doc	Female .doc
p123JAN2008h.mp3	p .mp3

The pattern is to ignore a numeric digit but because the expression is made Greedy by the inclusion of the '*', the pattern will ignore any numeric digits found in the string. Blank values shown to the left show New Name as a <space> character followed by the extension because this was added in the Replacement string. No characters were captured for those strings that began with numeric digits - the captured value is null "". Without the <space> character these would otherwise display as having the extension with no preceding <space>. This was done to show that these matched. Captured values of text show the text followed by the added <space> character.

i.e., aa123456zz.dll

Matches against the text characters 'aa', ignores the '123456'. The RegEx is satisfied. Captures the 'aa' because it is the first non-numeric text encountered.

i.e., 01JAN2009.txt

Ignores the '01'. RegEx is satisfied. The value captured is null and shows up as <space> in New Name.

RegEx Manual Volume II

Understanding Character Classes cont.

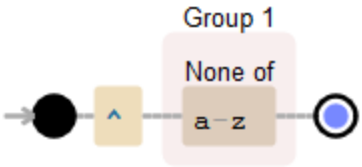
Negated Classes cont.

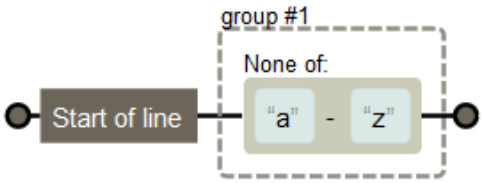
Example: Match against a filename that begins with any character except a lowercase letter

Match: ^([^a-z])
Replace: \1<space>

text 001.txt	text 001.txt
Text12abc3456more.jpg	T .jpg
000003.docx	0 .docx
01JAN2009.txt	0 .txt
02JAN2009.pdf	0 .pdf
1234text.docx	1 .docx
13NOV2013.ext	1 .ext
42.png	4 .png
aa123456zz.dll	aa123456zz.dll
Female.doc	F .doc
p123JAN2008h.mp3	p123JAN2008h.mp3

Matches against all of the filenames that begin with an uppercase letter or a numeric digit. The value captured is the first character matched. It does not match against any filename that begins with a lowercase letter.





Tim Notes:

1. The ^ outside of the negated class is an anchor and will be explained in the next section. It indicates that the match must occur at the beginning of the string (Beginning of Line or BOL).

Example Prevent a match on the numbers "10" and "12"

Match: (1[^02])
Replace: \1<space>

10	10
11	11
12	12
13	13
17	17
19	19
1a	1a
42.png	42.png
a1	a1

Matches against any numeric that begins with a '1' not followed by a '0' or '2' whichever comes first.

RegEx Manual Volume II

Understanding Character Classes cont.

Negated Classes cont.

Here is an example taken from an earlier section of the book that requires parsing:

Artist Name - Label Name - Track Name (Remix name)
to
Artist Name - Track Name - Remix name - Label Name

Match: ([^-]+) - ([^-]+) - ([^(]+) \(([^)]+)\)
Replace: \1 - \3 - \4 - \2)

Tim Analysis:

1. [^-]+ — Ignore the <hyphen> character if found but Greedy so…
Capture Group 1 = Artist Name<space>

- Captures up to the first <hyphen> encountered (capture anything but <hyphen>)

2. <space> <hyphen> <space> — Match against <space> <hyphen> <space> following 'Artist Name' but don't capture. This is done to move the RegEx forward through the string, otherwise the expression would be satisfied and the RegEx would stop. It also does one other thing. By matching against the first <space> following 'Name' of 'Artist Name', Capture Group 1 is reevaluated to 'Artist Name' removing that same <space>. Current position is 'L' of 'Label'

3. [^-]+ — Same as step 1.
Capture Group 2 = Label Name <space>

4. <space> <hyphen> <space> — Match against <space> <hyphen> <space> following 'Label Name' but don't capture. This is done to move the RegEx forward through the string, otherwise the expression would be satisfied and the RegEx would stop. It also does one other thing. By matching against the first <space> following 'Name' of 'Label Name', Capture Group 2 is reevaluated to 'Label Name' removing that same <space>. Current position = 'T' of 'Track'

5. [^(]+ — Similar to step 1 and step 3, it ignores the left parenthesis encountered but Greedy so…
Capture Group 3 = Track Name<space>

- Captures up to first left parenthesis following the current position 'T' of 'Track' so it moves forward. Current position = <space> of 'Track Name'.

Tim Notes:

1. Remember, a literal parentheses in a class doesn't require a preceding Escape Metacharacter, thus [(] or [)] can be used, but outside of the class, it does \(

RegEx Manual Volume II

Understanding Character Classes cont.

Negated Classes cont.

Tim Analysis: cont.

6. <space> \(	Match against a <space> followed by a literal left parenthesis that followed 'Name' of 'Track Name' but don't Capture. This is done to move the RegEx forward through the string, otherwise the expression would be satisfied and the RegEx would stop. It also does one other thing. By matching against the first <space> following 'Name' of 'Track Name', Capture Group 3 is reevaluated to 'Track Name' removing that same <space>. Current position = 'R' of 'Remix'
7. [^)]+	Similar to step 5, it ignores the first right parenthesis encountered but Greedy so… Capture Group 4 = Remix Name - Captures up to first right parenthesis following the current position 'R' of 'Remix' so it moves forward. Current position = 'e' of 'Name' of 'Remix Name'. Shouldn't current position = ') after 'Name' of 'Remix Name'? Normally if it wasn't for the negated class this would be True, but the negated class ignores the right parenthesis so it is not captured, thus the position of the RegEx engine stops at the 'e' and not the ')'
8. \)	Matches against right parenthesis but don't capture. Current position = EOL. Evaluation stopped. RegEx Engine stopped. Expression satisfied = TRUE.

Where:

\1 = Artist Name \2 = Label Name \3 = Track Name \4 = Remix Name

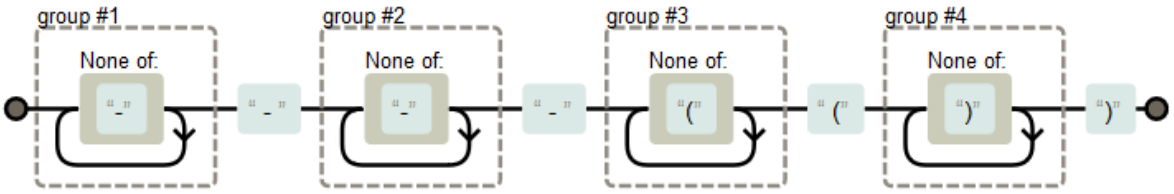
String is parsed. Now it can be re-arranged or added to using the Replace String.

Replace = \1 - \3 - \4 - \2

The Replace string is:

\1 = Artist Name <space> <hyphen> <space> \3 = Track Name <space> <hyphen> <space>
\4 = Remix Name <space> <hyphen> <space> \2 = Label Name

New Name = Artist Name – Track Name – Remix Name – Label Name



RegEx Manual Volume II

Understanding Character Classes cont.

Compound Character Classes

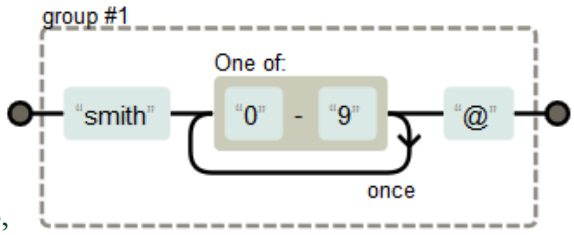
Character classes are a versatile tool when combined with various other metacharacter expressions. Compound character classes can help clarify and define sophisticated searches and test for certain conditions

Example: Find a partial e-mail address. Use a character class to denote a match for any number between 0 and 9 followed by a range to restrict the number of iterations a numeric digit matches

Match: (smith[0-9]{2}@)
Replace: <space>\1<space>

Smith34	Smith34
Smith0@a	Smith0@a
smith42@	smith42@
smith44@	smith44@
smith6	smith6

Pattern is to match against lowercase 's' 'm' 'i' 't' and 'h' followed by any numeric digit repeated twice (2 numeric digits) followed by the '@ sign.



Tim Notes:

1. Instead of stating each individual character, as was done here for clarification for new users, typically, I would use a statement similar to, Match against 'smith'.

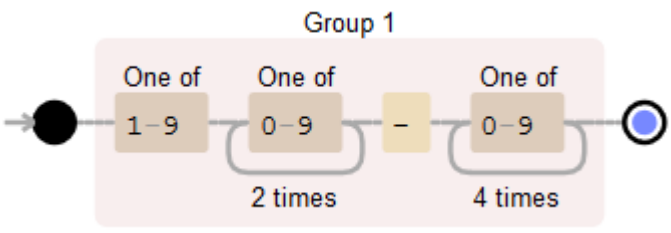
2. I just decided to throw in a preceding <space> in the Replacement string again for clarification of New Name matches, it doesn't mean anything.

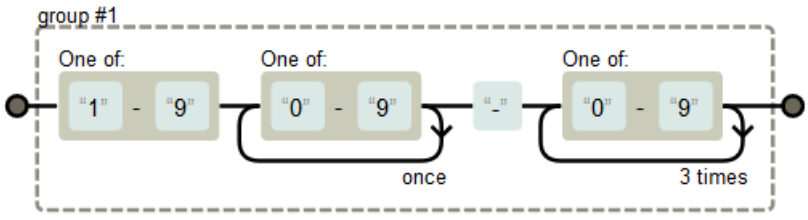
Example: Match a regular 7-numeric digit phone number in a filename. Prevent the digit "0" from leading the string

Match: ([1-9][0-9]{2}-[0-9]{4})
Replace: <space>\1<space>

023-1234	023-1234
123-4567	123-4567
1234-567	1234-567
555-5555	555-5555
555.5555	555.5555

Pattern is to match against any numeric digit except zero followed by any two numeric digits followed by a <hyphen> and ending with 4 numeric digits.





RegEx Manual Volume II

Understanding Character Classes cont.

Compound Character Classes cont.

Example: Match a valid e-mail address

Match: ^([a-z0-9])(.)([a-z0-9_-]+)(@)([a-z0-9].*)
Replace: <space>\1\2\3\4\5

	New Name
.smith@foo.net	.smith@foo.net
1234@mydomain.net	1234@mydomain.net
@foo.com	@foo.com
j.smith@bc.canada.ca	j.smith@bc.canada.ca
j_smith@foo.com	j_smith@foo.com
smith.@foo.org	smith.@foo.org
smith99@foo.co.uk	smith99@foo.co.uk
www.myemail.com	www.myemail.com

Pattern is an alphanumeric character followed by either an underscore, <hyphen> or literal dot character, followed by more alphanumeric text - or an underscore - or a <hyphen>, followed by the '@' sign, followed by more alphanumeric text.

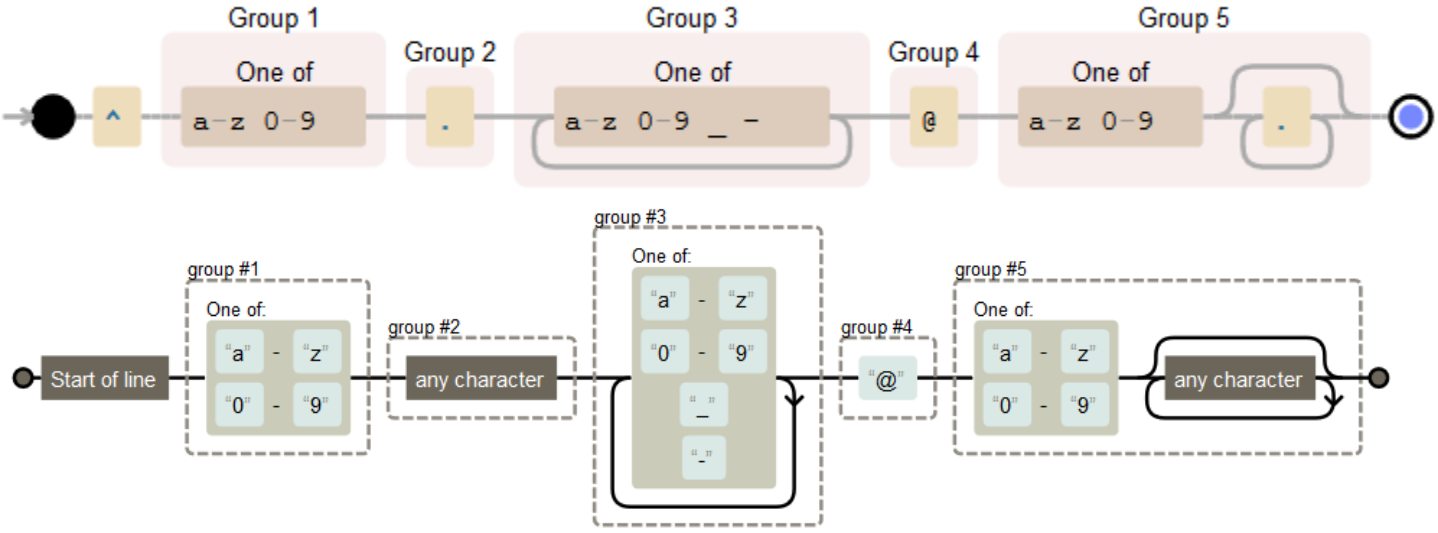
Before creating the RegEx, it is important to know what is a valid email address:

1. Consists of an email prefix and an email domain. The prefix appears to the left of the @ symbol while the domain appears to the right of the @ symbol.

2. Acceptable email prefix formats:
   1. Allowed characters: letters (a-z), numbers, underscores, periods, dashes and <hyphens>.
   2. An underscore, period, dash, or <hyphen> must be followed by one or more letters or numbers.

3. Acceptable email domain formats:
   1. Allowed characters: letters, numbers, <hyphens>, dashes.
   2. The last portion of the domain must be at least two characters, for example: .com, .org, .cc

Now that is established, the RegEx is created above with this pattern in mind.



RegEx Manual Volume II

Understanding Character Classes cont.

Compound Character Classes cont.

Using: j.smith@bc.canada.ca

Sample:

```
.............................
Match 1: j.smith@bc.canada
Group 1: j
Group 2: .
Group 3: smith
Group 4: @
Group 5: bc.canada
```

Breaks down:

[a-z0-9] . [a-z0-9_-]+ @ [a-z0-9].*

Where:

\1 = j \2 = . \3 = smith \4 = @ \5 = bc.canada

Tim Analysis:

1. ^	Start at beginning of string (Beginning of Line or BOL).	
2. [a-z0-9_-]	Match against an alphanumeric character, an underscore or a <hyphen> followed by …	Capture Group 1 = j
3. .	… match against the next character followed by … This is the dot character following the 'j' initial. This is important because sometimes an email address may use a dot, an underscore, <hyphen> or some other character that may otherwise not match just an alphanumeric specification (step 4). If the dot character was included as part of the class in step 4, the result in step 6 would match those filenames that we do not want to include. Try it yourself and see – refer to step 6.	Capture Group 2 = .
4. [a-z0-9_-]	… match against an alphanumeric character, an underscore or a <hyphen> This is the 's' of 'smith'.	Capture Group 3 = s
5. +	Make it Greedy followed by …	Capture Group 3 = smith
6. @	.. the '@' sign followed by …	Capture Group 4 = @
7. [a-z0-9]	Match against an alphanumeric character This is the 'b' of 'bc'.	Capture Group 5 = b
8. .*	Match against the remainder of string.	Capture Group 5 = bc.Canada

RegEx Manual Volume II

Understanding Character Classes cont.

Compound Character Classes cont.

Using: j.smith@bc.canada.ca

Tim Analysis: cont.

Using the dot character in the class of step 4 results in …

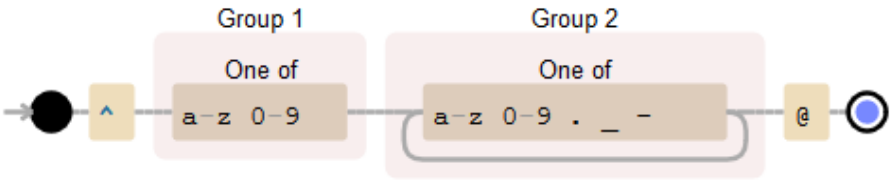
Match: ^([a-z0-9])([a-z0-9._-]+)@
Replace <space>\2

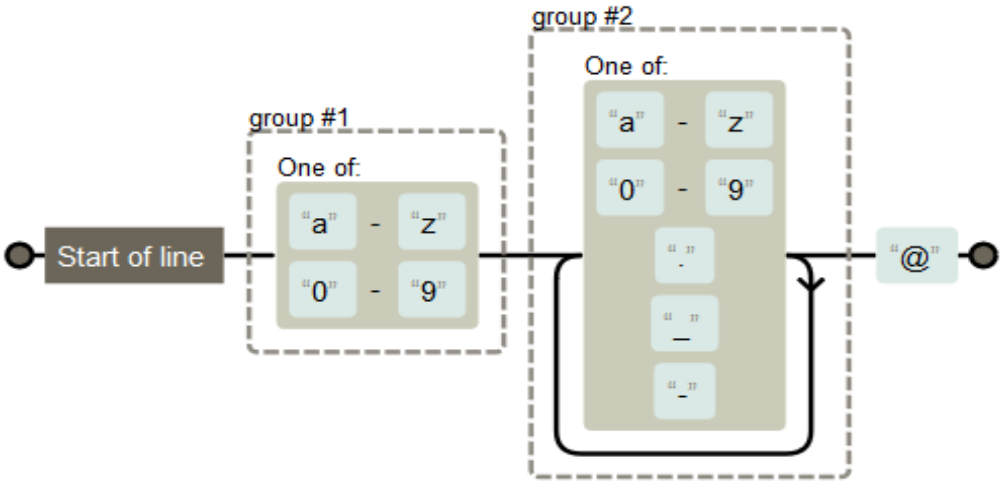
.smith@foo.net	.smith@foo.net
1234@mydomain.net	234.net
@foo.com	@foo.com
j.smith@bc.canada.ca	.smith.ca
j_smith@foo.com	_smith.com
smith.@foo.org	mith..org
smith99@foo.co.uk	mith99.uk
www.myemail.com	www.myemail.com

Instead of

Match: ^([a-z0-9])(.)([a-z0-9_-]+)@
Replace: <space>\2\3

.smith@foo.net	.smith@foo.net
1234@mydomain.net	234.net
@foo.com	@foo.com
j.smith@bc.canada.ca	.smith.ca
j_smith@foo.com	_smith.com
smith.@foo.org	smith.@foo.org
smith99@foo.co.uk	mith99.uk
www.myemail.com	www.myemail.com





Sample:

```
.....................
Match 1: j.smith@
Group 1: j
Group 2: .smith
```

Tim Notes:

1. Had to include backreference 3 in the Replace string on the right, to equal the results of where the evaluation is in the process on the left.

RegEx Manual Volume II

Understanding Character Classes cont.

Compound Character Classes cont.

Tim Notes:

1. Without the anchor , ^ the match would inadvertently include .smith@foo.net. By using the anchor, the RegEx HAS to match the first character against an alphanumeric therefore excluding any leading characters that are not alphanumeric.

2. <space> added to precede \1 backreference, otherwise, the RegEx matches entire string and New Name would show no change. This is done just to demonstrate that these files do match. I use this method a lot in this book. It doesn't mean anything except as a point of illustration.

3. I parsed the entire string and used Capture Groups for each section, again for purposes of illustration, but if I only wanted to use 4 Capture Groups instead of 5, I could have opted not to capture the '@' and instead include it in the Replacement string:

    Match: ^([a-z0-9])(_-|.)([a-z0-9_-]+)@([a-z0-9].*)
    Replace: \1\2\3@\4

The results would be the same.

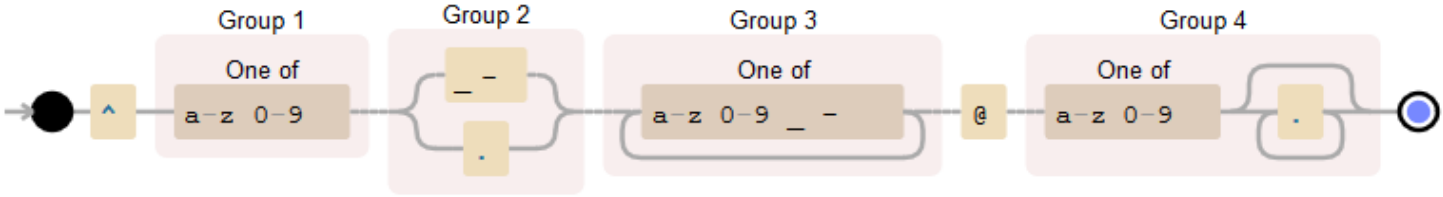
I also wouldn't have needed the added <space> in the Replacement string because the added '@' would have been enough of a change to get a value for New Name.

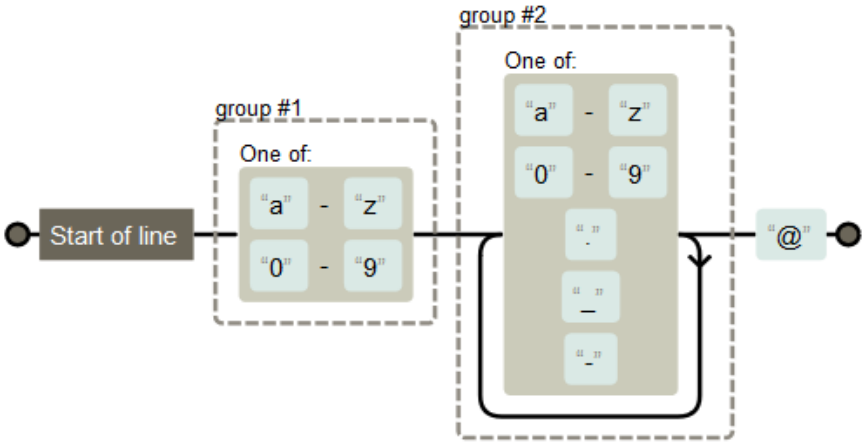
Where:

\1 = j \2 = . \3 = smith \4 = @ \5 = bc.canada

Replace: <space>\1\2\3\4\5

<space>j.smith@bc.canada.ca The .ca is treated as the file extension





Sample:

```
Match 1: j.smith@bc.canada
Group 1: j
Group 2: .
Group 3: smith
Group 4: bc.canada
```

RegEx Manual Volume II

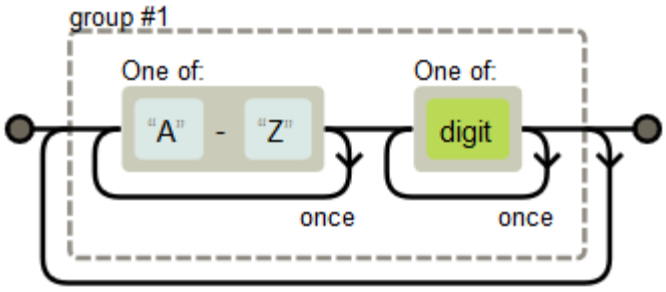
Understanding Character Classes cont.

Using Range Quantifiers with Classes

String = BRC32.exe

Match: ([A-Z]{2}[\d]{2})+
Replace: \1

BRC32.exe RC32.exe



Where:

\1 = RC32

Tim Analysis:

1. [A-Z]	Match against a class consisting of an uppercase letter = 'B' Current position = 'B'	Capture Group 1 =B
2. {2}	Repeat the match for 2 uppercase letters	Capture Group 1 = BR
3. \d	Match against numeric digit = '3' Changed Capture Group 1 because when it matched the '3' it then reevaluates based on the two uppercase letters that preceded the '3'	Changes Capture Group 1 = RC3
4. {2}	Repeat the match for 2 numeric digits	Capture Group 1 = RC32
5. +	Make it Greedy (already at EOL)	

Second match Third match: Fourth match:

B R C 3 2 B R C 3 2 B R C 3 2

RegEx Manual Volume II

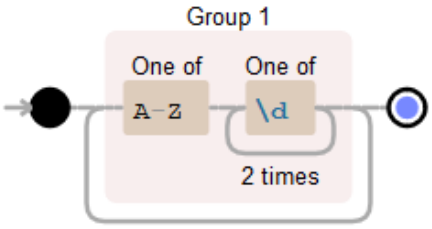
Understanding Character Classes cont.

Using Range Quantifiers with Classes cont.

String = BRC32.exe

Match: ([A-Z][\d]{2})+
Replace: \1

BRC32.exe	C32.exe



Where:

\1 = C32

Tim Analysis:

1. [A-Z]	Match against a class consisting of an uppercase letter = 'B' Current position = 'B'	Capture Group 1 =B
2. \d	Match against numeric digit = '3' Changed Capture Group 1 because when it matched the '3' it then reevaluates based on the uppercase letter that preceded the '3'	Changes Capture Group 1 = C3
4. {2}	Repeat the match for 2 numeric digits	Capture Group 1 = C32
5. +	Make it Greedy (already at EOL)	

First match Second match: Third match:

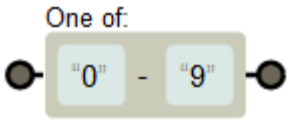
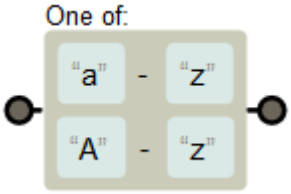
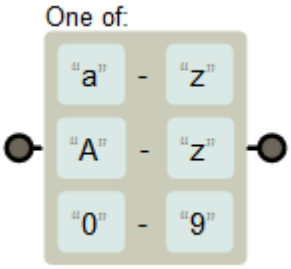
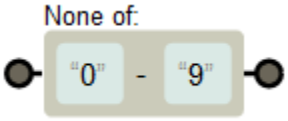
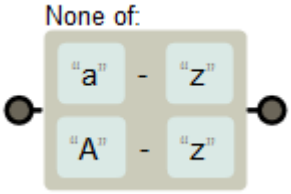
B R C 3 2 B R C 3 2 B R C 3 2

RegEx Manual Volume II

Understanding Character Classes cont.

Character Class Summary:

The following table defines various character class sequences. Use these alphanumeric patterns to simplify your regex searches.

Character Class	Description	
[0-9]	Matches any numeric digit from 0 to 9.	
[a-zA-z]	Matches any alphabetic character.	
[a-zA-z0-9]	Matches any alphanumeric character.	
[^0-9]	Matches any non-numeric digit.	
[^a-zA-z]	Matches any non-alphabetic character.	

RegEx Manual Volume II

Understanding Anchors

Also see, '**Example of parsing a group of numbers with no delimiters**'

These operators can constrain a pattern to match only at the beginning or end of the entire string. For this reason, both the caret "^" and dollar sign "$" are referred to as anchors in the regex syntax.

^ The caret indicates the beginning of the string (Beginning of Line or BOL).
$ The dollar sign indicates the end of the string (EOL or End of Line).

Tim Notes:

1. PCRE refers to the RegEx Engine used in BRU.

Example:

Match: (email)
Replace: \1<space>

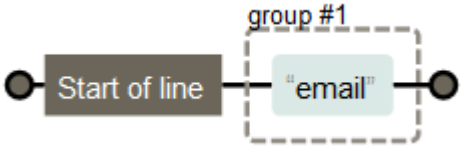
www.myemail.com	email .com
email	email
emailing	email
smithsemailaddress	email
bogus_emails	email

A RegEx containing 'email' will match 'email' regardless of where it is in the string.

Example:

Match: ^(email)
Replace: \1<space>

www.myemail.com	www.myemail.com
email	email
emailing	email
smithsemailaddress	smithsemailaddress
bogus_emails	bogus_emails

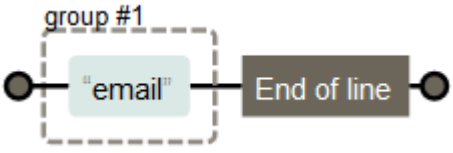


Using the ^ anchor, the match will only occur if the word 'email' is at the beginning of the string.

Example:

Match: (email)$
Replace: \1<space>

www.myemail.com	email .com
email	email
emailing	emailing
smithsemailaddress	smithsemailaddress
bogus_emails	bogus_emails



Using the $ anchor, the match will only occur if the word 'email' is at the end of the string EOL). Will not match 'emails' because it is restricted to 'email' unless it was changed to (email.*)$

Tim Notes:

1. The caret can also be used as a logical "NOT" to negate content in a character class, for example: [^...]. Refer to Negated Classes for more information.

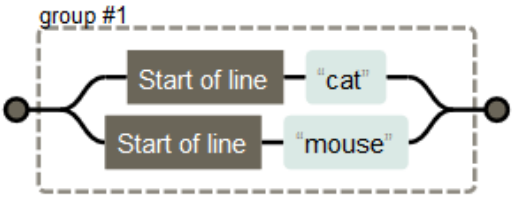
RegEx Manual Volume II

Understanding Anchors cont.

Example: Match against string using Alternatives at Beginning of Filename

Match: (^cat|^mouse)
Replace: \1

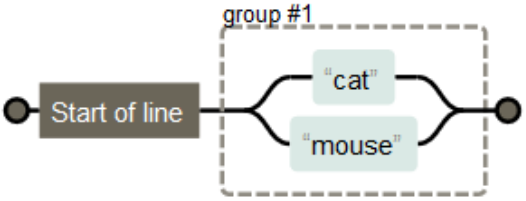
cat text	cat
toy cat	toy cat
mouse text	mouse
toy mouse	toy mouse



This could also have been written as:

Match: ^((?:cat|mouse))
Replace: \1

cat text	cat
toy cat	toy cat
mouse text	mouse
toy mouse	toy mouse



Tim Notes:

1. The (?:) is a Non-Marking Group – see Volume I

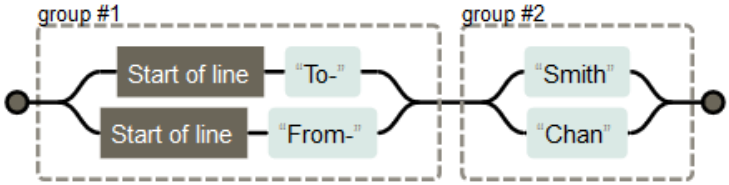
Here is another similar one –

Example: Use '^' in an Alternate Group to select match from beginning of string (BOL)

Match: (^To-|^From-)(Smith|Chan)
Replace: \1<space>\2

To Chan	To Chan
from - Smith	from - Smith
From- Chan	From- Chan
From-Chan	From-Chan
To-Chan	To-Chan
To-Smith	To-Smith

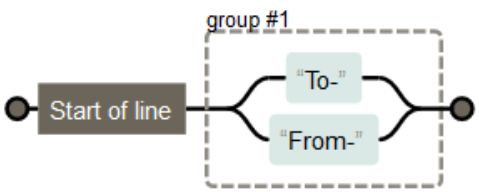
Match between the strings, 'To' followed by a <hyphen> and 'From' followed by a <hyphen> that must appear at the beginning of the filename followed by either, 'Smith' or 'Chan'.



Tim Notes:

1. Instead of using the caret inside the Capture Group for each item of the Alternate Group, it could have been written to obtain the same results:

^(To-|From-)



By placing the anchor outside, the anchor is applied to the entire expression within the Capture Group.

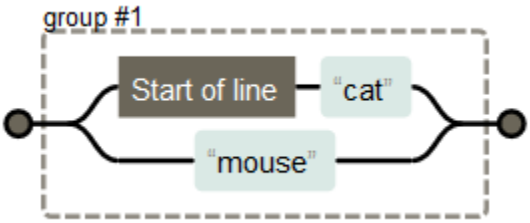
RegEx Manual Volume II

Understanding Anchors cont.

Example: Match against either a string found at the Beginning of Filename or a string anywhere in the Filename

Match: (^cat|mouse)
Replace: \1

cat text	cat
toy cat	toy cat
mouse text	mouse
toy mouse	mouse



Example: Match against the string 'cat' followed by either some alpha/numeric characters or the EOL (End of Line)

Match: (\bcat(\w+|$))
Replace: \1

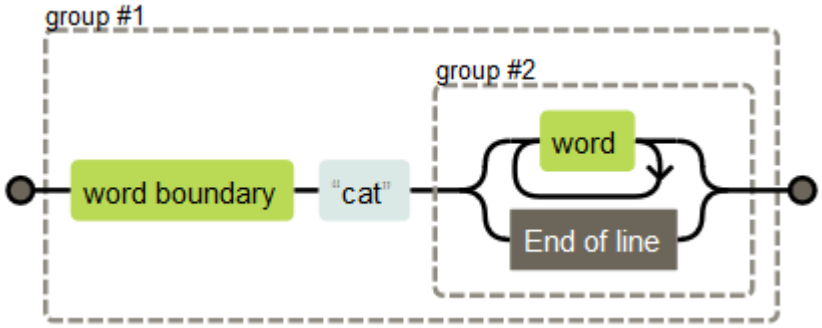
cat text	cat text
catch football	catch
made a catch	catch
toy cat	cat
mouse text	mouse text
toy mouse	toy mouse

Doesn't match against 'cat' because the <space> following 'cat' is considered whitespace not alphanumeric.

Matches 'cat' of 'toy cat' because EOL follows 'cat.

Sample:

```
.................
Match 1: catch
Group 1: catch
Group 2: ch
```



Where:

\1 = catch

Tim Analysis:

Using: catch football

1. \b	Match against a Word Boundary followed by…	
2. cat	.. string 'cat' followed by…	Capture Group 1 = cat
3. \w+ | $	.. either more alpha numeric text or the EOL whichever comes first.	Capture Group 1 = catch

RegEx Manual Volume II

Understanding Anchors cont.

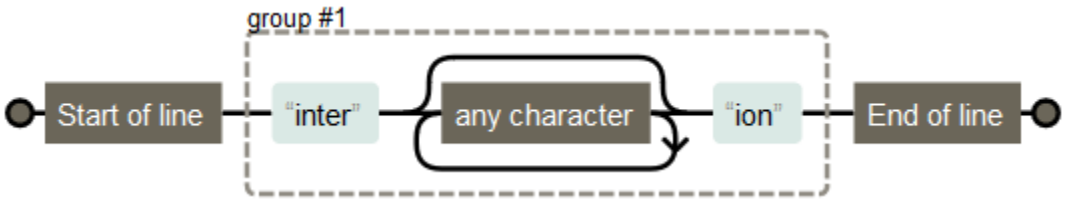
Now that the basics are out of the way…

Using both anchors in a RegEx

Example: Match the string, 'inter' that must appear at the beginning of the filename followed by any text, and end with the string 'ion' at the EOL (End of Line)

Match: ^(inter.*ion)$
Replace: <space>\1

reinternationalization	reinternationalization
internationalization	internationalization



Using: internationalization

Tim Analysis:

1. ^ Start at beginning of string (BOL)
2. inter Match against string, 'inter'
3. . Match against any character
4. * Make it Greedy
   Current position = EOL
5. ion Match against string, 'ion'
   Backtracks to match against 'ion'. Moves forward to EOL again. This captures the entire string again.
6. $ Must be matched from EOL.

Capture Group 1 = inter
Capture Group 1 = intern
Capture Group 1 = <entire string>

Capture Group 1 = internationalization

Tim Notes:

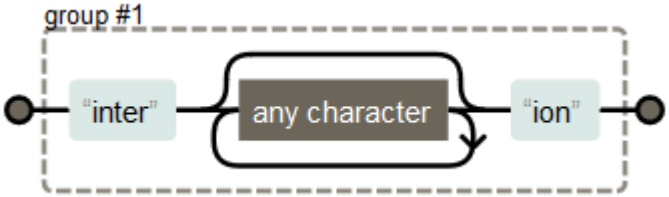
Because of the <space> in the Replace String, the match will display in New Name where normally it would show no change.

If you did not use the anchors,

Match: (inter.*ion)
Replace: <space>\1

reinternationalization	internationalization
internationalization	internationalization

both would match.



RegEx Manual Volume II

Understanding Lookarounds

Examining the Lookarounds

A Lookaround does not move the current position. It does not consume (capture) any characters but only asserts whether a match is possible or not. This is why Lookarounds are called 'Zero Length Assertions'.

- A Positive Lookahead instructs the RegEx engine to look to the immediate right for a match for FOUND items.
- A Negative Lookahead instructs the RegEx engine to look to the immediate right for a match for items NOT FOUND.

- A Positive Lookbehind instructs the RegEx engine to look to the immediate left for a match for FOUND items
- A Negative Lookbehind instructs the RegEx engine to the immediate left for a match for items NOT FOUND

Lookahead:

Behind Current Ahead

Positive = Match = True against Found
Negative = Match = True against Not Found

Lookbehind:

Behind Current Ahead

Syntax Summary:

(?<=) - positive lookbehind (?<!) - negative lookbehind
(?=) - positive lookahead (?!) - negative lookahead
(?:) - non-marking group

Syntax:

Look ahead positive (?=)

Find expression A. Look to the right *FOR* expression B:
A(?=B)

Look ahead negative (?!)

Find expression A. Look to the right *NOT* for expression B:
A(?!B)

Look behind positive (?<=)

Find expression A, Look to the left *FOR* expression B:
(?<=B)A

Look behind negative (?<!)

Find expression A, Look to the left *NOT* for expression B:
(?<!B)A

Tim Notes:

1. It is implied that when stated as 'look to the [right/left]', it is referring to the character or expression to the immediate [right/left] position.
2. In this section, Regex Buddy program was used a lot for testing. It uses a character position reference that starts from zero rather then from one. For example, the first character of a string is position zero, not one as BRU would use.
3. PCRE supports full expressions in Lookaheads, while in Lookbehinds there is only limited support for Character Classes and Alternates.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Tim Notes:

1. In a discussion with a colleague, it was thought best to include the following information so as not to cause confusion.

2. In this next section, I use references such as (?<=B)A. I want it understood that I am not being literal unless I am specifying that 'A' and 'B' are elements of the RegEx.

   For example, A = B is impossible. If you take A literally, then of course A can never = B, or A ≠ B, but I am, however, using this in the context of, for example, where A represents a character, an expression, equation. Or, if you like, think of A as a variable, and B is also representative.

   In the introduction, when stated as (?<=B)A, it refers to what A and B represent in the final RegEx.

   Could be that A represents one character and B represents another.
   Could be that A represents an expression and B represents another expression

   It's like A + B = C

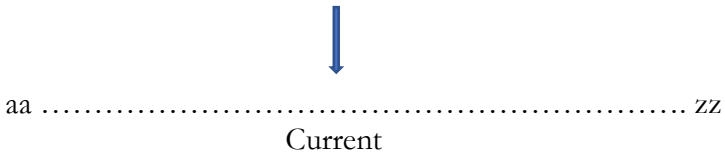
   or

   If A =B then C = A

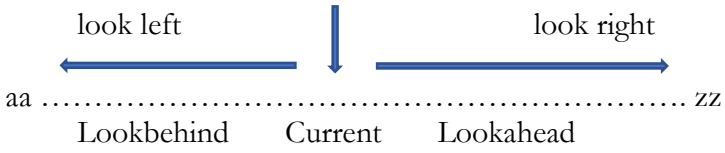
   That being said, you can't have literally, i.e., A(?<=B), where the RegEx requires A =B, because A can never = B. This is an example of the use of an improper syntax of a Positive Lookbehind. I will explain this shortly.

Introducing the Concepts:

You have a string that begins with 'aa' and ends with 'zz'. In between are 10 characters:



Next, think of a Lookbehind as a 'Look Left' and a Lookahead as a 'Look Right'



RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Introducing the Concepts: cont.

Given this RegEx – Look for a followed by a (where there are two different 'a's in the string)

Match: ((a)(?=a)(.....))
Replace: \1 <space>

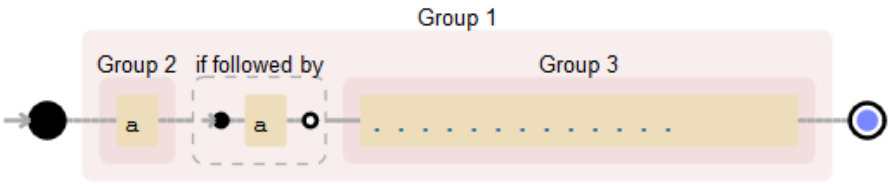
Breaks down:

C a p t u r e G r o u p 1

Capture Group 2 Capture Group 3
((a) (?=a) (.....))

a a1234567890zz

aa1234567890zz



Tim Notes:

1. (?=a) does not capture (Non-Marked Group)
2. Capture Group 1 takes its value inclusive of all Capture Groups within (Nested)

Where:

\1 = aa1234567890zz \2 = a \3 = a1234567890zz

Tim Analysis: Using Regex Buddy

1. (a)	Match against 'a' This is the first 'a' of 'aa'	Capture Group 2 = a Capture Group 1 = a
2. (?=a)	Lookahead. Look to the right. If next character is a single 'a' then expression = True and evaluation continues.= 'a' = TRUE	All Capture Groups = Unchanged
3. .{13}	Match against any character. This is the second 'a' of 'aa' Repeat the match for 13 characters.	Capture Group 3 = a Capture Group 3 = a1234567890zz Capture Group 2 = Unchanged Capture Group 1 = aa1234567890zz

Tim Notes:

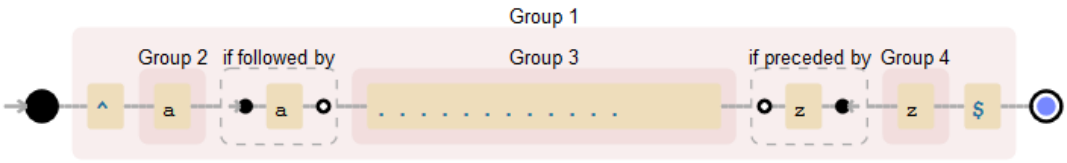
1. Capture Group 2 is the first 'a' of 'aa', while Capture Group 3 is made up of the second 'a' of 'aa'.
2. Capture Group 1 contains the value of the first 'a' from Capture Group 2 along with the value of Capture Group 3.
3. In the analysis, I simplified the (.....) to .{13}

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Introducing the Concepts: cont.



Now change it to this:

Match: (^(a)(?=a)(.....)(?<=z)(z)$)
Replace: \1 <space>

Breaks down:

C a p t u r e G r o u p 1

Capture Group 2		Capture Group 3		Capture Group 4		
(	(a)	(?=a)	(.....)	(?<=z)	(z)	$)
	a		a1234567890z		z	

aa1234567890zz

Where:

\1 = aa1234567890zz \2 = a \3 = a1234567890z \4 = z

Tim Analysis: Using Regex Buddy

1. ^	Start at beginning of string (BOL)	
2. (a)	Match against 'a' This is the first 'a' of 'aa'	Capture Group 2 = a Capture Group 1 = a
3. (?=a)	Lookahead. Look to the right. If next character is a single 'a' then expression = True and evaluation continues.= 'a' = TRUE	All Capture Groups = Unchanged
4. .{12}	Match against any character. This is the second 'a' of 'aa' Repeat the match for 12 characters.	Capture Group 3 = a Capture Group 3 = a1234567890z Capture Group 2 = Unchanged Capture Group 1 = aa1234567890z
5. (?<=z)	Lookbehind. Look to the left. If character is a single 'z' then expression = True and evaluation continues.= 'z' = TRUE	
6. z	Match against 'z' This is the second 'z' of 'zz'	Capture Group 4 = z Capture Group 2-3 = Unchanged Capture Group 1 = aa1234567890zz
7. $	Must be at EOL (End of Line)	

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Introducing the Concepts: cont.

Tim Notes:

1. Capture Group 2 is the first 'a' of 'aa', while Capture Group 3 is made up of the second 'a' of 'aa'.
2. Capture Group 3 is made up of the first 'z' of 'zz'
3. Capture Group 4 contains the value of the second 'z' of 'zz'
4. Capture Group 1 contains the value of the first 'a' from Capture Group 2 along with the value of Capture Group 3, and the value of capture Group 4. Capture Group 1 takes its value inclusive of all Capture Groups within (Nested).
5. In the analysis, I simplifid the (.....) to .{12}. It is 12 characters because the second 'z' of 'zz'is now captured separately in Capture Group 4.
6. Both the Positive Lookahead, (?=a), and the Positive Lookbehind, (?<=z), do not capture (Non-Marked Group)

Tim and Contributor Notes:

1. Positive Lookarounds require a match to the character or expression. Negative Lookarounds do not.

i.e. Positive Lookbehind

B,	of	(?<=B)A	To perform a match, find 'A' and look to the immediate left for 'B'.

Negative Lookbehind

(?<!B)A	as long as the character to the immediate left of 'A' is not 'B' it will match.

therefore..

With a Positive Lookbehind	(?<=B)A	- always needs the 2 characters 'B' and 'A' to perform a match.
With a Negative Lookbehind	(?<!B)A	- could match 'A' all by itself.

With Thanks to RegexNinja

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Look ahead positive (?=)

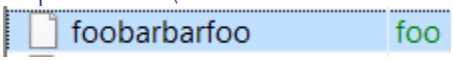
Find expression A. Look to the right *FOR* expression B:
A(?=B)

i.e. String = foobarbarfoo

1. Positive Lookahead

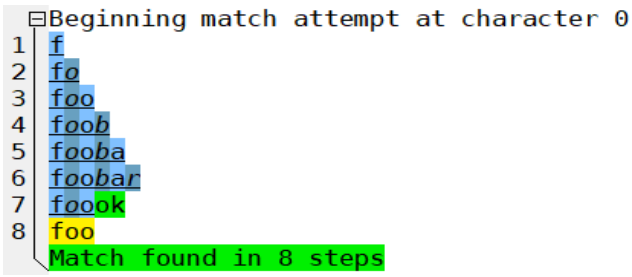
Locate 'foo'. Look to the right for 'bar'

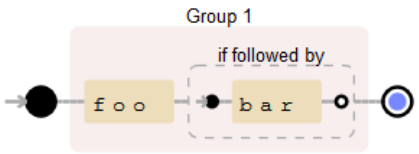
Match: (foo(?=bar))
Replace: \1



Match: (foo(?=bar))

Result: foobarbarfoo





1. Match against 'foo' at first 'f' in string
2. Test if 'bar' follows.
3. Zero Length Assertion satisfied

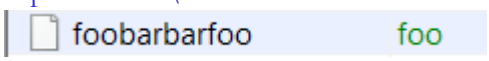
Look ahead negative (?!)

Find expression A. Look to the right *NOT* for expression B:
A(?!B)

2. Negative Lookahead

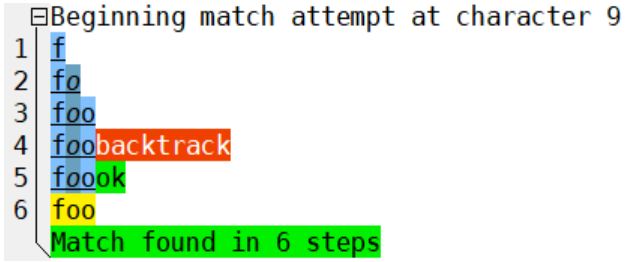
Locate 'foo'. Look to the right Not for 'bar'

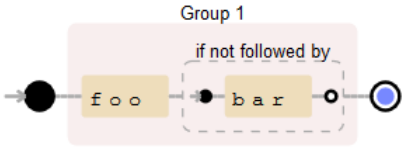
Match: (foo(?!bar))
Replace: \1



Match: (foo(?!bar))

Result: foobarbarfoo





1. Match against 'foo' at second 'f' in string.
2. Test for not 'bar' = EOL = True.
3. Zero Length Assertion satisfied

Tim Notes: Ambiguous Descriptions.

1. Look to the right could also be perceived as to what follows, just as Look to the left could be perceived to be what precedes. Just as A(?=B) could be perceived as look for A followed by B, or possibly look for B preceded by A, although the former would be more accurate.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Here is a more detailed explanation of what the RegEx Engine is doing in the Negative Lookahead example:

String = foobarbarfoo

Match: (foo(?!bar))

Where:

\1 = foo

Tim Analysis: Using Regex Buddy

1. foo	Matches against first 'foo' in string	
2. (?!bar)	Test for *not* 'bar' = 'bar' = Fail	
3.	Backtracks testing each position for 'foo'	
4.	Finds match at second 'foo' in string	Capture Group 1 = foo
5.	Test for Not 'bar' = EOL = True	

Tim Notes:

1. What happens after a match is that the testing continues at each character position until the string is exhausted – meaning no more matches possible or the end of the string (EOL – End of Line) has been reached.

   When this occurs, the match is given up (any matched values are removed as if the matches never happened) and the RegEx Engine returns back to the original position on the string before the Zero Length Assertion took place. For the first example this would be the second 'o' of the first 'foo' and in the second example, this would be the 'o' of the second 'foo' in the string. Any further RegEx evaluation would continue from those positions.

2. Reminder, Regex Buddy begins the first character at position zero not at position one. So when referring to the, i.e., 9th character above, it is not referring to the 'r' of the second 'bar', but the 'f' in the second 'foo' in the string - what I would have called the 10th character position. BRU, on the other hand, begins the count from position 1.

3. Zero Length Assertions do not change the position in the string and do not capture, consume or otherwise use up characters. Using the train analogy again, you can say, 'Train on a track, Runs to the end, Then back'. But again, for simplicity, suffice it to say in the analysis sections, that a match is found and evaluation continues.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining the Lookarounds cont.

Look behind positive (?<=)

Find expression A, Look to the left *FOR* expression B:
(?<=B)A

i.e. String = foobarbarfoo

1. Positive Lookbehind

Locate 'bar'. Look left *for* 'foo'

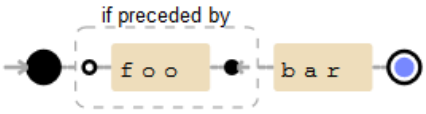
Match: ((?<=foo)bar)
Replace: \1

foobarbarfoo bar

Match: (?<=foo)bar

Result: foobarbarfoo

```
Beginning match attempt at character 3
1 f
2 fo
3 foo
4 ok
5 b
6 ba
7 bar
Match found in 7 steps
```



1. Match against 'first 'foo', character position 0
2. Match against 'bar' that follows
3. Zero Length Assertion is satisfied.

Look behind negative (?<!)

Find expression A, Look to the left *NOT* for expression B:
(?<!B)A

2. Negative Lookbehind

Locate 'bar'. Look left *not* for 'foo'

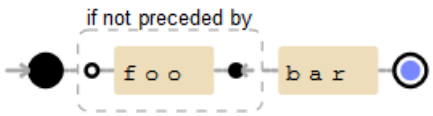
Match: ((?<!foo)bar)
Replace: \1

foobarbarfoo bar

Match: (?<!foo)bar

Result: foobarbarfoo

```
Beginning match attempt at character 6
1 backtrack
2 ok
3 b
4 ba
5 bar
Match found in 5 steps
```



1. Match against 'foo' at character position 0
2. Moves forward and tests for 'bar' = True = Fail
3. Moves forward and matches 'bar' at position 6 = True
4. Zero Length Assertion is satisfied.

Tim Notes:

1. In the Positive Lookbehind example, the engine moved forward first matching the 'foo' and then matched the following 'bar'. This proves that the engine does not always backtrack unnecessarily even if the nature of the RegEx is to 'Look behind'.

2. The RegEx Engine continues to test the remainder of the string for additional matches. When no more matches can be made or the EOL has been reached, whichever comes first, the Zero Assertion Lookaround gives up the match (any matched values are removed), and the RegEx Engine returns to the last character position before the Zero Assertion. Any additional RegEx evaluation would take place from this position.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads

Example: Find c. Look to the right *FOR* d

String = abcdedfABCDEFG

Match: `c(?=D)`

Result: `abcdedfABCDEFG`

It should be noted that the case used in Regex Buddy is *insensitive* in this example – that is why the RegEx, using a lowercase 'c' and an uppercase 'D', matched both the lowercase and uppercase 'C' in the string . If the case used was *sensitive* in Regex Buddy, this RegEx, `c(?=D)` would match against this string – `abcDedfABcDEFG.`

First Match:

```
Beginning match attempt at character 2
1 c
2 cd
3 cok
  Match found in 3 steps
```

Second Match:

```
Beginning match attempt at character 9
1 c
2 cd
3 cok
  Match found in 3 steps
```

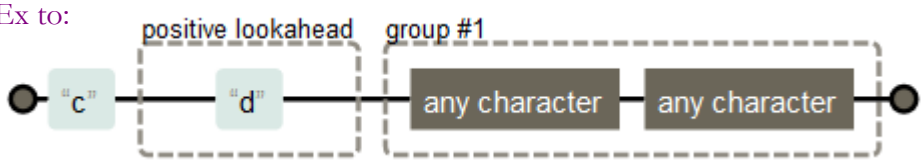
Where:

\1 = c

Tim Analysis: Using Regex Buddy

1. c — Test each character position looking for a 'c'. Matches against the 'c' at position 2
2. (?=D) — Look ahead for a 'D' by testing the next position. Matches at position 3. The Lookahead is satisfied.
3. c — The RegEx Engine continues moving through the string searching for other matches.
   a. At character position 9, it matches.
4. (?=D) — Look ahead for a 'D' by testing the next position. Matches at position 10. The Lookahead is satisfied.
5. — Continues to test each position until no more matches can be made or EOL whichever comes first.
6. — String is exhausted. The Lookahead gives up the matches and returns to the position where the first match was found, character position 3 at the 'd'

For BRU, which is case sensitive, change RegEx to:



Match: c(?=d)(..)
Replace: \1

abcdedfABCDEFG de

This shows that the current position after the Zero Length Assertion is the 'd' and Capture Group 1 = de

RegEx Manual Volume II

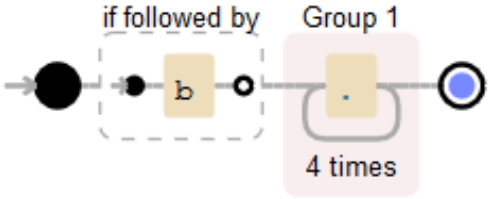
Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

Example: Locate a. Look to the Right for b. Match against the next 4 characters

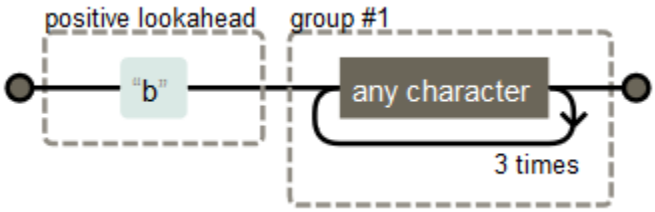
String = 123abc456abc789abc

Match: a(?=b)(.{4})
Replace: \1

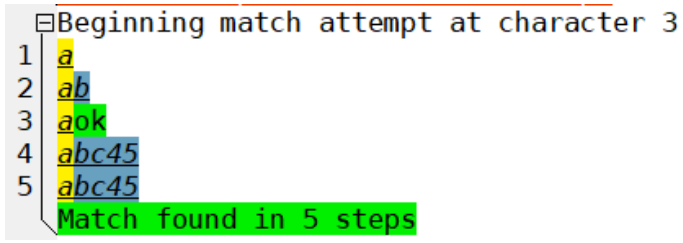


Match: a(?=b)(.{4})

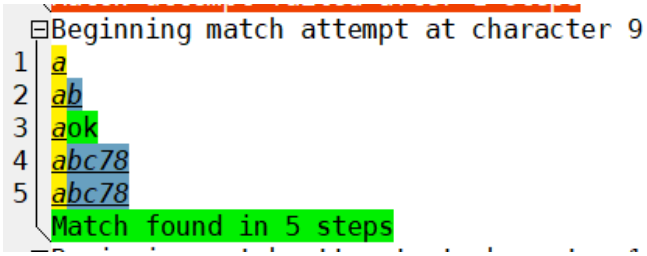
Result: 123abc456abc789abc



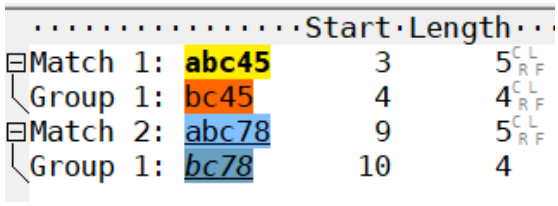
First Match:



Second Match:



Sample:



Tim Notes:

1. Although a Lookaround is a member of the Non-Marking Group, meaning they do not Capture, you can capture the value. i.e. Capture the 'b' inside the Lookaround:

   a(?=(b)) Capture Group 1 = b

   Where: The RegEx or the value you want to capture is grouped in parentheses *inside* rather than *outside* the Lookaround. If you try to capture the value by placing the parentheses outside of the Lookaround, you will only Capture the 'a':

   (a(?=b)) Capture Group 1 = a

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

String = 123abc456abc789abc

Match: a(?=b)(.{4})

Where:

\1 = bc45

Tim Analysis: Using Regex Buddy

1. a	Match against 'a' (Not Captured). This is the 'a' of '123a'	
2. (?=b)	Lookahead. Look to the right. If next character is a single 'b' then expression = True and evaluation continues = 'b' = TRUE The RegEx moves forward testing each character position for the 'b'. It finds it at character position 3, the first 'b' in the string following '123a'.	
3. .{4}	Match against the next 4 characters	Capture Group 1 = bc45
4. (?=b)	Continue to move through the string finding a second match at character position 9, the second 'b' in the string following '456a'. Changes Not Captured in step 1 to the 'a' of '456a'	
5. .{4}	Match against the next 4 characters	Changes Capture Group 1 = bc78
6.	Reaches EOL. String is exhausted. The Lookahead gives up the matches and returns to the position where the first match was found, character position 3 at the first 'a' in the string.	
10. .{4}	Matches against the next four characters. The rest of the RegEx, if any, continues from this point.	Capture Group 1 = bc45

Tim Notes:

1. The reason step 2 and step 4 include the 'b' is not because of the Lookbehind. The Lookbehind gives up its match so that the RegEx Engine returns to the point before the match, and this would be the 'a' of '123a' of step 1 and the 'a' of '456a' in step 3. Therefore the next 4 characters following 'a' of step 1 would be 'bc45' and the next 4 characters following 'a' of step 3 would be 'bc78'

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

Improper syntax of a Positive Lookahead (the expression to find should be on the left of the Lookahead):

This won't work: (?=B)A

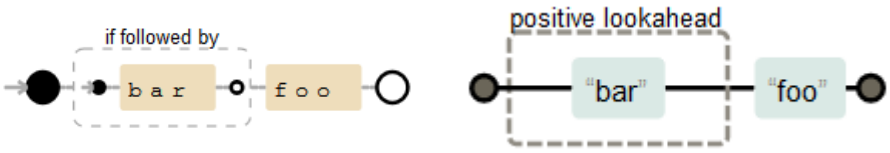
Example: Look ahead for 'bar' followed by 'foo'

String = foobarfoofoo

Match: (?=bar)foo

Result: Doesn't work

```
Beginning match attempt at character 3
1 b
2 ba
3 bar
4 ok
5 backtrack
Match attempt failed after 5 steps
```



Why?

This RegEx is not 'bar' followed by 'foo' but 'bar' followed by 'bar'. This can be seen in the above diagram where 'bar' is in the loop – 'bar' followed by 'bar.' 'Foo' is never evaluated. This is in a sense stating that expression A = expression B, which it doesn't and that is why it fails.

Observations:

With a positive Lookahead, you must have the expression to match preceding the Lookaround, i.e., foo(?=bar). Otherwise you have 'bar' followed by 'bar'. This will begin the discussion of improper usage of the Positive Lookahead.

Tim Analysis: Using RegexBuddy

1. (?=bar) Test each character position forward looking for 'bar'
   Matches at position 3, the first 'bar' of the string.
   Current position = 5, the 'r' of the first 'bar' of the string.
2. (?=bar) Evaluates next character for 'b' = 'f' = Fail

Tim Notes:

1. Using (?=B)A won't work unless expression A is something that is capable of evaluating to and matching 'B'.

   a. Wildcards i.e., ' …. ' will work
   b. A partial match of expression 'B'.

Both of these will be discussed next.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

Improper syntax of a Positive Lookahead cont:

Partial Matches & Wildcards

As already discussed, Using (?=B)A won't work. In this expression, (?=B)A , with 'A' following '(?=B)', given that 'A' is not the same or similar value as B and not a wildcard, would require a single character or expression to be both A and B at the same time which is impossible.

These will work:

Match:	Result:
(?=bar)b	foobarbarfoo
(?=bar)ba	foobarbarfoo
(?=bar)bar	foobarbarfoo
(?=bar)barf	foobarbarfoo
(?=bar)barfo	foobarbarfoo
(?=bar)barfoo	foobarbarfoo

Match:	Result:
(?=bar).	foobarbarfoo
(?=bar)..	foobarbarfoo
(?=bar)...	foobarbarfoo
(?=bar)....	foobarbarfoo
(?=bar).....	foobarbarfoo
(?=bar)......	foobarbarfoo
(?=bar).......	foobarbarfoo
(?=bar)........	foobarbarfoo
(?=bar).........	foobarbarfoo

The first column group of Match and Result display how a partial or similar match to 'bar' = True

The second column group of Match and Result display how a wildcard will also match against 'bar' = True.

These are the only way to match A = B. Examples of these follow on the next pages.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

Improper syntax of a Positive Lookahead cont:

Here is another example of using wildcards to match expression 'A' = expression 'B'

Example: Look to the right for 't' *followed* by 't'

Match: (?=t)(...)
Replace: \1

toy mouse	toy
Hello, there	the
Hello there	the
123E	123E
Catch football	tch
email	email
A1B1C1vsD1D2D3	A1B1C1vsD1D2D3
Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31	Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31

Lookahead finds 't' of 'toy' at the first position (the beginnning of string), as well as the 't' found within strings of other filenames.

Matches against first occurrence only so 't' in 'football' not matched.

When it finds a match, it gives up the match (Zero Length Assertion) and returns to the position before the match. This would be the 't'. The next three characters are captured from the current position. The RegEx Engine will continue moving forward and testing for any further matches until the string is exhausted or EOL whichever comes first.

i.e.

'Catch football' matches at the 't' in 'Catch', the 2nd character position, and again at the 't' in 'football, the 9th[h] character position. then after the string is exhausted (EOL), it returns back to the 2nd character position.

At this point it captures the next three characters – i.e., this would be 'tch' of the string 'Catch football'

Here is a Regex Buddy Analysis that confirms this:

Match: `(?=t)(...)` Result: `catch football`

Match at character position 2

```
Beginning match attempt at character 2
1 t
2 ok
3 t
4 tc
5 tch
6 tch
Match found in 6 steps
```

Match at character position 9

```
Beginning match attempt at character 9
1 t
2 ok
3 t
4 tb
5 tba
6 tba
Match found in 6 steps
```

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

Improper syntax of a Positive Lookahead: cont.

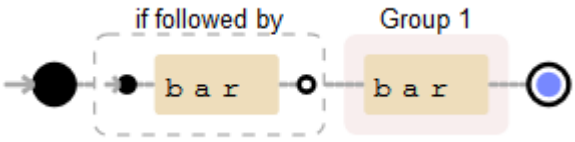
Example: Look to the right for 'bar' (inside Lookaround) preceded by 'bar' (outside Lookaround)

String = foobarbarfoo

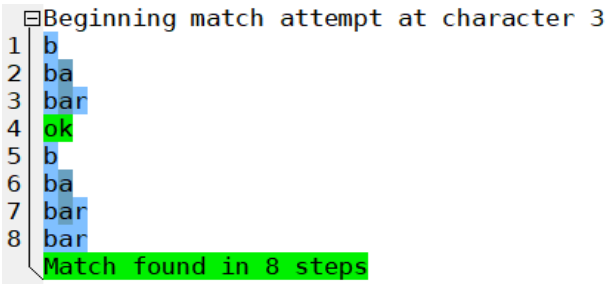
Match: (?=bar)(bar)
Replace: \1

Match: (?=bar)(bar)

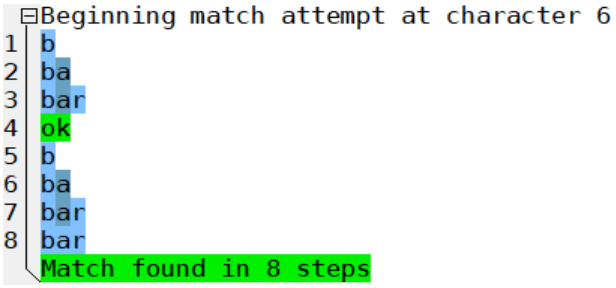
Result: foobarbarfoo



First Match:



Second Match:



Again, the only reason this matches is because this 'expression A', i.e., '(?=bar)' = 'expression B', i.e. 'bar'

Tim Analysis: Using Regex Buddy

1. (?=bar)	Test each character position forward looking for 'bar' Matches at position 3, the first 'bar' of the string. Current position = 5, the 'r' of the first 'bar' of the string. Gives up the match and returns to the position before the match. Position = 'b' of 'the first bar' in the string	
2. bar	Matches 'bar' of the first 'bar in the string at position 3	Capture Group 1 = bar
3.	Moves forward looking for further matches until the string is exhausted or EOL whichever comes first.	
4.(?=bar)	Finds a second match at the second 'bar' in the string at position 6 Gives up the match and returns to the position before the match. Position = 'b' of 'the second bar' in the string	
5. bar	Matches 'bar' of the second 'bar in the string at position 6	
6.	Moves forward looking for further matches until the string is exhausted or EOL whichever comes first.	

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

Improper syntax of a Positive Lookahead: cont.

You can however still match A= B using a Positive Lookahead in the proper manner.

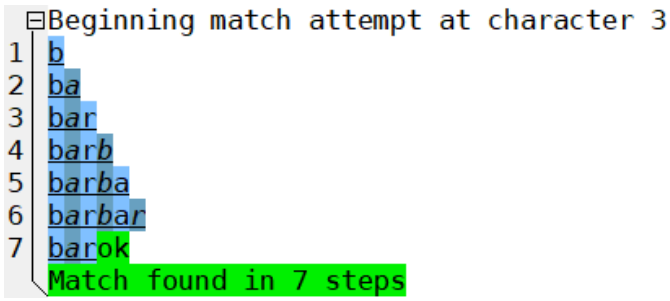
Example: Find 'bar' (outside of Lookaround). Look to the right for 'bar' (inside Lookaround)

String = foobarbarfoo

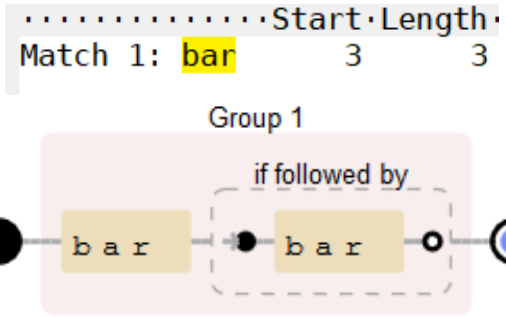
Match: (bar(?=bar))
Replace: \1

Match: bar(?=bar)

Result: foobarbarfoo



Sample:



Tim Analysis: Using Regex Buddy

1. bar	Match against 'bar' = True at position 3, the first 'bar' in string.	Capture Group 1= bar
2. (?=bar)	Look to the right for 'bar' = True Gives up the match and returns to the position in the string before the match. (does not affect Capture Group 1 which was matched outside of the Lookaround). Move forward through string seraching for more matches or EOL whichever comes first.	

Tim Notes:

1. I included this example just to finish off the A =B topic, or in these examples, bar = bar matching. This is the proper format for the Positive Lookaround so '(bar(?=bar))' could have been '(foo(?=bar))' and it would still match.

RegEx Manual Volume II

Understanding Lookarounds cont.

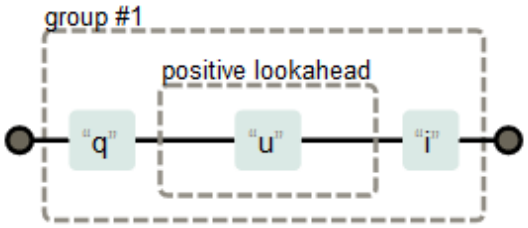
Examining Positive Lookaheads cont.

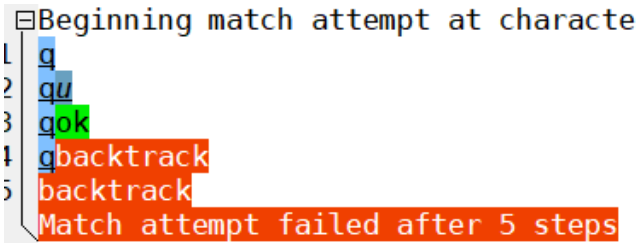
Ok. I lied, One more example of improper syntax usage of a Positive Lookahead: C(?=B)A

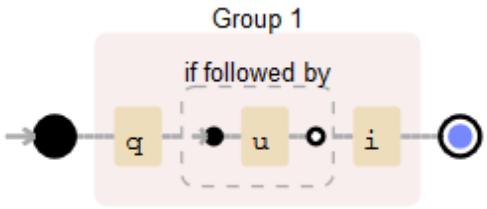
String = quiet cat quill quick

Match: (q(?=u)i)
Replace: \1 <space>

Fails







Tim Notes:

1. It will also fail to match against the 'qui' of 'quill' and 'quick' as well (not shown).

It may appear proper for a Positive Lookahead having the value to be matched to the left of the Lookahead 'B', but as long as there is a value, 'A', that is to the right of the Lookahead, it will never match.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookaheads cont.

String = quiet cat quill quick

Match: (q(?=u)i)

Why?

Same reason as originally stated, this RegEx is really asking for A = B, but A ≠ B. Here is a little more insight into how.

Tim Analysis: Using Regex Buddy

1. q	Matches 'q' of 'quiet'	Capture Group 1 = q
2. (?=u)	Positive Lookahead. Locate 'u' = True Gives up the Match and returns to the original position before the match. Current position = 'u' of 'quiet'	
3. i	Test for 'i' at current position = 'u' = Fail	
4.	RegEx will continue to move forward looking for more matches. Subsequent matches at 'quill' and 'quick' will also fail in the same manner. A ≠ B Continues until string is exhausted or EOL whichever comes first.	

Tim Notes:

1. The RegEx, 'q(?=u)i' , can never match anything. It tries to match 'u' and 'i' at the same position. If there is a 'u' immediately after the 'q', the lookahead = True, but 'i' Fails to match 'u'. If there is anything other than a 'u' that immediately follows after the 'q', the lookahead Fails.

2. Why all the waste of paper discussing what you shouldn't do vs what you should? Because it is important to know not only the right way but the wrong way. The wrong way, well that takes a lot of deep thought of understanding why you didn't get the expected results. To quote an overused but true expression, 'You learn more from your mistakes than from your successes.'

Source Material This Section: regular-expressions.info, rexegg.com

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookaheads

Find expression A, Look to the right *NOT* for expression B:

A(?!B)

Example: Locate 'a' followed by any three characters except 'cat'

Match: (a(?!cat)(...))
Replace: \1 \2

123456789abc	123456789abc
123abc456abc789abc	abc4 bc4
123d56789abcde12345	abcd bcd
AAC	AAC
abcc	abcc bcc
abcdedfABCDEFG	abcd bcd
b123456789adec	adec dec
BRC32.exe	BRC32.exe
dome.bat	dome.bat
fate any bar	ate te
foobarbarfoo	arba rba
Hello there	Hello there
Hello, there	Hello, there
My lj cat lk is furry	at l t l
pcre.dll	pcre.dll
quiet quick quaint	aint int
The Mouse ran up the clock	an u n u
toy mouse	toy mouse
xbcxapav	apav pav

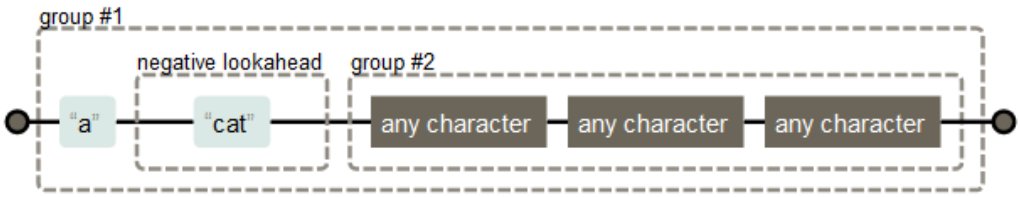
Sample:

```
..............
Match 1: abcc
Group 1: abcc
Group 2: bcc
```

Using: abcc

Where:

\1 = abcc \2 = bcc



Tim Analysis: Using Regex Buddy

1. a	Locate the first 'a' in the string	Capture Group 1 = a
2. (?!cat)	Test next character Not 'c' = 'b' = True Continue testing rest of string for 'cat until string exhausted or EOL whichever comes first. Give up the match and return to 'b'	
3. …	Match against next three characters; Capture Group 1 evaluates to … Capture Group 1 encloses the entire expression, therefore any matches within are part of Capture Group 1 – this includes the original 'a' + Capture Group 2's value of 'bcc'.	Capture Group 2 = bcc Capture Group 1 = abcc

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookaheads cont.

Example: Locate a Word Character. Look to the right for a 't' *not followed* by a 'Q'

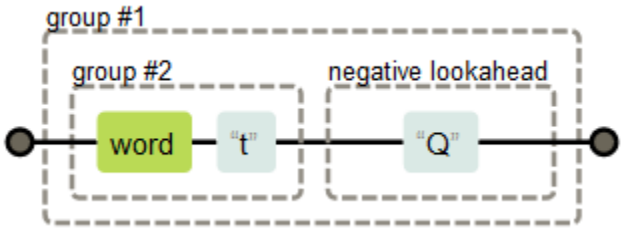
String = quiet quick quaint

Match: ((\wt)(?!Q))
Replace: \1

quiet quick quaint et

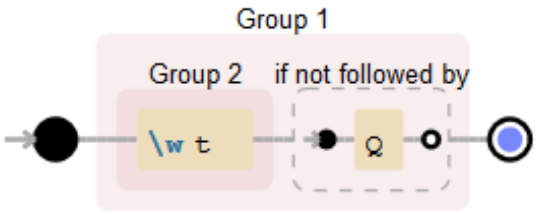
Match: (\wt)(?!Q)

Result: quiet quick quaint



Sample:

	Start	Length	
Match 1:	et	3	2
Group 1:	et	3	2
Group 2:	et	3	2
Match 2:	nt	16	2
Group 1:	nt	16	2
Group 2:	nt	16	2



First Match:

```
Beginning match attempt at character 3
1 e
2 et
3 etbacktrack
4 etok
Match found in 4 steps
```

Second Match:

```
Beginning match attempt at character 16
1 n
2 nt
3 ntbacktrack
4 ntok
Match found in 4 steps
```

Tim Analysis: Using Regex Buddy

1. \w	Test each character position looking for a Word Character. Matches 'q'. This is the 'q' of 'quiet'	
2. t	Test the next character for a 't' = u = Fail	
3.	Repeat until a match is found. Match found at the 'e' in 'quiet', character position 3.	Capture Group 1 = et
4. (?!Q)	Test the next character that it is not 'q' = <space> = True Zero Length Assertion satisfied.	
5.	Continues to test the rest of the string looking for a match. Matches at the 'n' at character position 16. Capture Group 1 is reevaluated	Changes Capture Group 1 = nt
6.	Continues to test the remainder of string until no more matches can be made (string is exhausted) or EOL whichever comes first.	
7.	The Lookbehind gives up the matches and returns to the position following the Zero Length Assertion, at the 'e' in Quiet. Capture Group 1 is reevaluated.	Changes Capture Group 1 = et

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookaheads cont.

Now for the improper syntax of the Negative Lookahead (the expression to find should be on the left of the Lookahead):

Find expression A. Test that A is NOT B:

(?!B)A

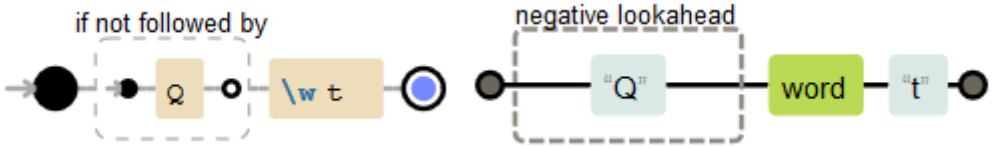
This will always be True as long as 'A' exists in the string.

Example: Locate a Word Character. Test that position is not a 'Q' and followed by a 't'

Match: (?!Q)\wt

Result: quiet quick quaint

quiet quick quaint et



First Match:

```
Beginning match attempt at character 3
1 backtrack
2 ok
3 e
4 et
Match found in 4 steps
```

Second Match:

```
Beginning match attempt at character 16
1 backtrack
2 ok
3 n
4 nt
Match found in 4 steps
```

Tim Analysis: Using Regex Buddy

1. (?!Q) Test each current character position looking for a character that is not a 'q'.
   a. Doesn't match at position 0, matches at the 'u' and 'i'
2. \w Tests each current character position – i.e. u, i, which are Word Characters and matches …
3. t Test for the following 't' after the initial Word Character match.
4. Fails until the 'e' of 'quiet' matches because it is followed by a 't'.
5. Continues searching through the string to find another match at the 'n' of 'quaint'
6. Continues searching until string is exhausted. Zero Length Assertion gives up the match, returns to the point before the first match, the 't' of 'quiet'. Any further evaluation (in this example there is none) would take place from this position.

To determine the current position, change the RegEx to (?!Q)\wt(..) - and the captured match will be the next two characters, the <space> following 'quiet) and the 'q' of 'quick'.

Tim Notes:

1. This works of course because as long as there is a Word Character (\w) that ends with 't', it will always be True.
2. This one is not totally useless because it does as intended – finds words that begin with 'q' (tests first position of the word) and followed by (end with) 't', as compared with the next example coming up.

RegEx Manual Volume II

Understanding Lookarounds cont.

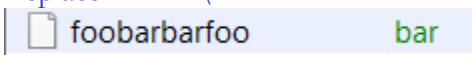
Examining Negative Lookaheads cont.

More examples of improper syntax usage:

String = foobarbarfoo

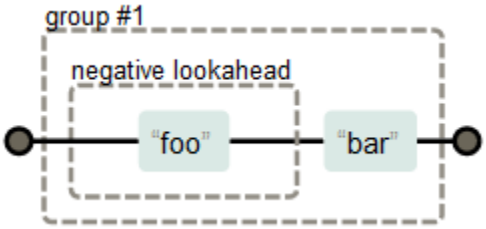
Example: Locate 'bar'. Test that 'bar' is Not 'foo'

Match: ((?!foo)bar)
Replace: \1

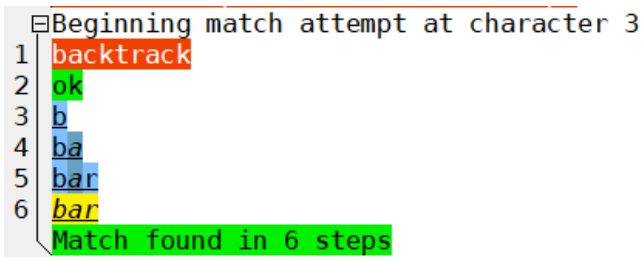


Match: (?!foo)bar

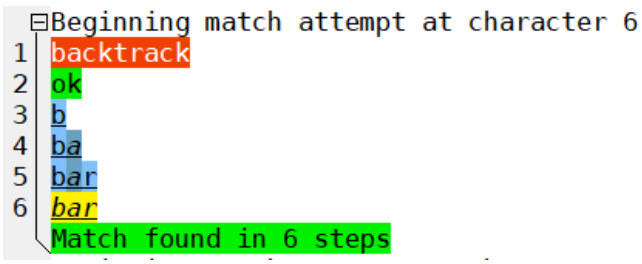
Result: foobarbarfoo



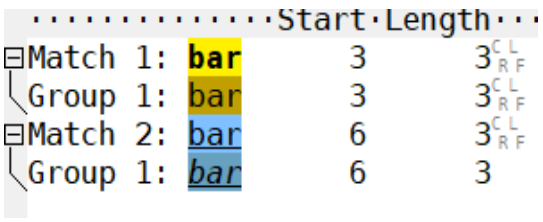
First Match

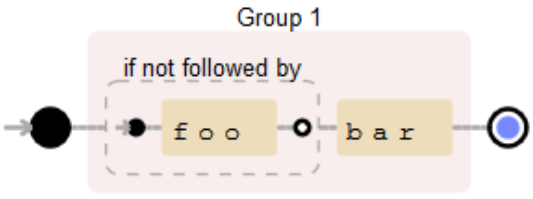


Second Match



Sample:





Tim Analysis: Using Regex Buddy

1. bar
(?!foo) Anywhere there is, (where 'A' =) 'bar' in the string, it will match
.. because A ≠ B (where 'B' = 'foo')
Matches first 'bar' in the string following 'foo'

Tim Notes:

1. Useless RegEx – perfect example of the improper use of the Lookaround that does nothing.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookaheads cont.

More examples of improper syntax usage: cont.

Example: Find <space>. Test that <space> is not 'b' followed by any three characters

Match: (?!b)\s(...)
Replace: \1

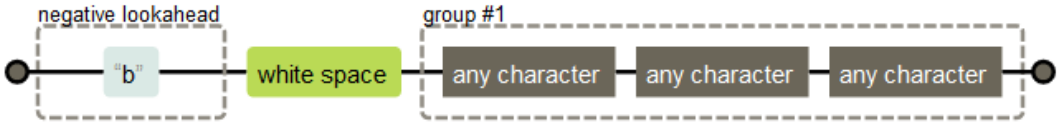
123d56789acbde12345	123d56789acbde12345
123E	123E
abcc	abcc
Catch football	foo
email	email
fate any bar	any
Hello there	the
Hello, there	the
My lj cat lk is furry	lj
SunFly116_12-The Beatles- I Want To Hold You...	Bea.mkv
Tarzan A.33 B.32 C.36 Jane A.35 B.33 C.31	A.3.31
toy mouse	mou
www.myemail.com	www.myemail.com

Sample:

```
.....Start.Length...
Match 1:  any     4       4
Group 1:  any     5       3
Match 2:  bar     8       4
Group 1:  bar     9       3
```

Match: (?!b)\s(...)

Result: fate any bar



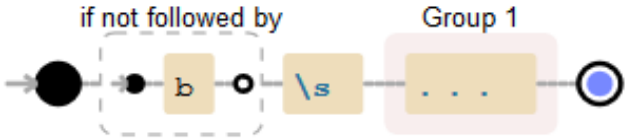
First Match:

```
Beginning match attempt at character 4
1 backtrack
2 ok
3
4  a
5  an
6  any
7  any
Match found in 7 steps
```

Second Match:

```
Beginning match attempt at character 8
1 backtrack
2 ok
3
4  b
5  ba
6  bar
7  bar
Match found in 7 steps
```

Tim Analysis:



Using fate any bar

1. bar (?!b) — Anywhere there is, (where 'A' =) <space> in the string, it will match .. because A ≠ B (where 'B' = 'b')
   Matches first <space> in string = <space> following 'fate'
2. … — Captures the three characters after the <space> — Capture Group 1 = any

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookbehinds

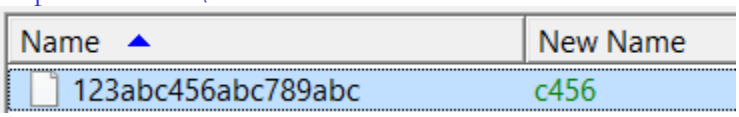
Find expression A. Look to the left for expression B:

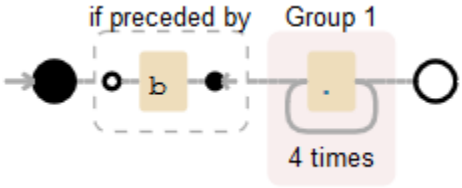
(?<=B)A

Example: Locate any four characters. Look to the left for 'b'

String = 123abc456abc789abc

Match: (?<=b)(.{4})
Replace: \1

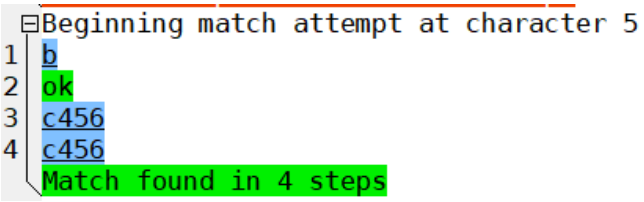




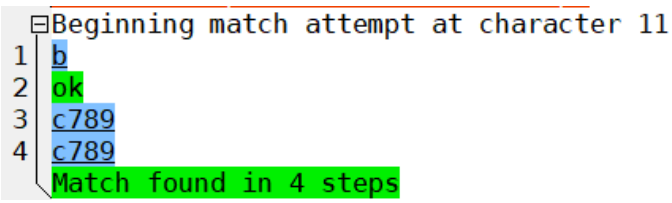
Match: (?<=b)(.{4})

Result: 123abc456abc789abc

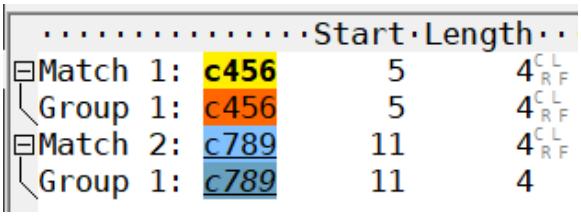
First Match:



Second Match:



Sample:



Tim Notes:

1. The way the RegEx Engine is depicted above, it looks to be moving forward through the string not backtracking even though this is a Positive Lookbehind. As already explained, the RegEx Engine doesn't always backtrack even if the expression is a Lookbehind.

2. BRU only matches against the first match. Subsequent matches, as displayed in Regex Buddy, are discarded.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookbehinds cont.

String = 123abc456abc789abc

Match: (?<=b)(.{4})

Where:

\1 = c456

Tim Analysis: Using Regex Buddy

1. (?<=b)	Lookbehind. Look to the left. If character is a single 'b' then expression = True and evaluation continues.= 'b' = TRUE The RegEx moves forward testing each character position for the 'b'. It finds it at character position 4, the first 'b' in the string following '123a'.	
2. .{4}	Captures the next 4 characters. 3 characters.	Capture Group 1 =c456

Tim Notes:

1. Unfortunately, Lookbehinds are not supported in the first Visualization program (the more descriptive diagram). Therefore, only the second Visualization can be used in this section.

2. The Lookarounds are Non-Marking Groups meaning they don't capture. This is why the 'b' is not captured. It asserts only that the 'b' is found.

3. Although I stop the analysis at the point that the BRU RegEx has been satisfied, it should be mentioned that the following continues to take place using any of these 'Zero Length Assertions':

3. (?<=b)	Will continue to test each character for an additional match until no more matches can be made (string exhausted) or EOL whichever comes first. Matches the 'c789'	Capture Group 1 = c789
4.	Gives up the Zero Length Assertion Matches and returns to the position in the string before the Lookaround, at the 'b'.	
6. {4}	Captures the next four characters. Additional RegEx evaluation (there is none) would continue fom this point.	Capture Group 1 = c456

RegEx Manual Volume II

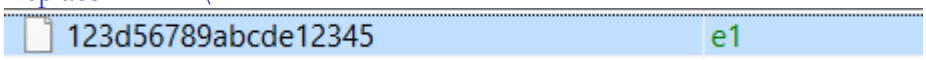
Understanding Lookarounds cont.

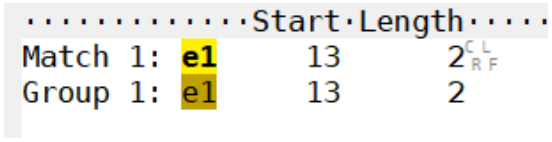
Examining Positive Lookbehinds cont.

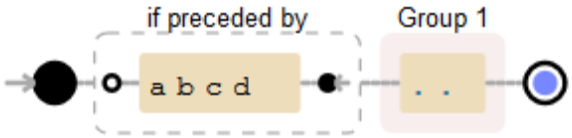
Example: Locate any two characters. Look to the left for 'abcde'

String = 123d56789abcde12345

Match: (?<=abcd)(..)
Replace: \1





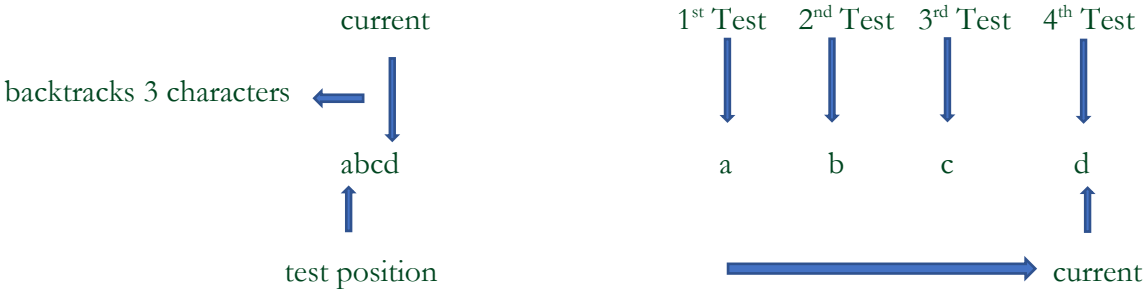


Tim Notes:

When I first started researching Lookbehinds, I was under the impression that it *always* backtracked or looked behind to find a match. It does not. With this assumption I first mistakingly used this as my analysis:

Tim Analysis: Using Regex Buddy

1. (?<=abcd) Tests the three preceding characters along with the current character for the match. Moves forward until match found at character position 12, the second 'd' of 'abcd' in the string. Here is how it works:



a. At current position, (starts at position 0, the beginning of the string - Beginning of Line or BOL), it backtracks to the preceding three characters.
b. The RegEx engine first tests for the 'a' from the third position preceding the current position= True
c. It moves forward and tests for the 'b' = True
d. It moves forward and tests for the 'c' = True
e. It moves forward back to its original character position and tests for the 'd' = True

But this was wrong. The correct analysis is found on the next page.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookbehinds cont.

Example: Locate any two characters. Look to the left for 'abcde' cont.

String = 123d56789abcde12345

Match: ((?<=abcd)(..))
Replace: \1

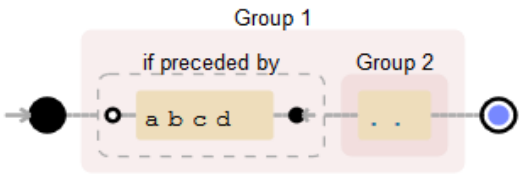
123d56789abcde12345 e1

```
            Start  Length
Match 1:    e1     13      2
Group 1:    e1     13      2
Group 2:    e1     13      2
```

Match: (?<=abcd)(..)

Result: 123d56789abcde12345

```
Beginning match attempt at character 13
1 a
2 ab
3 abc
4 abcd
5 ok
6 e
7 e1
8 e1
Match found in 8 steps
```



Where:

\1 = el

Tim Analysis: Using Regex Buddy

1. (?<=abcd) Lookbehind.
Look to the left. If character is a single 'a'
then continue to test for 'bcd'. If found,
expression = True and evaluation
continues.= 'abcd' = TRUE
Test each character position looking for an 'a'.
Matches at character position 13, the 'a' of 'abcd'
2. Tests the next positions for the 'b', 'c', 'd'. Matches.
Current position = 'd' at character position 16.
3. .. Matches against the next two characters Capture Group 1 = e1

Tim Notes:

1. Interesting to note that the Lookbehind does not look for 2 characters and then backtrack to find a match for the 'a', but instead looks ahead for the 'abcd', then proceeds to match against the next 2 characters to satisfy the Lookaround.

2. This was a perfect example of trying to make the evidence fit the theory rather than, the theory, based on where the evidence takes you. A mistake often made in the sciences and here, this is no different.

RegEx Manual Volume II

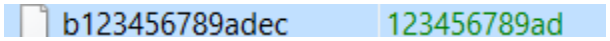
Understanding Lookarounds cont.

Examining Positive Lookbehinds cont.

Example: Locate any 11 characters. Look to the left for 'b'

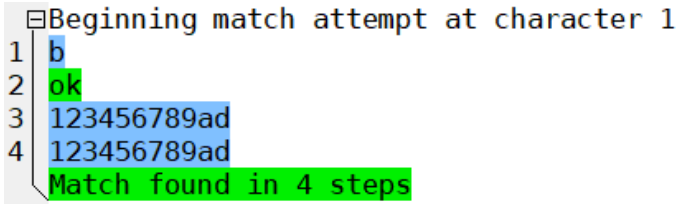
String = b123456789adec

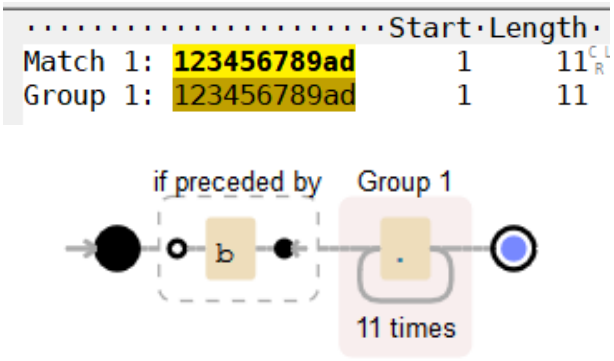
Match: (?<=b)(.{11})
Replace: \1



Match: (?<=b)(.{11})

Result: b123456789adec





Tim Analysis: Using Regex Buddy

1. (?<=b) Lookbehind.
Look to the left. If character is a single 'b'
then expression = True and evaluation
continues.= 'b' = TRUE
Looks for the 'b' in the string. Finds a match at position 0.

2. {.11} Matches against the next 11 characters (starting at character position 1). Current position = 'd'
Capture Group 1 = 123456789ad

3. Continue testing string for further matches of 'b'.

4. Finding none, it gives up the match (removes any matched value from the Lookaround – does not affect previous matches) and returns to the position before the Zero Length Assertion, the 'd'.

Tim Notes:

1. Interesting to note that the Lookbehind does not look for 11 characters and then backtrack to find a match for the 'b', but instead looks ahead for the b, then proceeds to match against the next 11 characters to satisfy the Lookaround.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookbehinds cont.

Improper syntax of the Positive Lookbehind (the expression to find should be on the right of the Lookbehind):

Locate expression A. Look left and test that expression B = expression A:

A(?<=B)

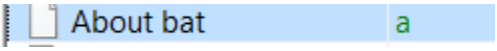
This is similar to the other RegExs' that have improper usage – comes down to A = B (or B = A) and A, in most instances can never equal B – which can be expressed as A ≠ B. So here are more examples, but listen – with each explanation doesn't it get easier to understand? This is because I am learning as you are learning.

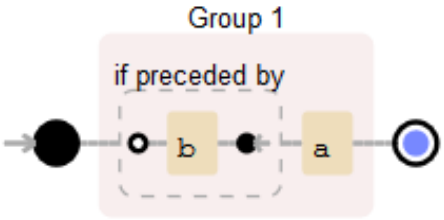
Let's say you have a string..

String = About bat

using a Positive Lookbehind in the proper manner -

Match: ((?<=b)a)
Replace: \1





Observations:

1. Look left after finding 'a'
2. Matches 'b' in bat

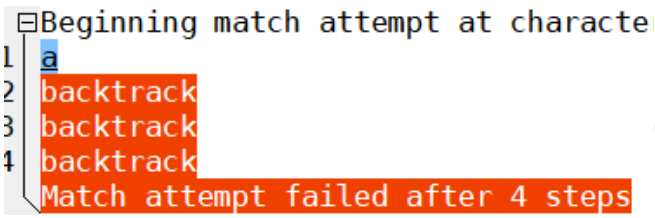
Tim Analysis: Using Regex Buddy

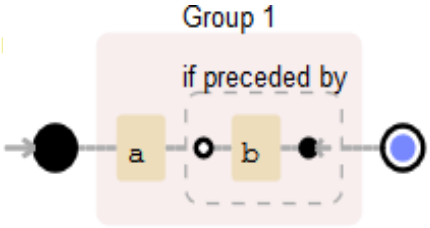
1. Moves forward and at each character position looks left (backtracks) to test for b.
2. Matches at b' in 'bat' b moves forward to match a.

now let's reverse it ...

Match: (a(?<=b))
Replace: \1

no match





Tim Analysis: Using Regex Buddy

1. Moves forward to each position to match for 'a'.
2. After matching 'a', in doing so, it moves to the next position.
3. Tests left (backtracks) to match for 'b' but because it already matched 'a' at this position, this becomes impossible because, 'a' previously matched, can never = 'b', whether it is at the position 'A' in 'About' or 'a' in 'bat'.

It will always be testing for 'b' in the exact same position that just matched 'a'.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookbehinds cont.

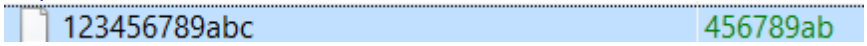
Improper syntax of the Positive Lookbehind: cont.

This is another example of the improper usage, but as previously explained under , 'Partial Matches & Wildcards', it will match because wildcards are used.

Example: Match against a character that is a 'b' preceded by any 8 characters.

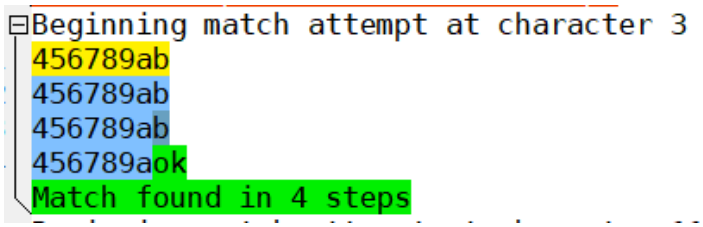
String = 123456789abc

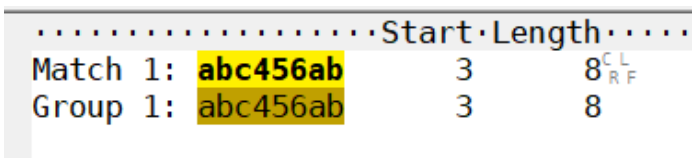
Match: (.{8})(?<=b)
Replace: \1

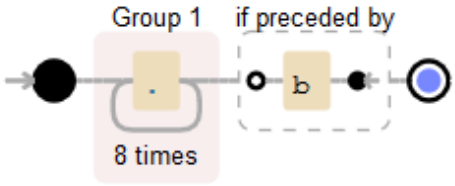


Match: (.{8})(?<=b)

Result: 123456789abc







Tim Notes:

1. In Regex Buddy, character 3 is actually the 4th character, '4'. This is because in Regex Buddy, it starts with the first character of the string at character position 0 and *not* at position 1. In other words, it begins the count of character positions beginning with zero. Position 0 is typically the Beginning of Line or BOL represented by the carat ' ^ ' Metacharacter in RegEx.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Positive Lookbehinds cont.

Improper syntax of the Positive Lookbehind: cont.

String = 123456789abc

Match: (.{8})(?<=b)

Where:

\1 = 456789ab

Tim Analysis: Using Regex Buddy

1..{8}	Matches against the first 8 characters. Current position = 8	Capture Group 1 = 12345678
2.(?<=b)	Tests each character looking for the 'b' Backtracks to character position 0 and tests for the "b' after matching the next 8 characters. Finding none, it will move forward through the string repeating until it has a match or has exhausted the string. This reevaluates Capture Group 1 each time.	

Character Position	Capture Group 1
0	12345678
1	23456789
2	3456789a
3	456789ab

	At character position 3, the '4', it finds a match, locating the 'b' at the current character position, position 10.	Changes Capture Group 1 = 456789ab
3.	Continue testing until no more matches can be made or EOL whichever comes first.	
4.	Lookbehind gives up any matches found, and returns to the position before the original match (this does not affect Capture Group 1 which is a separate match). Because of the previous Capture Group, the current position remains at 'b'. RegEx ends.	

Tim Notes:

1. You could also describe this as matching against 11 characters ending with a 'b'
2. For some people, Lookarounds (as well as other aspects of RegEx) are an easy subject, but for myself, it has been a struggle. I take you on this learning journey in real time – I ask the questions and try to find the answers, and in turn provide to you, that, which I have learned or still learning; from there you can determine if I am right or wrong.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookbehinds

Locate expression A. Look left NOT for B:

(?<!B)A

Example: Look for a 'c' not preceded by a 'q'

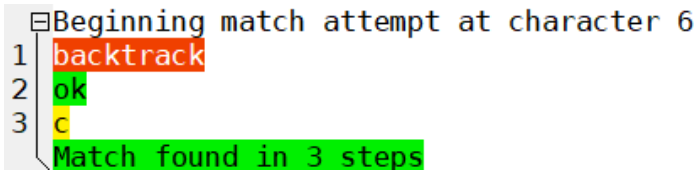
String = quiet cat quill quick

Match: ((?<!q)c)
Replace: \1

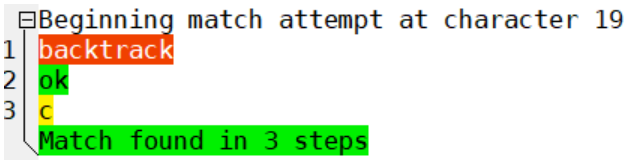
Match: (?<!q)c

Result: quiet cat quill quick

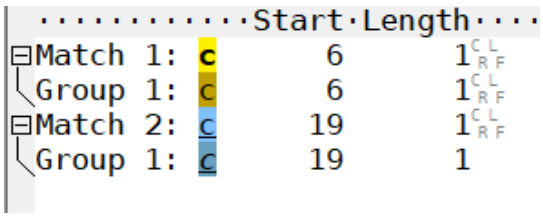
First Match:

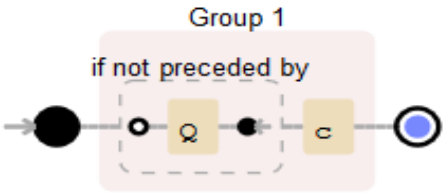


Second Match:



Sample:





RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookbehinds cont.

String = quiet cat quill quick

Match: ((?<!q)c)

Tim Analysis: Using Regex Buddy

1.(?<! q)	Negative Lookbehind. Look to the left. If character is NOT a single 'q' then expression = True and evaluation continues.= 'q' = FALSE At current character position backtrack and test the character is not a 'q'. a. At character position 0 = 'q' = Fail. b. At character position 1, 'u', backtrack to 0 = 'q' = Fail c. At character position 2, 'i', backtracks to 1 = 'u' = Matches.	
2. c	Move back to character position 2 and test for 'c' = 'i' = Fail.	
3.	Repeat for each subsequent position looking for a match. Matches at character position 6, the 'c' of 'cat'. a. the Current position was character 7, the 'a' of 'cat = not 'q' = Match. b. Backtracks to character position 6 and tests for 'c' = True	Capture Group 1 = c
4.	Continues to test the rest of the string looking for additional matches. Match found at character position 19. a. the Current position was character 20, the 'k' of 'quick' = not 'q' = Match. b. Backtracks to character position 19 and tests for 'c' = True	
5.	Continues to test the remainder of the string until no more matches can be made (string exhausted) or EOL whichever comes first.	
6.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 6, the 'c' of 'cat'. Further evaluation of the RegEx (there is none) would continue from this position.	

RegEx Manual Volume II

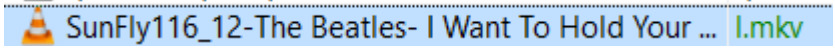
Understanding Lookarounds cont.

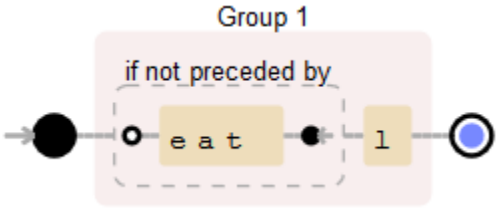
Examining Negative Lookbehinds cont.

Example: Look for 'l' not preceded by 'eat'

String = SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

Match: ((?<!eat)l)
Replace: \1

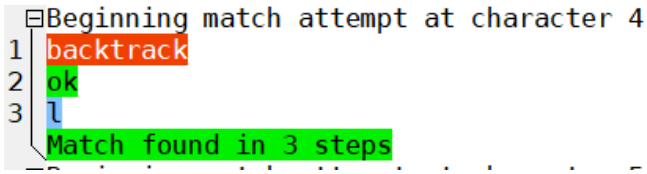




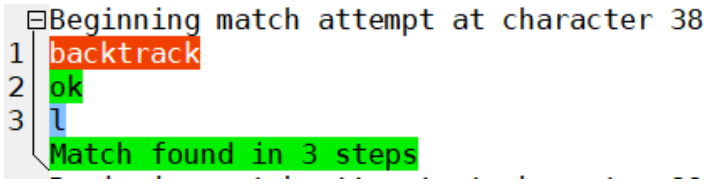
Match: ((?<!eat)l)

Result: SunFly116_12-The Beatles- I Want To Hold Your Hand.mkv

First Match:



Second Match:



Tim Analysis: Using Regex Buddy

1. (?<!eat)	Lookbehind. Look to the left. If character is NOT a single 'e' then expression = True and evaluation continues.= 'S' = TRUE Test each character position looking NOT for 'eat' followed by a 'l'.	
2. l	Backtracks testing each previous character for 'l' Matches at character position 4, the 'l' of 'Sunfly' Zero Length Assertion satisfied.	Capture Group 1 = l
3. (?>!eat)	Continue testing string looking for additional matches.	
4. l	Reaches EOL, finding no match begins to backtrack looking for an additional match for an 'l'. Matches at charcter position 38, the 'l' of 'Hold' Zero Length Assertion satisfied.	
5.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 4, the 'l' of 'Fly'. Further evaluation of the RegEx (there is none) would continue from this position.	

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookbehinds cont.

Improper Syntax of a Negative Lookbehind (the expression to find should be on the right of the Lookbehind):

Look for A ≠ B:

A(?<!B)

Example: Look for a 'c' that is not 'q'

Will always match as long as there is any 'c' in the string because 'c' ≠ 'q' and never will. Useless.

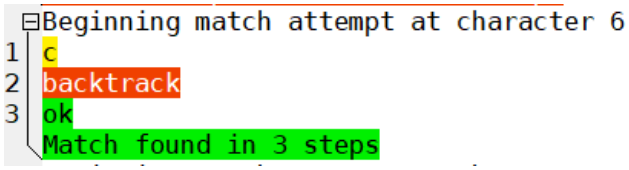
String = quiet cat quill quick

Match: (c(?<!Q))
Replace: \1

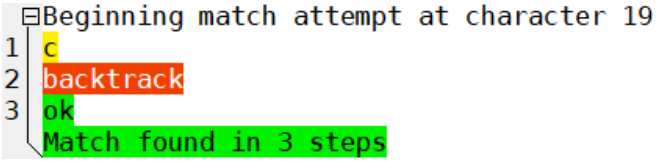
Match: c(?<!Q)

Result: quiet cat quill quick

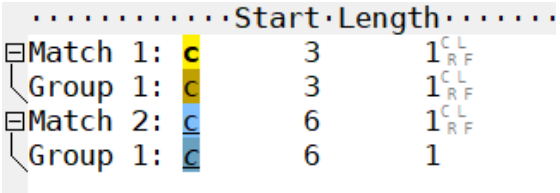
First Match:

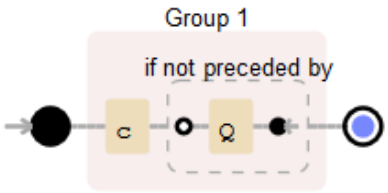


Second match:



Sample:





Although the result of (?<!Q)c is the same as c(?<!Q) how it got the result is not.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Negative Lookbehinds cont.

Improper Syntax of a Negative Lookbehind: cont.

Tim Analysis: Using Regex Buddy

1. c	Test each character position looking for a 'c'. Matches at character position 6, the 'c' of 'cat'.
2. (?<!Q)	Tests current position for Not 'Q' = 'c' = True True. Zero Length Assertion satisfied.
3.	Continues to test the rest of the string looking for additional matches. Match found at character position 19.
4.	Continues to test the remainder of the string until no more matches can be made (string exhausted) or EOL whichever comes first.
5.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 6, the 'c' of cat. Further evaluation of the RegEx (there is none) would continue from this position.

Summary:

(?<=)	- positive lookbehind	(?<!)	- negative lookbehind
(?=)	- positive lookahead	(?!)	- negative lookahead
(?:)	- non-marking group		

Syntax:

Look ahead positive (?=)

Find expression A. Look to the right *FOR* expression B:
A(?=B)

Look ahead negative (?!)

Find expression A. Look to the right *NOT* for expression B:
A(?!B)

Look behind positive (?<=)

Find expression A, Look to the left *FOR* expression B:
(?<=B)A

Look behind negative (?<!)

Find expression A, Look to the left *NOT* for expression B:
(?<!B)A

RegEx Manual Volume II

Understanding Lookarounds cont.

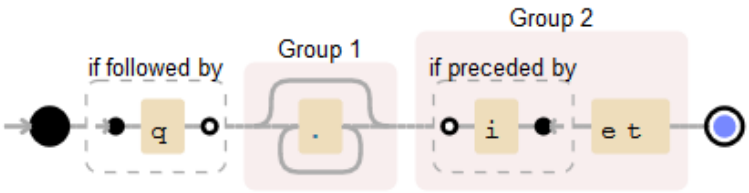
Here are some quick Examples to finish off this section:

Positive Lookahead with Positive Lookbehind

String = quiet cat quill quick

Match: (?=q)(.*?)((?<=i)et)
Replace: \2

quiet cat quill quick et



Where:
\1 = qui \2 = et

From this I can match any word that begins with 'q'. Just change the Lookbehind at the end.

Examples using same Replace string \2:

(?=q)(.*?)((?<=i)ll) matches against 'll' of 'quill' and (?=q)(.*?)((?<=i)ck) matches against 'ck' of 'quick

quiet cat quill quick ll

quiet cat quill quick ck

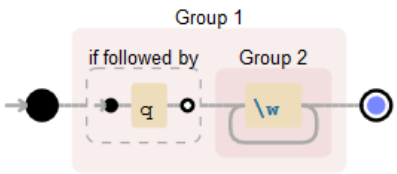
(?=c)(.*?)((?<=a)t) matches against 't' of 'cat'

quiet cat quill quick t

Positive Lookahead Examples to match word:

Match ((?=q)(\w+))
Replace \2 <space>

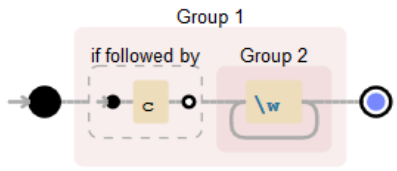
quiet cat quill quick quiet



1

Match ((?=c)(\w+))
Replace \2 <space>

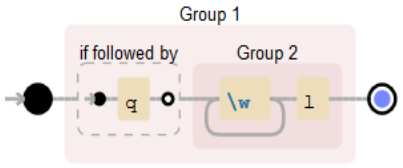
quiet cat quill quick cat



2

Match: ((?=q)(\w+l))
Replace: \2 <space>

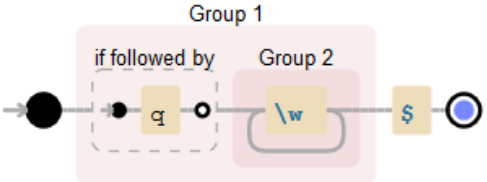
quiet cat quill quick quill



3

Match ((?=q)(\w+))$
Replace \2 <space>

quiet cat quill quick quick



4

Tim Notes:

1. The first example, matches the first word beginning with 'q'.
2. The Second example, matches 'cat' simply by changing the Lookahead.
3. The Third example, adds the 'l' because there is more than one word that begins with 'q'.
4. The fourth example matches last word of string that must begin with 'q'

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookarounds

You can also combine them: i.e., Positive Lookbehind and a Positive Lookahead

Example: Locate the 1st 'bar' in the string that is *preceded* by 'foo' *followed* by a 2nd occurrence of 'bar'

String = foobarbarfoo

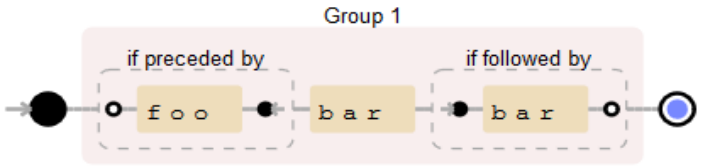
Match: ((?<=foo)bar(?=bar))
Replace: \1

foobarbarfoo	bar

```
Beginning match attempt at character 3
1  f
2  fo
3  foo
4  ok
5  b
6  ba
7  bar
8  barb
9  barba
10 barbar
11 barok
   Match found in 11 steps
```

Match: (?<=foo)bar(?=bar)

Result: foobarbarfoo



Tim Analysis: Using Regex Buddy

1. (?<=foo)	Locate 'foo' in string. Matches at character position 0. Current position = 2
2. bar	Test next character, position 3 for 'b' = True Test for 'ar' = 'bar' = True. Zero Length Assertion Satisfied. Current position = position 5, the 'r' of the first 'bar'
3. (?>foo)	Continue testing string looking for a second occurrence of 'foo' after position 5. Matches at position 9 = 'foo' = True
4.	Test next character, position 12 for 'b' = EOL = False
5.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 5, the 'r' of the first 'bar'. Further evaluation of the RegEx continues from this position.
6. (?=bar)	Look ahead tests the next character, 'b' at position 6 = 'b' = True. Test for 'ar' = 'bar' = True. Zero match Assertion Satisfied. Current position = position 8, the 'r' of the second 'bar'.
7.	Continue testing looking for further matches of 'bar until no more matches can be made or EOL whichever comes first.
8.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 8, the 'r' of the second 'bar'. Further evaluation of the RegEx would continue from this position (there is no more).

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookarounds cont.

Example: Look for a <space> *preceded* with 'lj' *followed* by any word *followed* by <space>'lk'

String = My lj cat lk is furry

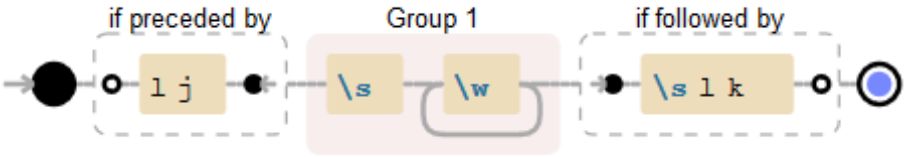
Match: (?<=lj)(\s\w+)(?=\slk)
Replace: \1

My lj cat lk is furry → cat

Match: (?<=lj)(\s\w+)(?=\slk)s

Result: My lj cat lk is furry

```
Beginning match attempt at character 5
1  l
2  lj
3  ok
4
5   cat
6   cat
7   cat
8   cat l
9   cat lk
0   catok
Match found in 10 steps
```



Tim Analysis: Using Regex Buddy

1. (?<=lj)	Positive Lookbehind Test each character position looking for an 'l'. Matches at character position 5. Tests that the next character is a 'j' = True.	
2. \s	Tests that the next character is a <space> = True Zero Length Assertion Satisfied. Current position = position 7, the <space> following 'lj'	Capture Group 1 = <space>
3. (?<=lj)	Continue testing looking for further matches of 'lj' until no matches no matches can be made or EOL whichever comes first.	
4.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 7, the <space> following 'lj'. Further evaluation of the RegEx continues from this position.	
5. \w	Test next character position for a Word Character = 'c'	Capture Group 1 = <space>c
6. +	Make it Greedy	Capture Group 1 = <space>cat
7. (?=\slk)	Test the next character for a <space> = True Test the following two character positions for 'lk' = True. Zero Length Assertion satisfied. Current position = 11.	
8.	Continue testing looking for further matches of '<space>lk' until no more matches can be made or EOL whichever comes first.	
9.	Lookaround gives up the match, returns to the previous position before the Zero Length Assertion, position 11, the 'k' of 'lk'. Further evaluation of the RegEx continues from this position (there is none).	

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

IMORTANT

The Next few pages are meant to show you what works and what doesn't and hopefully you will understand why.

Example: Look for any three characters *beginning* with a 't' *ending* with 'y'

String = toy mouse

Match: (?=t)(...)(?<=y)
Replace: \1

toy mouse	toy

Match: (?=t)(...)(?<=y)

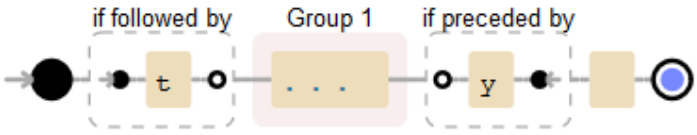
Result: toy mouse

Tim Analysis: Using Regex Buddy

1. (?=t)	Finds the 't' in 'toy'. Continue testing.	
2.	Exhausts string.	
3.	Gives up the match and returns to the 't' - beginning of string	
4. …	Matches 'toy'. Current position = 'y' of 'toy'	Capture Group 1 = toy
5. (?<=y)	Tests current character = 'y' = True. Lookbehind satisfied.	
6.	Exhausts string. Returns to original position = 'y' of 'toy'	

The Lookbehind at the end of the RegEx tests the current position for the 'y'

```
Beginning match attempt at character 0
1  t
2  ok
3  t
4  to
5  toy
6  toy
7  toy
8  took
   Match found in 8 steps
```



Tim Notes:

1. It may be easier to think of 'ending with' rather than 'preceding' because the Lookbehind tests the current character.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

If we add a (…) to the end of the RegEx, will that make a difference to the result of the Lookbehind?

Example: Look for any three characters *beginning* with a 't' *ending* with 'y' *followed* by 3 more characters

String = toy mouse

Match: `(?=t)(...)(?<=y)(...)`

Result: `toy mouse`

Tim Analysis: Using Regex Buddy

1. (?=t)	Finds the 't' in 'toy'. Continue testing.	
2.	Exhausts string.	
3.	Gives up the match and returns to the 't' - beginning of string	
4. …	Matches 'toy'. Current position = 'y' of 'toy'	Capture Group 1 = toy
5. (?<=y)	Tests current character = 'y' = True. Lookbehind satisfied.	
6.	Exhausts string. Returns to original position = 'y' of 'toy'	
7. …	Matches next three characters = '<space>mo'	Capture Group 2 = <space>mo

```
Beginning match attempt at character 0
1  t
2  ok
3  t
4  to
5  toy
6  toy
7  toy
8  took
9  to
10 to m
11 to mo
12 to mo
Match found in 12 steps
```

A Little difficult to comprehend, because 't' was in position zero.

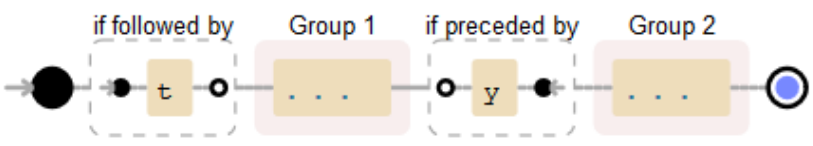
If the string was changed to:

String = a toy mouse

Would there be any difference?

Result: `a toy mouse`

```
Beginning match attempt at character 2
1  t
2  ok
3  t
4  to
5  toy
6  toy
7  toy
8  took
9  to
10 to m
11 to mo
12 to mo
Match found in 12 steps
```



The answer is no, on both counts, except the match is made at character position 2 instead of zero.

RegEx Manual Volume II

Understanding Lookarounds cont.

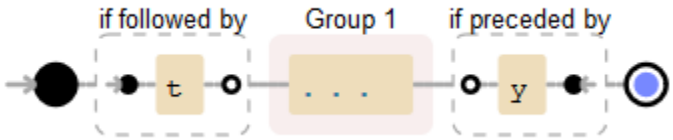
Examining Combination Lookaounds cont.

Example: Look for any three characters *beginning* with a 't' *ending* with 'y' cont.

Using an improper syntax allows the match to occur between the Lookarounds.

More Samples:

toy taking Tot
Tolstoy tally



Match: (?=t)(...)(?<=y)
Replace: \1 <space>

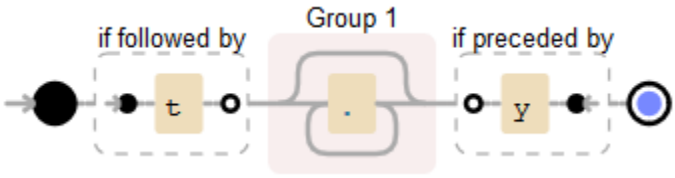
Name	New Name
taking	taking
tally	tally
Tolstoy	toy
Tot	Tot
toy	toy

Matches the lowercase 'toy' of 'Tolstoy'

Match: `(?=t)(...)(?<=y)`

Result: `Tolstoy`

Example: Look for any characters *beginning* with a 't' *ending* with 'y'



Match: (?=t)(.*)(?<=y)
Replace: \1 <space>

Name	New Name
taking	taking
tally	tally
Tolstoy	toy
Tot	Tot
toy	toy

Matches 'tally'

Match: `(?=t)(.*)(?<=y)`

Result: `tally`

If I change Tolstoy to lowercase, it will match entire string.

Name	New Name
tolstoy	tolstoy

It is the Capture Group (…) or (.*) that is performing the capturing not the Lookarounds. For the most part the analysis is pretty simple. Look for a lowercase t – Matches against the first 't' it finds. Capture the entire string in the case of (.*) or the next three characters (…) and keep the match only if at the end of that capture, the current character is a 'y'.

RegEx Manual Volume II

Understanding Lookarounds cont.

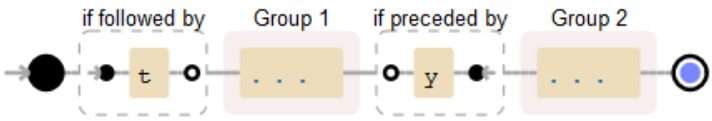
Examining Combination Lookaounds cont.

Using the same example, apply it to other files -

Example: Look for any three characters *beginning* with a 't' *ending* with 'y" *followed* by 3 more characters

Match: (?=t)(...)(?<=y)(...)
Replace: \1-\2

toy mouse	toy- mo
Hello, there	Hello, there
Hello there	Hello there
123E	123E
Catch football	Catch football
email	email
abcc	abcc
fate any bar	fate any bar



Tim Analysis: Using Regex Buddy

There are two filenames that contain both a 't' and a 'y': 'toy mouse' and 'fate any bar' but only 'toy mouse' matches.

1. (?=t) Look ahead for a 't'
2. … Match against three characters
3. (?<=y) Look behind (including current position) for a 'y'
4. … Match against three characters.

Catch Football

1. Finds the 't' in 'Catch' as well as the 't' in 'football'.
2. Continues testing until string exhausted.
3. Gives up the match and returns to the 't' in 'Catch'.
2. Matches 'tch' Current position = 'h' of 'Catch' Capture Group 1 = tch
3. Tests current character position for the 'y' = 'h' = Fail

fate any bar

1. Finds the 't' in 'fate'.
2. Continues testing until string exhausted.
3 Gives up the match and returns to the 't' of 'fate'
4. Matches 'te<space>'. Current position = <space> following 'fate' Capture Group 1 = te<space>
5. Tests current character position for the 'y' = <space> = Fail

Conclusion:

Because of the restrictive nature of the RegEx in this example, it would only match against 'toy mouse'

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

Example: Look for an any 7 characters *beginning* with a 'a' *ending* in a "y' *followed* by 3 more characters

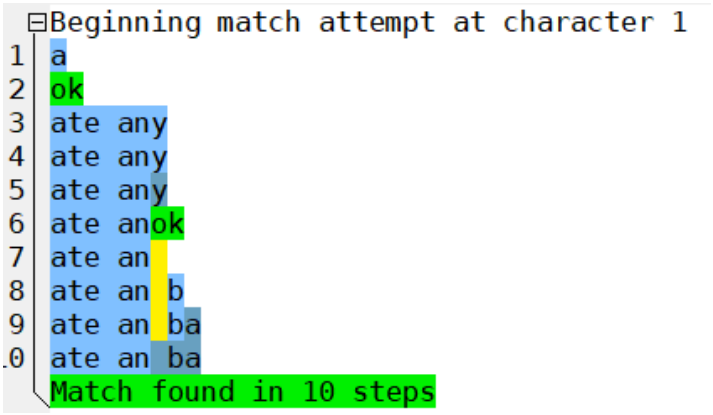
String = fate any bar

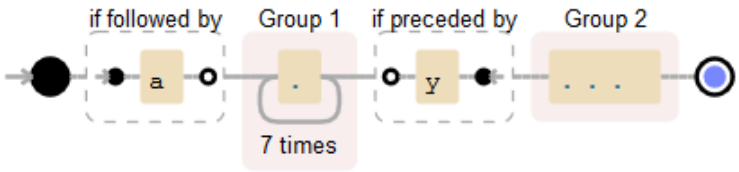
If I wanted to match 'fate any bar' I could use ….

Match: (?=a)(.{7})(?<=y)(...)
Replace: \1\2

Tim Analysis: Using Regex Buddy

1. (?=a)	Finds the 'a' in 'fate'.	
2.	Continue testing. Exhaust string. Give up the match and returns to the 'a of 'fate'	
3. .{7}	Matches next 7 charcaters. Current position = 'y' of 'any'	Group 1 = ate<space>any
4. (?<=y)	Tests current character = 'y' = True. Lookbehind satisfied.	
5.	Exhausts string. Returns to original position = 'y' of 'any'	
6. …	Matches next three characters = '<space>ba'	Group 2 = <space>ba





Tim Notes:

1. (?=a)(.{7}) is an improper syntax of a Positive Lookahead as previously discussed. The only reason it matches is because wildcards are used.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

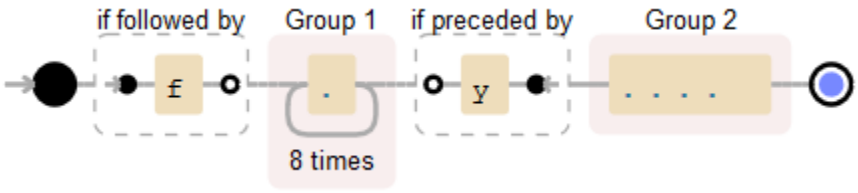
Example: Look for any 8 characters *beginning* with an 'f' *ending* in a 'y' *followed* by 3 more characters

String = fate any bar

… or this …

Match: (?=f)(.{8})(?<=y)(....)
Replace: \1\2<space>

fate any bar fate any bar



Match: (?=f)(.{8})(?<=y)(....)

Result: fate any bar

Tim Analysis: Using Regex Buddy

1. (?=f)	Finds the 'f' in 'fate'.	
2.	Continue testing. Exhaust string.	
3.	Give up the match and returns to the 'f of 'fate'	
4. .{8}	Matches 'fate<space>any'. Current position = 'y' of 'any'	Group 1 = fate<space>any
5. (?<=y)	Tests current character = 'y' = True. Lookbehind satisfied.	
6.	Exhausts string. Returns to original position = 'y' of 'any'	
7. ….	Matches next four characters = '<space>bar'	Group 2 = <space>bar

```
Beginning match attempt at character 0
  f
  ok
  fate any
  fate any
  fate any
  fate anok
  fate an
  fate an b
  fate an ba
  fate an bar
  fate an bar
Match found in 11 steps
```

Tim Notes:

1. (?=f)(.{8}) is an improper syntax of a Positive Lookahead as previously discussed. The only reason it matches is because wildcards are used.

RegEx Manual Volume II

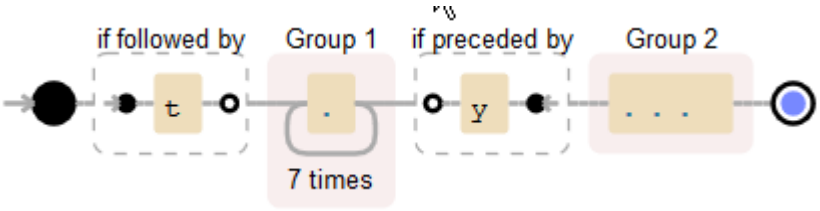
Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

Example: Look for a any 7 characters *beginning* with a 't' *ending* in a 'y' *followed* by 3 more characters

String = fate any bar

This won't work ….



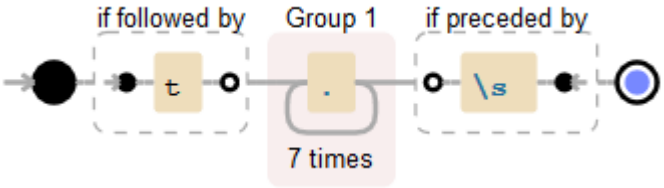
Match: (?=t)(.{7})(?<=y)(...)
Replace: \1

1. (?=t) Finds the 't' in 'fate'.
2. Continue testing. Exhaust string.
3. Gives up the match and returns to the 't' of 'fate'
4. .{7} Matches 'te'<space>'any' <space> Capture Group 1 = te<space>any<space>
   Current position = <space> following 'any'
5. (?<=y) Tests current position for 'y' = <space> = Fail.

… This will ….

Example: Look for a any 7 characters *beginning* with a 't' *ending* with a <space>

Match: (?=t)(.{7})(?<=\s)
Replace: \1



Match: (?=t)(.{7})(?<=\s)

Result: fate any bar

Tim Analysis: Using Regex Buddy

1. (?=t) Finds the 't' in 'fate'.
2. Continue testing. Exhaust string.
3. Give up the match and returns to the 't of 'fate'
4. .{7} Matches next 7 characters Group 1 = te<space>any<space>
   Current position = <space> after 'any'
5. (?<=\s) Tests current character = <space> = True. Lookbehind satisfied.
6. Exhausts string. Returns to original position = <space> after 'any'

```
Beginning match attempt at character 2
1 t
2 ok
3 te any
4 te any
5 te any
6 te anyok
Match found in 6 steps
```

Tim Notes:

1. (?=t)(.{7}) is an improper syntax of a Positive Lookahead as previously discussed. The only reason it matches is because wildcards are used.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

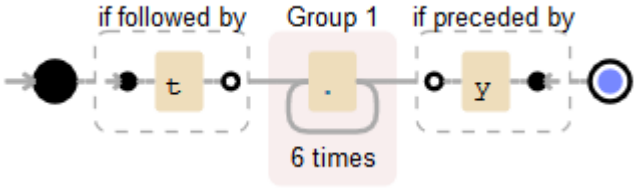
Example: Look for any 6 characters *beginning* with a 't', *ending* with 'y'

String = fate any bar

… and this will work as well …

Match: (?=t)(.{6})(?<=y)
Replace: \1

fate any bar te any



Match: (?=t)(.{6})(?<=y)

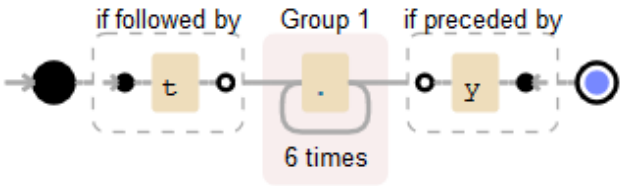
Result: fate any bar

Tim Analysis: Using Regex Buddy

1. (?=t) Finds the 't' in 'fate'.
2. Continue testing. Exhaust string.
3. Give up the match and returns to the 't of 'fate'
4. .{6} Matches next 6 characters — Group 1 = te<space>any
   Current position = 'y' of 'any'
5. (?<=y) Tests current character = 'y' = True. Lookbehind satisfied.
6. Exhausts string. Returns to original position = 'y' of 'any'

Beginning match attempt at character 2

```
1 t
2 ok
3 te any
4 te any
5 te any
6 te anok
Match found in 6 steps
```



Tim Notes:

1. (?=t)(.{6}) is an improper syntax of a Positive Lookahead as previously discussed. The only reason it matches is because wildcards are used.

RegEx Manual Volume II

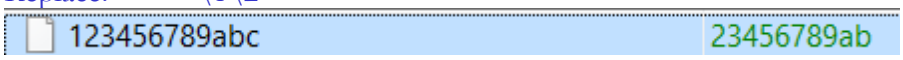
Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

Example: Look for any three characters *followed* by a '5' *followed* by any 7 characters *ending* with a 'b'

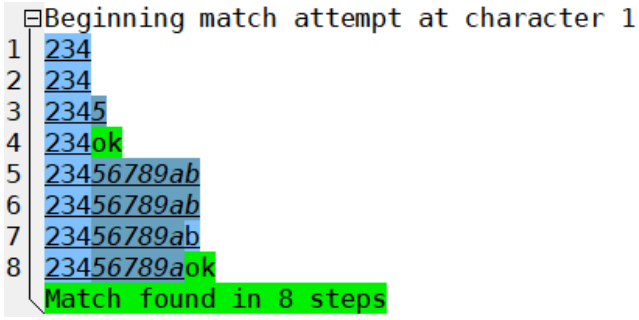
String = 123456789abc

Match: (.{3})(?=5)(.{7})(?<=b)
Replace: \1\2

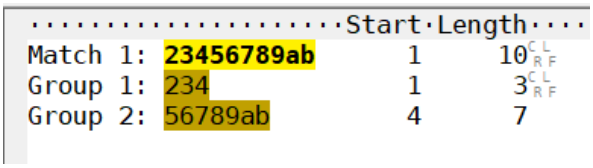


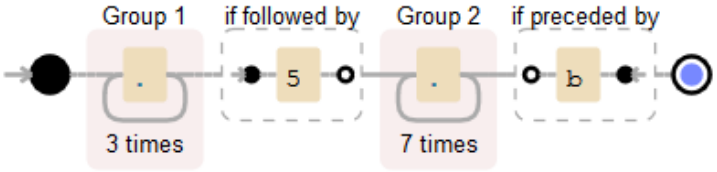
Match: (.{3})(?=5)(.{7})(?<=b)

Result: 123456789abc



Sample:





Tim Analysis: Using Regex Buddy

1. .{3}	Match against 1st three characters	Capture Group 1 = 123
2. (?=5)	Look ahead to find the '5'	Changes Capture Group 1 = 234
	Finds the '5' and reevaluates Group 1's value	
	Current position = character position 3, the '4'	
3.	Continue testing for additional '5'. Exhaust string.	
4.	Give up the match and returns to the '4'	
5. .{7}	Matches the next seven characters	Capture Group 2 = 56789ab
	Current position = 'b'	
6. (?<=b)	Test current character = b = True	
7.	Continue testing. exhaust string.	
8.	Give up the match and return to position 'b'	
	RegEx would continue from this position (none).	

RegEx Manual Volume II

Understanding Lookarounds cont.

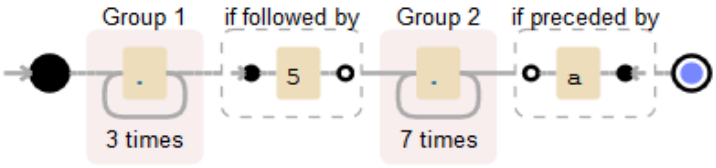
Examining Combination Lookaounds cont.

This doesn't work …

Example: Look for any three characters *followed* by a '5' *followed* by any 7 characters *ending* with an 'a'

String = 123456789abc

Match: `(.{3})(?=5)(.{7})(?<=a)`



Tim Analysis: Using Regex Buddy

1. .{3}	Match against 1st three characters	Capture Group 1 = 123
2. (?=5)	Look ahead to find the '5'	Changes Capture Group 1 = 234
	Finds the '5' and reevaluates Group 1's value	
	Current position = character position 3, the '4'	
3.	Continue testing for additional '5'. Exhaust string.	
4.	Give up the match and returns to the '4'	
5. .{7}	Matches the next seven characters	Capture Group 2 = 56789ab
	Current position = 'b'	
6. (?<=a)	Test current character = b = Fail	

```
⊟Beginning match attempt at character 0
1 123
2 123
3 123backtrack
4 123backtrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 1
1 234
2 234
3 2345
4 234ok
5 23456789ab
6 23456789ab
7 23456789abacktrack
8 23456789abacktrack
9 234backtrack
10 backtrack
  Match attempt failed after 10 steps
⊟Beginning match attempt at character 2
1 345
2 345
3 345backtrack
4 345backtrack
5 backtrack
  Match attempt failed after 5 steps
```

```
⊟Beginning match attempt at character 3
1 456
2 456
3 456backtrack
4 456backtrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 4
1 567
2 567
3 567backtrack
4 567backtrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 5
1 678
2 678
3 678backtrack
4 678backtrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 6
1 789
2 789
3 789backtrack
4 789backtrack
5 backtrack
  Match attempt failed after 5 steps
```

```
⊟Beginning match attempt at character 7
1 89a
2 89a
3 89abacktrack
4 89abacktrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 8
1 9ab
2 9ab
3 9abbacktrack
4 9abbacktrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 9
1 abc
2 abc
3 abcbacktrack
4 abcbacktrack
5 backtrack
  Match attempt failed after 5 steps
⊟Beginning match attempt at character 10
1 backtrack
2 backtrack
  Match attempt failed after 2 steps
⊟Beginning match attempt at character 11
1 backtrack
2 backtrack
  Match attempt failed after 2 steps
```

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

.. While this works …

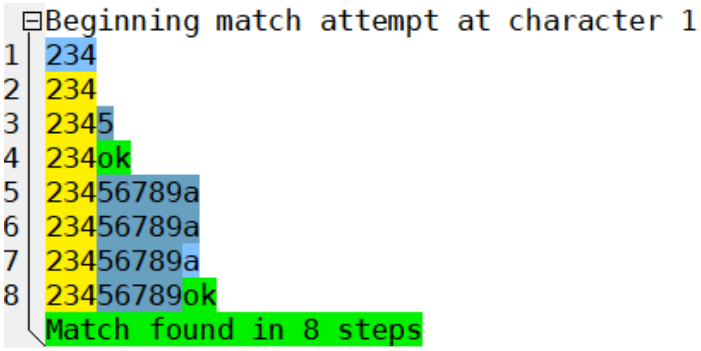
Example: Look for any three characters *followed* by a '5' *followed* by any 6 characters *ending* with an 'a'

String = 123456789abc

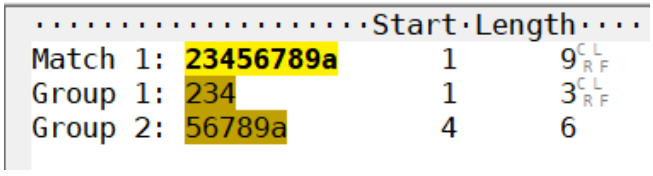
Match: (.{3})(?=5)(.{6})(?<=a)
Replace: \1\2

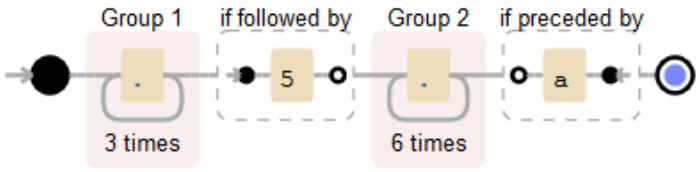
Match: (.{3})(?=5)(.{6})(?<=a)

Result: 123456789abc



Sample:





Tim Analysis: Using Regex Buddy

1. .{3}	Match against 1st three characters	Capture Group 1 = 123
2. (?=5)	Look ahead to find the '5'	Changes Capture Group 1 = 234
	Finds the '5' and reevaluates Group 1's value	
	Current position = character position 3, the '4'	
3.	Continue testing for additional '5'. Exhaust string.	
4.	Give up the match and returns to the '4'	
5. .{6}	Matches the next six characters	Capture Group 2 = 56789a
	Current position = 'a'	
6. (?<=a)	Test current character = a = True	
7.	Continue testing. exhaust string.	
8.	Give up the match and return to position 'a'	

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

… This doesn't ….

Example: Look for any three characters *followed* by a '5' *followed* by any 6 characters *ending* with a '9'

String = 123456789abc

Match: `(.{3})(?=5)(.{6})(?<=9)`

Tim Analysis: Using Regex Buddy

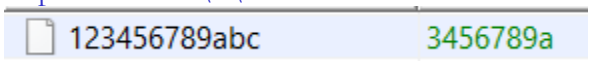
1. .{3}	Match against 1st three characters	Capture Group = 123
2. (?=5)	Look ahead to find the '5' Finds the '5' and reevaluates Group 1's value Current position = character position 3, the '4'	Changes Capture Group 1 = 234
3.	Continue testing for additional '5'. Exhaust string.	
4.	Give up the match and returns to the '4'	
5. .{6}	Matches the next six characters Current position = 'a'	Capture Group 2 = 56789a
6. (?<=9)	Test current character = a = Fail	

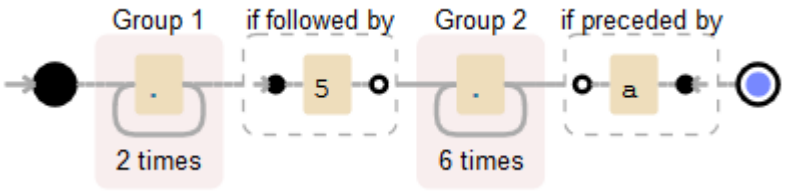
and finally, this works:

Example: Look for any two characters *followed* by a '5' *followed* by any 6 characters *ending* with an 'a'

String = 123456789abc

Match: (.{2})(?=5)(.{6})(?<=a)
Replace: \1\2





Where:

\1 = 34 \2 = 56789a

Match: `(.{2})(?=5)(.{6})(?<=a)`

Result: 123456789abc

Tim Notes:

1. Did not include analysis – by now, given the past examples, you should be able to work it out for yourself.

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

Another example: Look for a 't' followed by any 4 characters ending with a <space> followed by any three characters

Match: (?=t)(....)(?<=\s)(...)
Replace: \2-\4

toy mouse	toy -mou
Hello, there	Hello, there
Hello there	Hello there
123E	123E
Catch football	tch -foo
email	email
abcc	abcc
fate any bar	fate any bar

'toy mouse' = 'toy mou'
'Catch football' = 'tch foo'

Tim Analysis: Using Regex Buddy

1. (?=t)(….) The positive lookahead looks to locate a 't' followed by any 4 characters…

Matches:

toy mouse	toy
Hello, there	ther
Hello there	ther
123E	123E
Catch football	tch
email	email
abcc	abcc
123456789abc	123456789abc
fate any bar	te a

Tim Notes:

1. Only the first 't' encountered in the string and subsequent 3 characters are captured. You should know why by now. Hint – it has to do with the Zero Length Assertive.

2. (?=t)(….) is an improper syntax of a Positive Lookahead. The only reason it matches is because wildcards are used.

You end up with:

'toy mouse' = 'toy<space>' 'Hello, There' = 'ther' 'Hello There' = 'ther',
'Catch football' = 'tch<space>' 'fate any bar' = 'te<space>a'

and when you add this:

2. (?<=\s) … The last expression ends with a <space> (tests the current character)

you get this ….

(?=t)(....)(?<=\s)

toy mouse	toy
Hello, there	Hello, there
Hello there	Hello there
123E	123E
Catch football	tch
email	email
abcc	abcc
fate any bar	fate any bar

Disqualifies 'Hello, There', 'Hello There' and 'fate any bar', because there is no <space> character in the current position.

3. And when you add the (…) at the end, you end up with the final result (see above).

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

So far combinations of Positive Lookaheads with Positive Lookbehinds have been illustrated. Here is one using a Positive Lookahead with a Negative Lookahead.

Example: Locate 't' followed by any three characters that is not followed by a 'y', followed by three more characters

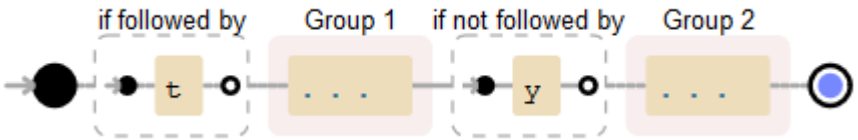
Match: (?=t)(...)(?!y)(...)
Replace: \2 -\4

toy mouse	toy- mo
Hello, there	Hello, there
Hello there	Hello there
123E	123E
Catch football	tch- fo
email	email
abcc	abcc
fate any bar	te -any

Tim Notes:

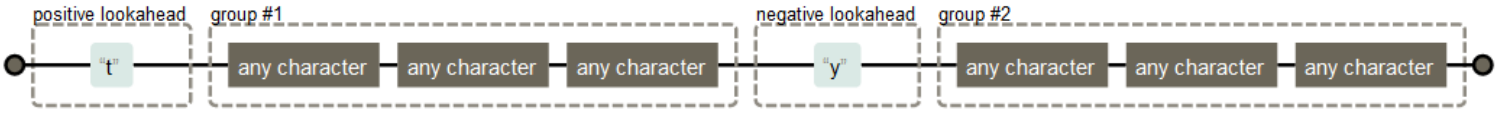
1. Only the first 't' encountered in the string and subsequent 3 characters are captured. You should know why by now. Hint – it has to do with the Zero Length Assertive.

2. (?=t)(...) is an improper syntax of a Positive Lookahead. The only reason it matches is because wildcards are used.

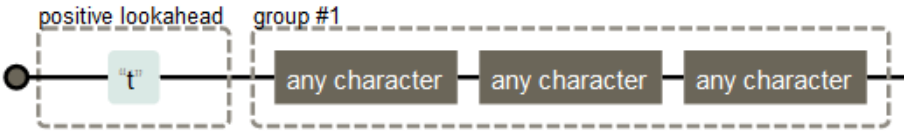


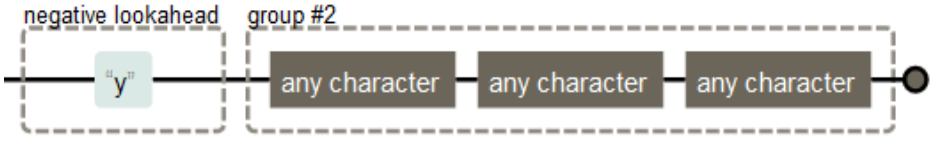
Sample:

```
..................
Match 1: toy mo
Group 1: toy
Group 2:  mo
```



Expanded:






```
Beginning match attempt at character 0
1  t
2  ok
3  t
4  to
5  toy
6  toy
7  toybacktrack
8  toyok
9  toy
10 toy m
11 toy mo
12 toy mo
Match found in 12 steps
```

RegEx Manual Volume II

Understanding Lookarounds cont.

Examining Combination Lookaounds cont.

Example: Locate 't' followed by any three characters that is not followed by a 'y', followed by three more.. cont.

Match: (?=t)(...)(?!y)(...)

Tim Analysis: Using Regex Buddy

1. (?=t)(…) Look for a 't' followed by any three characters.

toy mouse	toy
Hello, there	the
Hello there	the
123E	123E
Catch football	tch
email	email
abcc	abcc
fate any bar	te

'toy mouse' = 'toy'
'Hello, there' = 'the'
'Hello there' = 'the'
'Catch football' = 'tch'
'fate any bar' = 'te<space>'

2. (?!y) The Negative Lookahead tests the next character for a 'y'

toy mouse	toy
Hello, there	the
Hello there	the
123E	123E
Catch football	tch
email	email
abcc	abcc
fate any bar	te

Summary:

There is no 'y' following 'toy' = True
There is no 'y' following 'the' = True
There is no 'y' following 'tch' = True
There is no 'y' following 'te<space>' = True

3. And when you add the (…) at the end, you end up with the final result (see above).

Disqualifies:

'Hello there' - the three character match hits EOL (Current position was the 'e' of 'there')
'Hello there' - the three character match hits EOL (Current position was the 'e' of 'there')

RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors

When the RegEx engine is at the beginning of the string, there is nothing that precedes, so the value is the beginning of the string. Thus all of these match at the beginning of a string:

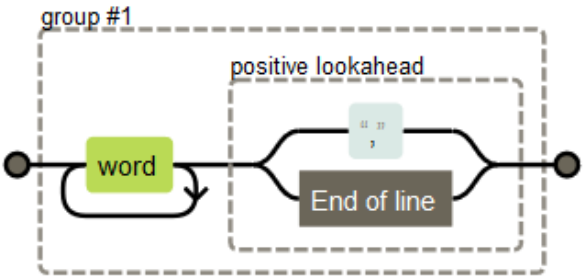
^	Indicates beginning of string (Beginning of Line or BOL)
(?<=^)	Lookbehind = Beginning of String
(?=^)	Lookahead = will match Beginning of String

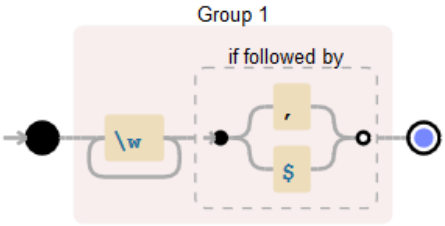
In the following example, we use the end-of-string anchor $ within a lookahead. The engine matches word characters, then asserts that what follows is either (the Alternate ' | ') a comma or the end of the string (End of Line or EOL).

Example: Locate a Word followed by either a comma or EOL

Match: (\w+(?=,|$))
Replace: \1

toy mouse	mouse
Hello, there	Hello
Hello there	there
123E	123E
Catch football	football
email	email
A1B1C1vsD1D2D3	A1B1C1vsD1D2D3
Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31	31





Essentially it matches all the filenames. Those that only have a single word are matched with the EOL alternate, but display no new name because the entire string is matched. Those that have more than one word with no comma have the last word matched by the EOL alternate, while those with a comma have the preceding word matched with the comma alternate – i.e., 'Hello, there' matches the first word, 'Hello', while 'Hello there' matches the last word, 'there'.

To display all matched names, you could add any change to the replacement string – for eaxmple, a <space>, to display a change that would generate a New Name value. If you did this then it would display matches for '123E', 'email' and 'A1B1C1vsD1D2D3'.

Sample:

```
................
Match 1: mouse
Group 1: mouse
```

RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

When the RegEx engine is positioned at the beginning of the string ^, you could say that what immediately precedes or follows this position will also indicate the beginning of the string. Character position 1, as we think of it as the first character of the string, is also representative of this (note that if you use the Regex Buddy program, a popular RegEx tool, it counts this first character as position zero). Just as you can't move forward past the $ that indicates the End of Line or EOL, you can't backtrack past the ^ (Beginning of Line or BOL).

Whenever an anchor is within a lookaround, the meaning of the anchor is the same as if the lookaround wasn't present. This is convenient for several reasons:

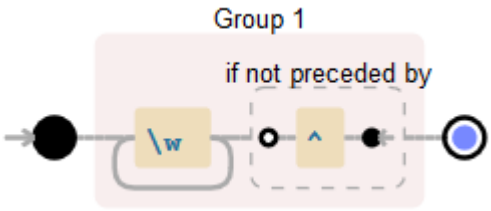
1. An anchor can be used in an alternation within a lookaround to make a complex delimiter, as in the \w+(?=,|$) example on the previous page.

2. A Negative Lookahead or a Negative Lookbehind can be used to assert that the current position does not correspond to an anchor. i.e., (?<!^) checks that the current position is not the beginning of the string (Beginning of Line or BOL).

Example: Locate a word not ending with the beginning of the string

Match: (\w+(?<!^))
Replace: \1<space>

toy mouse	toy
Hello, there	Hello
Hello there	Hello
123E	123E
Catch football	Catch
email	email
UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf	UK .pdf
123456789abc	123456789abc



This will match the first word of all filenames because the moment a Word Character is matched, the RegEx Engine moves forward, thus, the current character position can never be the beginning of the string (Beginning of Line or BOL).

Tim Analysis: Using Regex Buddy

Using string = 'Hello, there'

1. \w	Locates the 'H' of 'Hello there'	Capture Group 1 = H
2. +	Make it Greedy	Capture Group 1 = Hello
	Current position = 'o' of 'Hello'	
3. (?<!^)	Test current character for beginning of string (BOL)	
	= False = Match	
4.	Continue searching looking for other matches of beginning of string until string exhausted or EOL whichever is first.	
5.	Give up the match, move back to original position, the 'o' of 'Hello'	

RegEx Manual Volume II

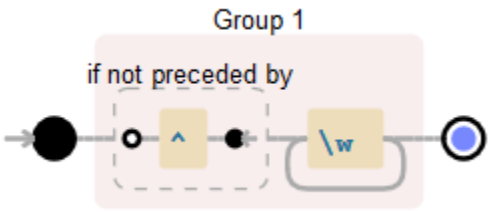
Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

Example: Not the beginning of the string (Beginning of Line or BOL) *followed* by a Word Character (or a Word Character not *preceded* by the beginnning of the string).

Match: ((?<!^)\w+)
Replace: \1<space>

toy mouse	oy
Hello, there	ello
Hello there	ello
123E	23E
Catch football	atch
email	mail
UK}Peter Ford, Jeremy Jordan, Lisa Wendt.pdf	K .pdf
123456789abc	23456789abc



Tim Analysis: Using Regex Buddy

Using string = 'Hello, there'

1. (?<!^) Backtrack from current character to test for the beginning of the string. 1st character position = 'H' = True = Fail

2. Move to next character, the 'e' of 'Hello'
Backtrack from current character to test for the beginning of the string. = 'H' = False = Match

Current position = 'H' of 'Hello'

Continue testing looking for further matches until string is exhausted or EOL whichever comes first.

3. Give up the match, move back to original position, the 'H' of 'Hello'
4. \w Test against next character for Word Character = 'e' = True — Capture Group 1 = e
5. + Make it Greedy — Capture Group 1 = ello

In each match, the first character of every first word of the string is cut off because it failed in step 1.

Tim Notes:

1. Weird descriptions on many of these examples in the book, I know. But they are just there to help illustrate.

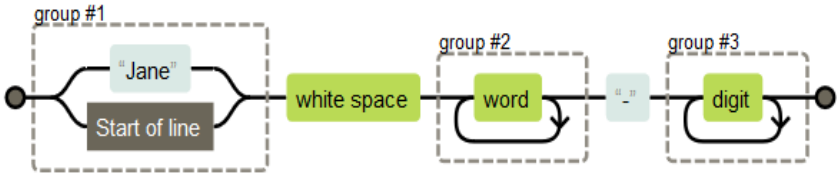
RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

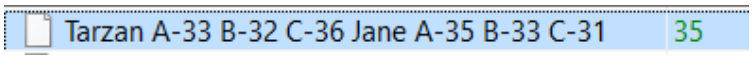
Jane & Tarzan have swim meet. Here are the final results:

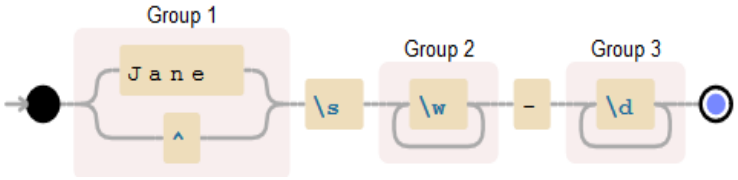
String = Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31



To match against Jane's first score:

Match: ((?:Jane|^))\s(\w+)-(\d+)
Replace: \3





Tim Analysis: Using Regex Buddy

1. ((?:Jane|^))	This is a non-marking group – typically not captured but I captured it anyway. Provides a choice between 'Jane' and the start of string. Matches ^ as the beginning of string because the beginning of the string comes before 'Jane' is encountered. Current position = Beginning of string (BOL)	Capture Group 1 = Matched but no Name Change
2. \s	Match against <space> (Not Captured) This is the <space> following 'Jane'. Because the next character ' T' of 'Tarzan', is not a <space>, it falls back to the (?:Jane|^) expression. This time the current position is not the start of the string, so it moves forward to 'Jane'. This reevaluates Capture Group 1 to 'Jane'. Current position is <space> following 'Jane'.	Changes Capture Group 1 = Jane
3. \w	Match against next Word character	Capture Group 2 = A
4. +	Make it Greedy until… The next character is <hyphen> which is not a Word Character so Capture Group's value remains unchanged.	Capture Group 2 = Unchanged = A
5. -	.. match against <hyphen> (Not Captured)	
6. \d	Match against first numeric digit This is the '3' following the 'Jane A-'	Capture Group 3 = 3
7. +	Make it Greedy	Capture Group 3 = 35

RegEx Manual Volume II

Understanding Lookarounds cont.

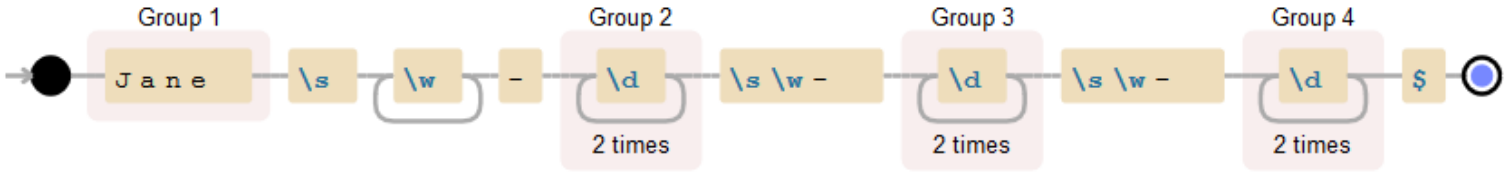
Using Lookarounds with Anchors cont.

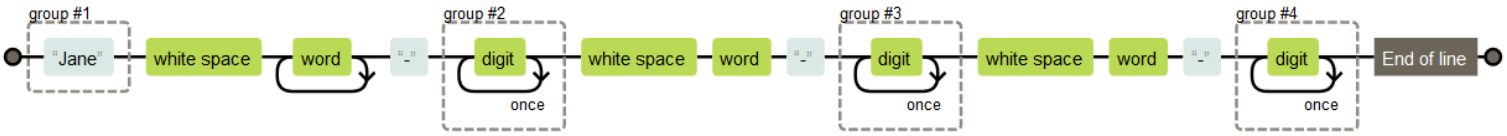
String = Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31

To match against all of Jane's scores:

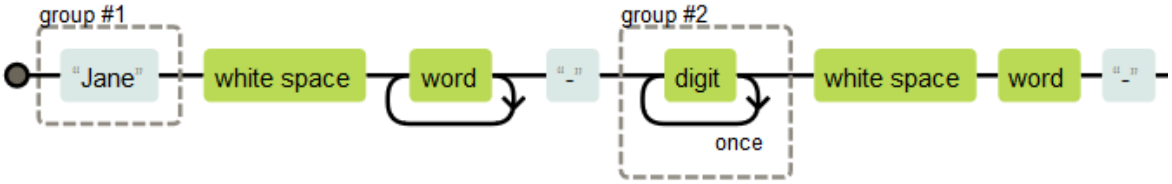
Match: ((?:Jane))\s\w+-(\d{2})\s\w-(\d{2})\s\w-(\d{2})$
Replace: \2 \3 \4

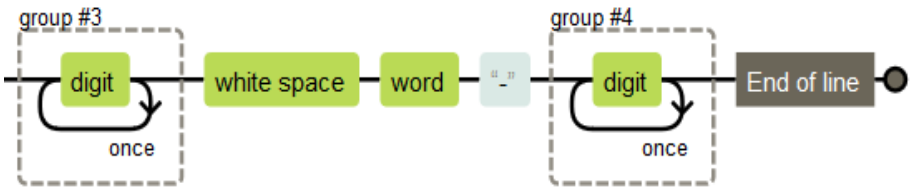
Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31	35 33 31





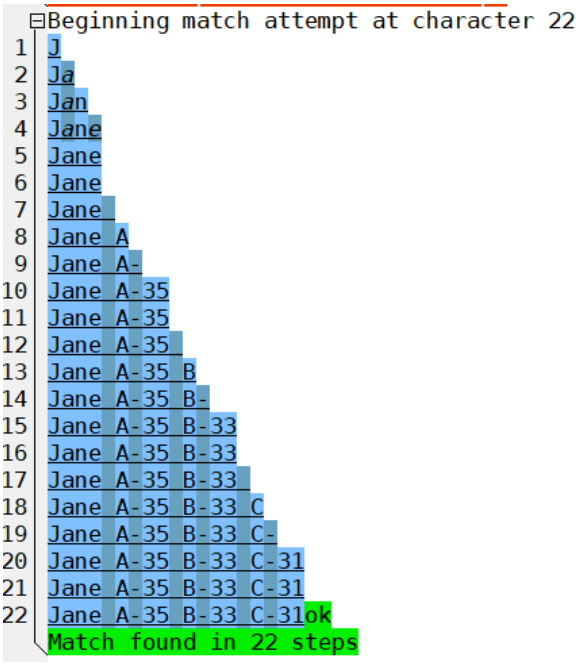
Expanded:





Sample:

```
................................
Match 1: Jane A-35 B-33 C-31
Group 1: Jane
Group 2: 35
Group 3: 33
Group 4: 31
```



RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

String = Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31

Match: ((?:Jane))\s\w+-(\d{2})\s\w-(\d{2})\s\w-(\d{2})$

Tim Analysis: Using Regex Buddy

1. ((?:Jane))	This is a non-marking group – typically not captured but I captured it anyway. Matches against 'Jane'. Current position = 'e' of 'Jane'	Capture Group 1 = Jane
2. \s	Match against <space> (Not Captured) This is the <space> following 'Jane'. Current position is <space> following 'Jane'.	Changes Capture Group 1 = Jane
3. \w 4. +	Match against next Word character (Not Captured) = A Make it Greedy (Not Captured) until… The next character is <hyphen> which is not a Word Character so the 'A' value uncaptured remains unchanged.	
5. -	.. match against <hyphen> (Not Captured).	
6. \d	Match against first numeric digit This is the '3' following the 'Jane A-'	Capture Group 2 = 3
7. {2}	Repeat it twice – Matches against two numeric digits.	Capture Group 2 = 35
8. \s\w-(\d{2})	Repeats the same expression to capture the next Score.	Capture Group 3 = 33
9. \s\w-(\d{2})	And once more to obtain the final score	Capture Group 4 = 31
10. $	Last expression must have matched at EOL (must be at End of Line)	

Source Material this section: Rexegg.com, princeton.edu, eloquentjavascript.net

RegEx Manual Volume II

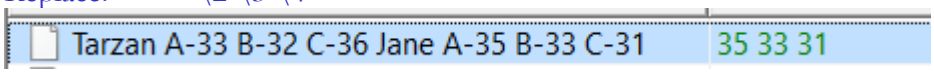
Understanding Lookarounds cont.

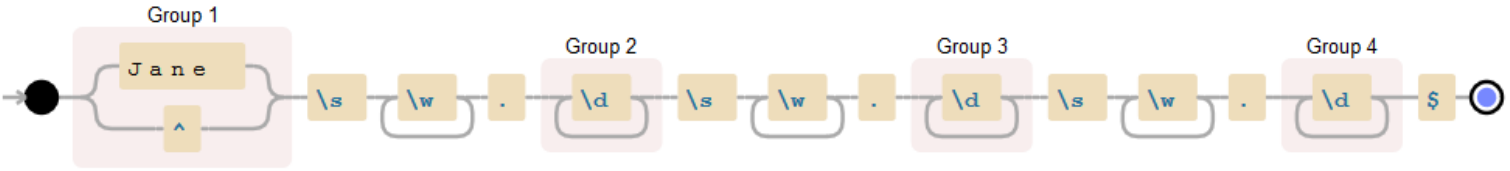
Using Lookarounds with Anchors cont.

String = Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31

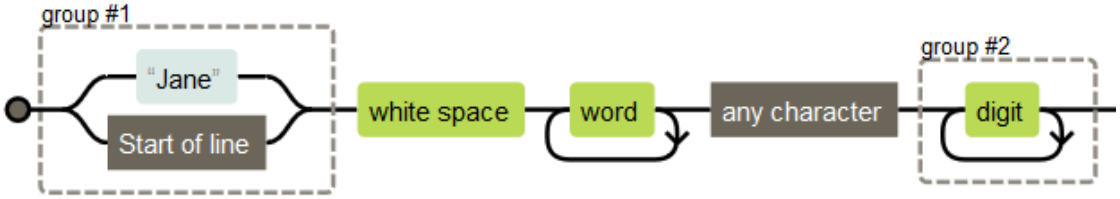
This will also match against all of Jane's scores:

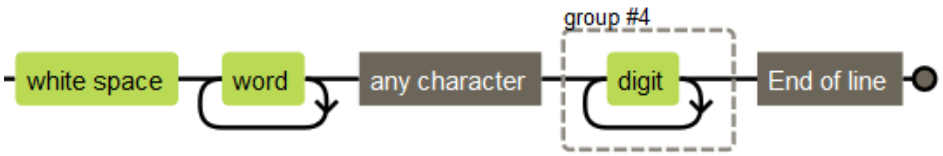
Match: ((?:Jane|^))\s\w+.(\d+)\s\w+.(\d+)\s\w+.(\d+)$
Replace: \2 \3 \4





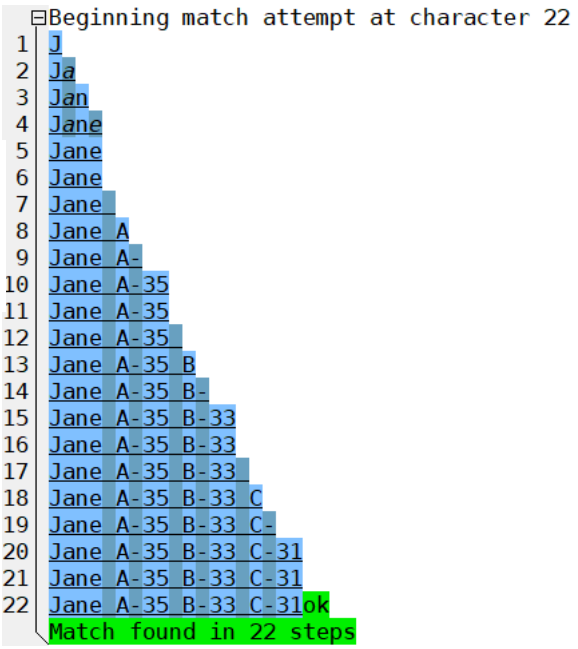
Expanded:





Sample:

```
................................
Match 1: Jane A-35 B-33 C-31
Group 1: Jane
Group 2: 35
Group 3: 33
Group 4: 31
```



RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

String = Tarzan A-33 B-32 C-36 Jane A-35 B-33 C-31

Match: ((?:Jane|^))\s\w+.(\d+)\s\w+.(\d+)\s\w+.(\d+)$

Tim Analysis: Using Regex Buddy

1. ((?:Jane|^))	Provides a choice between 'Jane and start of string. Matches ^ because the beginning of string comes before 'Jane'. Current position = Beginning of string (BOL)	Capture Group 1 = Matched but no Name Change
2. \s	Match against <space> (Not Captured) This is the <space> following 'Jane'. Because the next character ' T' of 'Tarzan', is not a <space>, it falls back to reevaluate the (?:Jane|^) expression and moves forward to 'Jane'. This reevaluates Capture Group 1 to 'Jane'. Current position is <space> following 'Jane'.	Changes Capture Group 1 = Jane
3. \w	Match against next Word character (Not Captured) = A	
4. +	Make it Greedy (Not Captured) until…	
5. -	.. match against <hyphen> (Not Captured).	
6. \d	Match against first numeric digit This is the '3' following the 'Jane A-'	Capture Group 3 = 3
7. +	Make it Greedy	Capture Group 3 = 35
8. \s\w-(\d{2})	Repeats the same expression to capture the next Score.	Capture Group 3 = 33
9. \s\w-(\d{2})	And once more to obtain the final score	Capture Group 4 = 31
10. $	Last expression must have matched at EOL	

RegEx Manual Volume II

Understanding Lookarounds cont.

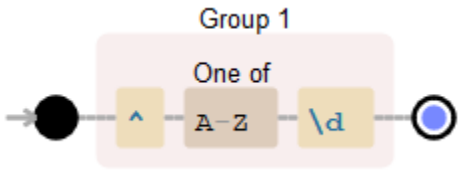
Using Lookarounds with Anchors cont.

Example: Extract the first alphanumeric characters from a string

String = A1B1C1vsD1D2D3

Match: (^[A-Z]\d)
Replace: \1

A1B1C1vsD1D2D3	A1



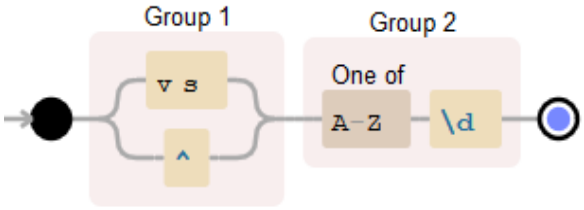
Tim Analysis: Using Regex Buddy

1. ^ Start at beginning of string (Beginning of Line or BOL)
2. [A-Z] Match against an uppercase letter followed by… Capture Group 1 = A
3. \d … numeric digit Capture Group 1 = A1

Let's build on this:

Match: ((?:vs|^))([A-Z]\d)
Replace: \2

A1B1C1vsD1D2D3	A1



Tim Analysis: Using Regex Buddy

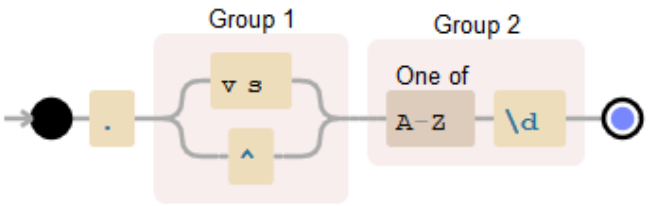
Where \1 = Matched Beginning of string

1. vs|^ Match between either the string 'vs' or the beginning of string whichever comes first. Currently at beginning of string so ^ (BOL) = TRUE.
2. [A-Z] Match against an uppercase letter followed by… Capture Group 1 = A
3. \d … numeric digit Capture Group 1 = A1

and …

Match: .((?:vs|^))([A-Z]\d)
Replace: \2

A1B1C1vsD1D2D3	D1



Where: \1 = vs \2 = D1

1. . Match against any character (Not Captured) = A
2. vs|^ Match between either the string 'vs' or the beginning of string whichever comes first. Position = 'A' so 'vs' = TRUE. Current position = 's' of 'vs'
2. [A-Z] Match against an uppercase letter followed by… Capture Group 1 = D
3. \d … numeric digit Capture Group 1 = D1

RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

Adding a Negative Lookahead

String = A1B1C1vsD1D2D3

Match something not followed by something else. When it is found at the beginning of string with an anchor character:

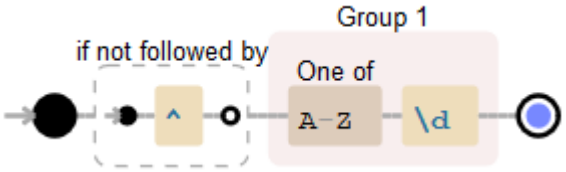
(?!^) - Tests that the current character is NOT at the beginning of the string (Beginning of Line or BOL)

(?!$) - Tests that the current character is NOT at the end of the string (End of Line or EOL)

Example: Locate an uppercase letter that is not at the Beginning of the string
s
Match: (?!^)([A-Z]\d)
Replace: \1

A1B1C1vsD1D2D3	B1

Where: \1 = Matched Beginning of string



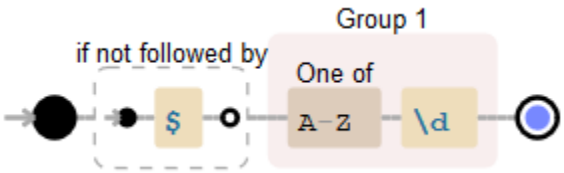
Tim Analysis: Using Regex Buddy

1. (?!^) Test that the current position is not the beginning of string (BOL) = 'A' = Fail.
Move to the next character. Current position = '1' of 'A1'
Tests that the current position is not the beginning of string (BOL) = '1' = True.
2. [A-Z] Match against a class consisting of an uppercase letter. Capture Group 1 = B
Matches against the next uppercase letter in the string, 'B'.
3. \d Match against numeric digit Capture Group 1 = B1

Example: Locate an uppercase letter that is not a the end of the string

Match: (?!$)([A-Z]\d)
Replace: \1

A1B1C1vsD1D2D3	A1



Tim Analysis: Using Regex Buddy

1. (?!$) Test that the current position is not the end of string (EOL) = 'A' = True.
2. [A-Z] Match against a class consisting of an uppercase letter. Capture Group 1 = A
Matches against the current position in the string, 'A'.
3. \d Match against numeric digit Capture Group 1 = A1

RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

Adding a Negative Lookahead cont.

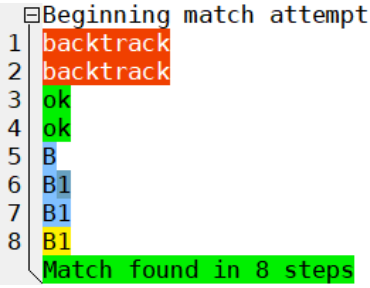
String = A1B1C1vsD1D2D3

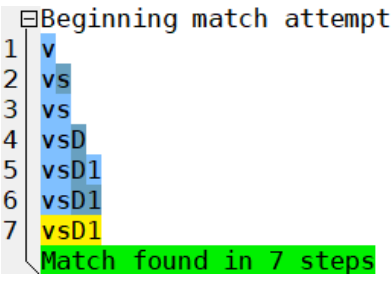
Example: Extract the second pair of alphanumeric characters from a string

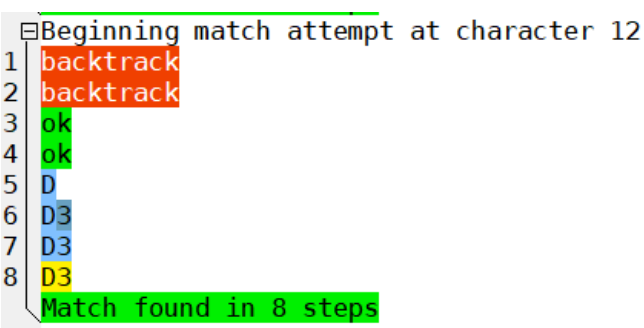
Match: ((?:vs|(?!^))([A-Z]\d)) (?:) = Non-Marking Group that I decided to Capture ((?:))
Replace: \2

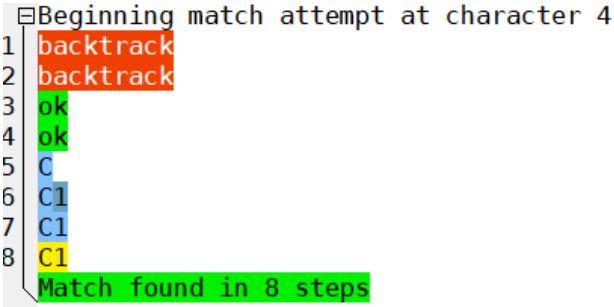
Match: ((?:vs|(?!^))([A-Z]\d))

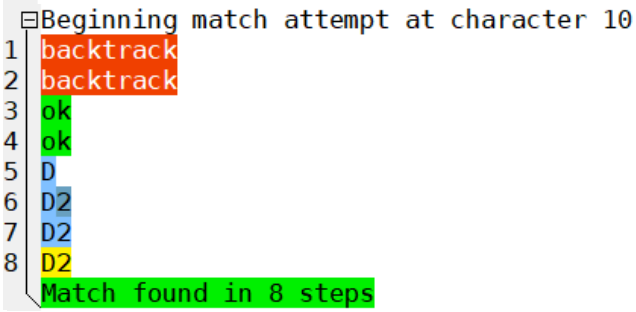
Result: A1B1C1vsD1D2D3











Tim Notes:

1. This constant matching demonstrates how after each match, the RegEx Engine falls back to the Alternates, testing the current position against the alternates and proceeding from there.

   You can see with the exception of Match at Character 6, where 'vs' = True, the (?!^) was matched for all others.

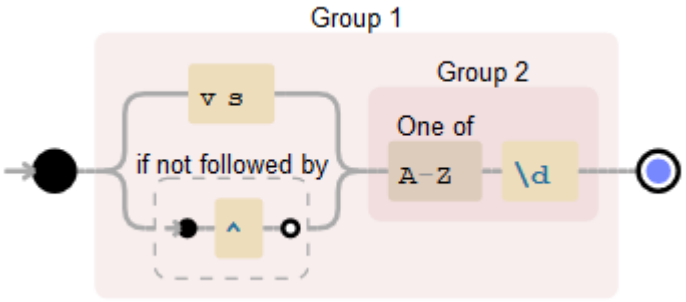
RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

Adding a Negative Lookahead cont.

String = A1B1C1vsD1D2D3



Tim Analysis: Using Regex Buddy

((?:vs|(?!^))([A-Z]\d))

Breaks down as: vs | (?!^) [A-Z] \d

This uses an Alternate for Capture Group 1:

This	OR	This
vs	|	(?!^)
Test that current position is at 'vs'		Test current character is not at beginning of string

The first alternate is always tested first. When the current position = 'vs' was True for (vs) at the 'Match at Character 6' of the previous page, it was also True for (?!^), but because the first alternate was tested first, it was matched against this first alternate (vs) and not the second alternate (?!^).

Alternate Test:

1. vs	Test the current position for the 'v' = 'A' = False	
2. (?!^)	Test that the current position is not the beginning of string = 'A' = Fail.	
	Move to the next character. Current position = '1' of 'A1'	
3. vs	Test the current position for the 'v' = '1' = False	
4. (?!^)	Test that the current position is not the beginning of string = '1' = True.	
5. [A-Z]	Match against a class consisting of an uppercase letter. Matches against the next uppercase letter in the string, 'B'.	Capture Group 1 = B
6. \d	Match against numeric digit	Capture Group 1 = B1
7.	Repeats back at the alternate after each character match	
..	At Character 6, the 'vs' matches –	
	a. Tests the current position for the 'v' = True	
	b. Tests the next position for the 's' = True	
	c. Match against a class consisting of an uppercase letter = 'D'	
	d. Match against numeric digit = '1'	

Only the first value was kept for all the matching that was done. The rest were disgarded when the Zero Length Assertion gave up the match. Beginning of string refers to Beginning of Line or BOL.

RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

Adding a Negative Lookahead cont.

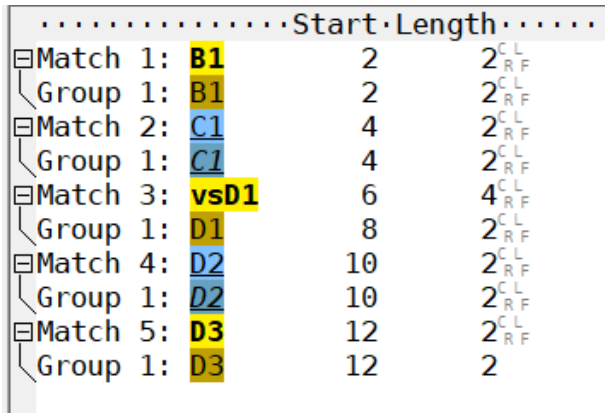
String = A1B1C1vsD1D2D3

A Negative Lookbehind would also have worked:

Example: Look for uppercase letter not preceded by beginnning of line (BOL)

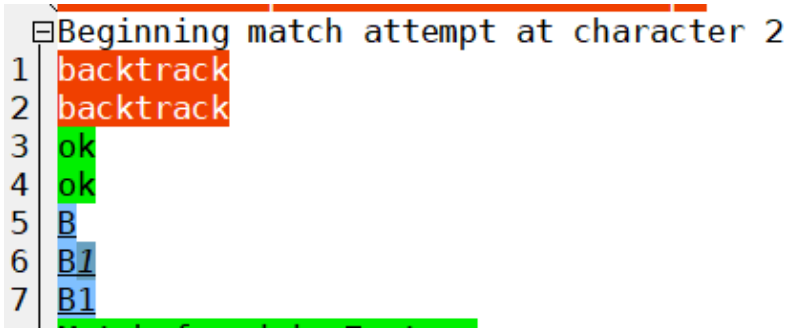
Match: (?:vs|(?<!^))([A-Z]\d)
Replace: \1

Sample:



Match: (?:vs|(?<!^))([A-Z]\d)

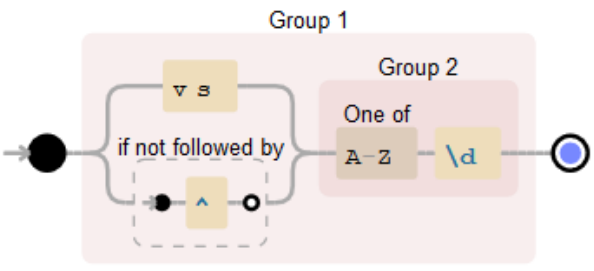
Result: A1B1D1vsD1D2D3



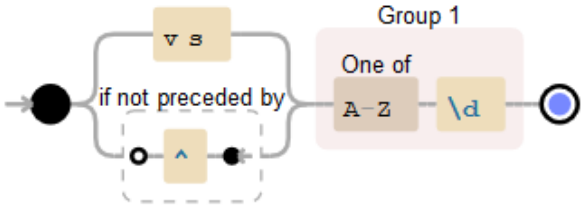
Tim Analysis: Using Regex Buddy

Doesn't change previous analysis. The use of the Negative Lookbehind in this instance performs the same function as the Negative Lookahead – test that the character is not at the beginning of the string (Beginning of Line or BOL).

Negative Lookahead



Negative Lookbehind:



RegEx Manual Volume II

Understanding Lookarounds cont.

Using Lookarounds with Anchors cont.

Finally, here is a real-world application:

Example: Determining a Valid Code on a Filename

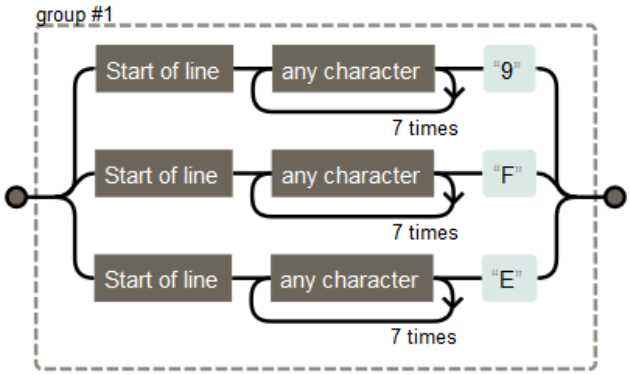
Observations:

1. Code signature attached to a filename.
2. You want to verify that a character in the 9th position in the string is either a '9', 'F' oe 'E'

String = 123d56789abcde12345

Match: (^.{8}9|^.{8}F|^.{8}E)
Replace: \1

123d56789abcde12345 123d56789

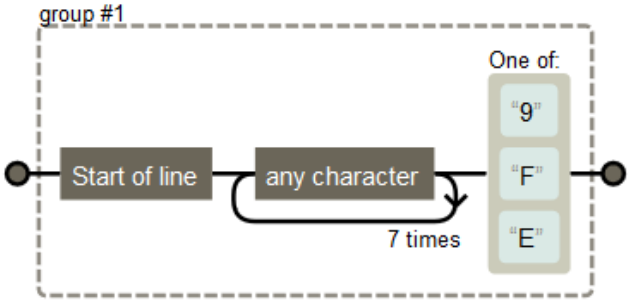


This uses an Alternate for Capture Group 1:

This	OR	This	OR	This
9	|	F	|	E

Tim Analysis: Using Regex Buddy

1. For each Branch test (Alternate), select the first 8 characters.
   Current position = 8
2. Test the next character for either a literal value of '9', 'F' or an 'E'



Tim Notes:

1. Could also have used a Character Class

Match: (^.{8}[9FE])
Replace: \1

123d56789abcde12345 123d56789

Source Material this section – Stackoverflow.com

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates)

	If
Syntax:	(?(Lookaround)then|else)

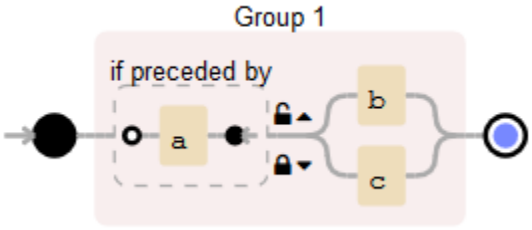
If the '*if*' evaluates to True, the RegEx engine will attempt to match the '*then*'. Otherwise, the '*else*' is attempted. The '*then*' must match for the overall RegEx to match. If the lookaround Fails, the '*else*' must match for the overall RegEx to match. The lookaround is zero-length (doesn't change the current position) but the '*then*' and '*else*' do consume their matches.

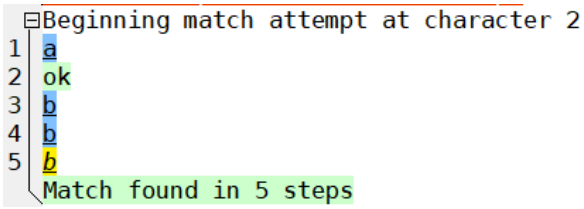
Samples:

abcc babxcac candy apple bcndy apple

Match: ((?(?<=a)b|c))
Replace: \1

Name	New Name
abcc	b
babxcac	b
candy apple	c
bcndy apple	c





Let's examine this closer

Match: ((?(?<=a)b|c))
Replace: \1

At first glance it looks like there are three expressions but there are not. The Lookbehind is actually part of the condition:

Condition	Then		Else
(?<=a)	b	|	c
Look for an 'a'	followed by 'b'		Match against a 'c'

String = candy apple

candy apple	c

Else matched = Matches the 'c' in 'candy'

String = bcady apple

bcndy apple	c

Else matched = Matches the 'c' of 'bcndy'

String = abndy apple

abndy apple	b

Then matched = Matches the 'b' of 'abndy'

String = babxcac

babxcac	b

Then matched = Matches the second 'b' of babxcac

Tim Notes:

1. Typically, as I understand it, a Lookbehind would match against the 'b' first, then look left to match for the 'a', but in a condiional it seems that it tests for the 'a' first then match against the 'b' as the '*then*', '*else*' look for a 'c'.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

You can use a Lookaround for the 'if'.

Example, using a Positive Lookahead -

Syntax: (?(?=regex)then|else)

Breaks down to:

(? (Positive Lookaround) then | else)

Because the lookaround has its own parentheses, the '*if*' and '*then*' elements are clearly separated.

For the '*then*' and '*else*', you can use any regular expression. If you want to use multiple alternation, you will have to group them:

Syntax: (? (Lookaround) (then1|then2|then3) | (else1|else2|else3))

Tim Notes:

1. There are no <spaces> in the above syntax, I added these for clarification.

2. Alternates are also referred to as 'branches'

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Let's expand the sample base:

Samples:

abcc babxcac candy apple bcndy apple bcandy apple
acndy apple abndy apple

Using same regex –

Match: ((?(?<=a)b|c))
Replace: \1

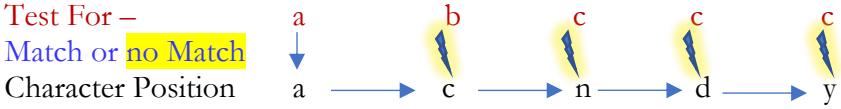
Name	New Name
abcc	b
babxcac	b
candy apple	c
bcandy apple	c
bcndy apple	c
acndy apple	acndy apple
abndy apple	b

Why didn't 'acndy apple' not match against the 'c' even though 'candy apple' did?

Remember at the beginning where it stated, 'The lookaround is zero-length (doesn't change the current position) but the '*then*' and '*else*' do consume their matches.'? There has to be one more character that is not 'a' following the 'c' to match for 'c'. Even another 'b' or 'c' would work.

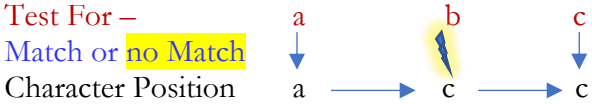
Name	New Name
ancndy	c
arcndy	c
accndy	c

This is because when it matches the 'a', the RegEx Engine moves forward to test for the 'b'. If False, the RegEx Engine moves to the next position to test for the 'c', but if when testing for the 'b', and the current position is held by the 'c', as it is here:



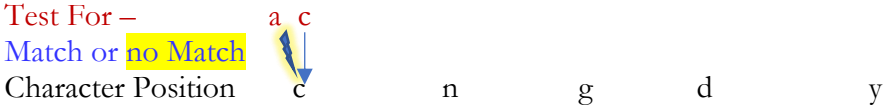
matches 'a' but not 'b' or 'c'

.. the 'c' will consequently never get tested for unless there is more than one 'c' in the string. i.e.,



matches 'a', not 'b' but matches 'c'

This doesn't matter if the string does not contain an 'a'. In which case, the 'c' will be matched. Tests at current position because Lookahead does not change the character position, so it tests for the 'c' at the same position as the Lookahead failed. This is the opposite behavior of when the Lookahead matches because at that point testing for the 'Then' and 'Else' does change the position. This is the reason the 'c' couldn't be matched using 'acndy'.



does not match 'a', so 'b' never tested, does match 'c'

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

abcc babxcac candy apple bcndy apple bcandy apple
acndy apple abndy apple

More examples:

Example: Positive Lookbehind locates a followed by b. If True, captures b. If not True, match against p

Match: ((?(?<=a)b|p))

Name	New Name
abcc	b
candy apple	p
abndy apple	b
acndy apple	p
babxcac	b
bcandy apple	p
bcndy apple	p

The Lookbehind matches the 'a' of 'abcc. Once matched, it matches the 'b' that follows it for the Then and captures that value. Remember that a Lookbehind tests for the next character for the Then following the match.

Example: Positive Lookbehind locates a followed by c. If True, captures c. If not True, match against p

Match: ((?(?<=a)c|p))
Replace: \1

acndy apple	c

Match: ((?(?<=a)c|p))

Result: acndy apple

The Lookbehind matches the 'a' of 'acndy'. Once matched, it matches against the 'c' that follows it for the Then and captures that value. Remember that a Lookbehind tests for the next character for the Then following the match.

Tim Notes:

1. Typically, as I understand it, a Lookbehind would match against the 'c' first, then look left to match for the 'a', but in a condiional it seems that it tests for the 'a' first then match against the 'c' as the '*then*', '*else*' look for a 'p'.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Additional samples:

bobby tally toy cat

The Lookaround can be other types as well.

Example: Positive Lookahead locates b followed by b. If True, captures b. If not True, match against y

Match: ((?(?=b)b|y))
Replace: \1

Name	New Name
bobby	b

Match: ((?(?=b)b|y))

Result: bobby

Matches against first 'b' of 'bobby'. The Lookahead once matched, gives up it's value and returns to the previous position which is the first 'b' of 'bobby'. It then tests that character position for the 'b' of the Then and matches and captures that value. Remember that a Lookahead tests the same character position as the match for the Then. In the result above it is the first 'b' of the string that is matched for the Then.

Example: Positive Lookahead locates bo followed by b. If True, captures b. If not True, match against y

Match: ((?(?=bo)b|y))
Replace: \1

Name	New Name
bobby	b
tally	y
toy	y
cat	cat

Match: ((?(?=bo)b|y))

Result: bobby

Matches against first 'b' of 'bobby' as well as the 'y' of 'tally' and 'toy' The Lookahead matches the 'bo' of 'bobby'. Once matched, gives up it's value and returns to the previous position which is the first 'b' of 'bobby'. It then tests that character position for the 'b' of the Then and matches and captures that value. Remember that a Lookahead tests the same character position as the match for the Then. In the result above it is the first 'b' of the string that is matched for the Then.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

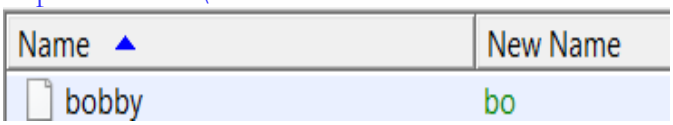
Additional samples:

bobby tally toy cat

The Lookaround can be other types as well. cont.

Example: Positive Lookahead locates b followed by bo. If True, captures bo. If not True, match against y

Match: ((?(?=b)bo|y))
Replace: \1



Matches against first 'b' of 'bobby. Once matched, the Lookahead gives up it's value and returns to the previous position which is the first 'b' of 'bobby'. It then tests that character position for the 'bo' of the Then and matches and captures that value.

Match: ((?(?=b)bo|y))

Result: **bo**bb**y**

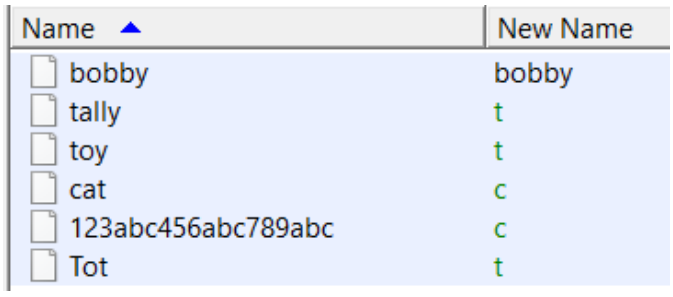
A Lookahead tests the same character position as the match for the Then. In the result above it is the first 'b' of the string followed by 'o' that is matched for the Then.

Two more sample additions

123abc456abc789abc Tot

Example: Positive Lookahead locates c followed by c. If True, captures c. If not True, match against t

Match: ((?(?=c)c|t))
Replace: \1



Matches against first 'c' of 'cat'. Once matched, the Lookahead gives up it's value and returns to the previous position which is the 'c' of 'cat'. It then tests that character position for the 'c' of the Then and matches and captures that value. Remember that a Lookahead tests the same character position as the match for the Then. In the result below it is the 'c' that is matched for the Then.

Match: ((?(?=c)c|t))

Result: **c**a**t**

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

bobby tally toy cat
123abc456abc789abc Tot

The Lookaround can be other types as well. cont.

Example: Positive Lookahead locates ca followed by c. If True, captures c. If not True, match against t

Match: ((?(?=ca)c|t))
Replace: \1

Name	New Name
bobby	bobby
tally	t
toy	t
cat	c
123abc456abc789abc	123abc456abc789abc
Tot	t

Matches against 'ca' of 'cat. Once matched, the Lookahead gives up it's value and returns to the previous position which is the first 'c' of 'cat'. It then tests that character position for the 'c' of the Then and matches and captures that value. For all other words, 't' is captured for the Else, except for 'bobby' and '123abc456abc789abc' because neither of those strings contain a 't'.

Match: ((?(?=ca)c|t))

Result: cat

A Lookahead tests the same character position as the match for the Then. In the result above it is the 'c' that is matched for the Then.

Example: Match against a b. Positive Lookahead locates o followed by a second b. If True, captures b. If not True, match against y

Match: (b(?(?=o)b|y))

Replace: \1

Name	New Name
bobby	by

Matches against and captures the first 'b' of 'bobby. The Lookahead matches the first 'o' of 'bobby'. Once matched, the Lookahead gives up it's value and returns to the previous position which is the first 'o' of 'bobby'. It then tests that same character position for the 'b' of the Then and Fails. The 'y' is matched for the Else. This in turn changes Capture Group 1 to the second 'b' of 'bobby'. Remember that a Lookahead tests the same character position as the match for the Then.

Match: (b(?(?=o)b|y))

Result: bobby

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

bobby tally toy cat
123abc456abc789abc Tot

The Lookaround can be other types as well. cont.

Example: Match against a b. Positive Lookahead locates o followed by a second o. If True, captures o. If not True, match against y

Match: (b(?(?=o)o|y))
Replace: \1

Name	New Name
bobby	bo

Matches against and captures the first 'b' of 'bobby. The Lookahead matches the first 'o' of 'bobby'. Once matched, the Lookahead gives up it's value and returns to the previous position which is the first 'o' of 'bobby'. It then tests that same character position for the 'o', matches for the Then and captures that value. Remember that a Lookahead tests the same character position as the match for the Then. Thus, in the result below, it is the 'o' that is matched for the Then.

Match: (b(?(?=o)o|y))

Result: **bo**bby

Example: Match against an a. Positive Lookahead locates l followed by a second l. If True, captures l. If not True, match against y

Match: (a(?(?=l)l|y))
Replace: \1

Name	New Name
tally	al

Matches against and captures the 'a' of 'tally. The Lookahead matches the first 'l' of 'tally'. Once matched, the Lookahead gives up it's value and returns to the previous position which is the first 'l' of 'tally'. It then tests that same character position for the 'l', matches for the Then and captures that value. Remember that a Lookahead tests the same character position as the match for the Then. Thus, in the result below, it is the first 'l' that is matched for the Then, not the second.

Match: (a(?(?=l)l|y))

Result: t**al**ly

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

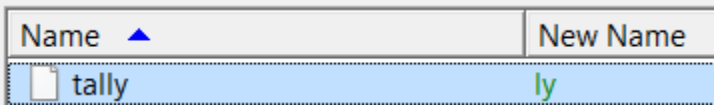
bobby tally toy cat
123abc456abc789abc Tot

The Lookaround can be other types as well. cont.

Here's the same example presented using two different Lookarounds.

Example: Match against an l. Positive Lookahead locates 'a' followed by a second l. If True, captures l. If not True, match against y

Match: (l(?(?=a)l|y))
Replace: \1



Matches against and captures the first 'l' of 'tally. The Lookahead Fails to match the 'a', because the current position is past the 'a'. The Then is never tested as a result. The Else matches against the 'y' of 'tally' and captures it. This is turn changes Capture Group 1 to the second 'l' of 'tally'.

Match: (l(?(?=a)l|y))

Replace: tally

Example: Match against an l. Positive Lookbehind locates 'a' followed by a second l. If True, captures l. If not True, match against y

Match: (l(?(?<=a)l|y))
Replace: \1

Matches against and captures the first 'l' of 'tally. The Lookbehind Fails to match the 'a', because the current position is past the 'a'. The Then is never tested as a result. The Else matches against the 'y' of 'tally' and captures it. This in turn changes Capture Group 1 to the second 'l' of 'tally'.

Match: (l(?(?<=a)l|y))

Replace: tally

Conclusion: Same result because in both cases Capture Group 1, outside of the Lookaround, matches past the point that 'a' of the Lookaround could never match.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

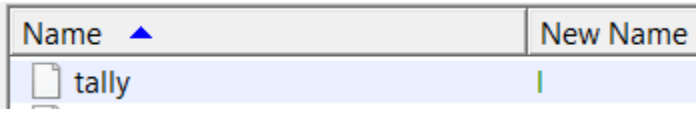
bobby tally toy cat
123abc456abc789abc Tot

The Lookaround can be other types as well. cont.

Here's another same example presented using two different Lookarounds.

Example: Positive Lookbehind locates 'a' followed by an l. If True, captures l. If not True, match against y

Match: ((?(?<=a)l|y))
Replace: \1



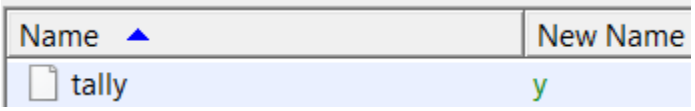
Match: ((?(?<=a)l|y))

Result: tally

The Lookbehind matches the 'a' of 'tally'. Gives up the match and returns to the position before the Lookaround which is the 'a' of 'tally'. Moves forward and tests for the 'l', matches and captures it.

Example: Positive Lookahead locates 'a' followed by an l. If True, captures l. If not True, match against y

Match: ((?(?=a)l|y))
Replace: \1



Match: ((?(?=a)l|y))

Result: tally

The Lookahead matches against the 'a' of 'tally'. The Lookahead gives up the match and returns back to the position before the match, the 'a' of 'tally'. The Then is tested at the same character position for the 'l' and Fails. The Else matches against the 'y' of 'tally' and captures it.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

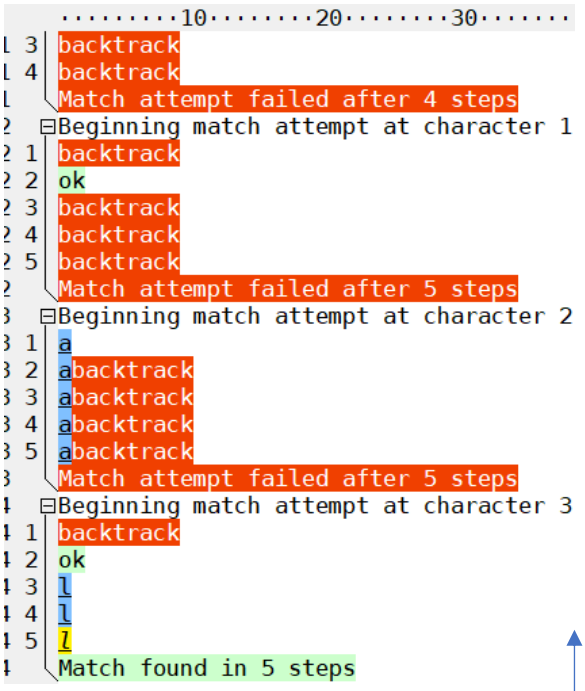
bobby	tally	toy	cat
123abc456abc789abc		Tot	

The Lookaround can be other types as well. cont.

Here's yet another example presented using two different Negative Lookarounds.

Example: Negative Lookahead locates not an 'a' followed by an l. If True, captures l. If not True, match against y

Match: ((?(?!a)l|y))
Replace: \1



Moves to each character position, testing for the not an 'a'. If matched, tests current position for the following 'l'.

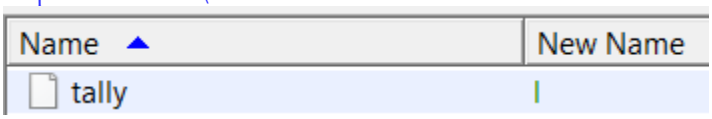
Match: ((?(?!a)l|y))

Result: tally

Matches the not an 'a' of 'tally' at the first 'l'. Gives up the match. Tests current character position for the 'l' of 'tally'. This matches against the first 'l' of 'tally'.

Example: Negative Lookbehind locates not an 'a' followed by an l. If True, captures l. If not True, match against y

Match: ((?(?<!a)l|y))
Replace: \1



Moves to each character positions, then backtracks to test each character for not an 'a'. If matched, continues to move forward to test for the 'l'.

Match: ((?(?<!a)l|y))

Result: tally

Matches the not an 'a' of 'tally' at the first 'l'. Gives up the match and moves forward to match against the second 'l' of 'tally'.

Tim Notes: Observations

1. Although the Examples are almost the same in their description, the difference lies in how the values are obtained by each. The Negative Lookahead tests for each current character position, while the Negative Lookbehind moves forward then backtracks to match. This is only my opinion of what I see happening.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

bobby tally toy cat
123abc456abc789abc Tot

The Lookaround can be other types as well. cont.

Example: Positive Lookbehind locates o followed by a b. If True, captures b. If not True, match against y

Match: ((?(?<=o)b|y))
Replace: \1

Name	New Name
tally	y
bobby	b

Match: ((?(?<=o)b|y))

Result: tally

Result: bobby

In the case of 'tally', the Positive Lookbehind does not match for the 'o' in the string , therefore the Then is never tested. The Else matches against the 'y'.

In the case of 'bobby', the Positive Lookbehind matches against the 'o', gives up the match and proceeds to test the next character for the 'b' and matches and captures the second 'b' of 'bobby'

Tim Notes:

1. It should be noted that as with all of my analysis, these are rather simple explanations that leave out a lot of the intermediary steps. For example, when the RegEx Engine is testing for values or evaluating sub-expressions, it moves back and forth between the in-between positions before and after each character position as described earlier in the book.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

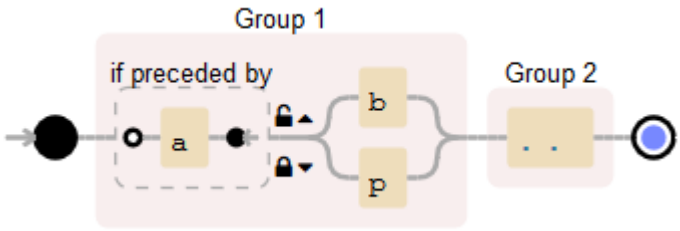
abcc acbphh

So after the Lookaround, what is the current position? Does it return to the 'a' in the next example, or to the position of the capture outside of the 'a' ?

Example: Positive Lookbehind locates a followed by a b. If True, captures b. If not True, match against p

Match: ((?(?<=a)b|p))(..)
Replace: \1 \2

Name	New Name
abcc	b cc



Match: ((?(?<=a)b|p))(..)

Result: abcc

```
Match 1: bcc
Group 1: b
Group 2: cc
```

```
Beginning match attempt at character 1
1 a
2 ok
3 b
4 b
5 b
6 bc
7 bcc
8 bcc
Match found in 8 steps
```

Same RegEx using this sample:

Name	New Name
acbphh	p hh

Result: acbphh

```
Match 1: phh
Group 1: p
Group 2: hh
```

```
Beginning match attempt at character 3
1 backtrack
2 backtrack
3 p
4 p
5 p
6 ph
7 phh
8 phh
Match found in 8 steps
```

In the 'abcc' example, the Positive Lookbehind matches the 'a', gives up the match and moves forward to match against the following 'b'. At which point, the next two following characters are captured – the 'cc'. In the 'acbphh' example, the Positive Lookbehind matches against the 'a', but Fails to match against the 'b' as the following character (= 'c'), so the Else matches against the 'p'. At which point, the next two following characters are captured – the 'hh'.

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Samples:

abcc acbphh

So after the Lookaround, what is the current position? Does it return to the 'a' in the next example, or to the position of the capture outside of the 'a' ? cont.

Conclusion:

It returns to the position of the capture made outside of the 'a' matched in the Lookaround.

Although the Lookaround does not affect character position, it gives up it's match, i.e. the 'a' in the above example, it returns to the point before the capture of the 'a', which is the 'b' matched in 'abcc' or the 'p' matched in 'acbphh'. At this point, the (..) captures the following two characters, the 'cc' of 'abcc' and the 'hh' of 'acbphh'.

I hope that I have made this clear enough. Some people get confused when they think that giving up the match means giving up both the 'a' and the subsequent capture of 'b' or 'p'. Furthermore, this confusion continues when they think that, not changing the character position after a lookaround, would always return back to the 'a'.

RegEx Manual Volume II

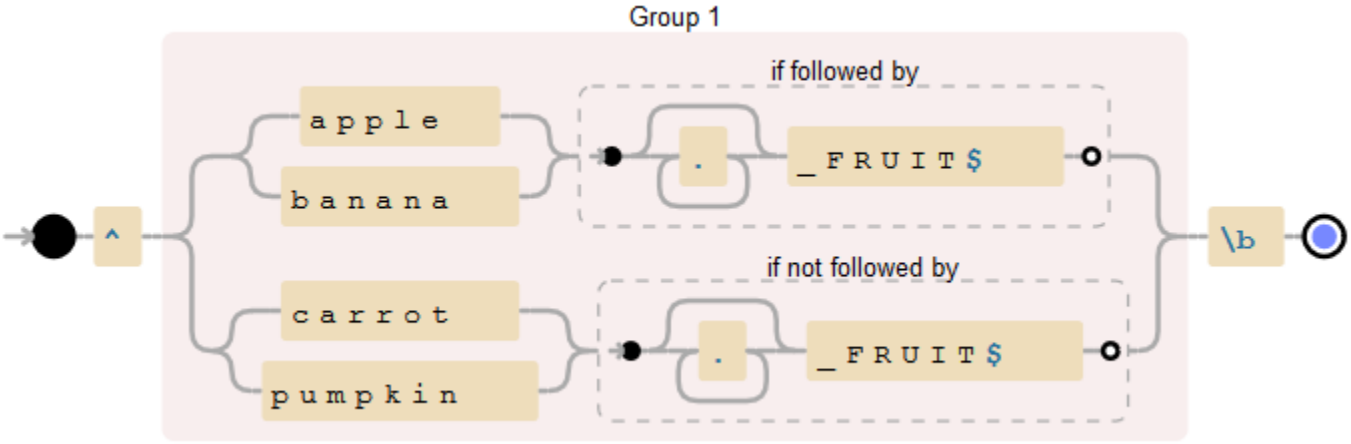
Understanding Lookarounds cont.

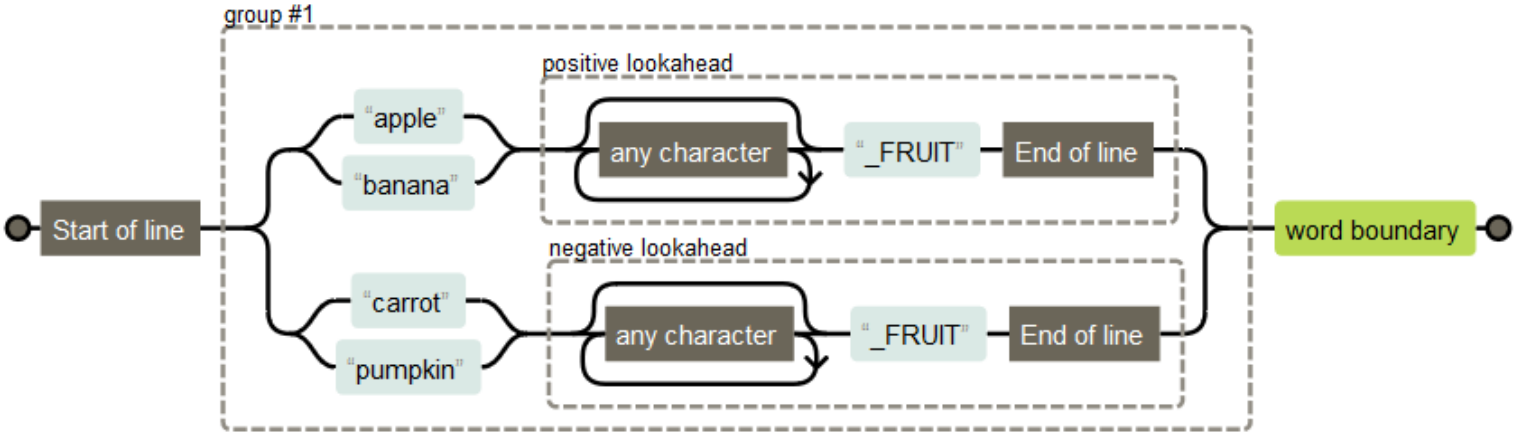
Lookaround conditional (with Alternates) cont.

Example:

Match: ^((?:(?:apple|banana)(?=.*_FRUIT$)|(?:carrot|pumpkin)(?!.*_FRUIT$)))\b
Replace: \1

apple is a _FRUIT	apple
apple jacks is not a _FRUIT	apple
banana is a _FRUIT	banana
carrot is a _VEGETABLE	carrot
pumpkin is a _VEGETABLE	pumpkin
Wooden apple is not a _FRUIT	Wooden apple is not a _FRUIT





RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Example:

Match: ^((?:(?:apple|banana)(?=.*_FRUIT$)|(?:carrot|pumpkin)(?!.*_FRUIT$)))\b

Breaks down:

C A P T U R E G R O U P O N E

Non-Marking Group

Non-Marking Group Non-Marking Group

Alternate

Alternate Positive Lookahead Alternate Negative Lookahead

Anchor Anchor Anchor WB

^ ((?: (?: apple | banana) (?= .* _ FRUIT $) | (?: carrot | pumpkin) (?! .* _ FRUIT $))) \b

Tim Analysis:

Using apple is a _FRUIT

1. ^	Match at beginning of string (Beginning of Line or BOL)	
2. apple | banana	Match for either 'apple' or 'banana'	Capture Group 1 = apple
3. (?=.*_FRUIT$)	Positive Lookahead. .* moves forward through string. _FRUIT$ looks for a match for '_FRUIT' at end of string. = True	
4. \b	Matches against a Word Boundary = EOL	

Using carrot is a _VEGETABLE

1. ^	Match at beginning of string (BOL)	
2. apple | banana	Match for either 'apple' or 'banana' = False	
3. or …		
4. carrot | pumpkin	Match for either carrot or pumpkin	Capture Group 1 = carrot
5. (?!.*_FRUIT$)	Negative Lookahead. .* moves forward through string. _FRUIT$ looks for a match for NOT '_FRUIT' at end of string. = '_VEGETABLE' = True	
4. \b	Matches against a Word Boundary = EOL	

Tim Notes:

Wherever end of string is used, it refers to End of Line or EOL

RegEx Manual Volume II

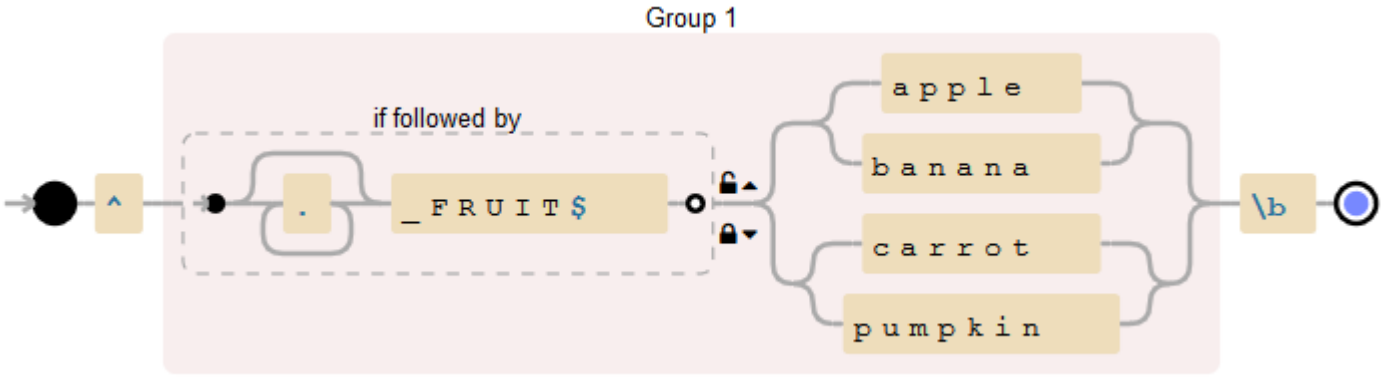
Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Using a Lookaround Conditional with Alternates, it can be simplified to:

Match: ^((?(?=.*_FRUIT$)(?:apple|banana)|(?:carrot|pumpkin)))\b
Replace: \1

apple is a _FRUIT	apple
apple jacks is not a _FRUIT	apple
banana is a _FRUIT	banana
carrot is a _VEGETABLE	carrot
pumpkin is a _VEGETABLE	pumpkin
Wooden apple is not a _FRUIT	Wooden apple is not a _FRUIT



Breaks down:

C A P T U R E G R O U P O N E

C O N D I T I O N A L

Non-Marking Group Non-Marking Group

Alternate

Positive Lookahead

Anchor Anchor Alternate WB

^ ((? (?= .* _ FRUIT$) (?: apple | banana) | (?: carrot | pumpkin))) \b

Conditional

IF THEN ELSE

(_FRUIT is at end of string) (apple or banana at beginning of string) (carrot or pumpkin at beginning of string)

RegEx Manual Volume II

Understanding Lookarounds cont.

Lookaround conditional (with Alternates) cont.

Match: ^((?(?=.*_FRUIT$)(?:apple|banana)|(?:carrot|pumpkin)))\b

Tim Analysis:

Using apple is a _FRUIT

1. ^	Match at beginning of string (Beginning of Line or BOL)	
2. (?	Begins Conditional..	
3. (?=.*_FRUIT$)	Positive Lookahead. .* moves forward through string. _FRUIT$ looks for a match for '_FRUIT' at end of string. = True	
4. apple | banana	'*Then*' Match for either 'apple' or 'banana'	Capture Group 1 = apple
5. \b	Matches against a Word Boundary = EOL	

Tim Notes:

1. There is a much more comprehensive treatise on these subjects that can be found at rexegg.com/regex-conditionals. Some of the examples and ideas researched in preparing this RegEx Manual manuscript have been sourced from there as well as from other sources of information in the public domain. If you do visit rexegg, check out their Regex Buddy program to help with your own analysis.

2. Examples and information sourced from this and others fall under the 'Fair Use' doctrine.

RegEx Manual Volume II

Understanding Referenced Conditional Groups

Using a Referenced Capturing Group for the 'if'

Syntax: (?(1)then|else)

You can check whether a capturing group has taken part in the match thus far by placing the number representing the capturing group inside parentheses, and use that as the '*if*'. It should be noted that this is a Non-Marking Group and does not create a Capturing Group, only references that value.

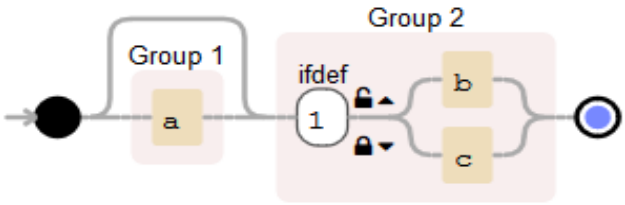
where:

1 is the number of a capturing group and '*then*' and '*else*' are any valid RegEx.

If the referenced capturing group took part in the match attempt thus far, the '*then*' must match for the overall RegEx to match. If the capturing group did not take part in the match thus far, the '*else*' must match for the overall RegEx to match. What I mean by this is, does the referenced capturing group hold a value? If it does, then it has taken part in the match up to this point in the evaluation.

Match: (a)?((?(1)b|c))
Replace: \1- \2

abcc	a- b
babxcac	a- b
candy apple	- c



Breaks down:

Capture Group 1	Capture Group 2								
(a)	?	(	(? (1)		b	|	c	))
a	?	(Condition with Reference 1		Then b		Else c	)		

Where:

1 references Capture Group 1 (a)

Tim Analysis:

Use abcc

1. a	Look for 'a' = True	Capture Group 1 = a
2. ?	Make it lazy and therefore optional	
3. (?(1)b|c)	Condition tests if Capture Group 1 took part in the match. What this means is – did (a) evaluate as True and therefore Capture Group 1 holds a value = True	
	'*Then*' Tests for a 'b' = True	Capture Group 2 = b

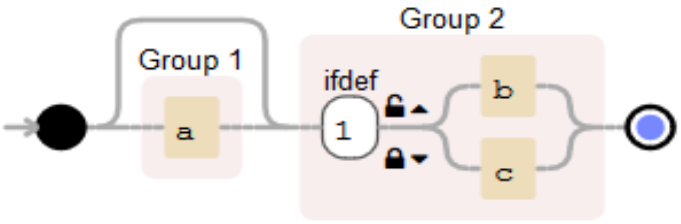
RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Match: (a)?((?(1)b|c))
Replace: \1- \2

abcc	a- b
babxcac	a- b
candy apple	- c



Tim Analysis:

Use candy apple

Match: (a)?((?(1)b|c))

Result: candy apple

Sample:

```
............Start·Length·····················
Match 1: c      0      1
Group 1 did not participate in the match
Group 2: c      0      1
```

```
Beginning match attempt at character 0
1 backtrack
2 ok
3 c
4 c
5 c
Match found in 5 steps
```

Where:

\1 = null \2 = c

1. a	Look for 'a' = 'c' = False
2. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..
	Capture Group 1 = null
3. (?(1)b|c)	.. Condition tests if Capture Group 1 took part in the match. What this means is – did (a) evaluate as True and therefore Capture Group 1 holds a value = False '*Else*' tests for 'c' = True
	Capture Group 2 = c

RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Tim Notes: Lazy & Zero Occurrence Matches

1. The RegEx would have failed and never would have tested the Conditional without the ? after (a). The ? makes the (a) test optional, so if it Fails, it moves on to the next expression – the Conditional in this case. Capture Group 1, however, has captured a null value despite the match failure. This is called a 'Zero Occurrence Match'. It Fails the conditional but retains a null value. So if you see a null value in any of these examples in this book it is probably due to the capture of a Zero Occurrence Match (but not always).

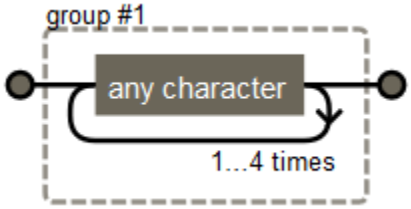
   A Zero Occurrence is matched typically if you have a Lazy Quantifier - ? which matches Zero or One time. If it matches Zero – the value is null and this is the 'Zero Occurrence Match'.

2. Clarify Lazy – the ? Quantifier is shorthand for {0,1} which could be stated as 'Greedy with a max of 1' and typically follows a Greedy Quantifier, i.e., *? or {0,5}. It also *mandates* the existence of a following character in the string.

   Examples:

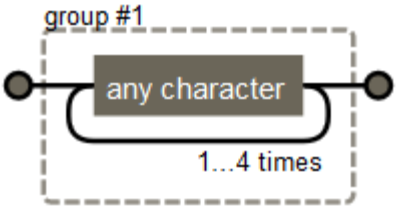
Match: (.{2,5}) = Final 2-thru-5 characters (Greedy, most possible = 5)
Replace: \1

abc	abc
abca	abca
abcbba	abcbb
abd	abd
ACABC	ACABC
babxcac	babxc
bc	bc
bd	bd



Match: (.{2,5}?) = First 2-characters (Lazy, least possible = 2)
Replace: \1

abc	ab
abca	ab
abcbba	ab
abd	ab
ACABC	AC
babxcac	ba
bc	bc
bd	bd



a. The above Lazy *mandates* the existence of a 3rd-character.. It cannot match a string of only 2 characters.
b. Since, however, the ? is defined by most all documentation as Lazy (or Non-Greedy), I will leave it in the book as is.

My Thanks to RegexNinja for clarification on notation 2

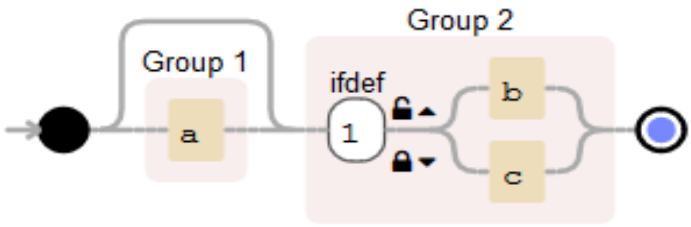
RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Match: (a)?((?(1)b|c))
Replace: \1- \2

abcc	a- b
babxcac	a- b
candy apple	- c



Tim Analysis:

Use babxcac

Where:
\1 = a \2 = b

1. a	Test for 'a' at first character = False	
2. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..	
3. (?(1)b|c)	.. Condition tests if Capture Group 1 took part in the match. What this means is – did (a) evaluate as True and therefore Capture Group 1 holds a value = False Else tests for a 'c' = 'b' = False	
4.	RegEx Engine moves to next character and begins the evaluation over.	
5. a	Tests for 'a' = True	Capture Group 1 = a
6.	'*Then*' Tests for 'b' = True	Capture Group 2 = b

Tim Notes:

1. If you want to know current position, just append (..) on to the RegEx where you want the location.

(a)?((?(1)b|c))(..)

babxcac	a - b xc

Indicates that the current position was the second 'x' in babxcac, meaning that the last character matched was the second 'b'.

RegEx Manual Volume II

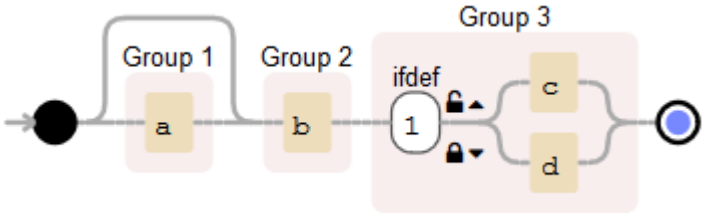
Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Difficult subject.. here are some more examples.

Match: (a)?(b)((?(1)c|d))
Replace: \1 - \2 - \3

bd	- b - d
bc	bc
abc	a - b - c



Tim Analysis:

Breaks down:

Capture Group 1 Capture Group 2 Capture Group 3

(a) ? (b) ((?(1)c|d))

Added an additional capture group to capture 'b' if 'b' = True

Use bd

Match: (a)?(b)((?(1)c|d))

Result: **bd**

Sample:

```
.............Start.Length.................
Match 1: bd      0      2
Group 1 did not participate in the match
Group 2: b       0      1
Group 3: d       1      1
```

```
Beginning match attempt at character 0
backtrack
ok
b
b
bd
bd
bd
Match found in 7 steps
```

Where:

\1 = null \2 = b \3 = d

RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Where:

\1 = null \2 = b \3 = d

Tim Analysis: cont.

1. a	Test for 'a' = 'b' = Fail	
2. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..	Capture Group 1 = null
3. b	..Test for 'b' = True	Capture Group 2 = b
4. (?(1)c|d)	Condition tests if Capture Group 1 took part in the match. What this means is – did (a) evaluate as True and therefore Capture Group 1 holds a value = False '*Else*' tests for 'd' = True	Capture Group 3 = d

Tim Notes:

1. If (a) failed, it would normally not have the opportunity to test for (b), but because it was made optional (using ' ? ') it moves on and evaluates for 'b'. This is True, so 'b' is captured but into Capture Group 2. Although the conditional failed, Capture Group 1 retains a null value because of a Zero Occurrence Match.

2. Refer to notes on Zero Occurrence Matches.

RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Match: (a)?(b)((?(1)c|d))
Replace: \1 - \2 - \3

bd	- b - d
bc	bc
abc	a - b - c

Where:

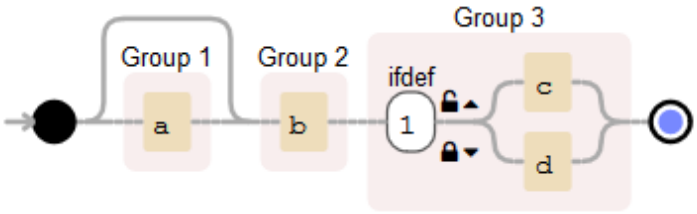
\1 = null \2 = b

Tim Analysis:

Use bc

Match: (a)?(b)((?(1)c|d))

Result: bc



1. a	Test for 'a' = 'b' = Fail	
2. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..	Capture Group 1 = null
3. b	..Test for 'b' = True	Capture Group 2 = b
4. (?(1)c|d)	Condition tests if Capture Group 1 took part in the match. What this means is – did (a) evaluate as True and therefore Capture Group 1 holds a value = False '*Else*' tests for 'd' = False Removes all values. Begins evaluation over again.	
5. a	Test next character for 'a' = 'c' = Fail	
2. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..	
3. b	..Test for 'b' = 'c' = Fail	

Tim Notes:

1. Refer to notes on Zero Occurrence Matches.

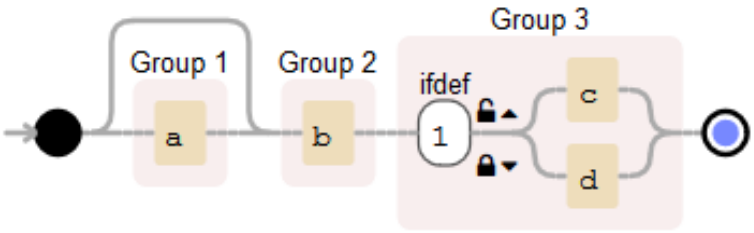
RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Match: (a)?(b)((?(1)c|d))
Replace: \1 - \2 - \3

bd	- b - d
bc	bc
abc	a - b - c



Where:

\1 = a \2 = b \3 = c

Tim Analysis:

Use abc

Match: (a)?(b)((?(1)c|d))

Result: abc

Beginning match attempt at character 0
a
a
ab
ab
abc
abc
abc
Match found in 7 steps

Sample

		Start	Length
Match 1:	abc	0	3
Group 1:	a	0	1
Group 2:	b	1	1
Group 3:	c	2	1

1. a	Test for 'a' = True	Capture Group 1 = a
2. ?	Make it lazy and therefore optional	
3. b	..Test for 'b' = True	Capture Group 2 = b
4. (?(1)c|d)	Condition tests if Capture Group 1 took part in the match. = True	
	'*Then*' tests for 'c' = True	Capture Group 3 = c

RegEx Manual Volume II

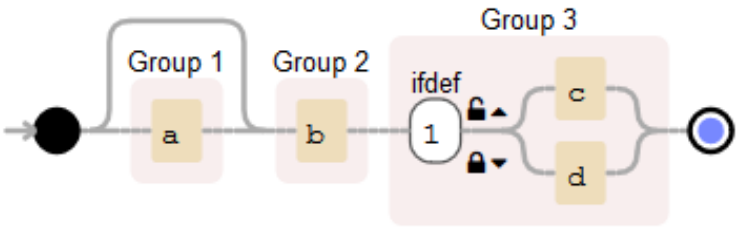
Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Tim Notes:

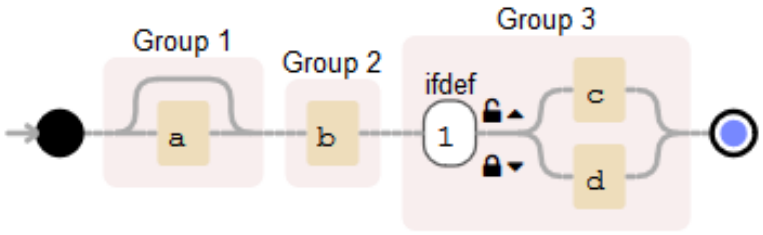
1. Note that (a)? is very different from (a?).

(a)? as used in (a)?(b)((?(1)c|d))



By placing the Lazy quantifier *outside* of the group (a) it makes the entire group optional – match zero or one time. If it doesn't match, it can move on to the next expression and evaluate for 'b'. This is how it has been used in these examples. If the match Fails, when the conditional is tested, it will evaluate the '*Else*' because (1) did not participate in the match.

(a?) as used in (a?)(b)((?(1)c|d))



By using the Lazy quantifier *inside* of the group (a), it indicates that within the group match zero or more times. This means that it will always capture a value – 'a' or 'null'. Therefore in the test for the conditional – it will always be True that (1) participated in the match always evaluating '*Then*' and never '*Else*'.

So do not make the mistake of using the Lazy quantifier inside of the group when used as a reference in a Conditional.

2. Refer to notes on Zero Occurrence Matches.

RegEx Manual Volume II

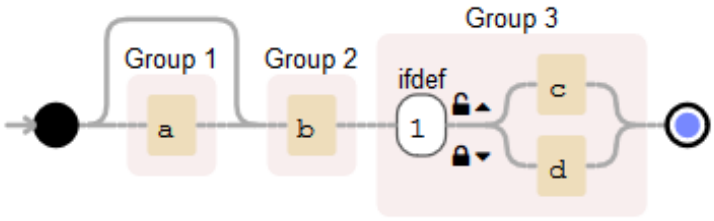
Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

String = abd

Match: (a)?(b)((?(1)c|d))
Replace: \1 - \2 - \3

abd - b - d



Where:

\1 = null \2 = b \3 = d

Tim Analysis:

Use abd

Match: (a)?(b)((?(1)c|d))

Result: abd

Sample:

```
.............Start·Length·................
Match 1: bd      1      2
Group 1 did not participate in the match
Group 2: b       1      1
Group 3: d       2      1
```

```
Beginning match attempt at character 1
1 backtrack
2 ok
3 b
4 b
5 bd
6 bd
7 bd
  Match found in 7 steps
```

1. a	Test for 'a' = True	Capture Group 1 = a
2. ?	Make it lazy and therefore optional	
3. b	..Test for 'b' = True	Capture Group 2 = b
4. (?(1)c|d)	Condition tests if Capture Group 1 took part in the match. = True '*Then*' tests for 'c' = 'd' = False Removes all values	

Tim Notes:

1. At this point in the evaluation, if (a) hadn't matched it would have tested '*Else*' 'd', but because (a) is matched this means that '*Then*' was tested so '*Else*' is not – yet.

RegEx Manual Volume II

Understanding Referenced Conditional Groups cont.

Using a Referenced Capturing Group for the 'if' cont.

Tim Analysis: cont.

5. a	RegEx Engine moves forward to next character, the 'b' and starts the expression over. Test for 'a' = 'b' = False	
6. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..	Capture Group 1 = null
7. b	..Test for 'b' = True	Capture Group 2 = b
4. (?(1)c|d)	Condition tests if Capture Group 1 took part in the match. = False '*Else*' tests for 'd' = True	Capture Group 3 = d

Tim Notes:

1. Refer to notes on Zero Occurrence Matches.

RegEx Manual Volume II

Understanding Relative Conditionals

Determines whether a capture group at a *relative* position to the *current* position has been set (contains a value). The relative position can be to the left (Back Relative Conditional) or to the right (Forward Relative Conditional) of the conditional.

Syntax:

The syntax is the same as that of a conditional that references a numbered capturing group with an added plus or minus sign before the group number.

Back Relative Conditional:

(-n)

Where n is the number referencing the Capture Group to the *left* of the current position.

E.g.

(?(-2)…|…)

Tests whether the second capture group to the left of the current position contains a value.

(?(-1)X|Y)

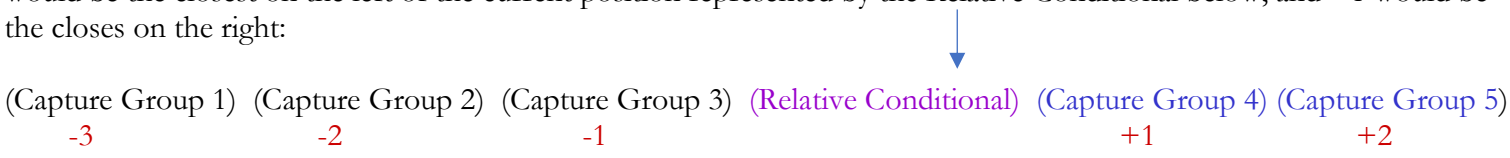
Tests whether the nearest capture group to the left of the current position contains a value, and because it is a conditional, if True because of the Alternate, '*Then*' match pattern X; '*Else*' match pattern Y.

Forward Relative Conditional:

(+n)

Where n is the number referencing the Capture Group to the *right* of the current position.

This is not necessarily the Capture Group 1 or 2 - 'n' refers to the distance from the current position. For example -1 would be the closest on the left of the current position represented by the Relative Conditional below, and +1 would be the closes on the right:



thus:

(-1) = Capture Group 3
(-2) = Capture Group 2
(-3) = Capture Group 1

(+1) = Capture Group 4
(+2) = Capture Group 5

RegEx Manual Volume II

Understanding Relative Conditionals cont.

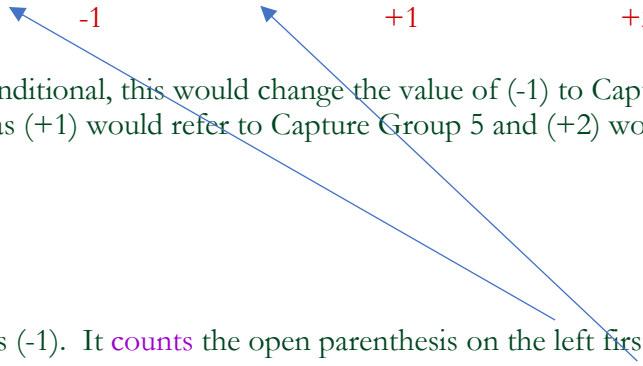
Tim Notes:

1. How the RegEx Engine determines the Capture Groups is by literally counting the number of 'Open Parenthesis' - the ' (' on the left side of it and the 'Open Parenthesis' on the right of it.

This is important because if you had this:

Capture Group 4

(Capture Group 1) (Capture Group 2) (Capture Group 3) ((Relative Conditional)) (Capture Group 5) (Capture Group6)
-4 -3 -2 -1 +1 +2



.. where a Capture Group was placed around the Relative Conditional, this would change the value of (-1) to Capture Group 4 and (-2) would refer to Capture Group 3, etc. Just as (+1) would refer to Capture Group 5 and (+2) would refer to Capture Group 6, etc.

Why isn't Capture Group 4 (+1)?

Capture Group 4 is considered to the *left* of the conditional as (-1). It counts the open parenthesis on the left first. Therefore Capture Group 4's left open parenthesis is determined to be on the left since there is only the *closed* parenthesis of Capture Group 4 to the right.

If you had this:

Capture Group 3 4 5

(Capture Group 1) (Capture Group 2) ((Relative Conditional (Then) (Else))) (Capture Group 6) (Capture Group7)
-3 -2 -1 +1 +2 +3 +4

Where a Capture Group was placed around the 'Then' and 'Else' values, (+1) refers to Capture Group 4, (+2) refers to Capture Group5, etc. (-1) would still refer to Capture Group 3, etc. Again, this is because both Capture Group 4 and Capture Group 5's open parenthesis are to the right of the conditional.

Also if you had this:

Capture Group 3

(Capture Group 1) (Capture Group 2) ((Relative Conditional))
-3 -2 -1

(+1) would come up with no match because there is no Capture Group to the right of the conditional.

2. Using a *relative* group in a conditional becomes important when you are working on a large RegEx expression, some of whose parts you may later decide to move. It can be easier to count a relative position such as -2 than an *absolute* position.

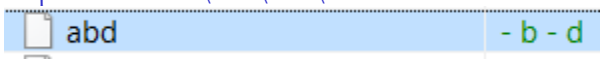
RegEx Manual Volume II

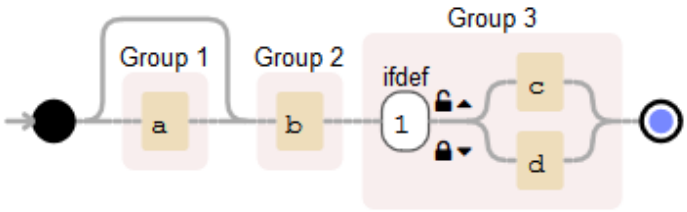
Understanding Relative Conditionals cont.

Take the last example:

String = abd

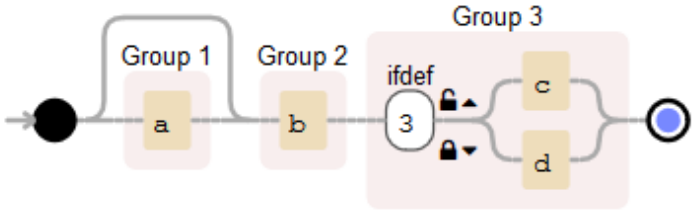
Match: (a)?(b)((?(1)c|d))
Replace: \1 - \2 - \3





Here is the equivalent:

Match: (a)?(b)((?(-1)c|d))
\1 - \2 - \3



Where:

\1 = a \2 = b \3 = d

RegEx Manual Volume II

Understanding Relative Conditionals cont.

Tim Analysis:

Use abd

Match: `(a)?(b)((?(-1)c|d))`

Result: **abd**

```
Beginning match attempt at character 0
1  a
2  a
3  ab
4  ab
5  abd
6  abd
7  abd
   Match found in 7 steps
```

Sample

```
·····Start·Length····
Match 1: abd    0      3
Group 1: a      0      1
Group 2: b      1      1
Group 3: d      2      1
```

Capture Group 1	Capture Group 2	Capture Group 3
(a)?	(b)	((?(-1)c|d))

1. a	Test for 'a' = True	Capture Group 1 = a
2. ?	Make it lazy and therefore optional	
3. b	..Test for 'b' = True	Capture Group 2 = b
4. (?(-1)c|d)	Condition tests if Capture Group 3 (closes to the Relative Conditional as (-1)) has a value. = False Capture Group 3 (The Conditonal) has no value at present. '*Else*' tests for 'd' = 'd' = True	Capture Group 3 = d

RegEx Manual Volume II

Understanding Relative Conditionals cont.

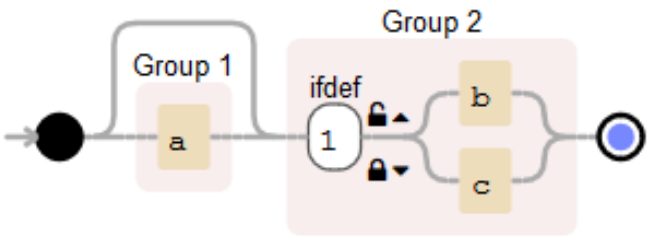
Here is another previous example modified to a Relative Conditional using Alternates:

String = babxcac

Original:

Match: (a)?((?(1)b|c))
Replace: \1 \2

babxcac a b



Sample:

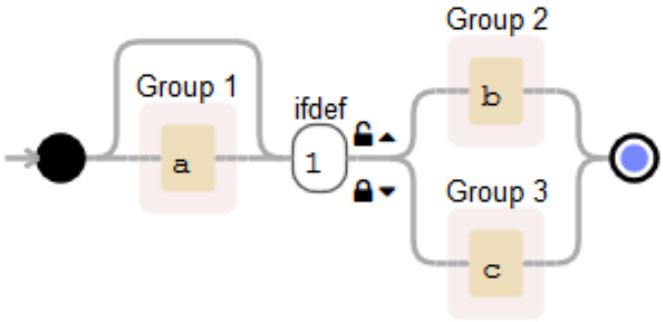
babxcac

```
            Start  Length
Match 1: ab     1       2
 Group 1: a     1       1
 Group 2: b     2       1
Match 2: c      4       1
 Group 1 did not participate in the match
 Group 2: c     4       1
Match 3: c      6       1
 Group 1 did not participate in the match
 Group 2: c     6       1
```

Modified:

Match: (a)?(?(-1)(b)|(c))
Replace: \1 \2

babxcac a b



Sample:

babxcac

```
            Start  Length
Match 1: ab     1       2
 Group 1: a     1       1
 Group 2: b     2       1
 Group 3 did not participate in the match
Match 2: c      4       1
 Group 1 did not participate in the match
 Group 2 did not participate in the match
 Group 3: c     4       1
Match 3: c      6       1
 Group 1 did not participate in the match
 Group 2 did not participate in the match
 Group 3: c     6       1
```

Where:
\1 = a \2 = b

RegEx Manual Volume II

Understanding Relative Conditionals cont.

Tim Analysis:

Use babxcac

Match: `(a)?(?(-1)(b)|(c))`

Result: `babxcac`

First Match:

```
Beginning match attempt at character 1
1  a
2  a
3  ab
4  ab
5  ab
Match found in 5 steps
```

Second match:

```
Beginning match attempt at character 4
1  backtrack
2  ok
3  c
4  c
5  c
Match found in 5 steps
```

Third Match:

```
Beginning match attempt at character 6
1  backtrack
2  ok
3  c
4  c
5  c
Match found in 5 steps
```

Sample First Match:

```
            Start  Length
Match 1: ab      1       2
Group 1: a       1       1
Group 2: b       2       1
Group 3 did not participate in the match
```

Sample: Second Match:

```
Match 2: c       4       1
Group 1 did not participate in the match
Group 2 did not participate in the match
Group 3: c       4       1
```

Sample Third Match:

```
Match 3: c       6       1
Group 1 did not participate in the match
Group 2 did not participate in the match
Group 3: c       6       1
```

RegEx Manual Volume II

Understanding Relative Conditionals cont.

Tim Analysis: cont.

Capture Group 1	Capture Group 2	Capture Group 3
(a)?	(b)	(c)

1. a	Test for 'a' at first character = False	
2. ?	Make it lazy and therefore optional When made optional, since it failed, it moves on to the next expression ..	
3. (?(-1)b|c)	.. Condition tests if Capture Group 1 (closes to the Relative Conditional as (-1)) has a value. = False Else tests for a 'c' = 'b' = False	
4.	RegEx Engine moves to next character and begins the evaluation over.	
5. a	Tests for 'a' = True	Capture Group 1 = a
6. (?(-1)b|c)	.. Condition tests if Capture Group 1 (closes to the Relative Conditional as (-1)) has a value. = True	
7.	'*Then*' Tests for 'b' = True	Capture Group 2 = b

Turns out that the end result didn't change, because both conditionals tested Capture Group 1, but how we got there did.

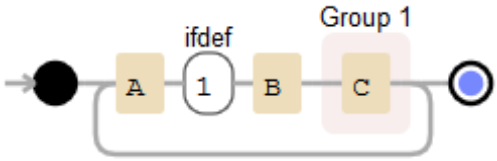
RegEx Manual Volume II

Understanding Relative Conditionals cont.

Example: without using Alternates

Match: (?:A(?(+1)B)(C))+
Replace: \1

ACABC C



Matches against the first 'C' in the string.

```
Beginning match attempt at character 0
 1  A
 2  Aok
 3  Aok
 4  AC
 5  AC
 6  AC
 7  ACA
 8  ACAB
 9  ACAB
10  ACABC
11  ACABC
12  ACABC
13  ACABCbacktrack
    Match found in 13 steps
```

Sample:

```
·····Start·Length··
Match 1: ACABC    0      5
Group 1: C        4      1
```

Tim Analysis:

String = ACABC

Match: (?:A(?(+1)B)(C))+

Result: **ACABC**

Non-Marking Group	Condition	Capture Group 1	End of Condition	End of Non-Marking Group	Greedy
(?:A	(? (+1) B)	(C)	)	)	+

1. A — Match against 'A' = True (Not Captured)
2. (?(+1)B)(C)) — .. Condition tests if Capture Group 1 (closes to the Relative Conditional on the right as (+1)) has a value. = False
   Capture Group 1 has not been evaluated yet so it won't test for 'B'
   Match against 'C' = True — Capture Group 1 = C
3. + — Greedy. Matches against remainder of string (Not Captured) = 'ACABC'

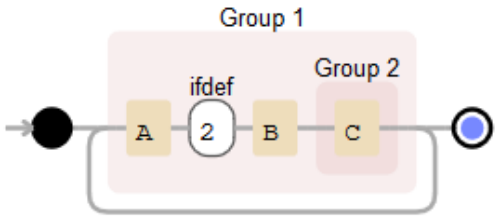
RegEx Manual Volume II

Understanding Relative Conditionals cont.

Example: If I enclose the entire expression in a Capture Group:

Match ((?:A(?(+1)B))(C))+
Replace: \1 \2

ACABC ABC C




```
Beginning match attempt at character 0
1  A
2  Aok
3  Aok
4  A
5  AC
6  AC
7  ACA
8  ACAB
9  ACAB
10 ACAB
11 ACABC
12 ACABC
13 ACABCbacktrack
14 ACABCbacktrack
15 ACABC
   Match found in 15 steps
```

Sample:

```
·················Start·Length····
Match 1: ACABC      0      5
Group 1: ABC        2      3
Group 2: C          4      1
```

Now –

(expression) becomes Capture Group 1
'C' is Capture Group 2

RegEx Manual Volume II

Understanding Relative Conditionals cont.

Example: If I enclose the entire expression in a Capture Group: cont.

Match ((?:A(?(+1)B))(C))+

Where:

\1 = ABC \2 = C

Tim Analysis:

1. A	Match against 'A' = True	Capture Group 1 = A
2. (?(+1)B)(C))	.. Condition tests if Capture Group 2 (closes to the Relative Conditional on the right as (+1)) has a value. Capture Group 2 has not been evaluated yet = False so it won't test for 'B'	
	Match against 'C' = True	Capture Group 2 = C
3.	Expression evaluates to[1]	Capture Group 1 = AC
4. +	Greedy. captures remainder of string.	Capture Group 1 = ACABC
5.	Repeat expression (the '+' made the entire expression Greedy) Because the string is at EOL, it Starts over at first character of string.	
6. A	Match against 'A' = True	Capture Group 1 = A
7. (?(+1)B)(C))	.. Condition tests if Capture Group 2 (closes to the Relative Conditional on the right as (+1)) has a value. = True	
	Match against 'B' = True	Capture Group 1 = AB
8. +	Greedy. captures remainder of string.	Capture Group 1 = ABC Capture Group 2= Unchanged

Tim Notes:

1. The expresion is finished but because Capture Group 1 was placed around the entire expression, the value of Capture Group 1 becomes *all* of the captured values that were matched within it (nested). This includes the 'A' and the 'C' even though 'C' was also captured in Capture Group 2.

RegEx Manual Volume II

Understanding Subroutine Calls

Syntax:

i.e., a(b(?1)?y)z

By Index:

Where:

(?1) refers to Capture Group 1

(?2) refers to Capture Group 2 etc.

i.e., a(b(?-1)?y)z

By Relative Position

Where:

(?-1) refers to the Capture Group closest to the left of the current position

(?-2) refers to the Second Closest Capture Group to the left of the current position

(?+1) refers to the Capture Group closest to the right of the current position

(?+2) refers to the Second Closest Capture Group to the right of the current position

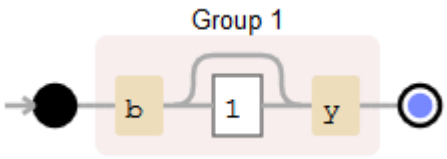
RegEx Manual Volume II

Understanding Subroutine Calls cont.

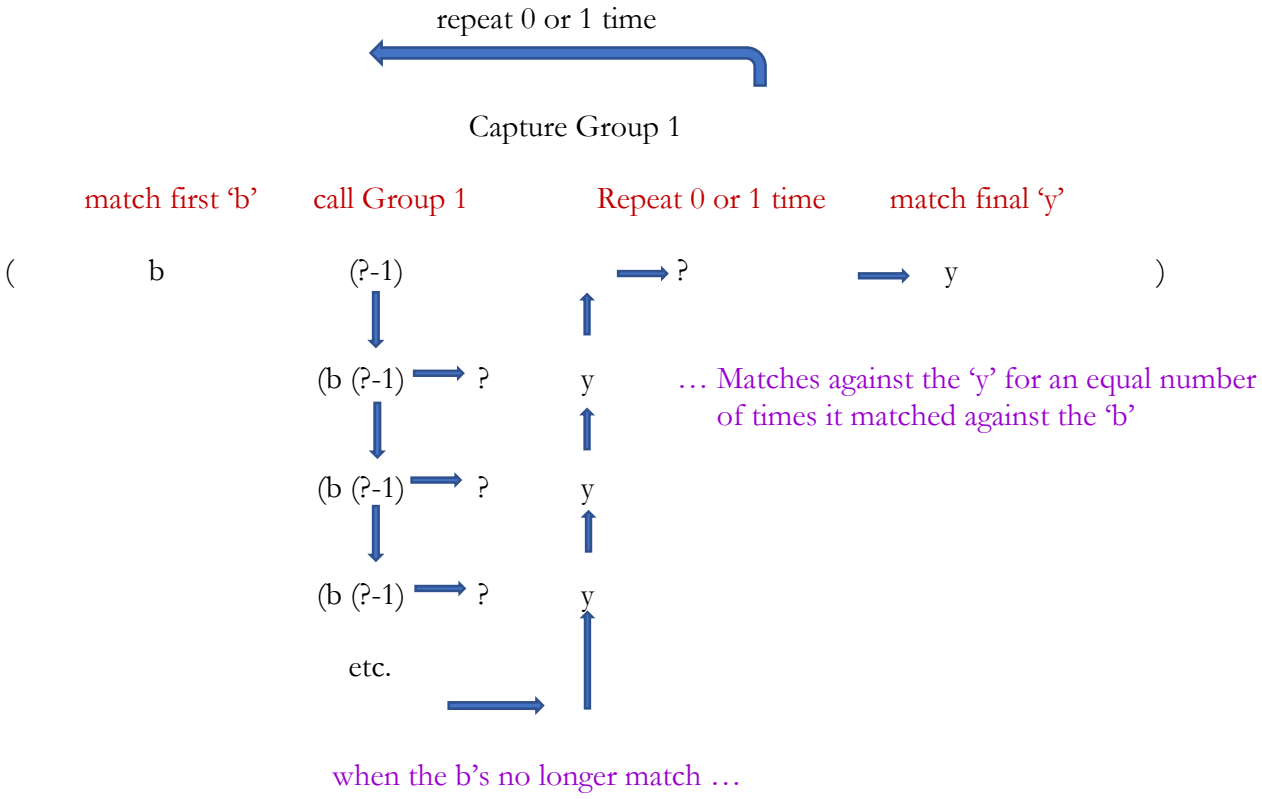
When you want to use a sub-expression multiple times without rewriting it, you can group it then call it as a subroutine.

Match: (b(?-1)?y)
Replace: \1 <space>

abbbyyyz	bbbyyy
abbyyz	bbyy
abyz	by



Breaks down:



Tim Notes:

1. it will only match against an equal number of y's as there were matches against the b's
   i.e.,

abbbyy	bbyy

or

abbyyy	bbyy

2. Because the Call to the Subroutine is placed within a Group, the call becomes 'Recursive', meaning that the Subroutine is called over and over again until it no longer matches. Then, the next part of the expression is evaluated, which is also within the same group, so it is evaluated an equal number of times as the Subroutine was called, because each evaluation represents one call within the same group.

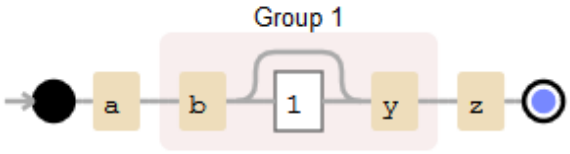
RegEx Manual Volume II

Understanding Subroutine Calls cont.

If we add some further matches to the expression it won't change the results:

Match: a(b(?-1)?y)z
Replace: \1

abbbyyyz	bbbyyy
abbyyz	bbyy
abyz	by




```
Beginning match attempt at character 0
1  a
2  ab
3  abb
4  abbb
5  abbbbacktrack
6  abbbok
7  abbby
8  abbby
9  abbbyy
10 abbbyy
11 abbbyyy
12 abbbyyy
13 abbbyyyz
   Match found in 13 steps
```

Sample:

```
·····················Start·Length···
Match 1: abbbyyyz       0      8
Group 1: bbbyyy         1      6
```

Breaks down:

match		Capture Group 1				match
		match	call Group 1	Repeat 0 or 1 time	match	
a	(	b	(?-1)	?	y	) z

Tim Analysis:

use abbbyyyz

1. a	Match against 'a' = True (Not Captured)	
2. b	Match against 'b' = True	Capture Group 1 = b
3. (?-1)	Call Group 1 (Recursive)	
	3a. ? Repeats for all 'b's it can match against	
	3b. y Match 'y' for each expression that matched against 'b'	
4. ?	Repeat the expression zero or more times	
	.. this repeats step 3 again.	Capture Group 1 = bbbyy
5. y	Match against last 'y' of string if found	Capture Group 1 = bbbyyyy
5. z	Match against 'z' = True (Not Captured)	

RegEx Manual Volume II

Understanding Subroutine Calls cont.

You've seen that this expression under 'Understanding Relative Conditionals' yields a value of 'ABC' for Capture Group 1 and a value of 'C' for Capture Group 2. The 'ABC' is because the expression was made Greedy and captured the 'ABC' value when it started over at the first character.

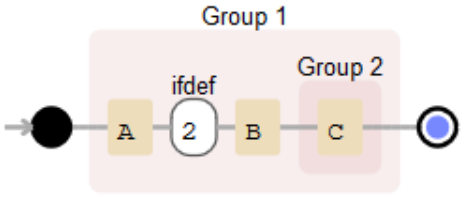
Match ((?:A(?(+1)B))(C))+
Replace: \1 \2

ACABC	ABC C

Here is the same RegEx, this time without the Greedy quantifier.

Match: (A(?(+1)B)(C))
Replace: \1 \2

ACABC	AC C




```
Beginning match attempt at character 0
1  A
2  Aok
3  Aok
4  AC
5  AC
6  AC
   Match found in 6 steps
```

Sample:

```
            Start  Length
Match 1: AC     0       2
Group 1: AC     0       2
Group 2: C      1       1
```

Tim Analysis:

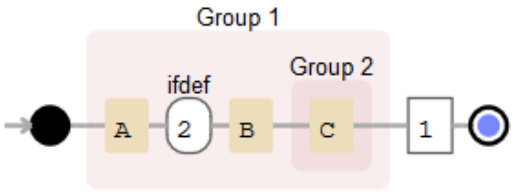
1. A	Match against 'A' = True	Capture Group 1 = A
2. (?(+1)B)(C))	.. Condition tests if Capture Group 2 (closes to the Relative Conditional on the right as (+1)) has a value. Capture Group 2 has not been evaluated yet = False so it won't test for 'B'	
	Match against 'C' = True	Capture Group 2 = C
3. (A(?(+1)B)(C))	Expression evaluates to The expresion is finished but because Capture Group 1 was placed around the entire expression, the value of Capture Group 1 becomes all of the captured values that were matched within it. This includes the 'A' and the 'C' even though 'C' was also captured in Capture Group 2 (nested).	Capture Group 1 = AC

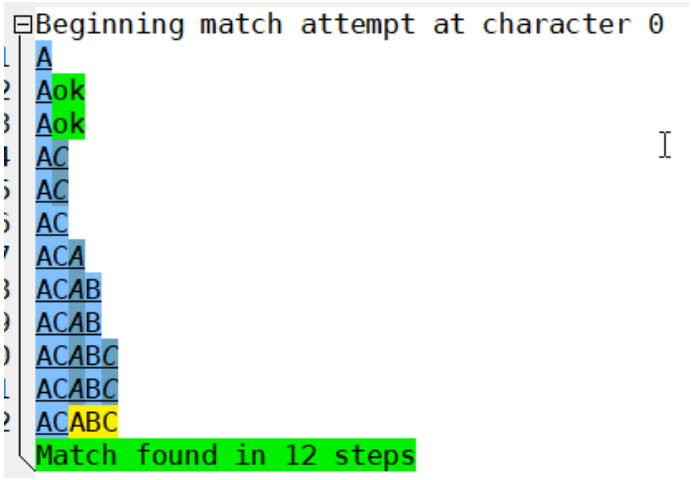
RegEx Manual Volume II

Understanding Subroutine Calls cont.

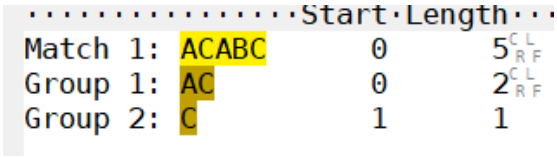
Replace the Greedy Quantifier with a Subroutine Call:

Match: (A(?(+1)B)(C))(?1)
Replace: \1 \2





Sample:



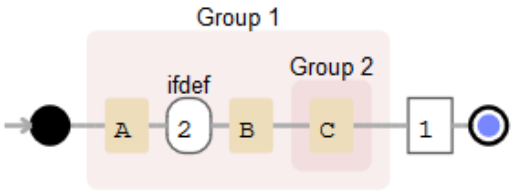
When used in this manner, the PCRE RegEx Engine restores the previous Capture Groups values upon exiting a Subroutine. Therefore even though the value of Group 1 changed to ABC during the subroutine, it restored back to 'AC'. Capture Group 2 restored as well, although this wasn't noticeable since both values were the same = 'C'.

RegEx Manual Volume II

Understanding Subroutine Calls cont.

Replace the Greedy Quantifier with a Subroutine Call: cont.

Match: (A(?(+1)B)(C))(?1)



Tim Analysis:

String = ACABC

Capture Group 1	Condition	Capture Group 2	End Condition	End Capture Group 1	Calls Capture Group 1
(A	(?	(+1) B)	(C)	)	(?1)

1. A	Match against 'A' = True	Capture Group 1 = A
2. (?(+1)B)(C))	.. Condition tests if Capture Group 2 (closes to the Relative Conditional on the right as (+1)) has a value. Capture Group 2 has not been evaluated yet = False so it won't test for 'B'	
	Match against first 'C' = True	Capture Group 2 = C
3.	Expression in Capture Group 1 evaluates =	Capture Group 1 = AC
3. (?-1)	Call Group 1 (Not Recursive)	
1. A	Match against 'A' = True	Capture Group 1 = A
2. (?(+1)B)(C))	.. Condition tests if Capture Group 2 (closes to the Relative Conditional on the right as (+1)) has a value = True	
	Match against 'B' = True	Capture Group 1 = AB
	Match against second 'C' = True	Capture Group 1 = ABC Capture Group 2= Unchanged
3.	Exits the subroutine and restores back previous values for Capture Groups 1 and 2.	Capture Group 1= AC Capture Group 2= C

Tim Notes:

1. This example was just for illustration. For practical purposes in renaming, it has no benefit over (A(?(+1)B)(C)).

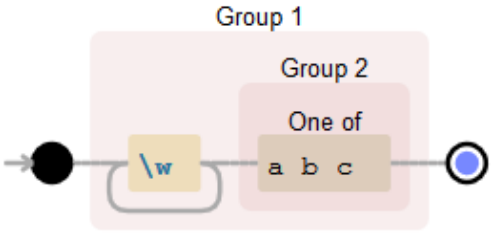
RegEx Manual Volume II

Understanding Subroutine Calls cont.

Example:

Match: (\w+([abc]))
Replace: \1 \2

123456789abc	123456789abc c
123abc456abc789abc	123abc456abc789abc c
123d56789abcde12345	123d56789abc c
ababc	ababc c
abbcab	abbcab b
abc	abc c
abca	abca a
abcbba	abcbba a
abd	ab b
ACABC	ACABC
babxcac	babxcac c
bc	bc c
bd	bd



Using 123d56789abcde12345

Match: (\w+([abc]))

Result: 123d56789abcde12345

Where:
\1 = 123d56789abc \2 = c

```
Beginning match attempt at character 0
 1 123d56789abcde12345
 2 123d56789abcde12345backtrack
 3 123d56789abcde12345backtrack
 4 123d56789abcde1234
 5 123d56789abcde1234backtrack
 6 123d56789abcde1234backtrack
 7 123d56789abcde123
 8 123d56789abcde123backtrack
 9 123d56789abcde123backtrack
10 123d56789abcde12
11 123d56789abcde12backtrack
12 123d56789abcde12backtrack
13 123d56789abcde1
14 123d56789abcde1backtrack
15 123d56789abcde1backtrack
16 123d56789abcde
17 123d56789abcdebacktrack
18 123d56789abcdebacktrack
19 123d56789abcd
20 123d56789abcdbacktrack
21 123d56789abcdbacktrack
22 123d56789abc
23 123d56789abcbacktrack
24 123d56789abcbacktrack
25 123d56789ab
26 123d56789abc
27 123d56789abc
28 123d56789abc
Match found in 28 steps
```

Sample:

```
                     Start  Length
Match 1: 123d56789abc    0      12
Group 1: 123d56789abc    0      12
Group 2: c              11       1
```

Tim Analysis:

To make literally, a long story short, \w+ matches the entire string. '123d56789abc'de12345' capturing the value in capture Group 1.

Then it begins to evaluate the ([abc]) and since it is at EOL, it begins backtracking until it matches the 'c'. This is captured in Capture Group 2.

Capture Group 1 is reevaluated to '123d56789abc'

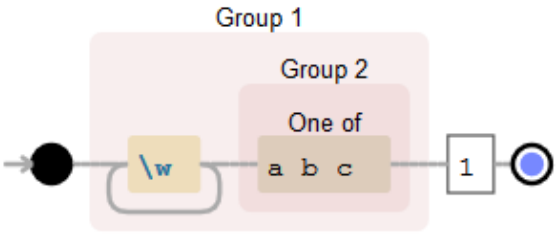
RegEx Manual Volume II

Understanding Subroutine Calls cont.

Example: Add a Subroutine call that calls Capture Group 1:

Match: (\w+([abc]))(?1)
Replace: \1 \2

123456789abc	123456789a a
123abc456abc789abc	123abc456abc789a a
123d56789abcde12345	123d56789a a
ababc	aba a
abbcab	abbc c
abc	abc
abca	ab b
abcbba	abcb b
abd	abd
ACABC	ACABC
babxcac	babxc c
bc	bc
bd	bd



Sample:

```
·····················Start·Length···
Match 1: 123d56789abc      0     12
Group 1: 123d56789a        0     10
Group 2: a                 9      1
```

Using 123d56789abcde12345

Where:

\1 = 123d56789abc \2 = a

Match: (\w+([abc]))(?1)

Result: 123d56789abcde12345

Tim Analysis:

As before, \w+ matches the entire string '123d56789abc'de12345'. Then begins backtracking until it encounters the 'c' of the Character Class consisting of 'a' 'b' or 'c' at the '123d56789abc' position.

The (?1) calls Capture Group 1 from the current position. From there, it returns back to the end of the string each time, backtracking until it matches the 'b' of '123d56789abc' and repeating to match the 'a' of '123d56789abc'.

Next, it repeats the backtracking but it can't match against the '9'. Capture Group 1 = '123d56789abc'. Finally, it evaluates the ([abc]) so it moves forward to the next character, the 'a' and matches. Capture Group 2 = 'a'.

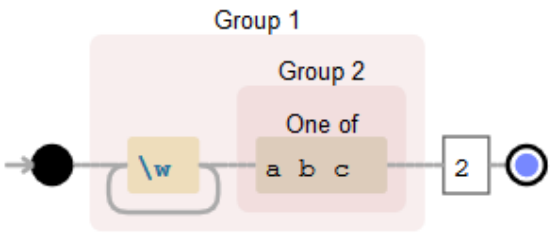
RegEx Manual Volume II

Understanding Subroutine Calls cont.

Example: Add a Subroutine call that calls Capture Group 2:

Match: (\w+([abc]))(?2)
Replace: \1 \2

123456789abc	123456789ab b
123abc456abc789abc	123abc456abc789ab b
123d56789abcde12345	123d56789ab b
ababc	abab b
abbcab	abbca a
abc	ab b
abca	abc c
abcbba	abcbb b
abd	abd
ACABC	ACABC
babxcac	babxca a
bc	bc
bd	bd



Sample:

```
.....................Start·Length···
Match 1: 123d56789abc       0      12
Group 1: 123d56789ab        0      11
Group 2: b                 10       1
```

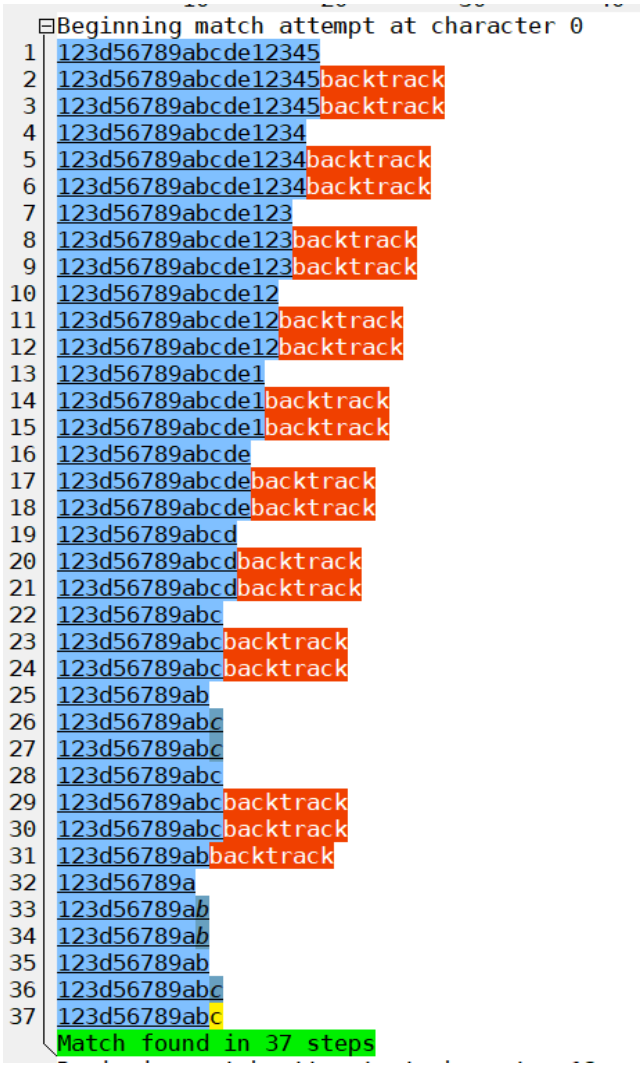
Using 123d56789abcde12345

Where:

\1 = 123d56789ab \2 = b

Match: (\w+([abc]))(?2)

Result: 123d56789abcde12345



Tim Analysis:

The \w+ matches the entire string '123d56789abc'de12345' in Capture Group 1. The (?2) calls Capture Group 2 from the current position. Because it is at EOL, it begins backtracking to match against, the 'b' of '123d56789ab'.

Capture Group 2 = 'b' and Capture Group 1 is reevaluated to '123d56789ab'.

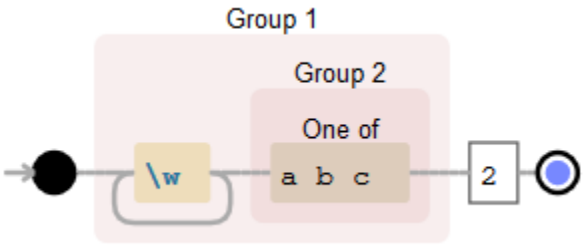
RegEx Manual Volume II

Understanding Subroutine Calls cont.

Example: Add a Subroutine that calls the first Capture Group on the left relative to the current position

Match: (\w+([abc]))(?-1)
Replace: \1 \2

123456789abc	123456789ab b
123abc456abc789abc	123abc456abc789ab b
123d56789abcde12345	123d56789ab b
ababc	abab b
abbcab	abbca a
abc	ab b
abca	abc c
abcbba	abcbb b
abd	abd
ACABC	ACABC
babxcac	babxca a
bc	bc
bd	bd

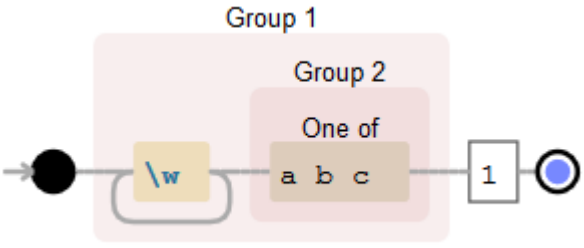


This is the equivalent of (\w+([abc]))(?-1), because (?-1) refers to Capture Group 2 as the closest relative to the current position.

Example: Add a Subroutine that calls the second Capture Group on the left relative to the current position

Match: (\w+([abc]))(?-2)
Replace: \1 \2

123456789abc	123456789a a
123abc456abc789abc	123abc456abc789a a
123d56789abcde12345	123d56789a a
ababc	aba a
abbcab	abbc c
abc	abc
abca	ab b
abcbba	abcb b
abd	abd
ACABC	ACABC
babxcac	babxc c
bc	bc
bd	bd



This is the equivalent of (\w+([abc]))(?1), because (?-2) refers to Capture Group 1 as the second closest relative to the current position.

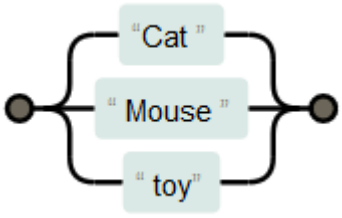
RegEx Manual Volume II

Understanding Branch Reset Group

Syntax: (?|RegEx)

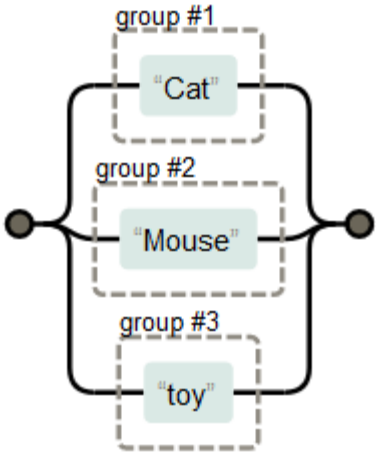
Branch Reset Groups are only used with Alternation.

Here is an alternation with three alternates:



Cat | Mouse | toy

If I wanted to capture the result of the alternate, I would use Capture Groups:



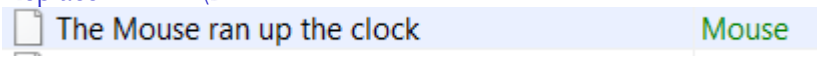
(Cat) | (Mouse) | (toy)

I now have three Capture Groups.

But when applied against a string, there can be only one outcome at a time:

String = The Mouse ran up the clock

Match: (Cat)|(Mouse)|(toy)
Replace: \2

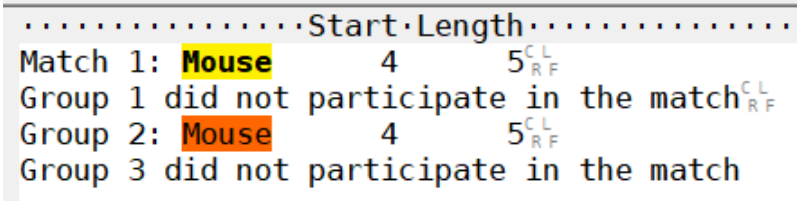


Where:

\1 = null \2 = Mouse \3 = Invalid

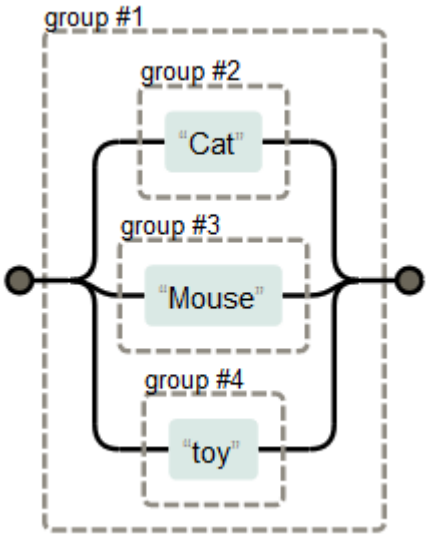
Backreference 2 has a value. Backreference 1, however, has a value of null beacuse Capture Group 1 is evaluated but proved False. Capture Group 3 is Invalid because 'toy' was never evaluated due to 'Mouse' matching prior. Capture Groups 1 and 3 are wasted because they don't match or were not evaluated. I say wasted only if you are hurting for Capture Groups - remember, only 9 allowed within a RegEx. This may be a good place to tweak the expression for better utilization of these Capture Groups.

Sample:



RegEx Manual Volume II

Understanding Branch Reset Group cont.



Placing the entire expression in a Capture Group results in -

Match: ((Cat)|(Mouse)|(toy))
Replace: \1

The Mouse ran up the clock Mouse

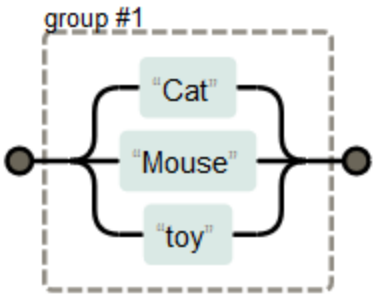
Where:

\1 = Mouse \2 = null \3 = Mouse

The nesting of Capture Groups 2, 3, and 4, within Capture Group 1 means that Capture Group 1 holds the evaluated values inclusive of these groups. Thus the 'True' evaluation of the Alternate 'Mouse' for Capture Group 3 is also the value of Capture Group 1. Capture Group 2 – 'Cat' is evaluated as 'false' so it holds a null value. Capture Group 4, 'Toy', is never evaluated because of the match of 'Mouse' prior. Because of these duplicated values, I say that these Capture Groups are 'wasted' – especially if you are hurting for Capture Groups. Let's presume that you are.

Sample:

```
·················Start·Length·················
Match 1: Mouse       4       5
Group 1: Mouse       4       5
Group 2 did not participate in the match
Group 3: Mouse       4       5
Group 4 did not participate in the match
```



This would solve the problem:

Match: (Cat|Mouse|toy)
Replace: \1

The Mouse ran up the clock Mouse

Where:

\1 = Mouse

Now only backreference 1 is defined and holds the value Mouse

Sample:

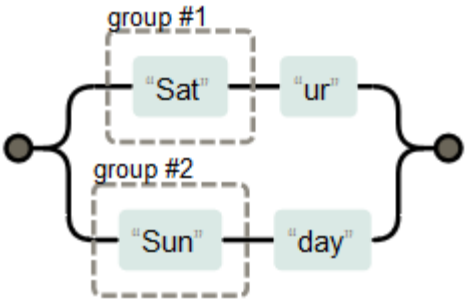
```
·················
Match 1: Mouse
Group 1: Mouse
```

RegEx Manual Volume II

Understanding Branch Reset Group cont.

… But what if I had this:

String = In the park on Sunday

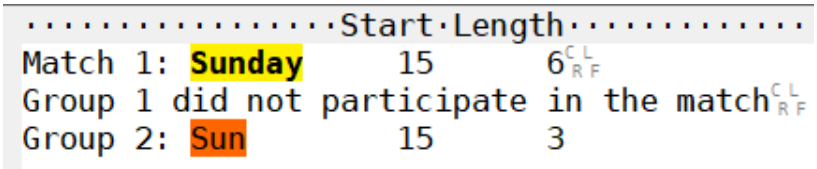


Match: (Sat)ur|(Sun)day
Replace: \2

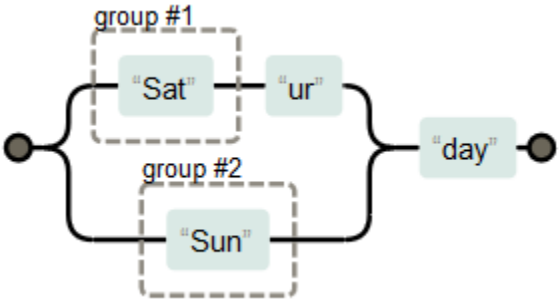
Where:

\1 = null \2 = Sun

Sample:



Using a Non-Marking Group doesn't make a difference because the result of the alternate still needs to be captured -

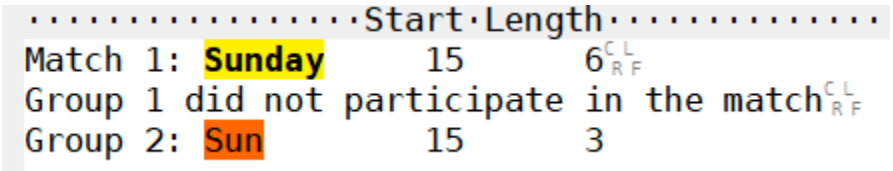


Match: (?:(Sat)ur|(Sun))day
Replace: \2

Where:

\1 = null \2 = Sun both result in a null value in Capture Group 1.

Sample:



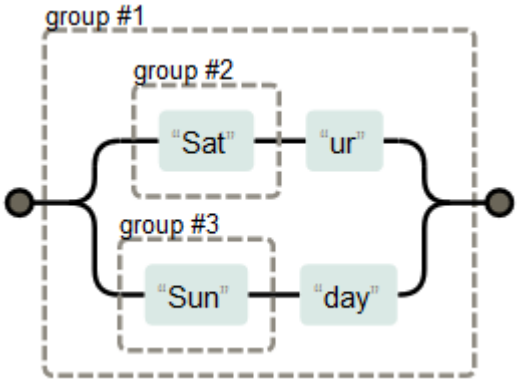
RegEx Manual Volume II

Understanding Branch Reset Group cont.

If we place the entire expression in a Capture Group as we did with Mouse:

Match: ((Sat)ur|(Sun)day)
Replace: \1

In the park on Sunday	Sunday



Where:

\1 = Sunday \2 = null \3 = Sun There is still a wasted Capture Group with a null value.

Sample:

```
·····················Start·Length·················
Match 1: Sunday      15      6
Group 1: Sunday      15      6
Group 2 did not participate in the match
Group 3: Sun         15      3
```

What is the solution? More tweaking? Yes, probably, but there is an easier method; Branch Reset Groups.

RegEx Manual Volume II

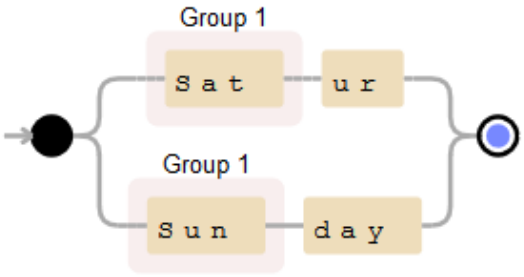
Understanding Branch Reset Group cont.

Introducing Branch Reset Groups

Syntax: (?|RegEx)

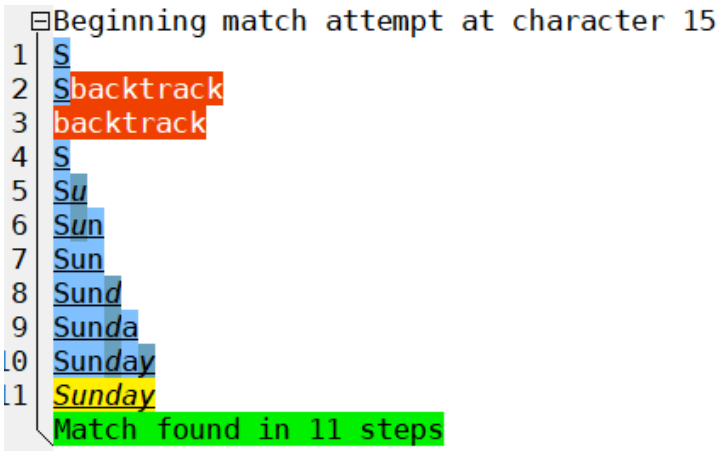
This is useful when you want to capture one, but not all, of a number of alternatives. Each test of an alternative is referred to as a 'Branch'.

Match: (?|(Sat)ur|(Sun)day)
Replace: \1

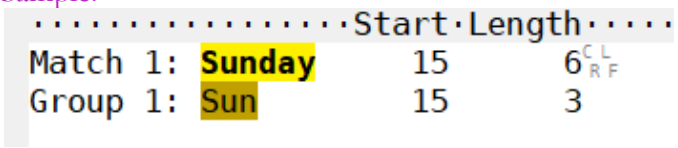


Match: (?|(Sat)ur|(Sun)day)

Result: In the park on **Sunday**



Sample:



Where: \1 = Sun

Now only 1 Capture Group is used.

RegEx Manual Volume II

Understanding Branch Reset Group cont.

How does it work?

Inside a branch reset pattern, capture groups are numbered as usual, but the number is reset at the start of each branch so even with multiple alternatives with capturing groups, the capturing group numbers are the same for all the alternatives.

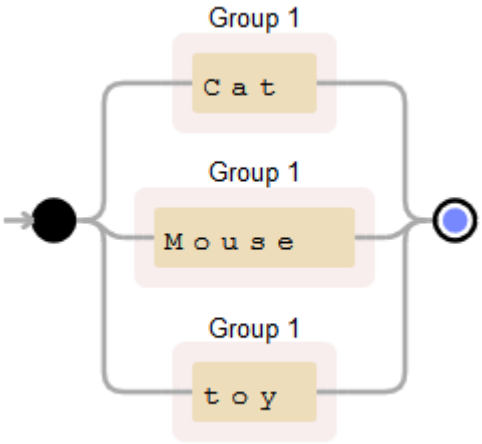
So for three alternatives with Capture Groups 1, 2 and 3 – the Match will be Capture Group 1.

String = The Mouse ran up the clock

Match: (?|(Cat)|(Mouse)|(toy))
Replace: \1

The Mouse ran up the clock	Mouse

Where: \1 = Mouse



Sample:

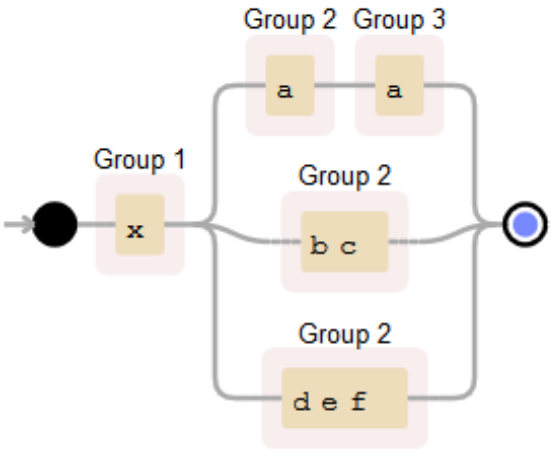
```
.................
Match 1: Mouse
Group 1: Mouse
```

Example: 3 alternatives but the first (a) (a) has two separate capture groups while the (bc) and (def) share one

String = xaa

Match: (x)(?|(a)(a)|(bc)|(def))
Replace: \1 \2

xdefdef	x def
xbcbc	x bc
xaa	x a



RegEx Manual Volume II

Understanding Branch Reset Group cont.

String = xaa

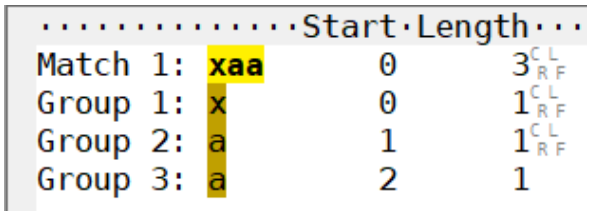
Observations:

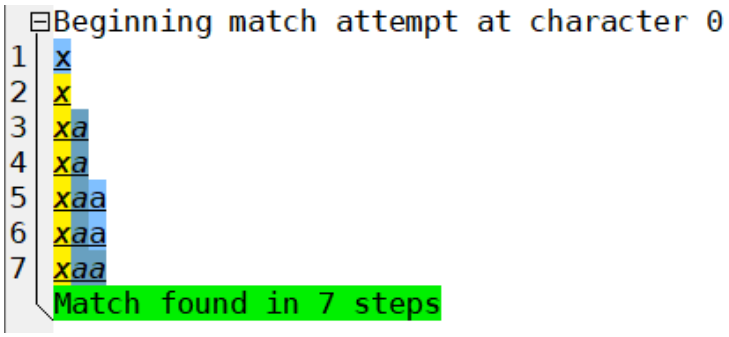
1. In order to match the first alternative, both values have to be present together following 'x'

Match: `(x)(?|(a)(a)|(bc)|(def))`

Result: **xaa**

Sample:





Where:

\1 = x \2 = a \3 = a

Why is there a third backreference?

The second value of 'a' creates a third Capture Group separate and apart from Capture Group 2 if matched. Remember, that the numbering of the Capture Groups is *reset* at the *start* of each branch – meaning, as each alternative is tested.

RegEx Manual Volume II

Understanding Branch Reset Group cont.

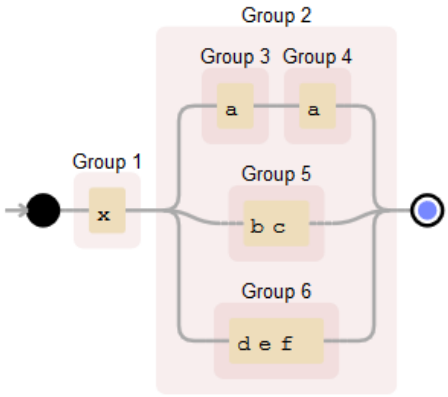
Example: 3 alternatives but the first (a) (a) has two separate capture groups while the (bc) and (def) share one cont.

So for xaa …

1. x	Test for 'x' = True	Capture Group 1 = x
2. a a | b c | d e f	Test next character is 'a' or 'b' or 'd' = 'a' = True	Capture Group 2 = a
3. a a | b c | d e f	Test next character is 'a' = True	Capture Group 3 = a

Because the evaluation of the '(a) (a)' Alternates was within the same branch being tested, the Capture Groups numbering was not reset, so Capture Group 1 captured the first 'a' and capture Group 2 captured the second 'a'. For the other values, xdefdef and xbccbc, only Capture Group 2 was used because for each branch tested, the numbering was reset.

Without using a Branch Reset Group:



Match: (x)((a)(a)|(bc)|(def))
Replace: \1 \2

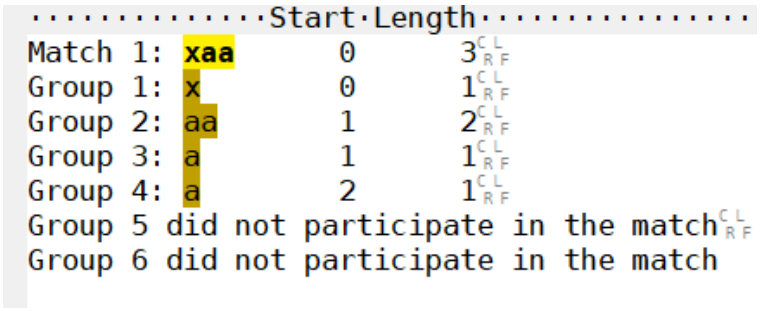
xdefdef	x def
xbcbc	x bc
xaa	x aa

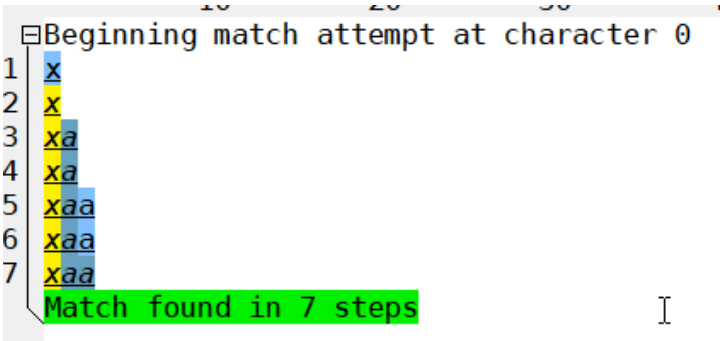
String = xaa

Match: (x)((a)(a)|(bc)|(def))

Result: **xaa**

Sample:





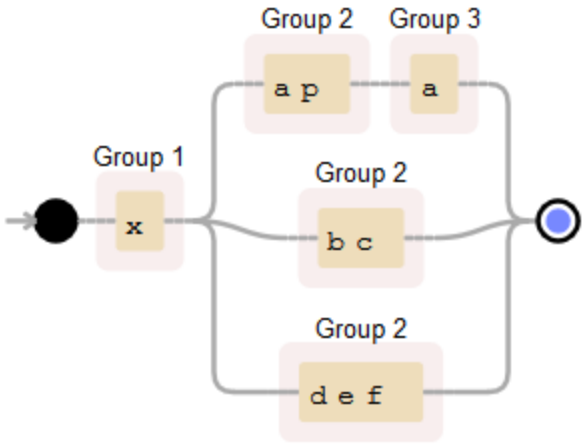
Where:
\1 = x \2 = aa \3 = a \4 = a

Both of the values of (a)(a) are captured as part of Capture Group 2. Capture Groups 3 and 4 represent the first and second 'a' captures respectively.

RegEx Manual Volume II

Understanding Branch Reset Group cont.

Here's another example:



String = xbcxapav

Match: (x)(?|(ap)(a)|(bc)|(def))
Replace: \1 \2

xbcxapav x bc

Where:
\1 = x \2 = bc

Match: (x)(?|(ap)(a)|(bc)|(def))

Result: xbcxapav

```
Beginning match attempt at character 0
1  x
2  x
3  xbacktrack
4  xbacktrack
5  xb
6  xbc
7  xbc
8  xbc
Match found in 8 steps
```

```
Beginning match attempt at character 3
1  x
2  x
3  xa
4  xap
5  xap
6  xapa
7  xapa
8  xapa
Match found in 8 steps
```

Okay – so this shows two matches, but we are not talking about multiple matches, this is, after all, still just an Alternate and there can only be one branch value in BRU. Why, however, didn't 'xapa' match before 'xbc'?

In the string, 'bc' comes before 'ap' although in the RegEx the alternate 'ap' 'a' is listed before 'bc. But it is the string that dictates the value selected, not the order in which they appear in the RegEx.

Review Time:

String = toy mouse

Match: (toy|mouse)
Replace: \1

toy mouse toy

Match: (mouse|toy)
Replace: \1

toy mouse toy

Conclusion:

Both have the value 'toy' because 'toy' comes before 'mouse' regardless of the alternate's position in the RegEx.

RegEx Manual Volume II

Understanding Branch Reset Group cont.

So now we got that out of the way and avoided any mass confusion, back to the analysis:

String = xbcxapav

Tim Analysis:

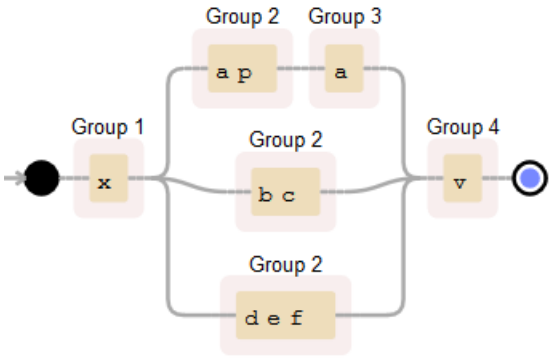
1. x	Test for 'x' = True	Capture Group 1 = x
2. ap a | b c | d e f	Test next character is 'a' or 'b' or 'd' = 'b' = True	Capture Group 2 = b
3. ap a | b c | d e f	Test next character = 'c' = True Current position = 'c"	Capture Group 2 = bc

Never mind any further matches. Only the first match counts in BRU in this instance.

So when does BRU test the second alternative?

Example:

String = xbcxapav



Match: (x)(?|(ap)(a)|(bc)|(def))(v)
Replace: \1\2 <space>

xbcxapav	xap

Replace: \1\2\3

xbcxapav	xapa

Replace: \1\2\3\4

xbcxapav	xapav

```
Beginning match attempt at character 3
1  x
2  x
3  xa
4  xap
5  xap
6  xapa
7  xapa
8  xapa
9  xapav
10 xapav
   Match found in 10 steps
```

Sample:

```
................
Match 1: xapav
Group 1: x
Group 2: ap
Group 3: a
Group 4: v
```

RegEx Manual Volume II

Understanding Branch Reset Group cont.

String = xbcxapav

Match: `(x)(?|(ap)(a)|(bc)|(def))(v)`

Result: `xbcxapav`

Sample:

```
·················Start·Length·····
Match 1: xapav       3       5
Group 1: x           3       1
Group 2: ap          4       2
Group 3: a           6       1
Group 4: v           7       1
```

Where:

\1 = x \2 = ap \3 = a \4 = v

Tim Analysis:

The first time it evaluates the string, it has not encountered the 'v'

1. x	Test for 'x' = True for first 'x' in the string.	Capture Group 1 = x
2. ap a | b c | d e f	Test next character is 'a' or 'b' or 'd' = 'b' = True	Capture Group 2 = b
3. ap a | b c | d e f	Test next character = 'c' = True Current position = 'c"	Capture Group 2 = bc
4. v	Matches against 'v'	
5.	Backtracks to re-evaluate string	

This time it will re-evaluate the string based on the 'v' value

6. x	Test for 'x' = True for second 'x'	Capture Group 1 = x
7. ap a | b c | d e f	Test next character is 'a' or 'b' or 'd' = second 'a' of string = True	Capture Group 2 = a
8. ap a | b c | d e f	Test next character = 'p' = True Current position = 'p'	Capture Group 2 = ap
9. ap a | b c | d e f	Test next character = third 'a' of string = True	Capture Group 3 = a
10. v	Matches against 'v'	Capture Group 4 = v

RegEx Manual Volume II

Understanding Atomic Groups with BRU

Syntax: (?>regex)

With an Atomic group, the RegEx Engine does not backtrack after a match has been found.

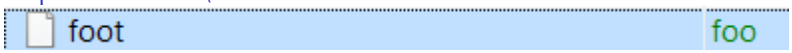
If the remainder of the RegEx Fails further down the string, the engine may backtrack if a quantifier or alternation makes it optional but it will not backtrack into the Atomic group to try other permutations (additional attempts) of that group.

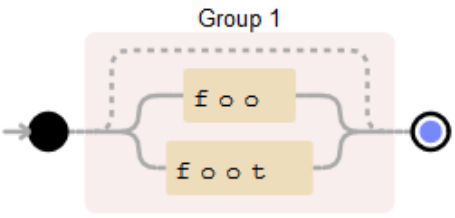
Typically, if the RegEx Fails further down the string and needs to backtrack, a capture group containing a quantifier would give up characters (what I refer to as 'reevaluating') one at a time, allowing the engine to try other matches. This is what causes a Capture Group to reevaluate its value or give it up entirely, invalidating the group. Likewise, if the group contained an alternation, the engine would attempt to match against the next branch. An atomic group won't do that: it's all or nothing. It will take the first successful value of the alternation only and will not test any other branches.

Example:

String = foots

Match: ((?>foo|foot))
Replace: \1

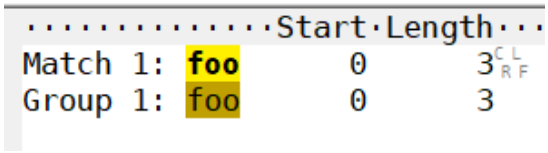




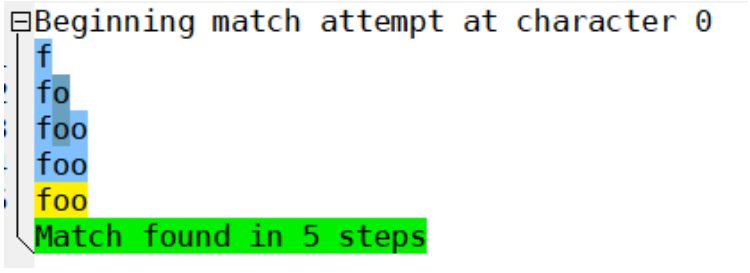
Match: ((?>foo|foot))

Result: foots

Sample:



Here you can see that the Match was based on one branch with no backtracking to test for a second branch.



Tim Notes:

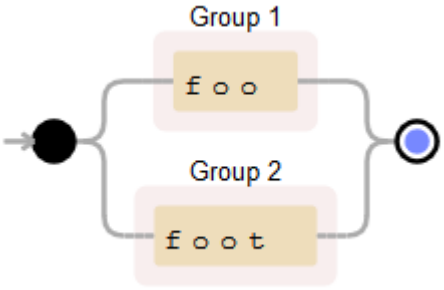
1. Atomic Groups are part of the Non-Marking (aka Non-Capturing, Non-Matching groups). I added the additional parentheses to capture the value. Other Atomic Groups include Lookarounds.

RegEx Manual Volume II

Understanding Atomic Groups with BRU cont.

String = foots

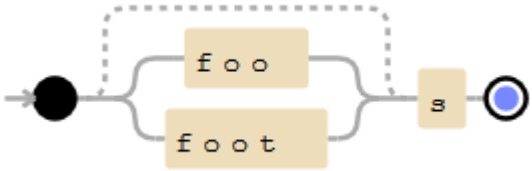
However, in BRU, only 'foo' will match regardless –



Match: (foo)|(foot)
Replace: \1

foot	foo

Here is an example that won't match

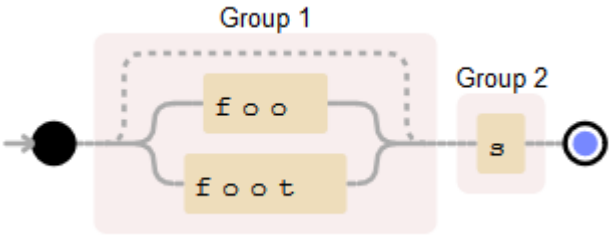


String = foots

Match: (?>foo|foot)s

Applied to foots will match its 1st alternative foo, then fail as 's' does not immediately follow, and stop as backtracking is disabled.

Likewise and for the same reasons, this won't work in Bru either:

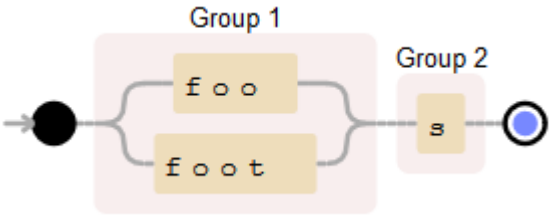


String = foots

Match: ((?>foo|foot))(s)

Because the value is 'foo' and therefore 'foo' + 's" = "foos' doesn't match against 'foot' or 'foots'

Equivalent in BRU that would work is without the Atomic Group:



Match: (foo|foot)(s)
Replace: \1\2 <space>

foots	foots

Foot is matched because BRU is allowed to backtrack and select the second alternative.

RegEx Manual Volume II

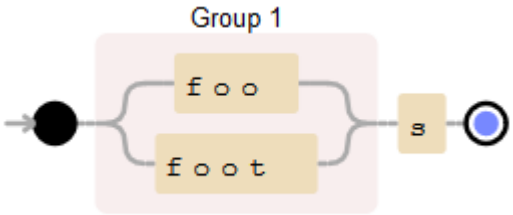
Understanding Atomic Groups with BRU cont.

String = foots

A non-atomic group will allow backtracking. If subsequent matching ahead Fails, it will backtrack and use alternative patterns until a match for the entire expression is found or all possibilities (permutations) are exhausted.

Match: `(foo|foot)s`

Result: `foots`



Both matches can be seen here:

```
Beginning match attempt at character 0
1  f
2  fo
3  foo
4  foo
5  foobacktrack
6  f
7  fo
8  foo
9  foot
10 foot
11 foots
Match found in 11 steps
```

Sample:

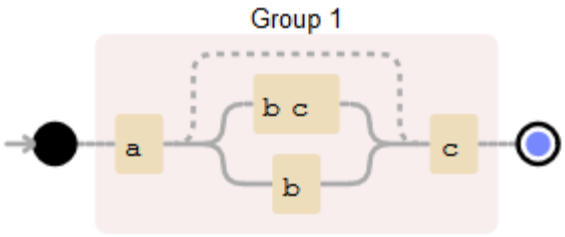
```
Match 1: foots
Group 1: foot
```

Tim Analysis:

The first alternative is tested and matches against 'foo', but because there is no 's' that follows immediately, it is allowed to backtrack and test against the second alternative which matches against 'foot' followed by the 's', satisfying the RegEx.

RegEx Manual Volume II

Understanding Atomic Groups with BRU cont.

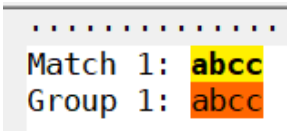


Without Quantifier but with Alternation –

Match: (a(?>bc|b)c)
Replace: \1 <space>

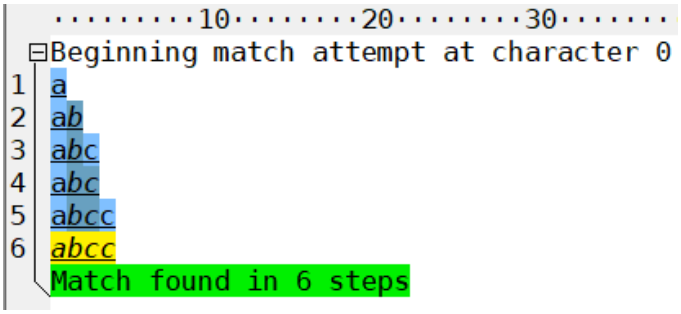
abc	abc
abcc	abcc

Sample:



This uses an Alternate for Capture Group 2:

This	OR	This
bc	|	b



Tim Analysis:

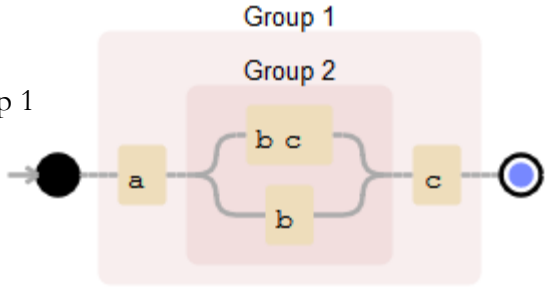
using 'abcc'

1. a	Match against a	Capture Group 1 = a
2. bc | b	Tests for 'bc' or 'c' = bc	Capture Group 1 = abc
3. c	Match against 'c'	Capture Group 1 = abcc

Why only one capture Group?

Because:

1. both the 'a' and 'c' match are within Capture Group 1
2. The Atomic Group holding the value 'bc' is also within Capture Group 1
3. Because the (?>bc|b) is atomic it is a Non-Marking Group.

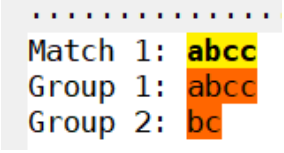


If you had this:

Match: (a(bc|b)c)
Replace: \1 <space>

abcc	abcc

Sample:



Where: \1 = abcc \2 = bc

Then the Alternate would be in its own Capture Group 2

RegEx Manual Volume II

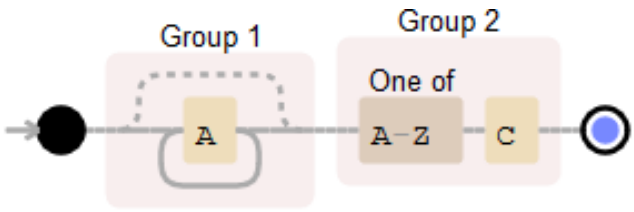
Understanding Atomic Groups with BRU cont.

With Quantifier –

String = AAC

Match: ((?>A+))([A-Z]C)
Replace: \1

AAC	AAC



Fails:

```
Beginning match attempt at character 0
1  AA
2  AA
3  AA
4  AAC
5  AACbacktrack
6  AAbacktrack
7  backtrack
8  backtrack
   Match attempt failed after 8 steps
Beginning match attempt at character 1
1  A
2  A
3  A
4  AC
5  ACbacktrack
6  Abacktrack
7  backtrack
8  backtrack
   Match attempt failed after 8 steps
```

Tim Analysis:

1. A	Look for an 'A'	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = AA
3. [A-Z]	Match against a class consisting of an uppercase letter = 'C'	Capture Group 2 = C
4. C	Match against 'C' = EOL = Fail	

After matching the 'AA' the engine matches the 'C' for the [A-Z], then attempts to match against the 'C' but Fails as the EOL has been reached. Because the group is atomic, it is unable to give up the second 'A', which would allow the rest of the pattern to match.

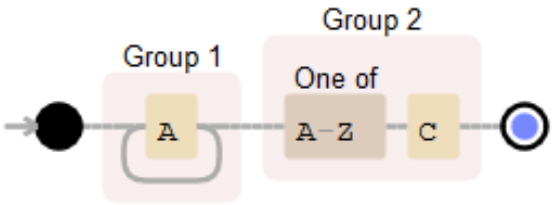
RegEx Manual Volume II

Understanding Atomic Groups with BRU cont.

With Quantifier – cont.

String = AAC

If you take the same RegEx and make it non Atomic, it will work:



Match: ((?:A+))([A-Z]C)
Replace: \1\2 <space>

Name	New Name
AAC	AAC

Sample:

```
·····Start·Length·····
Match 1: AAC      0      3
Group 1: A        0      1
Group 2: AC       1      2
```

Where:

\1 = A \2 = AC

Match: ((?:A+))([A-Z]C)

Result: **AAC**

```
Beginning match attempt at character 0
1  AA
2  AA
3  AA
4  AAC
5  AACbacktrack
6  AAbacktrack
7  A
8  A
9  A
10 AA
11 AAC
12 AAC
Match found in 12 steps
```

Tim Analysis:

1. A	Look for an 'A'	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = AA
3. [A-Z]	Match against a class consisting of an uppercase letter = 'C'	Capture Group 2 = C
4. C	Match against 'C' = False Backtracks to 2nd 'A'	
5.[A-Z]	Match against a class consisting of an uppercase letter = 'A'	Capture Group 2= A Changes Capture Group 1 = A
6. C	Match against 'C' = True	Capture Group 2 = AC

After matching the 'AA' the engine matches the 'C' for the [A-Z], then attempts to match against the 'C' but Fails as the EOL has been reached. It backtracks, matches against the second 'A' for the ([A-Z]) in Capture Group 2, reevaluating Group 1's value to 'A' (I call it reevaluating – most people refer to it as the Capture Group giving up the value), and matches the 'C'.

RegEx Manual Volume II

Understanding Atomic Groups with BRU cont.

With Quantifier – cont.

String = AAC

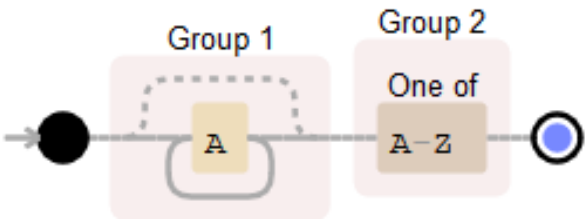
This will also match:

Match: ((?>A+))([A-Z])
Replace: \1 \2

AAC AA C

Match: ((?>A+))([A-Z])

Result: AAC



Sample:

```
.............Start·Length····
Match 1: AAC      0      3
Group 1: AA       0      2
Group 2: C        2      1
```

Tim Analysis:

1. A	Look for an 'A'	Capture Group 1 = A
2. +	Make it Greedy	Capture Group 1 = AA
3. [A-Z]	Match against a class consisting of an uppercase letter = 'C'	Capture Group 2 = C

By eliminating the match for the 'C', it will match at ([A-Z]). Basically, it is just another way of stating (A+([A-Z)) except using an Atomic Group.

```
Beginning match attempt at character 0
1 AA
2 AA
3 AA
4 AAC
5 AAC
Match found in 5 steps
```

RegEx Manual Volume II

Understanding Atomic Groups with BRU cont.

Changing the Atomic Group to a Possessive Quantifier –

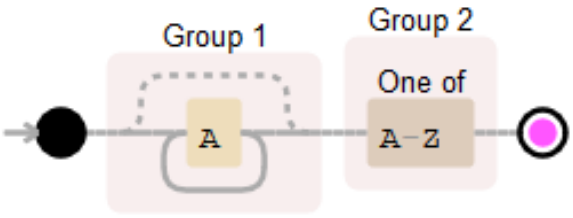
String = AACC

Match: (A++)([A-Z])
Replace: \1 \2

Match: (A++)([A-Z])

Result: **AAC**

Where: \1 = AA \2 = C



Sample:

```
Beginning match attempt at character 0
1  AA
2  AA
3  AAC
4  AAC
   Match found in 4 steps
```

```
              Start  Length
Match 1: AAC      0       3
Group 1: AA       0       2
Group 2: C        2       1
```

Tim Analysis:

1. A++	By adding the '+' to the Greedy Quantifier, '+', it makes the Quantifier 'Possessive'. This will be discussed fully in the next section. For now it Matches against the first 'A' and again on the second 'A' in the string.	Capture Group 1 = AA
2. [A-Z]	Tests the next character against a class consisting of an uppercase letter.	Capture Group 2 = C

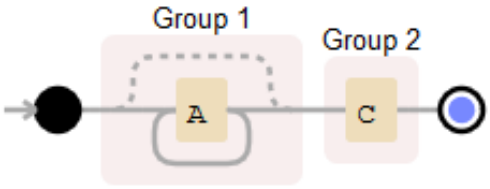
RegEx Manual Volume II

Understanding Atomic Groups with BRU cont.

Changing the Atomic Group to a Possessive Quantifier – cont.

String = AAC

This could also be written as:



Match: (A++)(C)
Replace: \1 \2

Match: (A++)(C)

Result: AAC

Where: \1 = AA \2 = C

Sample:

```
              Start  Length
Match 1: AAC      0       3
Group 1: AA       0       2
Group 2: C        2       1
```

```
Beginning match attempt at character 0
1  AA
2  AA
3  AAC
4  AAC
   Match found in 4 steps
```

When an atomic group only contains a sub-expression with a quantifier, an alternate syntax is a possessive quantifier, where a '+' is added to the quantifier.

Alternate Syntax: Possessive Quantifier:

? (?>A+) is equivalent to A++
? (?>A*) is equivalent to A*+
? (?>A?) is equivalent to A?+
? (?>A{…,…}) is equivalent to A{…,…}+

RegEx Manual Volume II

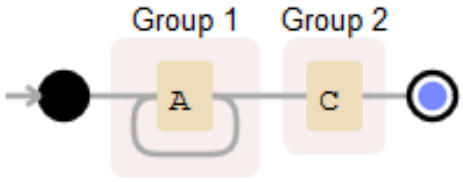
Understanding Atomic Groups with BRU cont.

Changing the Atomic Group to a Possessive Quantifier – cont.

String = AAC

As compared against the non-possessive Greedy Quantifer:

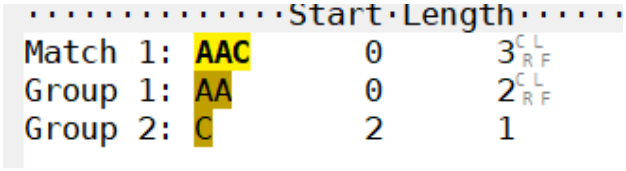
Match: (A+)([A-Z])
Replace: \1 \2

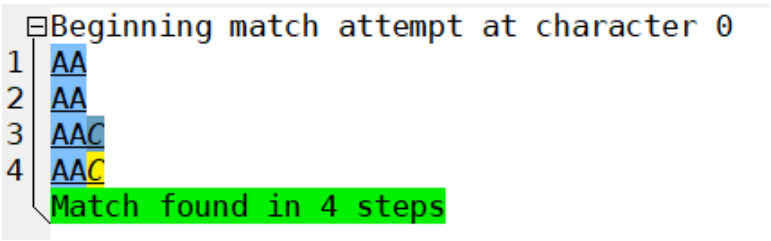


Where

\1 = AA \2 = C

Sample:





RegEx Manual Volume II

Examining the RegEx Engine Behavior On Quantifiers

A regular expression engine works by moving forward and backtracking. Essentially, it tests (eats, consumes,) a section of the string and attempts to match against the section. If it matches, great. If not, the engine alters its choice of the section of the string and attempts to match the RegEx against that section, and so on, until it's tried every possible choice. Each choice or attempt is referred to as a permutation.

The difference between greedy, reluctant, and possessive quantifiers becomes most evident when the engine is making its choices of what part of the string to attempt the match, and how to modify that choice if it doesn't work the first time. The rules are as follows:

Greedy

A greedy quantifier tells the engine to start with the entire string (or those parts of the string not previously matched) and test whether it matches.

If so, great; the engine can continue with the rest of the evaluation. If not, it tries again, by trimming (backtracking) one character (the last one) off the section of the string to be checked. If that doesn't work, it trims off another character, etc. So a greedy quantifier checks possible matches in order from longest to shortest.

Lazy (aka Non-Greedy, also called 'Reluctant')

A reluctant quantifier tells the engine to start with the shortest possible piece of the string. If it matches, the engine can continue; if not, it adds one character (moves forward one position) to the section of the string being checked and tries that, and so on until it finds a match or the entire string has been used up (Exhausted – meaning no more matches can be made, or EOL whichever comes first) . So a reluctant quantifier checks possible matches in the order from shortest to longest.

Possessive

A possessive quantifier is like a greedy quantifier on the first attempt: it tells the engine to start by checking the entire string. The difference is that if it doesn't work, the possessive quantifier reports that the match failed right then and there. The engine doesn't change the section of the string being looked at (move forward or backtrack), and it doesn't make any more attempts (no further permutations) .

I have already discussed Greedy and Lazy, but let's take a closer look at Possessive Quantifiers.

Source Material this section: David Z, Stackoverflow.com

RegEx Manual Volume II

Examining Possessive Quantifiers

Possessive Quantifiers can be used to replace traditional Greedy or Lazy quantifiers because they are more efficient. They do not backtrack and therefore run faster. This becomes important when you have a lot of files to process. Tweaking your RegEx in this way can greatly speed up the time required. Of course because they do not backtrack, they are not suitable for matches that require backtracking. That is why they are not used universally. That is the trade-off and that is why you have to tweak your RegEx to see if they will work for you.

Characteristics of Possessive Quantifiers:

- The match starts with the first unmatched character in the string.
- The possessive quantifier starts matching from left to right (only – no backtracking) one character at a time.
- The possessive quantifier matches as many characters as it can.
- Once a character has been matched, it remains in possession of the quantifier, even if this prevents other parts of the expression from matching (no backtracking).
- The quantifier is made possessive by appending a '+'

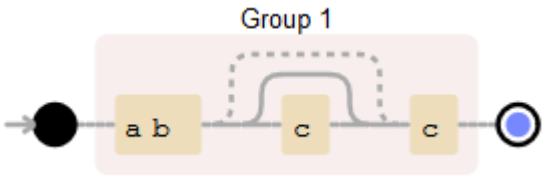
Possessive Quantifier:	Atomic Group Eqivalent to:	Non-Greedy Equivalent to:
?+	(?>A?)	?

Makes the preceding item optional. If the optional item can be matched, the quantifier won't give up its match even if the remainder of the RegEx Fails.

String = abcc

Match: (abc?+c)
Replace: \1 <space>

abc	abc
abcc	abcc

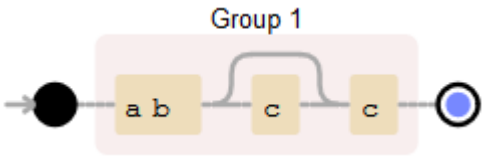


Match against 'abc' and then matches against the second 'a' in 'abcc'. In 'abc' it matches in the first part, but Fails when it cannot backtrack to match against the second 'a'.

This will match 'abc':

Match: (abc?c)
Replace: \1 <space>

abc	abc
abcc	abcc



By Making it non-possessive, it backtracks and matches against 'abc'.

RegEx Manual Volume II

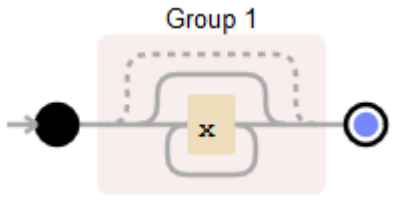
Examining Possessive Quantifiers cont.

Possessive Quantifier:	Atomic Group Eqivalent to:	Greedy Equivalent to:
Syntax: *+	(?>A*)	*

Repeats the previous item zero or more times. As many items as possible will be matched, without attempting any permutations with less matches even if the remainder of the RegEx Fails.

Match: (x*+)
Replace: \1

xbcxapav	x
xxbxx	xx
XXXXXcats	XXXXXcats
XXXXZ	XXXXZ
YYXXcatsXX	YYXXcatsXX



Tim Analysis:

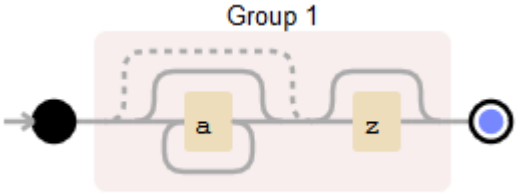
Using xxbxx

1. x	Match against the first 'x'	Capture Group 1 = x
2. *+	Make it Greedy Possessive	Capture Group 1 = xx

Another Example:

Match: (a*+z?)
Replace: \1 <space>

aaaaaaa	aaaaaaa
aaaaaz	aaaaaz
abcc	a



Tim Analysis:

Using aaaaaaa

By making the z lazy, it made it optional in the match.

1. a	Match against the first 'a'	Capture Group 1 = a
2. *+	Make it Greedy Possessive followed by … Current position EOL	Capture Group 1 = aaaaaaa
3. z	.. match against 'z' = EOL = Fail	
4. ?	Make it Lazy – last expression 'z' optional = True	Capture Group 1 = aaaaaaa

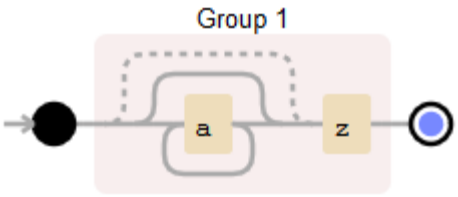
RegEx Manual Volume II

Examining Possessive Quantifiers cont.

If the z was not made lazy:

Match: (a*+z)
Replace: \1 <space>

aaaaaaa	aaaaaaa
aaaaaz	aaaaaz
abcc	abcc



Tim Analysis:

Using aaaaaaa

1. 1. a	Match against the first 'a'	Capture Group 1 = a
2. *+	Make it Greedy Possessive followed by … Current position EOL	Capture Group 1 = aaaaaaa
3. z	.. match against 'z' = EOL = Fail	

Using aaaaaz

1. 1. a	Match against the first 'a'	Capture Group 1 = a
2. *+	Make it Greedy Possessive followed by …	Capture Group 1 = aaaaa
3. z	.. match against 'z' = True	Capture Group 1 = aaaaaz

RegEx Manual Volume II

Examining Possessive Quantifiers cont.

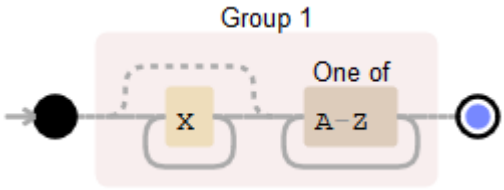
Possessive Quantifier:	Atomic Group Eqivalent to:	Greedy Equivalent to:
Syntax: ++	(?>A+)	+

Repeats the previous item once or more. As many items as possible will be matched, without trying any permutations with less matches even if the remainder of the RegEx Fails.

String = XXXXZ

Match: (X++[A-Z]+)
Replace: \1 <space>

XXXX	XXXX
XXXXZ	XXXXZ



Tim Analysis:

1. X	Match against the First 'X'	Capture Group 1= X
2. ++	Make it Greedy Possessive	Capture Group 1 = XXXX
3. [A-Z	Match against a class consisting of an uppercase letter = 'Z'	Capture Group 1 = XXXXZ
4. +	Make it Greedy	

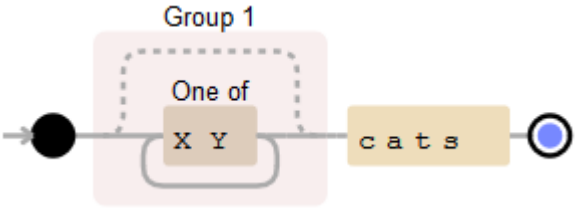
It failed in the string 'XXXX' because the possessive quantifier does not backtrack so after step 2, it cannot match against the [A-Z].

Another Example:

String = YYXXcatsXX

Match: ([XY]++)cats
Replace: \1

YYXXcatsXX	YYXX



Tim Analysis:

1. X	Match against a class consisting of either an uppercase 'X' or 'Y' = 'Y'	Capture Group 1= Y
2. ++	Make it Greedy Possessive followed by ...	Capture Group 1 = YYXX
3. cats	.. Match against a string 'cats' (Not Captured)	

RegEx Manual Volume II

Examining Possessive Quantifiers cont.

Possessive Quantifier:	Atomic Group Eqivalent to:	Range Quantifier Equivalent to:
Syntax: {n,m}+	(?>A{.. , ..})	{min, max}

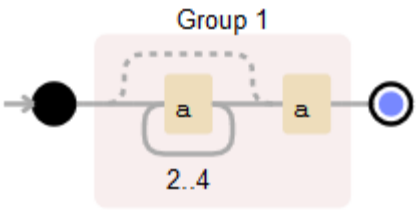
Where:
n >= 0 and m >= n

Repeats the previous item between n and m iterations. As many items as possible up to m will be matched, without trying any permutations with less matches even if the remainder of the regex Fails.

String = aaaaa

Match: (a{2,4}+a)
Replace: \1<space>

aaaa	aaaa
aaaaa	aaaaa



Tim Analysis:

Matches aaaaa but not aaaa

In the string 'aaaaa', It matches between 2 to 4 'a's. There are 5 a's in this string, so the first part matches. The second match of the 'a' matches because there is a 5th 'a' in the string, whereas in 'aaaa', there are only 4 a's, therefore it can't match against the final 'a' and Fails because it cannot backtrack.

RegEx Manual Volume II

Examining Possessive Quantifiers cont.

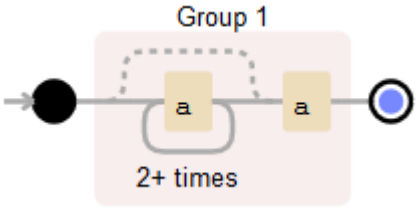
Possessive Quantifier:	Atomic Group Eqivalent to:	Range Quantifier Equivalent to:
Syntax: {n,}+	(?>A{.. ,}	{n,}

Where:
n >= 0

Repeats the previous item n or more iterations. As many items as possible will be matched, without trying any permutations with less matches even if the remainder of the regex Fails.

String = aaaaa

Match: (a{2,}+a)
Replace: \1<space>

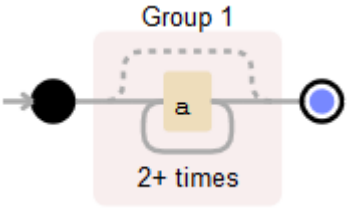


Tim Analysis:

Fails. Can never match anything. Because after the a{2,}+ (Match against 2 or more a's) it matches the entire string, but because it is possessive (the added '+') it cannot backtrack to match the final 'a'. This would also fail with any string that contained more than two a's – i.e. aaaaabbbbaaa.

This will match:

Match: (a{2,}+)
Replace: \1 <space>



aaaaa	aaaaa

By removing the 'a' at the end, it will match against 2 or more 'a's, matching against the entire string.

String = aaaabbbbaa

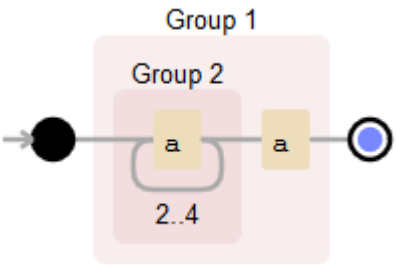
Match: (a{2,}+)
Replace: \1 <space>

aaaabbbbaa	aaaa

Matches against the first group of a's as you would expect.

As will this:

Match: ((a{2,4})a)
Replace: \1 <space>



aaaaa	aaaaa
aaaabbbbaa	aaaa

By making it non-possessive, it backtracks and matches against the additional 'a' of 'aaaaa'.

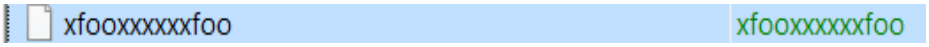
RegEx Manual Volume II

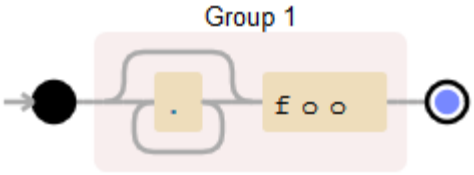
The Differences Between Greedy, Lazy and Possessive Affect on a Test String

String = xfooxxxxxxfoo

Greedy:

Match: (.*foo)
Replace: \1 <space>





Lazy:

Match: (.*?foo)
Replace: \1 <space>

Tim Notes:

1. Unfortunately, the Second program's Visualization doesn't discern the differences between Greed aand Lazy as the first program would but Possessive Quantifiers are not supported. I say unfortunate because the Second Visualization program provides more detail.

RegEx Manual Volume II

The Differences Between Greedy, Lazy and Possessive Affect on a Test String cont.

String = xfooxxxxxxfoo

Tim Analysis:

1. .	Match against any character	Capture Group 1 = x
2. *	Make it Greedy. Current position = EOL	Capture Group 1 = entire string
3. ?	But not too Greedy (Zero Occurrence Match) Made the .* optional. Backtracks.	Capture Group 1 = null
4. foo	Match against string 'foo' = True	Capture Group 1 = xfoo

```
Beginning match attempt at character 0
1 ok
2 backtrack
3 x
4 xf
5 xfo
6 xfoo
7 xfoo
Match found in 7 steps
Beginning match attempt at character 4
```

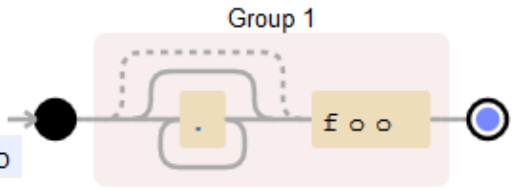
Why was the 'x' added back into Capture Group 1? Because it was reevaluated. Essentially, when it backtracked, it was at the first 'x' position in the string again, then matched against the x character with the <dot> and moved forward to match 'foo'.

Technically speaking, it will repeat this same algorithm against the remainder of the string, and also match the 'xxxxxxfoo' as well, but as far as BRU is concerned, it will only capture the first match.

Possessive:

Match: (.*+foo)
Replace: \1 <space>

xfooxxxxxxfoo xfooxxxxxxfoo



Tim Analysis:

Fails because after matching the entire string in the first part, the current position becomes EOL, and because it is possessive, the RegEx Engine cannot backtrack to match against 'foo'.

Regular Expressions (RegEx) Manual Volume II

THE ~ END

BRU JavaScript Reference Manual

Researched and Edited by Timothy R. Mongeon

BRU JavaScript Reference Manual

The following has to do with some of BRU's implementation of JavaScript. The examples presented are for your understanding of how a method or property functions. The examples won't mean much in practicality, but this is only to illustrate how they work and a simple method of implementing them in BRU.

BRU JavaScript Reference Manual

Understanding the JavaScript Code Entry Form Color Coding

The color coding system used is to help with syntax and errors (color is as close as I can get).

Within function

Item	Color	Correct	Bad	Neutral
Keywords	blue	*	Black & White	
strings	Brown-red	*	“	
Variables	Black & White			*
Numerals	Black & White			*
Function name	Black & White			*

Outside Function

Item	Color	Correct	Bad	Neutral
Variables	purple	*		

All

Item	Color	Correct	Bad	Neutral
Operators	silver-green	*		
parenthese/brackets	silver-green	*		
Braces, Semicolon				

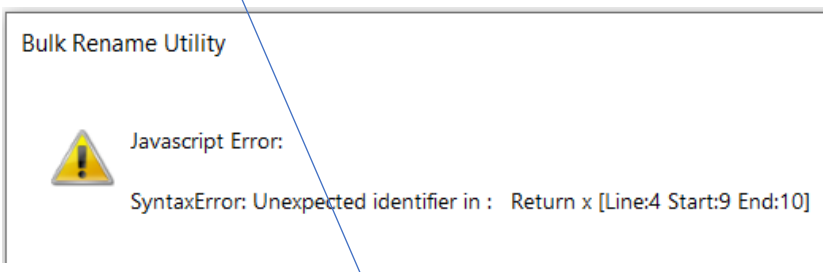
Avoid Common Errors: CASE SENSITIVITY

Many times entering the script into the JavaScript Code Entry form I encounter errors:

Javascript Code:

```
1 function myFunction() {
2   var y = 5;
3   var x = y + 2;
4   Return x
5 }
6
7 newName = myFunction();
8
9
```

One of the most common is this:



Why? Because I capitalized a keyword and most JavaScript language is in Lowercase and Case Sensitive. One of the clues is that the word in error is in black and white and should be blue or some other color depending on the keyword.

BRU JavaScript Reference Manual

Unicode

JavaScript has no ASCII strings, they're intrinsically UTF-based.

The Unicode Character Sets

Character-set	Description
UTF-8	A character in UTF8 can be from 1 to 4 bytes long. UTF-8 can represent any character in the Unicode standard. UTF-8 is backwards compatible with ASCII.
UTF-16	16-bit Unicode Transformation Format is a variable-length character encoding for Unicode, capable of encoding the entire Unicode repertoire.

ASCII

The first 128 characters of Unicode correspond one-to-one with ASCII with the same binary value as ASCII, making valid ASCII text valid UTF-8-encoded Unicode as well.

What this means for instance is that UNICODE 72 is the same as ASCII 72 or 'H'

The following Charts display the most common Unicode characters.

source this Section: Stackoverflow, w3schoolscom (Fair Use)

Unicode UTF-8 Chart (For JavaScript)

Control Characters

Char	Dec	Name
NUL	00	null character
SOH	01	start of header
STX	02	start of text
ETX	03	end of text
EOT	04	end of transmission
ENQ	05	enquiry
ACK	06	acknowledge
BEL	07	bell (ring)
BS	08	backspace
HT	09	horizontal tab
LF	10	line feed
VT	11	vertical tab
FF	12	form feed
CR	13	carriage return
SO	14	shift out
SI	15	shift in
DLE	16	data link escape
DC1	17	device control 1
DC2	18	device control 2
DC3	19	device control 3
DC4	20	device control 4
NAK	21	negative acknowledge
SYN	22	synchronize
ETB	23	end transmission block
CAN	24	cancel
EM	25	end of medium
SUB	26	substitute
ESC	27	escape
FS	28	file separator
GS	29	group separator
RS	30	record separator
US	31	unit separator

Tim Notes:

1. Many Control Characters may not be viewable or reproducible.

Unicode UTF-8 Chart (For JavaScript)

Latin Basic

Char	Dec	Hex	Name
	32	0020	SPACE
!	33	0021	EXCLAMATION MARK
"	34	0022	QUOTATION MARK
#	35	0023	NUMBER SIGN
$	36	0024	DOLLAR SIGN
%	37	0025	PERCENT SIGN
&	38	0026	AMPERSAND
'	39	0027	APOSTROPHE
(	40	0028	LEFT PARENTHESIS
)	41	0029	RIGHT PARENTHESIS
*	42	002A	ASTERISK
+	43	002B	PLUS SIGN
,	44	002C	COMMA
-	45	002D	HYPHEN-MINUS
.	46	002E	FULL STOP
/	47	002F	SOLIDUS
0	48	0030	DIGIT ZERO
1	49	0031	DIGIT ONE
2	50	0032	DIGIT TWO
3	51	0033	DIGIT THREE
4	52	0034	DIGIT FOUR
5	53	0035	DIGIT FIVE
6	54	0036	DIGIT SIX
7	55	0037	DIGIT SEVEN
8	56	0038	DIGIT EIGHT
9	57	0039	DIGIT NINE
:	58	003A	COLON
;	59	003B	SEMICOLON
<	60	003C	LESS-THAN SIGN
=	61	003D	EQUALS SIGN
>	62	003E	GREATER-THAN SIGN
?	63	003F	QUESTION MARK
@	64	0040	COMMERCIAL AT
A	65	0041	LATIN CAPITAL LETTER A
B	66	0042	LATIN CAPITAL LETTER B
C	67	0043	LATIN CAPITAL LETTER C
D	68	0044	LATIN CAPITAL LETTER D
E	69	0045	LATIN CAPITAL LETTER E
F	70	0046	LATIN CAPITAL LETTER F
G	71	0047	LATIN CAPITAL LETTER G
H	72	0048	LATIN CAPITAL LETTER H
I	73	0049	LATIN CAPITAL LETTER I
J	74	004A	LATIN CAPITAL LETTER J
K	75	004B	LATIN CAPITAL LETTER K
L	76	004C	LATIN CAPITAL LETTER L

Unicode UTF-8 Chart (For JavaScript)

Latin Basic

Char	Dec	Hex	Name
M	77	004D	LATIN CAPITAL LETTER M
N	78	004E	LATIN CAPITAL LETTER N
O	79	004F	LATIN CAPITAL LETTER O
P	80	0050	LATIN CAPITAL LETTER P
Q	81	0051	LATIN CAPITAL LETTER Q
R	82	0052	LATIN CAPITAL LETTER R
S	83	0053	LATIN CAPITAL LETTER S
T	84	0054	LATIN CAPITAL LETTER T
U	85	0055	LATIN CAPITAL LETTER U
V	86	0056	LATIN CAPITAL LETTER V
W	87	0057	LATIN CAPITAL LETTER W
X	88	0058	LATIN CAPITAL LETTER X
Y	89	0059	LATIN CAPITAL LETTER Y
Z	90	005A	LATIN CAPITAL LETTER Z
[	91	005B	LEFT SQUARE BRACKET
\	92	005C	REVERSE SOLIDUS
]	93	005D	RIGHT SQUARE BRACKET
^	94	005E	CIRCUMFLEX ACCENT
_	95	005F	LOW LINE
`	96	0060	GRAVE ACCENT
a	97	0061	LATIN SMALL LETTER A
b	98	0062	LATIN SMALL LETTER B
c	99	0063	LATIN SMALL LETTER C
d	100	0064	LATIN SMALL LETTER D
e	101	0065	LATIN SMALL LETTER E
f	102	0066	LATIN SMALL LETTER F
g	103	0067	LATIN SMALL LETTER G
h	104	0068	LATIN SMALL LETTER H
i	105	0069	LATIN SMALL LETTER I
j	106	006A	LATIN SMALL LETTER J
k	107	006B	LATIN SMALL LETTER K
l	108	006C	LATIN SMALL LETTER L
m	109	006D	LATIN SMALL LETTER M
n	110	006E	LATIN SMALL LETTER N
o	111	006F	LATIN SMALL LETTER O
p	112	0070	LATIN SMALL LETTER P
q	113	0071	LATIN SMALL LETTER Q
r	114	0072	LATIN SMALL LETTER R
s	115	0073	LATIN SMALL LETTER S
t	116	0074	LATIN SMALL LETTER T
u	117	0075	LATIN SMALL LETTER U
v	118	0076	LATIN SMALL LETTER V
w	119	0077	LATIN SMALL LETTER W
x	120	0078	LATIN SMALL LETTER X
y	121	0079	LATIN SMALL LETTER Y

Unicode UTF-8 Chart (For JavaScript)

Latin Basic

Char	Dec	Hex	Name
z	122	007A	LATIN SMALL LETTER Z
{	123	007B	LEFT CURLY BRACKET
|	124	007C	VERTICAL LINE
}	125	007D	RIGHT CURLY BRACKET

Unicode UTF-8 Chart (For JavaScript)

Control Characters

Char	Dec	Hex	Name
~	126	007E	TILDE
DEL	127		delete (rubout)
€	128	0080	CONTROL
	129	0081	CONTROL
‚	130	0082	BREAK PERMITTED HERE
ƒ	131	0083	NO BREAK HERE
„	132	0084	INDEX
…	133	0085	NEXT LINE (NEL)
†	134	0086	START OF SELECTED AREA
‡	135	0087	END OF SELECTED AREA
ˆ	136	0088	CHARACTER TABULATION SET
‰	137	0089	CHARACTER TABULATION WITH JUSTIFICATION
Š	138	008A	LINE TABULATION SET
‹	139	008B	PARTIAL LINE FORWARD
Œ	140	008C	PARTIAL LINE BACKWARD
	141	008D	REVERSE LINE FEED
Ž	142	008E	SINGLE SHIFT TWO
	143	008F	SINGLE SHIFT THREE
	144	0090	DEVICE CONTROL STRING
‘	145	0091	PRIVATE USE ONE
’	146	0092	PRIVATE USE TWO
“	147	0093	SET TRANSMIT STATE
”	148	0094	CANCEL CHARACTER
•	149	0095	MESSAGE WAITING
–	150	0096	START OF GUARDED AREA
—	151	0097	END OF GUARDED AREA
˜	152	0098	START OF STRING
™	153	0099	CONTROL
š	154	009A	SINGLE CHARACTER INTRODUCER
›	155	009B	CONTROL SEQUENCE INTRODUCER
œ	156	009C	STRING TERMINATOR
	157	009D	OPERATING SYSTEM COMMAND
ž	158	009E	PRIVACY MESSAGE
Ÿ	159	009F	APPLICATION PROGRAM COMMAND

Tim Notes:

1. Many Control Characters may not be viewable or reproducible.

Unicode UTF-8 Chart (For JavaScript)

Latin Supplement

Char	Dec	Hex	Name
	160	00A0	NO-BREAK SPACE
¡	161	00A1	NVERTED EXCLAMATION MARK
¢	162	00A2	CENT SIGN
£	163	00A3	POUND SIGN
¤	164	00A4	CURRENCY SIGN
¥	165	00A5	YEN SIGN
¦	166	00A6	BROKEN BAR
§	167	00A7	SECTION SIGN
¨	168	00A8	DIAERESIS
©	169	00A9	COPYRIGHT SIGN
ª	170	00AA	FEMININE ORDINAL INDICATOR
«	171	00AB	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
¬	172	00AC	NOT SIGN
-	173	00AD	SOFT HYPHEN
®	174	00AE	REGISTERED SIGN
¯	175	00AF	MACRON
°	176	00B0	DEGREE SIGN
±	177	00B1	PLUS-MINUS SIGN
²	178	00B2	SUPERSCRIPT TWO
³	179	00B3	SUPERSCRIPT THREE
´	180	00B4	ACUTE ACCENT
µ	181	00B5	MICRO SIGN
¶	182	00B6	PILCROW SIGN
·	183	00B7	MIDDLE DOT
¸	184	00B8	CEDILLA
¹	185	00B9	SUPERSCRIPT ONE
º	186	00BA	MASCULINE ORDINAL INDICATOR
»	187	00BB	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
¼	188	00BC	VULGAR FRACTION ONE QUARTER
½	189	00BD	VULGAR FRACTION ONE HALF
¾	190	00BE	VULGAR FRACTION THREE QUARTERS
¿	191	00BF	INVERTED QUESTION MARK
À	192	00C0	LATIN CAPITAL LETTER A WITH GRAVE
Á	193	00C1	LATIN CAPITAL LETTER A WITH ACUTE
Â	194	00C2	LATIN CAPITAL LETTER A WITH CIRCUMFLEX
Ã	195	00C3	LATIN CAPITAL LETTER A WITH TILDE
Ä	196	00C4	LATIN CAPITAL LETTER A WITH DIAERESIS
Å	197	00C5	LATIN CAPITAL LETTER A WITH RING ABOVE
Æ	198	00C6	LATIN CAPITAL LETTER AE
Ç	199	00C7	LATIN CAPITAL LETTER C WITH CEDILLA
È	200	00C8	LATIN CAPITAL LETTER E WITH GRAVE
É	201	00C9	LATIN CAPITAL LETTER E WITH ACUTE
Ê	202	00CA	LATIN CAPITAL LETTER E WITH CIRCUMFLEX
Ë	203	00CB	LATIN CAPITAL LETTER E WITH DIAERESIS
Ì	204	00CC	LATIN CAPITAL LETTER I WITH GRAVE

Unicode UTF-8 Chart (For JavaScript)

Latin Supplement

Char	Dec	Hex	Name
Í	205	00CD	LATIN CAPITAL LETTER I WITH ACUTE
Î	206	00CE	LATIN CAPITAL LETTER I WITH CIRCUMFLEX
Ï	207	00CF	LATIN CAPITAL LETTER I WITH DIAERESIS
Ð	208	00D0	LATIN CAPITAL LETTER ETH
Ñ	209	00D1	LATIN CAPITAL LETTER N WITH TILDE
Ò	210	00D2	LATIN CAPITAL LETTER O WITH GRAVE
Ó	211	00D3	LATIN CAPITAL LETTER O WITH ACUTE
Ô	212	00D4	LATIN CAPITAL LETTER O WITH CIRCUMFLEX
Õ	213	00D5	LATIN CAPITAL LETTER O WITH TILDE
Ö	214	00D6	LATIN CAPITAL LETTER O WITH DIAERESIS
×	215	00D7	MULTIPLICATION SIGN
Ø	216	00D8	LATIN CAPITAL LETTER O WITH STROKE
Ù	217	00D9	LATIN CAPITAL LETTER U WITH GRAVE
Ú	218	00DA	LATIN CAPITAL LETTER U WITH ACUTE
Û	219	00DB	LATIN CAPITAL LETTER U WITH CIRCUMFLEX
Ü	220	00DC	LATIN CAPITAL LETTER U WITH DIAERESIS
Ý	221	00DD	LATIN CAPITAL LETTER Y WITH ACUTE
Þ	222	00DE	LATIN CAPITAL LETTER THORN
ß	223	00DF	LATIN SMALL LETTER SHARP S
à	224	00E0	LATIN SMALL LETTER A WITH GRAVE
á	225	00E1	LATIN SMALL LETTER A WITH ACUTE
â	226	00E2	LATIN SMALL LETTER A WITH CIRCUMFLEX
ã	227	00E3	LATIN SMALL LETTER A WITH TILDE
ä	228	00E4	LATIN SMALL LETTER A WITH DIAERESIS
å	229	00E5	LATIN SMALL LETTER A WITH RING ABOVE
æ	230	00E6	LATIN SMALL LETTER AE
ç	231	00E7	LATIN SMALL LETTER C WITH CEDILLA
è	232	00E8	LATIN SMALL LETTER E WITH GRAVE
é	233	00E9	LATIN SMALL LETTER E WITH ACUTE
ê	234	00EA	LATIN SMALL LETTER E WITH CIRCUMFLEX
ë	235	00EB	LATIN SMALL LETTER E WITH DIAERESIS
ì	236	00EC	LATIN SMALL LETTER I WITH GRAVE
í	237	00ED	LATIN SMALL LETTER I WITH ACUTE
î	238	00EE	LATIN SMALL LETTER I WITH CIRCUMFLEX
ï	239	00EF	LATIN SMALL LETTER I WITH DIAERESIS
ð	240	00F0	LATIN SMALL LETTER ETH
ñ	241	00F1	LATIN SMALL LETTER N WITH TILDE
ò	242	00F2	LATIN SMALL LETTER O WITH GRAVE
ó	243	00F3	LATIN SMALL LETTER O WITH ACUTE
ô	244	00F4	LATIN SMALL LETTER O WITH CIRCUMFLEX
õ	245	00F5	LATIN SMALL LETTER O WITH TILDE
ö	246	00F6	LATIN SMALL LETTER O WITH DIAERESIS
÷	247	00F7	DIVISION SIGN
ø	248	00F8	LATIN SMALL LETTER O WITH STROKE
ù	249	00F9	LATIN SMALL LETTER U WITH GRAVE

Unicode UTF-8 Chart (For JavaScript)

Latin Supplement

Char	Dec	Hex	Name
ú	250	00FA	LATIN SMALL LETTER U WITH ACUTE
û	251	00FB	LATIN SMALL LETTER U WITH CIRCUMFLEX
ü	252	00FC	LATIN SMALL LETTER U WITH DIAERESIS
ý	253	00FD	LATIN SMALL LETTER Y WITH ACUTE
þ	254	00FE	LATIN SMALL LETTER THORN
ÿ	255	00FF	LATIN SMALL LETTER Y WITH DIAERESIS

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

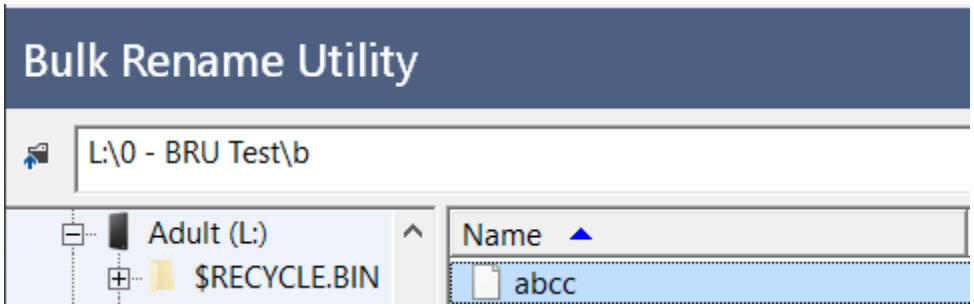
Constants

origName

```
origName
```

This constant contains the original name of the object. This is before any Bulk Rename Utility order of operations have been performed as specified by the renaming criteria sections 1 to 13 and all section 14 specials.

This is the equivalent of Name in BRU.



origExt

```
origExt
```

This constant contains the original extension of the object. This is before any Bulk Rename Utility order of operations have been performed as specified by the renaming criteria sections 1 to 13 and all section 14 specials.

ext

```
ext
```

This constant contains the extension of the object (i.e. file or folder name) being processed. This is after any Bulk Rename Utility order of operations have been applied as specified by the renaming criteria sections 1 to 13 and all section 14 specials.

If the extension has not been modified, then ext will be the same as the value of the constant, origExt.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Constants

name

```
name
```

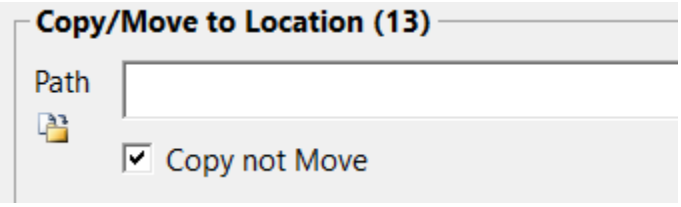
This constant contains the name of the object (i.e. file or folder name) being processed. This is after any Bulk Rename Utility order of operations have been applied as specified by the renaming criteria sections 1 to 13 and all section 14 specials.

If the original name of the object has not been modified, then name will be the same as the value of the constant, origName.

location

```
location
```

This constant contains the location of the object as specified in 'Section #13: Copy/Move To Location'.



counter

```
counter
```

A counter that is incremented during the renaming operation, starting from 1. To retrieve the Bulk Rename Utility 'Autonumber' as used in 'Section #10: Numbering' , use the BRU object function.

BRU JavaScript Reference Manual

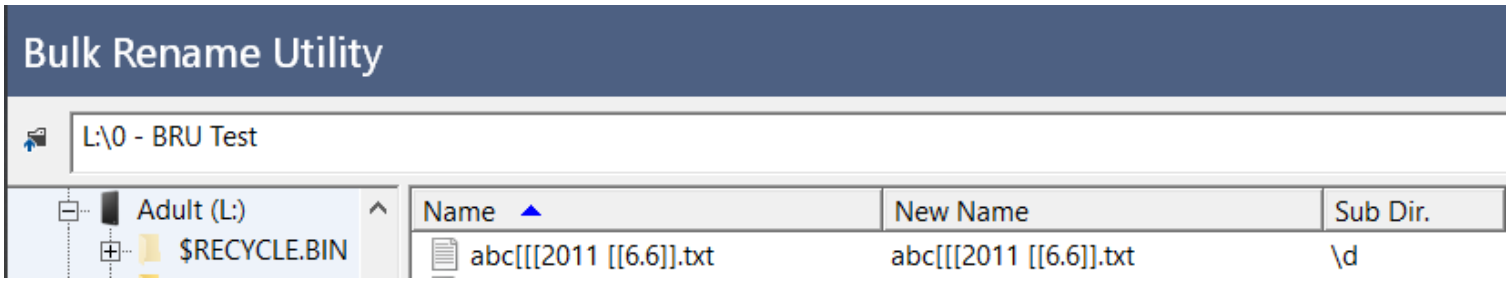
BRU Specific JavaScript Features

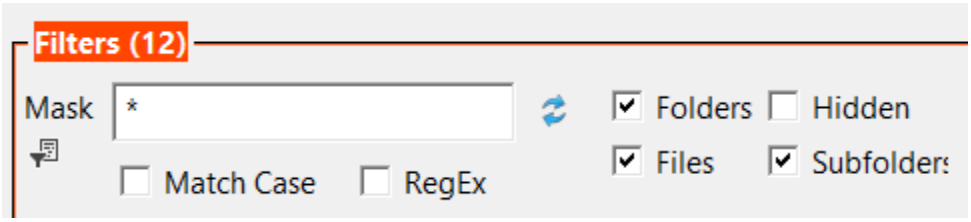
Constants

subDir

subDir

This constant contains the object sub directory (it matches the subdir column in the file list). It is only applicable if Subfolders (recursion) have been enabled in 'Section #12: Filters'.





By enabling Subfolders, it turns on recursion and also makes the Subdir Column data available in the Content Pane.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

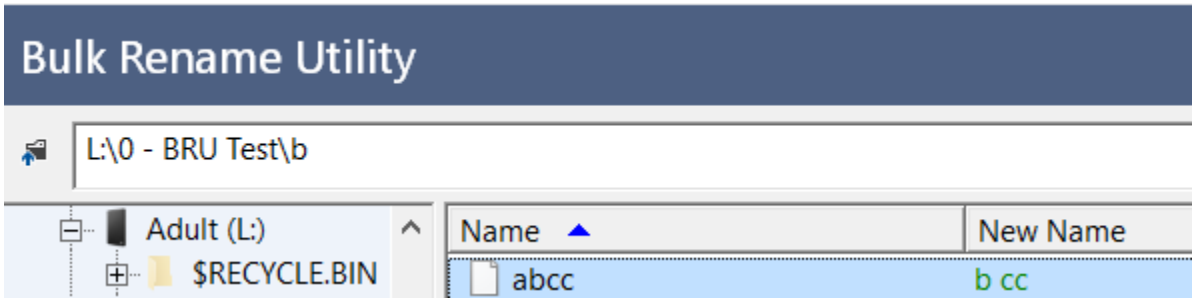
Variables

newName

newName

This variable contains the new name to be applied to the object. Modify this variable to modify the object name. If you do not modify the variable newName, then the JavaScript code has no effect on the name of an object.

newName is the equivalent of New Name in BRU. New Name must show a change (in green) for the renaming function to take place.



newExt

newExt

This variable contains the new extension to be applied to the object. Modify this variable to modify the object extension. If you do not modify the variable newExt, then the JavaScript code has no effect on the extension of an object.

BRU JavaScript Reference Manual

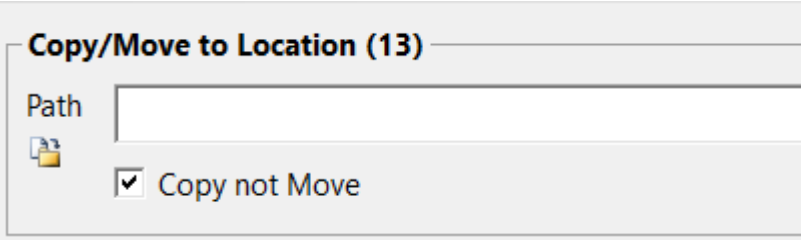
BRU Specific JavaScript Features

Variables

newLocation

newLocation

This variable contains any modified value of the 'Section #13: Copy/Move to Location' criteria for the object. If you do not modify this variable, then the JavaScript code has no effect. In other words, newLocation will only contain a value if 'Section #13: Copy/Move to Location' was modified.



BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

object("value")

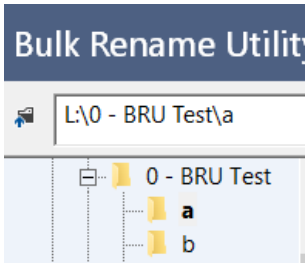
```
object("value")
```

This function returns specific values for the object being processed.

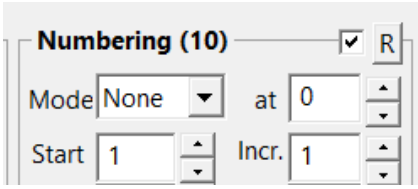
Where:

value =

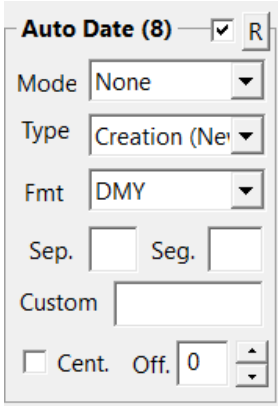
"folder" returns the folder for the current object being processed.



"autonumber" returns the autonumber for the current object being processed, taken from 'Section #10: Numbering'



"autodate" returns the autodate for the current object being processed, taken from 'Section #8: Auto Date'



BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

object("value") cont.

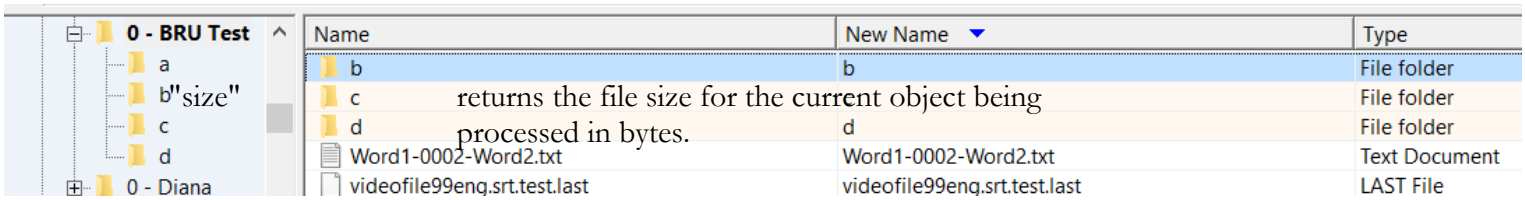
```
object("value")
```

Where:

value =

The following values correspond with Column Titles in the Content Pane:

"isdir" returns true if the current object being processed is a directory. Corresponds to the 'Type' column only as far as identification as a File Folder is concerned.



"size" returns the file size for the current object being processed in bytes.

"modified" returns the modified timestamp for the current object being processed
"created" returns the created timestamp for the current object being processed.
"accessed" returns the accessed timestamp for the current object being processed.

All three of these values correspond to the columns of the same name.

Created	Modified	Accessed
4/15/2020 11:29:59 AM	6/29/2020 8:01:35 PM	6/29/2020 8:01:35 PM

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Substitution Tags

exif("value")

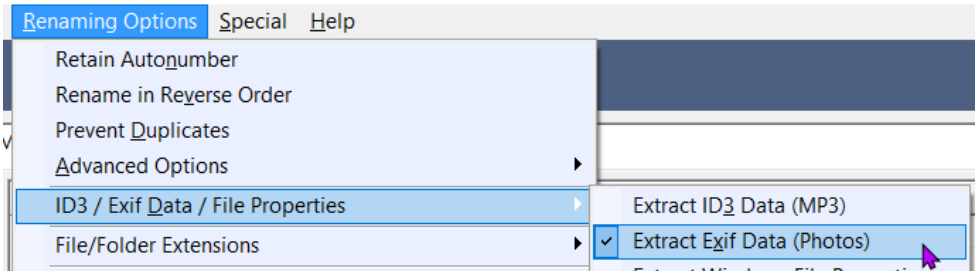
```
exif("value")
```

This function returns EXIF values for the object being processed. These refer to the substitution tags used in 'Section #7: Add', as well as Date/Time Taken.

Where:

value =

"%a"	Aperture
"%c"	Comments
"%e"	Exposure
"%f"	Focal Length
"%xb"	Exposure Bias
"%ma"	Camera Make
"%mo"	Camera Model



In BRU, would normally need to enable, 'Extract Exif Data (Photos)'. It is not specified that this is required under JavaScript and I would have to presume that no mention otherwise would mean that it is not required.

In addition, the following values are recognized for 'Date/Time Taken', not to be confused with %d, 'Day of Month' under Custom Date Formats of, 'Section #8: Auto Date'. These particular values do not correspond with any substitution tags in BRU.

"%d"	Date/Time Taken
"taken"	Date/Time Taken, same as %d

Example:

Javascript Code:

```
1 newName = exif("taken")
```

Returns: (Invalid in red because of the colon character)

Name	New Name	Taken (Original)
DSCN0689.JPG	Sun Mar 15 2015 15:06:17 GMT-0400 (Eastern Daylight Time)	3/15/2015 3:06:17 PM
RSCN0514.JPG	Tue Jan 01 2013 14:38:27 GMT-0500 (Eastern Standard Time)	1/1/2013 2:38:27 PM
FSCN0517.JPG	Tue Jan 01 2013 14:47:20 GMT-0500 (Eastern Standard Time)	1/1/2013 2:47:20 PM
DSCN0688.JPG	Sat Mar 14 2015 17:12:47 GMT-0400 (Eastern Daylight Time)	3/14/2015 5:12:47 PM

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

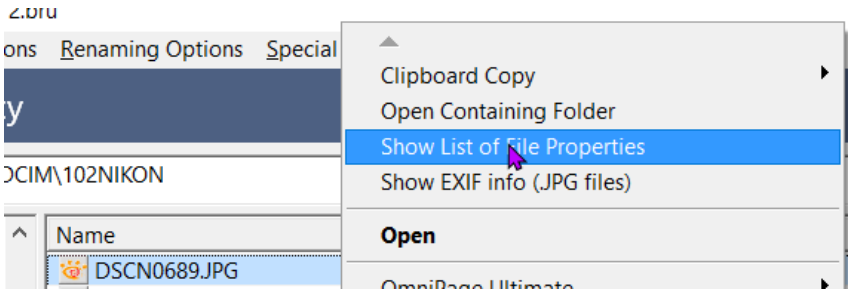
Windows File Properties

fileProperty("name") or fileProperty("label")

fileProperty("name") fileProperty("label")

Every file has metadata values that are assigned through Windows. These are called 'Windows Properties', not to be confused with other metadata, i.e. ID3 and Exif. These functions return the Windows File Property as a text string either by 'Name' or 'Label'. If the property is not found, the function will return null.

To see the available properties for a file, right click and select, 'Show List of File Properties' from the context menu:



Properties

Label	Value	Name
Folder name	102NIKON	System.ItemFolderNameDisplay
Type	JPG File	System.ItemTypeText
Name	DSCN0689.JPG	System.ItemNameDisplay
Size	906 KB	System.Size
Attributes	A	System.FileAttributes
Date modified	3/15/2015 4:06 PM	System.DateModified
Date created	3/15/2015 4:06 PM	System.DateCreated
Date accessed	8/7/2017 1:00 AM	System.DateAccessed
	DSCN0689	System.ItemNameDisplayWithoutExtension
Content type	image/jpeg	System.ContentType
Title		System.Title

Total Count: 108

This brings up the Windows Properties for the selected file.

As you can see there can be quite a lot of File Properties associated with one file. This file has 108, and they vary by file.

Here you can see the 'Label' and 'Name' properties and the current value. Note that they share one value meaning that regardless if you specify the 'Name' or 'Label' for a particular property, it returns the same value.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Windows File Properties cont.

fileProperty("name") or fileProperty("label") cont.

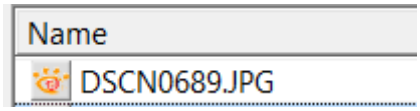
fileProperty("name")	fileProperty("label")

Where:

"name" or "label" correspond to a property found in the Windows Property List for a particular file

i.e.,

Using the same file as previous,



Label	Value
Folder name	102NIKON

Value	Name
102NIKON	System.ItemFolderNameDisplay

fileProperty("Folder name") Using the "label'

or

fileProperty("System.ItemFolderNameDisplay") Using the "name"

Returns:

102NIKON

Note: It doesn't matter what the file type is, this example just happened to be for an image type.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Windows File Properties cont.

fileProperty("name") or fileProperty("label") cont.

fileProperty("name") fileProperty("label")

In addition, this JavaScript function can also be used to return the Exif Properties for a file. The complete list of Exif Metadata can be found in Volume I in the Reference Section. If the property is not found, the function will return null.

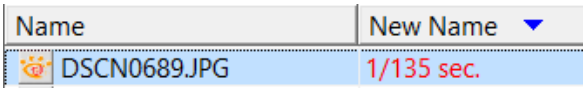
The format is:

fileProperty("exif:<property>")

i.e.,

```
newName = fileProperty("System.Photo.ExposureTime")
```

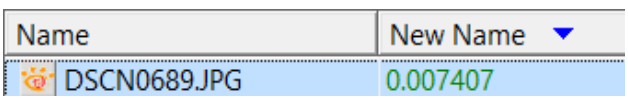
Returns:



In the Windows File Properties list for that file, this corresponds to:

```
newName = fileProperty("Exif:ExposureTime")
```

Returns:



In the Exif Metdata Reference, this is listed as:

Exif:ExposureTime - Exposure time, given in seconds.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Windows File Properties "Date"

filePropertyDate("<date property>")

```
filePropertyDate("<date property>")
```

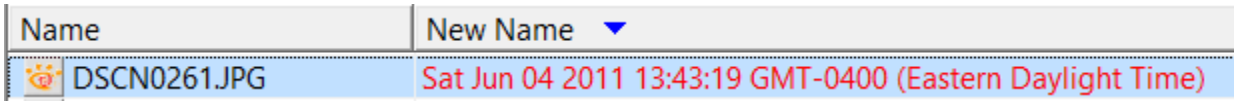
By adding the "Date" to the fileProperty, it defaults to using JavaScript's format of providing the date in full written text string format rather than just as a series of numbers.

i.e.,

With Date:

filePropertyDate("exif:DateTimeOriginal")

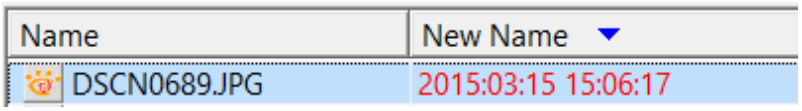
```
newName = filePropertyDate("exif:DateTimeOriginal")
```



Without Date:

fileProperty("exif:DateTimeOriginal")

```
newName = fileProperty("exif:DateTimeOriginal")
```



Notes:

1. Date can be used with any date Exif or Windows File Property.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Windows File Properties "Date" cont.

filePropertyDate("<date property>") cont.

```
filePropertyDate("<date property>")
```

More examples:

```
newName = filePropertyDate("System.Document.DateCreated")
```

Name	New Name	Taken (Original)	Created
DSCN0689.JPG	Sun Mar 15 2015 16:06:00 GMT-0400 (Eastern Daylight Time)	3/15/2015 3:06:17 PM	3/15/2015 4:06:16 PM

```
newName = filePropertyDate("exif:DateTimeOriginal")
```

Name	New Name	Taken (Original)
DSCN0261.JPG	Sat Jun 04 2011 13:43:19 GMT-0400 (Eastern Daylight Time)	6/4/2011 1:43:19 PM

```
newName = filePropertyDate("System.DateAccessed")
```

Name	New Name	Accessed
DSCN0261.JPG	Mon Aug 07 2017 01:00:00 GMT-0400 (Eastern Daylight Time)	8/7/2017 1:00:00 AM

Notes:

1. Invalid (red) because of the colon character. The only way to remove this is to use the individual JavaScript Date and Time methods to extract the parts {i.e. get Month() }, and piece it back togther using the delimiter you want. As it stands now, it won't be renamed.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Windows File Properties "Num"

filePropertyNum("<property>")

```
filePropertyNum("<property>")
```

By appending the "Num" to the fileProperty, it converts any numeric string data into a numeric and returns only those values disregarding any text or symbols up until a <space> character.

i.e.,

```
newName = fileProperty("exif:DateTimeOriginal")
```

Returned as a String -

Name	New Name	Taken (Original)
DSCN0261.JPG	2011:06:04 13:43:19	6/4/2011 1:43:19 PM

```
newName = filePropertyNum("exif:DateTimeOriginal")
```

Returned as a Numeric Value (time portion not returned because of the preceding <space>) –

Name	New Name	Taken (Original)
DSCN0261.JPG	20110604	6/4/2011 1:43:19 PM

Because it is numeric data, it can be treated as such (20110604 + 2 = 20110606):

```
newName = filePropertyNum("exif:DateTimeOriginal") + 2
```

Name	New Name	Taken (Original)
DSCN0261.JPG	20110606	6/4/2011 1:43:19 PM

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

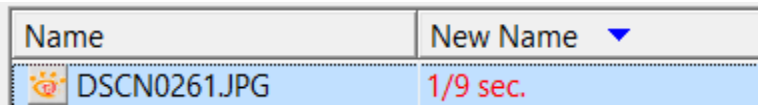
Windows File Properties "Num" cont.

filePropertyNum("<property>") **cont.**

```
fileProperty Num("<property>")
```

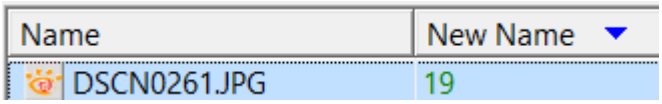
In this example, System.Photo.ExposureTime property, taken from the Exif Reference Manual of Volume I, returns the exposure time of the original file as it was created. The value returned is 1/9 of a second.

```
newName = fileProperty("System.Photo.ExposureTime")
```



With the appending of 'Num' to the fileProperty, the function returns the numeric value of the System.Photo.ExposureTime property.

```
newName = fileProperty Num("System.Photo.ExposureTime")
```



The return of values is not a conversion of the PhotoExposureTime into it's numeric equivalent, but instead, an extraction of the numeric data only. Thus '1/9 sec.' is extracted as '1' and '9' returning '19' and not 0.111111111111, the numeric equivalent derived from 1 divided by 9.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

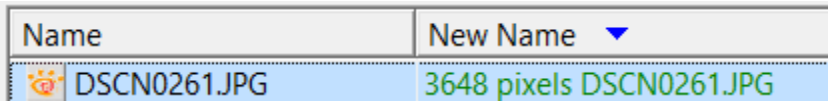
Windows File Properties "Num" cont.

filePropertyNum("<property>") cont.

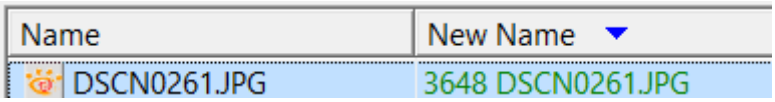
```
filePropertyNum("<property>")
```

Here are a few more examples:

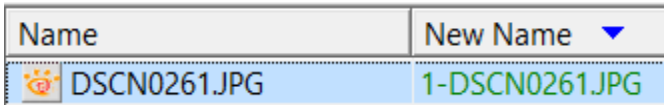
```
newName = fileProperty("Width") + " " + name;
```



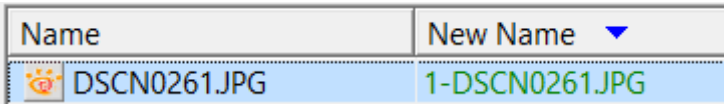

```
newName = filePropertyNum("Width") + " " + name;
```




```
newName = fileProperty("exif:Orientation") + "-" + name;
```




```
newName = filePropertyNum("exif:Orientation") + "-" + name;
```



BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

Windows File Properties "Num" cont.

fileProperty("<property>") cont.

```
fileProperty("<property>")
```

In this example the value '1' returned is the same for both, but in the example without Num, it is returned as a string value:

If I try to add the numeric 2 to the value of the returned 1 ….

```
newName = fileProperty("exif:Orientation") + 2;
```

It retruns '12' not '3', the result of adding '1' + '2'

Name	New Name
DSCN0261.JPG	12

Why didn't it come up with a type mismatch error? Because the '+' is seen as a concatenation string operator, and not as a mathematical operator.

FYI - Where did the '1' value come from? It is the Orientation flag taken from the property:

Exif:Orientation - The image orientation viewed in terms of rows and columns

Source Material This section: //impulseadventure.com/photo/exif-orientation.html

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

clipboard()

```
clipboard()
```

This function returns the text that is currently in the Windows Clipboard. This is the equivalent of using '<clip>' in the 'Section #7: Add'.

i.e.,

Captured 'A Little History of Religion' into Windows Clipboard –

```
newName = clipboard() + " - video";
```

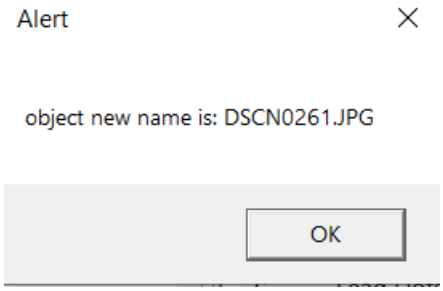
alert("text")

```
alert("text")
```

This function shows a popup message.

i.e,

```
alert("object new name is: " + newName);
```



BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

regEx("text","match","replace")

```
regEx("text","match","replace")
```

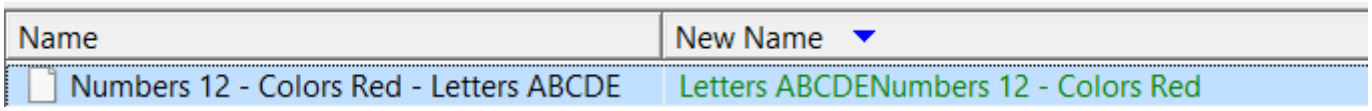
This function processes a regular expression in the same way as it is done in section 'Section #1:RegEx'.

Using this function, it is possible to process multiple regular expressions. Something that BRU cannot do, and until now was only available in BRC.

i.e.,

This code will switch a file name around:

```
newName = regEx(name, "(.*)-(.*)", "\\2\\1");
```



Where:

text =	name	BRU JavaScript constant
match =	"(.*)-(.*)"	RegEx code in quotes
replace =	"\\2\\1"	Replace String

In BRU, this would be the equivalent of:

Match:	(.*)-(.*)
Replace:	\2\1

Note:

The backslash character ' \ ' must be doubled \\ in JavaScript strings.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

setEnv("varname","value")

```
setEnv("varname","value")
```

Sets the value of an environment variable.

i.e,

```
setEnv('BRUNAME', name);
```

Where:

varname = BRUNAME
value = name

Note:

1. Once set, I don't believe the value can be deleted. I tried using the JavaScript delete statement using a few different methods unsuccessfully. Perhaps the setEnv is equivalent to using a var statement for declaration? Variables created using var cannot be removed using delete.

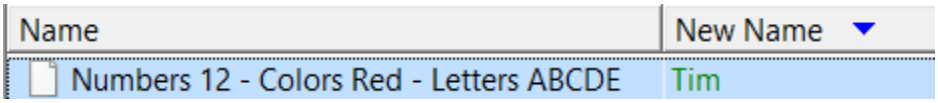
getEnv("varname")

```
getEnv("varname")
```

Gets the value of an environment variable.

i.e.,

```
var value = getenv('USERNAME');
newName = value
```



BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

include("<filepath>/filename")

```
include("<filepath>/filename")
```

Includes and runs an extra JavaScript file. The file location is relative to the current working directory, usually the Bulk Rename Utility installation folder, unless Bulk Rename Utility is running from a different directory.

Where:

"<filepath>/filename"	filepath is optional. It can be included as part of the string, otherwise it will attempt to load the file from the current directory and as explained above, this most likely is the program directory of BRU. On my system, it is, "C:\Program Files2\Bulk Rename Utility" If filepath is specified it is relative to the current directory, meaning that if you specify the filepath as js, it will look for the file in the subdirectory, 'js' located beneath the current directory. On my system this is, "C:\Program Files2\Bulk Rename Utility\js\". Note that specifying: "C:\Program Files2\Bulk Rename Utility\js\<name of library file.js> Is referred to as 'absolute' or full file path specification. filename is the name of the file, typically, a JavaScript library file.

Examples:

```
include('js/sugar.js');
```

```
include('file.js');
```

```
include('../afile.js');
```

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Functions

include("<filepath>/filename") cont.

```
include("<filepath>/filename")
```

Using the include function is the replacement for the normal JavaScript src method of:

```
src="js/lib/jquery.min.js"
```

However, the above code will also work.

Note:

1. Using 'include' (or 'src') does not verify that the library file exists. No error message is generated if it doesn't locate the file at the location specified.

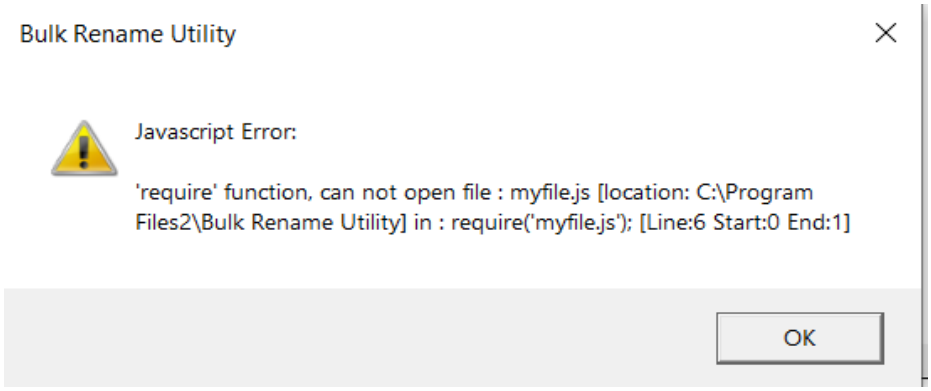
require("filename")

```
require("filename")
```

Same as include but if the file is not found, JavaScript execution is stopped and an error is generated.

```
require('js/date.js');
require('myfile.js');
```

myfile.js does not exist so it generates this message:



BRU JavaScript Reference Manual

BRU Specific JavaScript Features

JavaScript Extension Libraries

JavaScript Extension Libraries

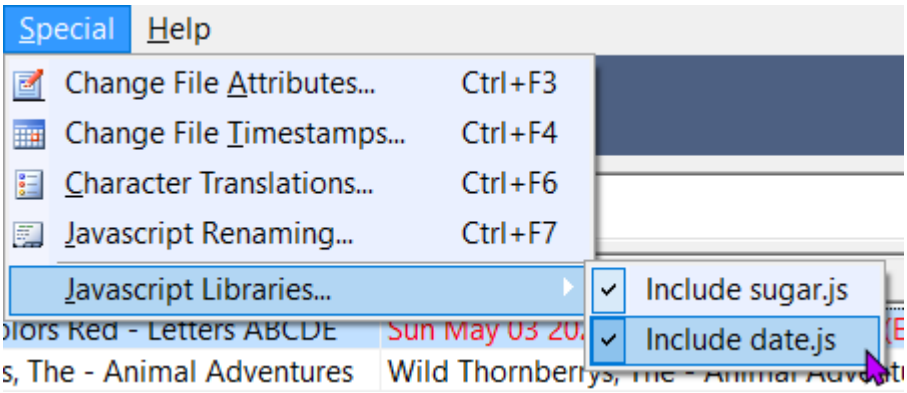
BRU comes with two available extension libraries located in a subdirectory under the program directory:

"..\ Bulk Rename Utility\js"

The library file names are sugar.js and date.js.

BRU makes it easy to include these into your JavaScript code. You don't even have to use the 'include' statement.

From the Special Menu, under the JavaScript Libraries submenu, just enable one or both libraries as needed. It is recommended to only use them if needed because inclusion of the libraries can slow down performance if you are renaming a lot of files at once.



This is the equivalent to using the 'require' at the start of your JavaScript code.

About Sugar.js library

It adds many useful functions to work with dates, text and more in JavaScript.
For more information, refer to http://sugarjs.com/api

About Date.js library

It adds many useful functions to work with dates in JavaScript.
For more information, refer to http://datejs.com/

Tim Notes:

1. The libraries are not available with the portable version of BRU.
2. Not all functions and methods from these libraries are supported in BRU. Many need adapting.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

Conditional Renaming

About Conditional Renaming

Setting the variable newName back to the origName is like canceling the renaming of an object, as the new name is set back to the object's original name. This feature allows you to perform conditional renaming.

For example, you could set newName back to origName only for objects with a certain modified timestamp. The modified timestamp is accessed via object("modified"). For more information, see object("value").

The same is valid for newExt and newLocation.

Example:

```
newName = 'A';
 if (true) newName = origName;     // revert to original name so no change
```

the statement if (true) can be anything that evaluates to a condition.

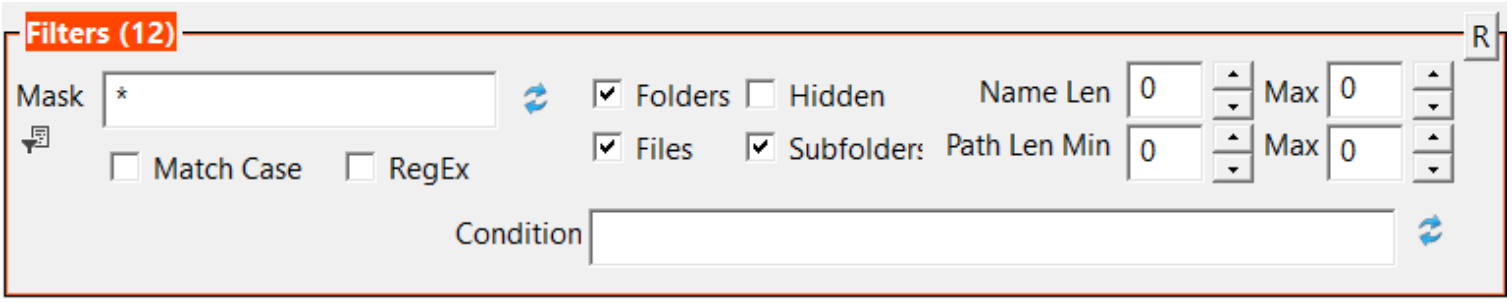
BRU JavaScript Reference Manual

BRU Specific JavaScript Features

JavaScript Filter Condition

JavaScript Filter Condition

The JavaScript filter condition is part of the 'Section #12: Filters'.



Whereas the Mask field with RegEx enabled allows you to enter a RegEx to filter out file and folder names from the Content Pane, the Condition field allows you to use JavaScript code as a filter in the same manner. If the condition evaluates to true for an object in the file list, then that object is included, otherwise it is not.

- Supports all the functions, variables and constants that are available for JavaScript Renaming.
- Allows selection/inclusion of files/folders based on objects' name and also date, size, time, exif, attributes, etc.
- Use F8 to expand the data entry field for easier entry.

Here are some examples of JavaScript conditions that can be used:

```
name.endsWith(/[q-z]/)
```

Include all objects that end with 'q' to 'z'. Requires sugar.js be enabled.

```
name.startsWith(/[a-d]/, null, false)
```

Include all objects that start with 'a' to 'd', case insensitive. Requires sugar.js be enabled.

```
ext.isBlank()
```

Include all objects that have no extension. Requires sugar.js be enabled.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

JavaScript Filter Condition

JavaScript Filter Condition cont.

```
object('modified').getTime() == exif('%d').
getTime()
```

Include all files that have the Windows modified timestamp matching the Exif date taken. Requires no additional libraries.

```
object('modified').getTime() != exif('%d').
getTime() && exif('%d').getTime() != 0
```

Include all files that have the Windows modified timestamp not matching the Exif date taken and the Exif date taken is not empty. Requires no additional libraries.

```
object('size') == 0
```

Include all objects with zero size. Disable option to include folders in 'Section #12: Filters' and to include only files. Requires no additional libraries.

```
object('size') > 10000
```

Include all files larger than 10000 bytes. Requires no additional libraries.

```
object('modified').daysAgo() < 31
```

Include all objects with the Windows modified timestamp more recent than 31 days ago. daysAgo() requires sugar.js be enabled.

BRU JavaScript Reference Manual

BRU Specific JavaScript Features

JavaScript Filter Condition

JavaScript Filter Condition cont.

```
object('modified').weeksAgo() < 4
```

Include all objects with the Windows modified timestamp more recent than 4 weeks ago. weeksAgo() requires sugar.js be enabled.

```
object('modified').isBetween('yesterday',
'tomorrow');
```

Include all objects with the Windows modified timestamp between yesterday and tomorrow. isBetween() requires sugar.js be enabled.

```
object('modified').isBetween('the beginning of
last month', 'today');
```

Include all objects with the Windows modified timestamp between the beginning of last month and today. isBetween() requires sugar.js be enabled.

```
exif('taken').isBetween('the beginning of last
month', 'today');
```

Include all objects with the EXIF taken date between the beginning of last month and today. isBetween() requires sugar.js be enabled.

BRU JavaScript Reference Manual

END

ASCII Chart & ISO 1252 Latin-1 Char Set pg1

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
^@	0	0x00	0000	^@	NUL **nul**
^A	1	0x01	0001	^A	SOH **s**tart **o**f **h**eader
^B	2	0x02	0002	^B	STX **s**tart of **tex**t
^C	3	0x03	0003	^C	ETX **e**nd of **tex**t
^D	4	0x04	0004	^D	**EOT e**nd **o**f **t**ransmission
^E	5	0x05	0005	^E	ENQ **enq**uiry
^F	6	0x06	0006	^F	ACK **ack**nowledge
^G	7	0x07	0007	^G	BEL **bel**l
^H	8	0x08	0010	^H	BS **b**ack**s**pace [\b]
^I	9	0x09	0011	^I	**HT h**orizontal **t**ab [\t]
^J	10	0x0a	0012	^J	**LF l**ine **f**eed [\n]
^K	11	0x0b	0013	^K	VT **v**ertical **t**ab
^L	12	0x0c	0014	^L	FF **f**orm **f**eed [\f]
^M	13	0x0d	0015	^M	**CR c**arriage **r**eturn [\r]
^N	14	0x0e	0016	^N	SO **s**hift **o**ut
^O	15	0x0f	0017	^O	SI **s**hift **i**n
^P	16	0x10	0020	^P	DLE **d**ata **l**ink **e**scape
^Q	17	0x11	0021	^Q	DC1 **d**evice **c**ontrol **1**, XON resume transmission
^R	18	0x12	0022	^R	DC2 **d**evice **c**ontrol **2**
^S	19	0x13	0023	^S	DC3 **d**evice **c**ontrol **3**, XOFF pause transmission
^T	20	0x14	0024	^T	DC4 **d**evice **c**ontrol **4**
^U	21	0x15	0025	^U	NAK **n**egative **ack**nowledge
^V	22	0x16	0026	^V	SYN **syn**chronize
^W	23	0x17	0027	^W	ETB **e**nd **t**ext **b**lock
^X	24	0x18	0030	^X	CAN **can**cel
^Y	25	0x19	0031	^Y	EM **e**nd **m**essage
^Z	26	0x1a	0032	^Z	SUB **sub**stitute
^[	27	0x1b	0033	^[	ESC **esc**ape
^	28	0x1c	0034	^\	FS **f**ile **s**eparator
^]	29	0x1d	0035	^]	**GS g**roup **s**eparator
^^	30	0x1e	0036	^^	**RS r**ecord **s**eparator
^_	31	0x1f	0037	^_	US **u**nit **s**eparator
	32	0x20	0040		space
!	33	0x21	0041	!	bang, exclamation

ASCII Chart & ISO 1252 Latin-1 Char Set pg 2

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
"	34	0x22	0042	"	quote
#	35	0x23	0043	#	sharp, number sign
$	36	0x24	0044	$	dollar sign
%	37	0x25	0045	%	percent
&	38	0x26	0046	&	ampersand
'	39	0x27	0047	'	apostrophe
(	40	0x28	0050	(	left parenthesis
)	41	0x29	0051	)	right parenthesis
*	42	0x2a	0052	*	star, asterisk
+	43	0x2b	0053	+	plus
,	44	0x2c	0054	,	comma
–	45	0x2d	0055	–	minus
.	46	0x2e	0056	.	period
/	47	0x2f	0057	/	slash, **not backslash!**
0	48	0x30	0060	0	numeric digit 0
1	49	0x31	0061	1	numeric digit 1
2	50	0x32	0062	2	numeric digit 2
3	51	0x33	0063	3	numeric digit 3
4	52	0x34	0064	4	numeric digit 4
5	53	0x35	0065	5	numeric digit 5
6	54	0x36	0066	6	numeric digit 6
7	55	0x37	0067	7	numeric digit 7
8	56	0x38	0070	8	numeric digit 8
9	57	0x39	0071	9	numeric digit 9
:	58	0x3a	0072	:	colon
;	59	0x3b	0073	;	semicolon
<	60	0x3c	0074	<	less than
=	61	0x3d	0075	=	equals
>	62	0x3e	0076	>	greater than
?	63	0x3f	0077	?	question mark
@	64	0x40	0100	@	at sign
A	65	0x41	0101	A	upper case A
B	66	0x42	0102	B	upper case B
C	67	0x43	0103	C	upper case C

ASCII Chart & ISO 1252 Latin-1 Char Set pg 3

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
D	68	0x44	0104	D	upper case D
E	69	0x45	0105	E	upper case E
F	70	0x46	0106	F	upper case F
G	71	0x47	0107	G	upper case G
H	72	0x48	0110	H	upper case H
I	73	0x49	0111	I	upper case I
J	74	0x4a	0112	J	upper case J
K	75	0x4b	0113	K	upper case K
L	76	0x4c	0114	L	upper case L
M	77	0x4d	0115	M	upper case M
N	78	0x4e	0116	N	upper case N
O	79	0x4f	0117	O	upper case O
P	80	0x50	0120	P	upper case P
Q	81	0x51	0121	Q	upper case Q
R	82	0x52	0122	R	upper case R
S	83	0x53	0123	S	upper case S
T	84	0x54	0124	T	upper case T
U	85	0x55	0125	U	upper case U
V	86	0x56	0126	V	upper case V
W	87	0x57	0127	W	upper case W
X	88	0x58	0130	X	upper case X
Y	89	0x59	0131	Y	upper case Y
Z	90	0x5a	0132	Z	upper case Z
[	91	0x5b	0133	[	left square bracket
\	92	0x5c	0134	\	backslash, **not slash!**
]	93	0x5d	0135	]	right square bracket
^	94	0x5e	0136	^	hat, circumflex
_	95	0x5f	0137	_	underscore
`	96	0x60	0140	`	grave, rhymes with have
a	97	0x61	0141	a	lower case a
b	98	0x62	0142	b	lower case b
c	99	0x63	0143	c	lower case c
d	100	0x64	0144	d	lower case d
e	101	0x65	0145	e	lower case e
f	102	0x66	0146	f	lower case f

ASCII Chart & ISO 1252 Latin-1 Char Set pg 4

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
g	103	0x67	0147	g	lower case g
h	104	0x68	0150	h	lower case h
i	105	0x69	0151	i	lower case i
j	106	0x6a	0152	j	lower case j
k	107	0x6b	0153	k	lower case k
l	108	0x6c	0154	l	lower case l
m	109	0x6d	0155	m	lower case m
n	110	0x6e	0156	n	lower case n
o	111	0x6f	0157	o	lower case o
p	112	0x70	0160	p	lower case p
q	113	0x71	0161	q	lower case q
r	114	0x72	0162	r	lower case r
s	115	0x73	0163	s	lower case s
t	116	0x74	0164	t	lower case t
u	117	0x75	0165	u	lower case u
v	118	0x76	0166	v	lower case v
w	119	0x77	0167	w	lower case w
x	120	0x78	0170	x	lower case x
y	121	0x79	0171	y	lower case y
z	122	0x7a	0172	z	lower case z
{	123	0x7b	0173	{	left curly brace
|	124	0x7c	0174	|	vertical bar
}	125	0x7d	0175	}	right curly brace
~	126	0x7e	0176	~	tilde
•	127	0x7f	0177	&#127;	DEL **del**ete
€	128	0x80	0200	€	
•	129	0x81	0201		
‚	130	0x82	0202	‚	
ƒ	131	0x83	0203	ƒ	
„	132	0x84	0204	„	
…	133	0x85	0205	…	
†	134	0x86	0206	†	
‡	135	0x87	0207	‡	
ˆ	136	0x88	0210	ˆ	
‰	137	0x89	0211	‰	

ASCII Chart & ISO 1252 Latin-1 Char Set pg 5

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
Š	138	0x8a	0212	Š	
‹	139	0x8b	0213	‹	
Œ	140	0x8c	0214	Œ	
•	141	0x8d	0215		
Ž	142	0x8e	0216	Ž	
•	143	0x8f	0217		
•	144	0x90	0220		
‘	145	0x91	0221	‘	
’	146	0x92	0222	’	
“	147	0x93	0223	“	
”	148	0x94	0224	”	
•	149	0x95	0225	•	
–	150	0x96	0226	–	
—	151	0x97	0227	—	
˜	152	0x98	0230	˜	
™	153	0x99	0231	™	
š	154	0x9a	0232	š	
›	155	0x9b	0233	›	
œ	156	0x9c	0234	œ	
•	157	0x9d	0235		
ž	158	0x9e	0236	ž	
Ÿ	159	0x9f	0237	Ÿ	
	160	0xa0	0240		
¡	161	0xa1	0241	¡	PostScript (¡) exclamdown
¢	162	0xa2	0242	¢	PostScript (¢) cent
£	163	0xa3	0243	£	PostScript (£) sterling
¤	164	0xa4	0244	¤	PostScript (/) fraction
¥	165	0xa5	0245	¥	PostScript (¥) yen
¦	166	0xa6	0246	¦	PostScript (ƒ) florin
§	167	0xa7	0247	§	PostScript (§) section
¨	168	0xa8	0250	¨	PostScript (¤) currency
©	169	0xa9	0251	©	PostScript (') quotesingle
ª	170	0xaa	0252	ª	PostScript (“) quotedblleft
«	171	0xab	0253	«	PostScript («) guillemotleft
¬	172	0xac	0254	¬	PostScript (<) guilsinglleft

ASCII Chart & ISO 1252 Latin-1 Char Set pg 6

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
	173	0xad	0255		PostScript (>) guilsinglright
®	174	0xae	0256	®	PostScript fi ligature
¯	175	0xaf	0257	¯	PostScript fl ligature;
°	176	0xb0	0260	°	
±	177	0xb1	0261	±	PostScript (–) endash
²	178	0xb2	0262	²	PostScript (†) dagger
³	179	0xb3	0263	³	PostScript (·) periodcentered
´	180	0xb4	0264	´	
µ	181	0xb5	0265	µ	
¶	182	0xb6	0266	¶	PostScript (¶) paragraph
·	183	0xb7	0267	·	PostScript (•) bullet
¸	184	0xb8	0270	¸	PostScript (‚) quotesinglbase
¹	185	0xb9	0271	¹	PostScript („) quotedblbase
º	186	0xba	0272	º	PostScript (”) quotedblright
»	187	0xbb	0273	»	PostScript (») guillemotright
¼	188	0xbc	0274	¼	PostScript (…) ellipsis
½	189	0xbd	0275	½	PostScript (‰) perthousand
¾	190	0xbe	0276	¾	
¿	191	0xbf	0277	¿	PostScript (¿) questiondown
À	192	0xc0	0300	À	
Á	193	0xc1	0301	Á	PostScript (`) grave
Â	194	0xc2	0302	Â	PostScript (´) acute
Ã	195	0xc3	0303	Ã	PostScript (^) circumflex
Ä	196	0xc4	0304	Ä	PostScript (~) tilde
Å	197	0xc5	0305	Å	PostScript (¯) macron, overbar accent
Æ	198	0xc6	0306	Æ	PostScript (˘) breve, flattened u-shaped accent
Ç	199	0xc7	0307	Ç	PostScript (·) dotaccent
È	200	0xc8	0310	È	PostScript (¨) dieresis
É	201	0xc9	0311	É	
Ê	202	0xca	0312	Ê	PostScript (°) ring
Ë	203	0xcb	0313	Ë	PostScript (¸) cedilla
Ì	204	0xcc	0314	Ì	
Í	205	0xcd	0315	Í	PostScript (”) hungarumlaut
Î	206	0xce	0316	Î	PostScript (˛) ogonek, reverse comma

ASCII Chart & ISO 1252 Latin-1 Char Set pg 7

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
Ï	207	0xcf	0317	Ï	PostScript (˅) caron, flattened v-shaped accent
Ð	208	0xd0	0320	Ð	PostScript (—) emdash
Ñ	209	0xd1	0321	Ñ	
Ò	210	0xd2	0322	Ò	
Ó	211	0xd3	0323	Ó	
Ô	212	0xd4	0324	Ô	
Õ	213	0xd5	0325	Õ	
Ö	214	0xd6	0326	Ö	
×	215	0xd7	0327	×	
Ø	216	0xd8	0330	Ø	
Ù	217	0xd9	0331	Ù	
Ú	218	0xda	0332	Ú	
Û	219	0xdb	0333	Û	
Ü	220	0xdc	0334	Ü	
Ý	221	0xdd	0335	Ý	
Þ	222	0xde	0336	Þ	
ß	223	0xdf	0337	ß	
à	224	0xe0	0340	à	
á	225	0xe1	0341	á	PostScript (Æ) AE
â	226	0xe2	0342	â	
ã	227	0xe3	0343	ã	PostScript (ª) ordfeminine
ä	228	0xe4	0344	ä	
å	229	0xe5	0345	å	
æ	230	0xe6	0346	æ	
ç	231	0xe7	0347	ç	
è	232	0xe8	0350	è	PostScript (L/) Lslash, L with / overstrike
é	233	0xe9	0351	é	PostScript (Ø) Oslash
ê	234	0xea	0352	ê	PostScript (Œ) OE
ë	235	0xeb	0353	ë	PostScript (º) ordmasculine
ì	236	0xec	0354	ì	
í	237	0xed	0355	í	
Î	238	0xee	0356	î	
Ï	239	0xef	0357	ï	
Ð	240	0xf0	0360	ð	
Ñ	241	0xf1	0361	ñ	PostScript (æ) ae

ASCII Chart & ISO 1252 Latin-1 Char Set pg 8

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

ASCII Table, ISO 1252 Latin-1 Chart & Character Set

Char	Dec	Hex	Octal	HTML	Function / Description / Notes
Ò	242	0xf2	0362	ò	
Ó	243	0xf3	0363	ó	
Ô	244	0xf4	0364	ô	
Õ	245	0xf5	0365	õ	PostScript (1) dotlessi, i without dot
Ö	246	0xf6	0366	ö	
÷	247	0xf7	0367	÷	
Ø	248	0xf8	0370	ø	PostScript (l/) l with / overstrike
Ù	249	0xf9	0371	ù	PostScript (ø) oslash
Ú	250	0xfa	0372	ú	PostScript (œ) oe
Û	251	0xfb	0373	û	PostScript (ß) germandbls
Ü	252	0xfc	0374	ü	
Ý	253	0xfd	0375	ý	
Þ	254	0xfe	0376	þ	
Ÿ	255	0xff	0377	ÿ	

ASCII Chart & ISO 1252 Latin-1 Char Set pg 9

ASCII characters 32 to 128 are common among most languages and character sets, while characters above ASCII 127 are different for almost every character set. In the USA, Windows systems use the Latin-1 character set by default.

Note that this Latin-1 Character set is not the same as the ASCII charts presented in the Appendix of Bulk Rename Utility Operations Manual Volume II. These codes above ASCII 127 will work with the RegEx.

Tim Notes:

1. When entering codes in RegEx, use syntax:

    \x <code>

Where:

    \x = Metacharacter to enter a hex value
    <code> = refers to the hex code.

Entered as the last two characters in the hex code of the chart above.

For example, to enter é, decimal code 193, hex code is …

    0xC1, you enter \xC1 – be aware it is case sensitive.

2. In the original Bulk Rename Utility Operations Manual, the 'Extended ASCII Chart' (in green typeface) – not the 'PC Extended ASCII Chart' (in blue typeface) is an excerpt from the Latin-1 chart.

   It gets a little confusing, because in BRU, you enter an ASCII code for the same character as (ALT on numeric keypad) + 130 (decimal) using the PC Extended ASCII chart. But for RegEx, you use \xC1 which is 193 decimal.

   a. The Extended ASCII Chart (in green typeface) from Volume I corresponds to the chart to use for RegEx, but the hex values are only found in this chart (could not locate them at the time of the writing of the first book).

   b. The PC Extended ASCII Chart from Volume I corresponds to the chart to use for BRU using the ALT + Decimal Coding.

Source Material this Section: //barcodefaq.com/ascii-chart-char-set/

A Word : The Regular Expressions Engine: PCRE

Perl Compatible Regular Expressions (PCRE) is a library written in C by Philip Hazel, which implements a regular expression engine, inspired by the capabilities of the Perl programming language. A regular expression, RegEx, is a sequence of characters that define a search pattern. The entire concept of Regular Expressions is, in fact, pattern matching. The RegEx Engine, a regex processor, translates the regular expression into an internal representation (compiled) which can be executed and matched against a string representing the text being searched upon.

Much of what Regular Expressions do as far as renaming, is text 'parsing'. Parsing, syntax analysis, or syntactic analysis, as it applies here, is the process of analyzing a string and breaking it down into its components.

There are two major versions of the PCRE library. The current version, PCRE2, first released in 2015, is now at version 10.34. The older, but still widely deployed PCRE library, originally released in 1997, is at version 8.44. BRU is currently supporting PCRE v7.5.

RegEx Engine Types

There are two fundamentally different types of regex engines 'DFA' (Deterministic Finite Automatons) and 'NFA' (Nondeterministic Finite Automatons). A Third, POSIX is an NFA classification. There are even Hybrid Engines.

DFA engines run in linear time because they do not require backtracking (and thus they never test the same character twice and are very fast). However, a DFA engine cannot match a pattern with backreferences, and because it does not construct an explicit expansion, it cannot capture subexpressions – aka Capture Groups. DFA engines are found in almost all versions of egrep and awk, as well as lex and flex, MySQL and Procmail.

Traditional NFA engines run so-called "greedy" match backtracking algorithms, testing all possible expansions of a regular expression in a specific order and accepting the first match. Traditional NFA engines include the .NET languages, PHP, Ruby, Perl, Python, GNU Emacs, ed, sed, vi, most versions of grep, and even a few versions of egrep and awk. as well as Java and the PCRE library. Although slower than DFA, you can steer them to find matches in linear or polynomial time using patterns that reduce ambiguities and limit backtracking – in other words tweak the RegEx with Possessive quantifiers and redo the code in general to limit backtracking. Using a program like Regex Buddy can help in this respect.

The POSIX standard clarifies not only which metacharacters and features an engine should support, but also the results you can expect from them. POSIX NFA engines are like traditional NFA engines, except that they continue to backtrack until they guarantee that the longest match possible has been found. As a result, a POSIX NFA engine is significantly slower than a traditional NFA engine. POSIX NFA Engines include mawk, Mortice Kern Systems' utilities and GNU Emacs (when requested).

Source Material this section: Wikipedia, //www.pcre.org/ http://msdn.microsoft.com/en-us/library/0yzc2yb0.aspx

The

Following Software was acknowledged in the Forums along with BRU to provide Solutions used in this Book:

Everything	VoidTools.com
Ztree	Ztreewin.com
Microsoft Excel	Microsoft.com
Visual Basic	Microsoft.com

Additional Testing Tools:

Heger Buddy	Just-Great-Software.com
Sed	GNU.org

Hear Ye ~ By Order of Proclamation, The Author Wishes to Bestow his Praise

Admin,
bru (aka RegexNinja)
therube,
Jane,
Stefan,
truth,
viral316,
growltiger,
Gabriel,
jimwillsher,
KenP,
Hasse,
@pepemendes,
gocyclones,
spotburst,
Glenn,
ajdedo,
the_Badluck,
@trekerie,
BRUlvr100,
Narcissus,
gilessmith,
j.zulkeski,
RobsterUK,
coyotewrw,
bitmonger,
TheRegExpMaster,
GMA,
Saintoctopus,
deselby,
DavidLago,
Abteri_X,
Tolek12,
jrstech,
CharlesFromVA,
olwynn,
bobkoure,
thewordguy,
Jack2,
xavier5678,
notused,
jgbrown54,
jcolton,
JST200,
JMM,
glennrune,
HNS,
matejmeglic,
Ardwych,
hannah141
AndrewRH
Michael

Last Word

Is this the Last Word? Will there be a Volume 3, I don't know. As the forums become more active, this may provide a future volume if enough material becomes available. There is also the possibility that I may try to tackle the BRC (Bulk Rename Command) program or I may expound upon the JavaScript Manual.

Hopefully these books will stimulate others to get involved and even purchase the commercial version. This is a misnomer. You don't have to be in business to enjoy these benefits. I hope I have proved that to you. Remember that other companies ask you to register their freeware and pay a fee to support the development and show your support. This company is no different. Show your support and register for the commercial version.

JavaScript...
When I started this volume, I had NO experience with it whatsoever. In writing this volume, a large part of my writing involved intensive research, testing, analyzing and adding my own ideas The JavaScript section was no different. I tackled bits and pieces at first stumbling my way through (as I still continue to do) ... but hey, you know what? I was able to finally understand some of it to the point where I could begin analyzing and troubleshooting scripts. I even made a few of my own which you can find scattered here and there! The same goes with Regular Expressions.

That is the Success of what I am striving to do for others. Help them... and Help you!

Nothing is better in this respect than a Program by Example type volume. I remember when that was a popular theme in technical books - This by example, that by example. I really cherished those books. One of my favorites was "Paradox by Example" - when Paradox, a fantastic Relational Database Manager was a DOS program. Although they still write 'by Example Books', they just are not the same as I remember.

This volume is a 'BRU by Example' that harkens back to that time in my life, brought to you by myself, the forums, and your own contributions.

Learn, and most off all - Enjoy.

Timothy R. Mongeon

[This Page Intentionally Blank]

Index

A

About the Format, 2, 45

Append

Append, 2, 6, 8, 16–17, 19, 23, 52, 57, 72, 157, 159–160, 260, 344–345, 884–917, 973, 982, 1009, 1017, 1028, 1102, 1108–1118, 1282–1283, 1400, 1407, 1409–1416, 1545, 1707

Appendix

Appendix, 25–30, 70, 104, 112, 155, 348, 479, 492, 648, 804, 809, 917, 1061, 1064, 1067, 1073, 1076, 1082, 1086, 1093, 1095, 1102, 1211, 1223, 1225, 1240, 1250, 1286, 1290, 1296, 1516, 1518–1519, 1521–1522, 1524–1532, 1534–1537, 1539–1542, 1544–1545, 1547–1554, 1556–1560, 1562–1566, 1568–1572, 1577, 1806–1814

- A Word on Order of Evaluation, 25, 1517–1519
- A Word on Pad Alpha, 26, 1561–1566
- A Word: The Regular Expressions Engine: PCRE, 30, 1815
- Accessing a Network Drive, 25, 1533–1537
  - BRU Does not Support UNC Paths, 25, 1534
  - Freeze or Slow Response when Accessing Mapped Network Drive, 25, 1536
  - Mapped Drives not Showing in BRU, 25, 1537
  - Mapping a Network Drive, 25, 1535

ASCII Chart & ISO 1252 Latin-1 Char Set, 30, 1806–1814

Author Proclamation to Contributors, 30, 1817

BRU JavaScript Manual, 30, 1766–1805

- BRU Specific JavaScript Features, 30, 1778–1804
- Understanding the JavaScript Code Entry Form Color Coding, 30, 1768
- Unicode, 3, 22, 26, 30, 73, 1378–1383, 1548, 1769–1777
- Unicode UTF-8 Chart (For JavaScript), 30, 1770–1777

Documenting Tools, 25, 1520–1522

Last Word, 2, 13, 19, 30, 53, 590–594, 769–770, 772–773, 835–837, 1138–1142, 1243, 1654, 1672, 1818

Long File Paths, 26, 1546–1554

- Symbolic Links (Hard and Soft Links), 26, 1551–1554

Unicode and UNC (Universal Naming Convention), 26, 1548
Using Substituted Drives, 26, 1549–1550
Methods of Creating File Lists for Import Rename-Pairs, 25, 1523
Excel Technique, 25, 1524
Methods of Creating File List Structure, 25, 1524–1526
Methods of Creating the Complete File Lists, 25, 1527–1532
The AutoHotkey Script, 25, 1530–1532
Utilize DOS to Create a List of Files, 25, 1526
Utilize Powershell, 25, 1527–1529
You can use BRU itself to Create file list for Import into Rename Pair, 25, 1524–1525
Ztree or other File Manager that has a Create Batch file method, 1525–1526
Privileges - Administrator/User Access, 26, 1555
Copy/Move Files to the C:\ Root Directory, 26, 1560
Error 740 with 'Bulk Rename Here' from Windows Explorer, 26, 1556–1558, 1560
Remove 'Bulk Rename Here' from Windows Explorer Context Menu, 26, 1559
Renaming with Dates Prior to January 1, 1971, 25, 1538–1542
Selection Issue in Content Pane using BRU, 26, 1567–1572
Software Used In Testing for this Volume, 30, 1816
Working with Folders (Directories), 25, 1543–1545

ASCII

ASCII, 21–22, 30, 39, 44, 1141–1142, 1159, 1204–1226, 1378–1383, 1769, 1806–1814
ASCII Class, 1211
High ASCII, 22, 39, 44, 1141–1142, 1159, 1219, 1225, 1378–1383
Latin-1 ASCII Chart, 1211

B

batch file

batch file, 25, 158, 161, 224, 227, 235, 890–891, 907, 981, 986–987, 1241–1242, 1353, 1454–1455, 1467, 1525–1526

BRC

BRC, 5, 21, 40–41, 156–162, 224, 233–234, 462–463, 469–470, 890–908, 938, 981–987, 1108, 1142, 1225, 1241–1243, 1247, 1352–1353, 1454–1456, 1465, 1467–1471, 1493, 1498, 1519, 1562, 1796, 1818
%, 61, 158, 161–162, 981, 983–985, 1108, 1111–1115, 1353, 1446, 1448–1450, 1453–1455, 1457, 1525, 1539, 1541–1542, 1771, 1807
%max10%, 224, 232, 235, 890–891, 907, 1241–1242
/Appenddate, 157–162
/Changecase, 1465, 1467, 1469
/CutFirstN, 1465, 1467, 1469
/Execute, 40, 224, 227, 235, 462–463, 469–470, 891–892, 894, 907, 981–982, 1241–1242, 1454, 1456, 1465, 1467
/FixedExt, 157–160
/Insert, 1465, 1467–1468
/MinLen, 1465, 1467
/PasteAtStart, 1465, 1467, 1470
/Pattern, 21, 156–160, 224, 232, 235, 462–463, 469–470, 890–892, 907, 981–982, 985, 1108, 1115, 1142, 1241, 1352–1353, 1454–1456
/Replace, 907, 1386
/TidyDS, 1465, 1467, 1471

DisableDelayedExpansion, 981, 984, 1108, 1112, 1114–1115
EnableDelayedExpansion, 981, 983–985, 1108, 1111, 1115, 1454–1455
Parameters, 161, 1519
Pipe, 728, 811, 894–898, 900, 902, 904, 906–907, 1466, 1468, 1470–1471, 1495, 1497, 1501, 1505, 1509–1510, 1512, 1525
ReplaceCI, 890–891, 907, 1465, 1467–1468, 1471
Switches, 156–157, 1352, 1394
Using Custom Date with BRC, 5, 156–162
Variables, 43, 158, 161, 884, 907, 981, 983, 1113–1114, 1453–1455, 1768, 1781–1782, 1797, 1802

Bulk Rename Utility

Add Alpha Order Name (Alphabet A-Z Serial Numbering), 2, 48
Add Creation Date as Suffix, 2, 61
Adding Space between Hyphens Anywhere in String, 2, 60
Appending Parent Folders Names to All SubFolders Names, 3, 71–72
Autonumber Files based on When Text in String Changes, 3, 67–68
Bulk Rename Utility, 2, 46–83
Changing Parentheses to Brackets, 2, 47
Completely renaming files sequentially (1,2,3...etc), 2, 51
Convert Case to Uppercase for Significant Words Only, 3, 74–79
Cropping File Extension, 2, 58
How to Add Title from Mp3 files to File Name Using ID3 Data Extraction, 3, 66
Import Rename-Pairs using Unicode (UTF-16) Files, 3, 73
Pad with Alpha not Numeric, 3, 80
Remove .com, All Brackets, Hyphens and Dashes and Maintain <space> before Year, 2, 49–50
Remove Extraneous Length of Text at End of Filename when Length is Known, 3, 69
Remove First Word, 2, 59
Remove Underscore and Add Brackets around Last Word in String, 2, 53
Rename Duplicate Filenames W/original Filename and Delete Them, 2, 55
Rename Files Based on Folder Name, 2, 56–57
Rename Photo Files with Prefix and date of Photo, 3, 81–83
Rename Same Number Sequence Alpha, 3, 70
Renaming Folders During Move, 2, 54
Renaming with Odd or Even Numerals using Autonumber, 2, 62
Replace Filename with Folder Name and Append Sequential Number, 2, 52
Use 'Folder' to Reset Autonumber for each Subdirectory's Sequential Numbering of Files, 2, 63–65

C

Case

Case, 3, 8, 16, 23, 41, 45, 74–79, 86–88, 90, 93, 104, 106, 116, 145, 156, 177, 192, 222–223, 244–245, 248, 316, 321, 344, 369, 410–411, 463, 474–476, 501, 532, 557, 623, 667, 704, 710, 735, 780, 821, 907, 918–938, 945, 948, 988–989, 1059, 1102–1103, 1113, 1211, 1249–1250, 1280, 1307, 1365, 1370, 1383, 1401, 1422, 1455, 1460–1463, 1465–1471, 1477, 1483, 1519, 1524, 1534, 1574–1575, 1589, 1594, 1626, 1659, 1688, 1694, 1697, 1706, 1768, 1802, 1807–1809, 1814
Case Insensitive, 90, 410–411, 907, 1102–1103, 1461–1463, 1483, 1574, 1802
(?i), 410–411, 1102–1103, 1460–1464, 1482–1484

Case Sensitive, 41, 156, 410–411, 463, 1574, 1594, 1626, 1768, 1814
(?-i), 410–411

Character Position

Character Position, 4, 96, 99–101, 198, 373–374, 378–379, 788, 829, 1013, 1107, 1273, 1286, 1290, 1293, 1296, 1618, 1624–1626, 1628–1629, 1631–1632, 1637–1638, 1642–1644, 1646–1648, 1650–1651, 1653, 1655–1656, 1658, 1660, 1665–1668, 1673–1674, 1688, 1690–1693, 1695–1697, 1699
Common Errors, 2, 35–44, 1768

Context Menu

Bulk Rename Here, 26, 1556–1560
Clipboard Copy, 1522
Context Menu, 26, 1556, 1558–1559, 1786

Contributor List

@pepemendes, 124, 1817
@trekerie, 207, 1817
AbteriX, 781, 1817
Admin, 33, 47, 52, 59, 69–70, 73, 114, 117, 192, 201–202, 264, 410, 486, 534, 546, 830–831, 908, 1241, 1372–1375, 1377, 1383–1389, 1391–1392, 1396, 1398–1400, 1403, 1406–1408, 1410, 1416, 1418, 1433–1435, 1439, 1441, 1445–1447, 1450–1451, 1461, 1534–1535, 1537, 1817
ajdedo, 195, 1817
AndrewRH, 83, 1817
Ardwych, 1410, 1817
bitmonger, 301, 346, 348–349, 662–663, 668, 676, 732, 1006, 1817
bobkoure, 1094, 1817
bru, 4, 25–26, 28, 30, 32–33, 35–41, 43, 46, 54, 58, 62, 70, 73, 78–80, 86, 88, 96–100, 102, 104–105, 108, 141, 156, 159, 213, 224, 234–235, 270, 309, 311, 361, 368, 385, 410, 447, 474, 520, 529, 542, 661, 696, 770, 781, 890–917, 938, 958, 1005, 1101, 1103, 1154, 1160, 1166, 1198, 1204, 1206, 1225, 1240, 1242, 1247, 1263, 1266, 1269, 1271, 1276, 1280, 1282, 1347–1351, 1370, 1433, 1438, 1448, 1453, 1456, 1461, 1464, 1467, 1470–1471, 1488–1489, 1493–1495, 1497–1499, 1502–1503, 1506–1509, 1524–1532, 1534–1537, 1539–1541, 1544, 1548, 1550, 1554, 1558, 1560, 1562, 1567–1572, 1586–1588, 1614, 1618, 1624, 1626, 1641–1642, 1743–1744, 1746–1755, 1764, 1766–1769, 1778–1805, 1814–1818
BRUlvr100, 241, 308, 1817
CharlesFromVA, 951, 957, 1817
coyotewrw, 297, 1817
DavidLago, 646, 1817
deselby, 601, 1817
Gabriel, 57, 1817
gilessmith, 266, 1817
Glenn, 155, 392, 518, 525, 667, 734, 821, 1099, 1120, 1817
glennrune, 1377, 1817
GMA, 72, 353, 371, 374, 1283, 1817
gocyclones, 138, 1817
growltiger, 54, 1817
hannah141, 1445, 1817
Hasse, 115, 1817
HNS, 642, 1396, 1817
j.zulkeski, 281, 1817
Jack2, 79, 1817
Jane, 194, 243, 245, 247–248, 252–254, 260, 317, 344–345, 358, 1000, 1003, 1096–1097, 1675–1679, 1817
jcolton, 1537, 1817
jgbrown54, 1537, 1817
jimwillsher, 62, 141, 309, 361, 696, 770, 1101, 1817
JMM, 1276, 1817
jrstech, 799, 1817
JST200, 1540, 1817
KenP, 66, 68, 80, 121, 133, 197, 237, 261, 267, 269, 271, 273, 275, 277, 279, 311, 315, 505, 507, 510, 519, 529, 571, 579, 581, 583, 599, 605, 608, 612, 616, 620, 795, 799, 801, 803, 808, 810, 813, 846, 848, 1102, 1187, 1191, 1194–1195,

1201, 1206, 1414, 1817
matejmeglic, 1403, 1817
Michael, 263–265, 303–304, 317–318, 1475, 1485, 1817
Narcissus, 264, 1817
notused, 1817
olwynn, 1007, 1817
RegexNinja, 33, 161, 192, 432, 472, 485, 644, 675, 830, 981, 1058, 1108, 1138, 1142, 1217, 1351, 1353, 1367, 1457, 1471–1473, 1475, 1485, 1492, 1514, 1591, 1622, 1706, 1817
RobsterUK, 286, 642, 1396, 1398, 1817
Saintoctopus, 453, 455, 1817
spotburst, 141, 1817
Stefan, 48, 50, 128, 130, 239, 290, 293, 299, 305, 340, 344, 351, 652, 658, 667, 683, 695, 727, 738, 743, 770, 863, 872–873, 883, 933, 935, 941, 961, 989, 995, 1007, 1037, 1096, 1104, 1196, 1277, 1353, 1530, 1532, 1817
the_Badluck, 198, 1817
TheRegExpMaster, 323, 669, 671, 951, 953, 1817
therube, 53, 55, 58, 60–61, 65, 112, 122, 126, 192, 195, 201–202, 205, 207, 211, 213, 363, 409, 453, 455, 488, 499, 501, 503, 510, 513, 537–538, 540, 542, 545, 549–550, 552, 559, 561, 563, 568, 610, 631, 729, 789, 830, 840, 842, 853, 858, 860, 1182–1184, 1204, 1206, 1228–1229, 1240, 1351, 1372, 1393, 1398, 1433, 1439, 1441, 1473, 1817
thewordguy, 75, 1817
Tolek12, 793, 1817
truth, 51, 138, 366, 614, 686, 691, 933, 935, 955, 957, 963, 966, 969, 1271, 1817
viral316, 52, 1817
xavier5678, 1204, 1817

Create Batch File

Create Batch File, 1525–1526

Criteria

Criteria, 35, 46, 86, 97–98, 830, 988, 1280, 1282, 1370, 1442, 1444, 1453, 1519, 1522, 1568–1572, 1778–1779, 1782

D

Dir /B

Dir /B, 76, 1526

J

JavaScript

Add Incrementing Counter within String, 24, 1475–1486
A discussion using .toLowerCase() and .toUpperCase() with arrays, 24, 1476–1486
Add Random String of Letters and Numbers to End of a Filename, 23, 1407
Add Today's Date and Time to Filename, 22, 1399
Append Folder Name Levels and Rearrange Them, 23, 1411–1416
Capitalize First Letter of each Word, 23, 1406
Change date Format, 22, 1389–1391
Change Letter in any Position in String from Lowercase to Uppercase, 22, 1374
Change to Title case from 40th character, 23, 1465–1471
Convert High ASCII and Unicode to ASCII, 22, 1378–1383
Convert Written date to YMD, 22, 1371–1372
Convert YMD to Written date, 22, 145, 1373
Example of Using string.repeat(number) of Javascript, 22, 1388
JavaScript, 22–24, 30, 32, 41–44, 145, 156, 463, 884, 908, 1142, 1225, 1241, 1247,

1370–1514, 1541, 1766–1805, 1818
JavaScript Condition Filter All Files that Contain name, 'walk', 23, 1408
JavaScript Condition Filter Files with Modified date not Matching Exif Date Taken Timestamp, 23, 1435
Move Everything after Name to Beginning of Filename and Add Underscore, 22, 1394–1396
Move only Renamed Files to a New Specified Folder, 23, 1442–1445
Remove (e##) from File Name, 23, 1473–1474
Remove <dots> between words only (Not numbers) in File Name and replace it with <space>, 24, 1487–1514
Remove Hyphens Between Date, 15, 22, 824–830, 1375
Remove if the Prefix or Suffix Character of a Filename is Either a Space or a Dot, 23, 1417–1433
Remove Multiple Hyphens Leaving One, 22, 1392–1393
Remove Redundant Information, 22, 1376–1377
Remove the Second Repetition of Filename from the End, 23, 1460–1464
Removing . (dot) from Beginning of Files, 23, 1440–1441
Removing Repeated Words in File Names, 22, 1384
Rename Filename to Match the Folder Name, 22, 1400
Rename Filenames with Random String of Letters and Numbers, 23, 1409–1410
Rename Part of File Name from: dd-mm-yyyy to: yyyy-mm-dd, 22, 1401–1405
Renumber in Groups of Five, 23, 1472
Replace a Series of Hyphens with Spaces, but Leave the last Hyphen, 22, 1386
Replace Filenames with Values based on Nested Loops, 23, 1446–1457
A Word about ~~, 23, 1457–1459
Replace Prefix '18' with '20' in String, 23, 1436–1439
Replace Specific Characters after a Specific Position (Replace Hyphens with Dots at Section Following date), 22, 1397–1398
Separate Run-On Sentences with Spacing #2, 22, 1385
Shorten Music Directory Names by Removing Artist and Leaving only Album Names, 23, 1434
Uppercase First Letter of Name, 22, 1387

JavaScript Topics

Bitwise Operators, 1457–1458
Comparison Operators, 1456
counter, 24, 65, 1273, 1446, 1448–1453, 1455, 1457, 1472, 1475–1486, 1779
Global Switch, 41, 156, 938, 1225, 1488, 1494, 1498, 1500, 1503–1504, 1506–1508
modulo division, 1448
mod, 1448–1450
nested loop, 23, 1446–1457
pseudocode, 1446
sugar.js library, 1419, 1800

L

Long File Path

Hard Link, 1551–1552, 1554
Junction, 1552–1554
Long File Path, 1549
mkLink, 1551–1552, 1554
Soft Link, 26, 1551–1554
Subst, 1550–1551, 1554
Symbolic Link, 26, 1537, 1551–1554
UNC, 25–26, 1534, 1537, 1548

M

Mapping

Mapping, 25, 1535, 1537

Menus

Actions Menu, 57, 79–80, 1524
Debug New Name, 1518, 1522
Import Rename-Pairs, 3, 25, 73, 78–79, 1523–1532
Rename-Pairs, 3, 25, 73, 78–79, 1523–1532
Display Options Menu, 51, 80, 1444, 1545
Sort, 35, 51, 80, 227, 509, 511–512, 981, 984, 1108, 1115, 1281
'Logical Sorting', 51
Apply Random Sort to Current List, 80
Renaming Options Menu, 55, 58, 66, 80–81, 122, 147, 275, 394, 412, 418, 425, 427, 488, 534, 636, 671, 969, 1184, 1391, 1433, 1435, 1489, 1536, 1540, 1545, 1586
Extract ID3 Data (MP3), 66
ID3 / Exif Data / File Properties, 66, 636
Ignore File Extension, 36, 39, 58, 422, 424, 433, 441, 447, 581, 1489–1490
Log Activity, 1521
Prevent Duplicates, 36, 182, 418, 425, 802
Rename File Extensions As Being Part of File Name, 36, 145, 194, 275, 412, 418, 424, 427, 534, 542, 636, 735, 780, 793, 969, 1184, 1391, 1433
Rename Folder Extensions As Being Part of Folder Name, 58, 1545
Rename in Reverse Order, 80

O

Order of Evaluation

Order of Evaluation, 25, 88, 156, 165, 1470–1471, 1517–1519

p

Permissions

Administrator Access, 1556–1558
Elevated Privilege, 1537, 1557–1558, 1560
Permissions, 1535, 1537, 1556
User Access, 26, 1555–1556, 1558

Prefix

Prefix, 3, 12, 16–19, 23, 48, 51–52, 57, 61, 64, 72, 80–83, 97–98, 132, 157–160, 364–365, 460–461, 559–560, 802, 862–864, 953, 955, 963, 969–988, 1019, 1045–1093, 1108–1118, 1122–1179, 1273–1275, 1282, 1411–1412, 1417–1433, 1436–1439, 1468, 1470, 1472, 1518–1519, 1548, 1607

Programs

'Everything', 363, 1206, 1347–1351, 1383, 1440
AutoHotKey, 25, 1530–1532
Excel, 25, 76–79, 1524, 1816
PowerShell, 25, 1527–1529
Regex Buddy, 4, 34, 96, 150–151, 153, 165, 237, 286, 366, 453, 458, 465, 467, 474, 476, 675, 895, 897, 958, 1166, 1204, 1271, 1463, 1494, 1498, 1585, 1618, 1624, 1626, 1631, 1641, 1647, 1673, 1703, 1815–1816
Sed, 1815–1816
Ztree, 25, 76, 1525–1526, 1816

R

RegEx Topics

Anchor, 14, 27–28, 96, 99, 101, 134, 150, 153, 167–168, 187–188, 198–199, 237, 266, 286, 335, 366, 420, 447, 453, 458, 474, 477, 653, 676, 678, 795, 798, 806–807, 875, 896, 910, 965, 967, 987, 1004, 1047, 1051, 1055, 1059, 1064, 1070, 1076, 1082, 1086, 1091, 1100, 1118, 1120, 1123, 1127, 1130, 1133, 1136, 1139, 1149–1150, 1206, 1238, 1240, 1321, 1324, 1327, 1332, 1340, 1355, 1358, 1363, 1424, 1462, 1603, 1610, 1614–1617, 1672–1685, 1701–1702

BOL, 96–97, 99–101, 135, 138–139, 143, 168, 172, 188–189, 196–198, 200–201, 203, 206, 220, 228, 230–231, 236, 238, 242, 244, 260, 266, 270, 280, 294–295, 307, 318, 327–329, 331, 334, 380–382, 390–392, 396–397, 399, 401, 403, 405, 407, 416, 422, 433, 441, 452, 464–466, 468, 474–475, 502, 504, 511–512, 516, 523, 526–527, 532, 534, 544, 556, 562, 564, 589, 639, 645, 647, 653, 662, 664–665, 667, 670, 678, 682, 684, 687, 698, 701, 706, 710, 728–730, 733, 735, 737, 740–741, 750, 755, 757, 760, 767, 785, 788, 794, 798, 800, 804, 809, 811, 819, 827, 829, 841, 849, 874, 893, 955, 965, 967, 971, 974, 976, 979, 987, 994, 996, 998, 1003–1004, 1013, 1048, 1052, 1056, 1060, 1064–1066, 1070–1071, 1073, 1077, 1083, 1087, 1092, 1100, 1118, 1120–1121, 1124, 1128, 1131, 1134, 1137, 1140, 1145, 1151, 1154, 1158–1159, 1183, 1193–1194, 1243–1246, 1249, 1278, 1286, 1290, 1293, 1296, 1299, 1306, 1308, 1310, 1315, 1321–1322, 1324, 1333, 1341–1342, 1356, 1360, 1365, 1424, 1440, 1462–1463, 1474, 1496, 1603, 1608, 1614–1615, 1617, 1621, 1643, 1647, 1672–1675, 1679–1681, 1683–1684, 1701, 1703

Beginning of Line, 99, 135, 138, 197–198, 220, 228, 230–231, 236, 238, 242, 244, 260, 266, 270, 318, 329, 452, 464–466, 468, 474–475, 532, 544, 647, 662, 667, 670, 687, 701, 706, 728–729, 735, 737, 785, 794, 798, 800, 804, 809, 811, 819, 965, 967, 971, 1003, 1243–1246, 1249, 1278, 1321, 1424, 1440, 1603, 1608, 1614, 1643, 1647, 1672–1674, 1680–1681, 1683–1684, 1701, 1703

EOL, 96, 99–101, 108–111, 129, 131, 135, 137–139, 147–151, 153–154, 164, 168, 177, 188, 192, 198, 201, 203, 221, 228, 230–231, 236, 238, 240, 260, 266, 268, 270, 276, 280, 282, 286, 288, 294, 296, 304, 310, 321–322, 324, 329, 335, 343, 345, 355–356, 359, 367–370, 375–377, 380–382, 390–392, 396, 398, 400, 402, 404, 406, 408, 416, 422, 433, 436, 439, 441, 443, 446–447, 453, 458, 465, 467–468, 473, 477, 512, 516, 532, 534, 545, 547–548, 551, 554, 564, 574, 576, 602, 607, 613, 615, 617, 621, 623, 634, 639, 645, 653, 659, 665, 670, 672, 676, 678, 680–682, 684, 688, 690, 692, 698, 701, 706, 710, 714, 723–724, 726, 728, 734, 741, 747–749, 751, 755, 760, 780–781, 787–788, 791–792, 794–796, 809, 811, 819, 821–823, 834, 843, 847, 866, 869, 871, 875, 878–879, 884, 887–889, 893, 896–906, 916, 930, 934, 947–950, 952, 955, 961, 965, 967, 971, 987, 994, 998, 1004, 1006, 1013, 1015,

1017–1018, 1037–1038, 1044, 1048, 1052, 1060, 1065–1066, 1071–1073, 1077, 1083, 1087–1088, 1093, 1095, 1101, 1103, 1106, 1118, 1131, 1134, 1137, 1140, 1145, 1154–1155, 1162, 1169, 1171, 1178–1179, 1182, 1186, 1193–1196, 1200, 1203, 1206, 1208, 1210, 1219, 1228, 1231, 1233, 1235, 1238, 1240, 1243, 1256, 1259, 1264–1265, 1267–1268, 1272, 1278, 1297, 1299, 1308, 1310, 1315, 1321, 1323, 1327, 1335, 1341, 1346, 1354, 1356, 1360, 1365, 1367, 1424, 1462–1463, 1466, 1496, 1505, 1575, 1577, 1579–1582, 1585–1586, 1588–1590, 1594–1596, 1605, 1611–1612, 1614, 1616–1617, 1621, 1623–1626, 1628, 1631–1633, 1635–1637, 1642, 1648, 1650–1651, 1653, 1655–1656, 1671–1674, 1677, 1679, 1681, 1701, 1703, 1724, 1731, 1733, 1750–1751, 1756, 1758–1759, 1764

End of Line, 96, 99, 108, 138–139, 149, 151, 154, 164, 177, 192, 228, 230–231, 236, 238, 260, 266, 270, 288, 329, 335, 345, 367–368, 375–377, 380–382, 392, 465, 534, 551, 670, 676, 680–682, 684, 701, 706, 726, 728, 734, 794, 809, 811, 819, 821, 866, 869, 875, 947–950, 952, 955, 961, 965, 967, 971, 987, 1004, 1006, 1013, 1018, 1038, 1044, 1048, 1066, 1073, 1077, 1083, 1088, 1093, 1101, 1118, 1131, 1134, 1137, 1140, 1145, 1155, 1196, 1240, 1256, 1278, 1321, 1356, 1360, 1367, 1424, 1463, 1575, 1581–1582, 1589–1590, 1614, 1616–1617, 1621, 1624, 1672–1673, 1677, 1681, 1701

Back Reference, 88, 202, 585, 687, 769, 872, 1463

Branch Reset, 28, 1735–1745

Capture Groups, 27, 88–89, 111, 135, 151, 154, 176, 191, 214, 216–217, 223, 290, 322, 335, 337, 348, 353, 358, 369, 380, 417, 423–424, 435, 437, 439, 443–446, 456, 469, 483, 494, 502, 508, 530, 532, 546, 585, 595, 636, 655, 678, 697, 701, 728, 750–754, 757, 760, 763, 767, 794, 884, 886, 928, 942, 987, 1036, 1051, 1055, 1058–1059, 1064, 1070, 1073, 1076, 1082, 1086–1089, 1091, 1093, 1128, 1149, 1151, 1178, 1192–1193, 1202, 1218–1222, 1224, 1231, 1233, 1235, 1237, 1247, 1249, 1252–1254, 1307, 1362, 1364–1365, 1462–1463, 1576, 1589–1591, 1593, 1601, 1610, 1620–1622, 1729–1730, 1735–1736, 1740–1742, 1815

Invalid, 108, 233, 290, 304, 336–337, 419, 426, 446–447, 473–474, 478, 481–483, 485, 572, 628, 788, 894, 897, 1158–1159, 1251–1254, 1373, 1466, 1468, 1585, 1735, 1785, 1790

Nested, 23, 116, 135, 166, 168, 185, 188, 416, 420, 481–482, 750–754, 757, 760, 763, 767, 917, 926, 1202, 1361, 1364–1367, 1446–1457, 1463, 1620, 1622

Undefined Capture Group, 337

trick of preventing an Invalid '\4', 419, 426

Conditional, 28, 90, 106, 1686–1724, 1728, 1730, 1801

Alternate, 28, 92, 106, 134, 167, 170–171, 176–177, 187, 191–192, 264–265, 327, 329, 331, 334–335, 415–416, 420, 422, 429, 433–434, 441–442, 474–475, 603, 644–645, 802, 804, 925–931, 936–937, 1052, 1058, 1064, 1073, 1076, 1082, 1086, 1093, 1127–1128, 1149–1151, 1202–1203, 1321–1323, 1332–1333, 1340–1343, 1424, 1496, 1615, 1618, 1672, 1682–1683, 1685–1703, 1715, 1719, 1722, 1735–1737, 1742–1743, 1749, 1754

Branches, 93, 177, 192, 804, 1687, 1746
Relative Conditional, 28, 90, 1715–1724, 1728, 1730
Back Relative Conditional, 1715
Forward Relative Conditional, 1715
Evaluation, 25, 39, 88, 104, 108, 129, 156, 165, 177, 192, 198, 243, 335, 367–369, 390–391, 396–397, 399, 401, 403, 405, 407, 446, 483, 526–527, 572, 628, 645, 647–648, 670, 674, 678–679, 688, 804, 894, 1013, 1040, 1044, 1124, 1128, 1151, 1155, 1240, 1284, 1288, 1291, 1294, 1297, 1300, 1317, 1324, 1328, 1336, 1365, 1470–1471, 1517–1519, 1576, 1605, 1609, 1620–1621, 1624–1625, 1628, 1638, 1642, 1644–1645, 1650–1651, 1653, 1655–1656, 1707, 1710, 1721, 1726, 1736, 1742, 1756
'Expanding' Characters, 27, 1590–1591
add characters, 102, 1591
Evaluated, 88, 92, 100, 103, 111, 135, 151, 322, 447, 483, 627, 629, 645, 648, 653, 710, 714, 750, 754, 878–879, 894, 920, 929–930, 1179, 1365–1366, 1456, 1481, 1585, 1587–1588, 1629, 1722, 1724, 1726, 1728, 1730, 1735–1736
Giving up, 27, 1589, 1591, 1699, 1751
Reevaluate, 655, 659, 679, 701, 741, 1463, 1589, 1611–1612, 1648, 1665–1668, 1675, 1679, 1746
Reevaluation, 369
Literal Characters, 88, 112, 197, 1001
Lookaround, 28, 88, 155, 389, 392–408, 468–470, 648, 1095, 1618–1703
Lookahead, 18, 28, 92, 153–154, 167–168, 170, 176, 187–188, 191, 390–391, 396–397, 399, 401, 403, 405, 407–408, 468–470, 526–527, 987, 1040, 1094–1095, 1099–1100, 1118, 1120–1121, 1123–1124, 1127–1128, 1149–1152, 1155, 1286, 1290, 1294, 1296, 1340, 1345, 1488, 1618–1624, 1626–1640, 1653–1655, 1661–1664, 1669–1673, 1681–1684, 1687–1688, 1690–1696, 1701–1703
Lookbehind, 18, 28, 90, 168, 171, 177, 188, 192, 390–391, 396–408, 468–470, 526–527, 647–648, 987, 1047–1048, 1061, 1064–1067, 1070–1073, 1094–1095, 1099–1100, 1118, 1120–1121, 1336–1341, 1343–1344, 1346, 1618–1619, 1621–1622, 1625, 1628, 1637, 1641–1658, 1661–1664, 1670, 1672–1673, 1684, 1686, 1689, 1694–1698
Negative Lookahead, 18, 28, 92, 153–154, 167–168, 170, 176, 187–188, 191, 468–470, 987, 1040, 1094–1095, 1100, 1118, 1121, 1340, 1345, 1488, 1618, 1623–1624, 1636–1640, 1653, 1670–1671, 1673, 1681–1684, 1696, 1701
Negative Lookbehind, 18, 28, 168, 171, 177, 188, 192, 468–470, 987, 1047–1048, 1061, 1064–1067, 1070–1073, 1094–1095, 1099–1100, 1118, 1120–1121, 1336–1341, 1343–1344, 1346, 1618, 1622, 1625, 1649–1653, 1673, 1684, 1696
Metacharacter, 27, 34, 99, 107–109, 165, 255–256, 270, 288, 293–294, 296, 299, 311, 322, 324, 348, 358–359, 420, 440, 449, 492, 497, 500, 513–514, 518, 520, 555, 581–582, 593–594, 597, 606–607, 636, 664, 670, 678, 727–728, 735, 750–751, 754, 757, 760, 767, 781–782, 791, 802, 809, 825, 838, 852, 950, 976, 979–980, 1001, 1100, 1120–1121, 1162, 1169, 1185–1186, 1204, 1206, 1314, 1574–1576, 1593, 1601, 1604, 1606, 1647, 1814–1815
<dot>, 17, 27, 36, 39, 91, 94, 100, 108, 116, 119–120, 123, 143, 145, 147, 149, 151–152, 154, 164, 171, 268, 271, 273, 275–276, 299, 324, 394–408, 412, 417–418, 423–424, 427, 435, 437–439, 443, 445–447,

525–527, 534, 543, 582, 611, 615, 627, 630, 637, 639, 645, 653, 677–684, 794, 877–879, 959–960, 1001, 1008, 1011, 1016–1022, 1025, 1027, 1030, 1033, 1036, 1139–1140, 1154, 1184, 1186, 1200, 1231, 1258, 1260, 1420, 1488, 1491, 1496–1497, 1499–1501, 1503–1506, 1511, 1525, 1574–1576, 1581–1582, 1585–1586, 1588, 1592, 1764
<space>, 2, 6, 9–10, 13–15, 18, 21, 24, 34, 39, 49–50, 53, 55, 59–60, 66, 72, 74, 88–89, 99, 108–112, 114, 116–117, 119–120, 123, 132, 134–135, 147–149, 151–152, 154, 156–157, 159, 172, 186–191, 196–197, 201–204, 207, 209–210, 212, 228–229, 232, 236, 240, 242, 244–247, 249, 255–256, 258, 262–263, 265–266, 268, 270, 276, 287–288, 291, 295–296, 298–299, 304–305, 307, 310–312, 314–316, 318, 320, 322, 324, 338, 341–348, 355–356, 359, 362, 364, 385, 387–388, 408–410, 415–417, 420, 422–423, 429–447, 449, 451–459, 464–468, 475–477, 482–483, 488–498, 500, 504–506, 508, 514, 516, 518, 520, 523–527, 536–539, 544, 551–552, 560, 562, 565, 567, 570, 572, 583–604, 606–607, 610–611, 617–620, 632, 634, 641–643, 647–648, 650, 655, 657, 659–660, 666–668, 670, 672, 676, 678–680, 682, 684–685, 687–688, 705–706, 714–715, 718, 721, 725–726, 730–735, 737, 744–745, 747–761, 763–765, 767–768, 772–773, 777, 779–783, 785, 787–788, 791–792, 797–800, 802, 804, 809, 813–820, 822–823, 832, 834, 836, 838, 840–841, 843, 849–850, 852, 854, 856, 861, 864, 866, 869, 886–887, 893–894, 896–898, 902–905, 910–911, 913–916, 919–921, 925, 927–930, 932, 934, 936–937, 939–942, 944, 946, 948–952, 958–960, 972, 974, 976, 978–980, 985, 990, 1016, 1018–1020, 1022, 1025, 1027, 1029–1031, 1033, 1035–1038, 1040, 1044, 1055–1056, 1064–1066, 1070–1073, 1076–1078, 1082–1083, 1086–1089, 1091–1093, 1095–1100, 1106, 1119–1121, 1139–1140, 1145, 1150–1151, 1154, 1158, 1165–1166, 1189, 1193, 1195, 1202–1203, 1210–1211, 1218, 1220, 1223–1224, 1226, 1228, 1231, 1233, 1235, 1242–1246, 1257, 1259, 1273–1274, 1277–1279, 1303, 1306–1307, 1313, 1321–1323, 1332, 1334, 1340, 1344, 1352–1353, 1371–1373, 1404, 1417–1418, 1421–1422, 1426–1427, 1429, 1454–1455, 1462–1463, 1465–1466, 1468, 1487–1514, 1562, 1574, 1578–1579, 1581–1586, 1592, 1594–1597, 1601–1602, 1604–1607, 1609–1610, 1616–1617, 1620–1621, 1634, 1637–1638, 1640, 1654, 1656, 1659–1660, 1663, 1669, 1672, 1675, 1677, 1679, 1726, 1744, 1747, 1749, 1751, 1757–1760, 1762–1764, 1791
Whitespace, 18, 92, 270, 293, 311–312, 314, 358, 489, 500, 520, 555–556, 594, 597, 670, 781, 1096–1097, 1164–1165, 1167, 1171, 1175, 1178, 1616
Class, 17, 27, 139, 165, 167, 169–172, 175–177, 187, 189–192, 215, 220–221, 228, 230–233, 236, 242–244, 249–250, 255–256, 259, 275–276, 278, 288, 294, 307–308, 314, 318, 362, 367, 369–370, 372, 392–393, 396–397, 399, 401, 403, 405, 407, 420, 422, 424, 433–434, 439, 441–442, 446, 464–468, 475–476, 487, 497, 500, 511, 534, 544, 584, 602, 604, 631–632, 641, 670, 697–698, 701, 706, 710–711,

714, 716–725, 728–730, 733, 735, 737–739, 741, 763, 767–768, 772, 782, 784, 786–788, 792, 795–796, 811, 819, 827, 866, 877–879, 887–889, 891, 893, 896–906, 913–916, 920–921, 926–928, 934, 937, 944, 950, 955, 959–960, 962, 987, 994, 1000–1001, 1004, 1018, 1051–1052, 1055–1056, 1058–1060, 1064, 1073, 1076–1077, 1082–1083, 1086–1088, 1091–1093, 1099–1101, 1118, 1121, 1123–1124, 1127–1128, 1130–1131, 1133–1134, 1136–1137, 1139–1140, 1145, 1148–1151, 1154, 1158, 1162, 1164, 1169, 1171, 1173, 1175, 1178, 1187, 1204–1205, 1208, 1210–1211, 1228, 1231, 1233, 1235, 1238–1239, 1243–1246, 1249–1256, 1262, 1285–1286, 1289–1290, 1292–1293, 1295–1296, 1298–1299, 1301–1303, 1305–1306, 1308, 1310, 1314–1315, 1320, 1322–1323, 1326–1327, 1329, 1333–1335, 1342–1346, 1354–1356, 1358, 1360, 1363, 1365–1367, 1483, 1488, 1496, 1503, 1506, 1583, 1594, 1599–1614, 1618, 1681, 1683, 1685, 1732, 1750–1753, 1760

Escape Metacharacter, 420, 728, 852, 1001, 1601, 1604

Negated Class, 228, 230–231, 236, 243–244, 249, 255–256, 259, 276, 278, 314, 318, 420, 424, 439, 446, 464–467, 475–476, 487, 497, 511, 534, 584, 716–725, 733, 735, 737, 739, 741, 787, 811, 827, 893, 896, 898, 900, 902, 904, 906, 916, 950, 1001, 1004, 1051–1052, 1055–1056, 1058–1060, 1064, 1073, 1076–1077, 1082–1083, 1086–1088, 1091–1093, 1162, 1169, 1171, 1173, 1175, 1178, 1187, 1205, 1208, 1210, 1243–1246, 1303, 1322–1323, 1333, 1483, 1488, 1503, 1506, 1583, 1602–1603, 1605

Non-Whitespace Metacharacter, 311, 500, 520, 555

Non-Word Metacharacter, 593

Special Metacharacter, 727–728, 1204

Metacharacters that have reserved status, 728

Wildcard, 27, 251, 283, 285, 357, 436, 443, 474, 491–492, 1279, 1352–1353, 1574–1576, 1629–1631, 1647, 1661–1664, 1669–1670

Word Boundaries, 207

\b, 207, 669–670, 782, 787, 1156–1159, 1616, 1700–1703, 1806

Word Metacharacter, 255–256, 288, 293, 296, 358–359, 513–514, 518, 581–582, 593–594, 597, 606–607, 735, 750, 754, 757, 760, 767, 838, 976, 979–980, 1100, 1120–1121, 1162, 1169, 1185–1186

Non-Marking Groups, 88, 347, 494, 1051, 1058, 1076, 1082, 1086, 1093, 1642

Atomic Group, 28, 430, 432, 436, 438, 443, 728, 1049, 1051, 1053, 1057–1058, 1076, 1082, 1086, 1093, 1223, 1591, 1746–1755, 1757–1758, 1760–1762

Non-Numeric Digit, 215, 348, 350, 449–453, 457, 459, 538–539, 723–725, 737, 750, 754, 760, 763, 767, 838, 869, 1037, 1303, 1312, 1613

Parse, 7, 16, 102, 108, 122, 232–233, 240, 292–296, 338, 489, 513, 524–525, 557, 585, 595, 658–660, 697, 707, 751–754, 872–883, 1016–1017, 1179, 1371–1372

Pattern Matching, 325, 492, 1815

Pattern, 13–14, 21, 67, 81, 86, 93–95, 104–105, 138–139, 148, 156–160, 163–164, 224, 232, 235, 245, 298, 311, 321, 325–328, 335, 389, 456, 462–463, 469–470, 479, 491–492, 524, 530, 552, 569, 572, 605–607, 618, 625, 645, 654, 666–668, 671, 673–674, 677–685, 696, 703, 711, 734–735, 745, 801–803, 820, 824, 826, 850, 884, 890–892, 907, 911,

918, 923, 938, 953, 963, 967, 972, 981–982, 985, 1016, 1051, 1058, 1061, 1067, 1076, 1078, 1080, 1082, 1086, 1089, 1093, 1108, 1115, 1119, 1142, 1179, 1227, 1229, 1232, 1238, 1241, 1261, 1277, 1284, 1288, 1291, 1294, 1297, 1300, 1317, 1324, 1328, 1336, 1352–1354, 1357, 1360–1362, 1364, 1366, 1385, 1403, 1446, 1454–1456, 1488, 1491, 1509, 1575–1577, 1589–1590, 1599–1602, 1606–1607, 1613–1614, 1715, 1740, 1748, 1750, 1815

PCRE, 30, 88, 90, 96, 103, 348, 520, 938, 1347, 1614, 1618, 1729, 1815

Possessive, 28–29, 90, 1240, 1753–1764, 1815

Quantifier, 27–29, 90–95, 108, 129, 175, 190, 207, 283, 299, 301, 321–322, 342, 357, 369, 479, 485, 491, 564, 628–630, 664, 678, 728, 751, 756, 758–759, 761–762, 764–766, 780, 917, 951, 967, 981, 983, 986–987, 1011, 1013, 1023, 1141, 1159, 1239–1240, 1249, 1251, 1260, 1287, 1303, 1307, 1309, 1311–1312, 1316, 1354, 1356–1357, 1360–1361, 1366–1367, 1577, 1579–1582, 1585–1590, 1592–1598, 1602, 1611–1612, 1706, 1712, 1728–1730, 1746, 1749–1763, 1815

Greedy, 15, 27, 29, 90–91, 93–95, 108, 111, 116, 119–120, 123, 129, 131, 137, 143, 147, 149, 151–152, 154, 164, 172, 189, 196, 201, 203, 209–210, 212, 215, 220–221, 228, 230–232, 236, 240, 242, 244, 246, 249, 255–256, 263, 265–266, 268, 272, 274, 276, 278, 280, 282–283, 285, 287–288, 291, 295–296, 301, 304–305, 307, 310, 312, 314, 318, 322, 324, 329, 331, 334–335, 341–343, 347, 350, 355–357, 359, 362, 367, 369, 372, 375–377, 390–392, 396–397, 399, 401, 403, 405, 407, 409–411, 416–417, 422–424, 433, 435–437, 439, 441, 443–446, 451, 457, 464–468, 475–476, 479, 487, 489, 493, 500, 502, 504, 506, 508, 511–512, 514, 516, 520, 523–524, 526–527, 530, 532, 534, 538–541, 543–544, 547–548, 551, 554, 556, 560–562, 564, 567, 569–570, 574, 576, 578, 580, 582, 584, 586, 589, 593–594, 597, 602, 604, 606–607, 609, 611–613, 615, 617, 619, 621, 623, 627, 630, 632, 634, 637, 639, 641, 643, 645, 647, 650, 653, 655, 657, 659–660, 664–665, 670, 672, 675–676, 678–680, 682, 684, 687–688, 690, 692, 694, 697–698, 701, 706, 710–711, 714, 723–726, 728, 733, 735, 737–741, 747–748, 750, 754, 757, 760, 763, 767, 770, 772–773, 777, 779–781, 784–788, 791–792, 794, 796, 800, 811, 815, 817, 819, 822–823, 832, 834, 838, 841, 843, 845, 847, 852, 854, 856, 859, 861, 864, 866, 869, 871, 874–875, 877–882, 887–889, 893, 896–906, 913–917, 919–921, 926–929, 932, 934, 936–937, 940, 942, 944, 947–948, 950–952, 954, 959–962, 967, 971, 974, 976, 979–980, 987, 990, 994, 996, 998–999, 1001, 1004, 1006, 1011, 1013, 1015, 1018, 1020, 1022, 1025, 1027, 1030, 1033, 1036, 1038, 1040, 1044, 1048, 1052, 1056, 1060, 1065–1066, 1071–1073, 1077, 1083, 1087–1088, 1092–1093, 1100–1101, 1103, 1106, 1118, 1121, 1124, 1128, 1131, 1134, 1137, 1140–1141, 1145, 1151, 1154, 1158–1159, 1162, 1165, 1167, 1169, 1171, 1178, 1182–1183, 1186–1187, 1193–1195, 1200, 1203, 1208, 1210, 1228, 1231, 1233, 1235, 1238–1240, 1243–1246, 1249–1256, 1259–1260, 1262, 1272, 1278, 1287, 1299, 1302–1303, 1306–1312,

1315–1316, 1322–1323, 1327, 1333, 1342, 1347, 1354, 1356, 1360, 1365–1367, 1462–1463, 1474, 1483, 1496, 1503, 1506–1508, 1574, 1577–1591, 1597–1598, 1602, 1604–1605, 1608, 1611–1612, 1617, 1656, 1673–1675, 1677, 1679, 1706, 1722, 1724, 1728–1730, 1750–1753, 1755–1760, 1763–1764, 1815
Lazy, 15, 27, 29, 90–91, 93–94, 129, 301, 322, 369, 468, 475, 479, 489, 491, 493, 564, 629, 678, 680, 682, 684, 728, 738–741, 770, 1013, 1239–1240, 1518–1519, 1574, 1577–1591, 1704–1707, 1709–1714, 1718, 1721, 1756–1759, 1763–1764
Non-Greedy, 93–95, 479, 489, 493, 538, 540–541, 544, 561, 593–594, 726, 728, 738, 882, 919, 932, 951, 1238–1240, 1249–1255, 1574, 1577, 1590, 1706, 1756–1757
Reluctant, 728, 1590, 1756
Range Quantifier, 27, 369, 1249, 1592–1598, 1611–1612, 1761–1762
RegEx Engine, 29, 88, 91–92, 96, 100–101, 103–104, 106, 108–111, 131, 151, 197, 240, 243, 265, 288, 324, 348, 367–369, 390, 408, 420, 435–436, 443, 453, 520, 557, 593, 606, 625, 629, 648, 655, 665, 678–679, 724, 727–728, 741, 788, 801, 809, 812, 822–823, 829, 847, 864, 929–930, 934, 936–937, 994, 1013, 1154, 1159, 1231, 1233, 1235, 1238–1240, 1243–1246, 1249–1256, 1360, 1366–1367, 1463, 1500, 1576, 1579–1580, 1582–1583, 1585, 1587–1589, 1591, 1602, 1605, 1614, 1618, 1624–1626, 1628, 1631, 1641, 1643, 1672–1673, 1682, 1686, 1688, 1697, 1707, 1714, 1721, 1729, 1746, 1756, 1764, 1815
Replacement String, 4, 39, 88–89, 99, 102, 108, 110, 112, 114, 118, 146, 228–229, 236, 298–299, 303, 335, 419, 426, 450, 452–454, 456, 459, 464, 466, 475, 483, 489, 494, 497, 524, 553, 572, 705, 737, 824, 835–836, 917, 951, 958, 968, 990, 1000, 1017, 1041, 1078, 1089, 1103, 1116, 1179, 1247, 1285, 1289, 1292, 1295, 1298, 1301, 1320, 1326, 1329, 1488, 1502, 1504, 1507–1508, 1574, 1602, 1606, 1610, 1672
Subroutine Call, 28, 90, 1725–1734
Token, 27, 103, 981, 984, 1108, 1113–1114, 1576
Validate, 168, 171, 176–177, 188, 191–192
Validation, 169–171, 180
Zero Assertion, 1625
Zero Length Assertions, 1618, 1624, 1642
Zero Length matches, 129
Zero Occurrence Match, 129, 290, 321–322, 439, 451, 457, 464–468, 472–473, 476, 479, 489, 493, 500, 576, 593–595, 628–630, 700–701, 709, 713, 718, 722–724, 780, 782–783, 787–788, 917, 1000, 1131, 1134, 1179, 1367, 1585, 1587, 1590, 1706, 1709–1710, 1712, 1714, 1764

Regular Expressions

A Note on Replacement Strings, 4, 88, 1574
Additional Notes, 4, 104
BRU and String Position, 4, 97–98
Character Position vs String Position, 4, 99–101
My Philosophy, 4, 105–107
Railroad Diagrams - Visualizations, 4, 90–95
Regex Buddy, 4, 34, 96, 150–151, 153, 165, 237, 286, 366, 453, 458, 465, 467, 474, 476, 675, 895, 897, 958, 1166, 1204, 1271, 1463, 1494, 1498, 1585, 1618, 1624, 1626, 1631, 1641, 1647, 1673, 1703, 1815–1816
Regular Expressions, 4–21, 27–30, 41, 86, 99–101, 104, 107, 155, 348, 491, 494, 728, 1095, 1368, 1573, 1765, 1796, 1815, 1818
Testing the Waters, 4, 108–112
The RegEx Sections, 4, 102
Tim Additions, 4, 103

Regular Expressions (RegEx) Manual Volume II

A Word on Tokens, 27, 1576
Additional RegEx Information on Greedy and Lazy, 27, 1577–1591
A Short Discussion on Capture Groups 'Expanding' Characters - Lazy, 27, 1590–1591
A Short Discussion on Capture Groups 'Giving Up' Characters - Greedy, 27, 1589
Additional RegEx Information on Range Quantifiers, 27, 1592–1598
Examining Possessive Quantifiers, 29, 1757–1762
Examining the RegEx Engine Behavior on Quantifiers, 29, 1756
Literal Text Match, 27, 1574
Regular Expressions (RegEx), 27–29, 99–101, 1573, 1765
The <dot> Metacharacter (aka Wildcard), 27, 1574–1576
The Differences Between Greedy, Lazy and Possessive Affect on a Test String, 29, 1763–1764
Understanding Anchors, 27, 1614–1617
Using both Anchors in a RegEx, 27, 1617
Understanding Atomic Groups with BRU, 28, 1746–1755
Changing the Atomic Group to a Possessive Quantifier, 28, 1753–1755
With Quantifier, 28, 756, 1750–1752
Without Quantifier but with Alternation, 28, 1749
Understanding Branch Reset Group, 28, 1735–1745
Understanding Character Classes, 27, 1599–1613
Character Class Summary, 27, 1613
Compound Character Classes, 27, 1606–1610
Negated Classes, 27, 243, 250, 275, 782, 891, 1602–1605, 1614
Using Range Quantifiers with Classes, 27, 1611–1612
Understanding Lookarounds, 28, 1618–1703
Examining Combination Lookarounds, 28, 1655–1656
Examining Negative Lookaheads, 28, 1636–1640
Examining Negative Lookbehinds, 28, 1649–1653
Examining Positive Lookaheads, 28, 1626–1635
Examining Positive Lookbehinds, 28, 1641–1648
Examining the Lookarounds, 28, 1618–1625
Lookaround Conditional (with Alternates), 28, 1686–1703
Using Lookarounds with Anchors, 28, 1672–1685
Understanding Referenced Conditional Groups, 28, 1704–1714
Using a Referenced Capturing Group for the 'If', 28, 1704–1714
Understanding Relative Conditionals, 28, 1715–1724, 1728
Understanding Subroutine Calls, 28, 1725–1734

Regular Expressions - Add

Add, 2–3, 6, 8–10, 13, 16–19, 21–24, 37, 40, 48, 52–53, 61, 66, 74, 97, 100, 102–104, 106, 108, 141, 146, 202–204, 224, 229, 236, 283, 298, 336, 357, 377, 382, 393–408, 412–447, 449–450, 454–485, 501–502, 505–506, 521–524, 572, 620–630, 708, 712, 755, 768, 820, 836, 839–1179, 1182, 1189, 1237, 1240, 1243, 1247, 1257, 1261, 1275, 1284–1351, 1394–1396, 1399, 1407, 1437–1438, 1448, 1456, 1464, 1466, 1468, 1470–1472, 1475–1486, 1499–1500, 1503, 1513, 1518–1519, 1524–1526, 1528, 1539, 1542, 1545, 1590–1592, 1654, 1658, 1669, 1671–1672, 1727, 1732–1734, 1756, 1785, 1794–1795, 1800

A Word on the 'Section #10: Numbering' - Insert At - Option, 19, 1107
Add '-' AFTER all upper case (Artist Name) Or BEFORE uppercase/lowercase (title), 16, 106, 918–937
Add 0 to Single Numeric Digit in String (Position Does not Matter), 16, 953–954
Add a Hyphen before First Numeric Digit in String, 16, 860–861
Add additional 'Year' as second item, remove hyphen, 16, 850–857
Add Autonumber in Middle of String (Variable Placement), 17, 988–990
Add Comma after First Item, i.e., Last Name, 16, 840–841
Add Leading Zero after #, 16, 842–843
Add prefix Based on name, 19, 1122–1179
Take last character in string and prefix it to front, 19, 1160–1162
Take second character at beginning of string and prefix to front, 19, 1174–1175
Take second character from end of string and prefix to front, 19, 1168–1171
Take second character of string and prefix it to front, 19, 1163–1167
Take second letter at beginning of string and prefix to front, 19, 1172–1173
Take the first letter (upper or lowercase) and prefix it to the front, 19, 1143–1145, 1152
Take the initial of the first word (upper or lowercase) and prefix it to the front, 19, 1146–1159
Take the initial of the last word (upper or lowercase) and prefix it to the front, 19, 1138–1142
Take the last alphanumeric character of string and prefix it to front, 19, 1135–1137
Take the last lowercase or uppercase letter of string and prefix it to front, 19, 1129–1131, 1134
Take the last Uppercase character of string and prefix it to front, 19, 1132–1134
Take third character from end of string and prefix to front, 19, 1176–1178
Take uppercase letter of the first uppercase word and prefix it to front, 19, 1122–1128
Add Quotation Signs to String, 16, 956–960
Add Space Before Numeric at End of Filename, Ignore Those That Already have, 16, 947–952
Add Spaces Between First Hyphen, 18, 1098–1100, 1120
Add Text before Numeric Digits in String (Parses Text Portion from Numeric), 16, 872–883
Add the Century to YY in String and Move to End, 16, 848–849
Add the Text in Parentheses (not Move it) to Beginning of Filename, 17, 1002–1004
Add the Word DVD to Files (Demonstrate Negated Character Class), 17, 1000–1001
Append Zeros to Prefix of Numeric Numbers of varying Lengths in String, 19, 1108–1118
Autonumber and Add Brackets to Suffix Number, 19, 1104–1107
Changing YYMMDD to YYYY-MM-DD (Restrictive Vs Universal), 17, 991–999
Changing YYMMDD to YYYYMMDD and Rearrange (Demonstrate \s Whitespace for <space> Character), 18, 1096–1097
How to add '-' between letters and digits, 16, 961–962
Insert Spaces between First Occurrence of Hyphen in String, 19, 1119–1121

Just for Fun - an Impossible Task - Append Initials of Song Title to End of Filename, 16, 884–917
Pad 0 Before Variable Numbers (Represented by #), 16, 938–946
Pad Digits at End of Filename, 16, 865–871
Pad Double Digit at End of String (in one Pass!), 17, 1007–1037
Append Only Single Numeric Digits in Mixed, 17, 1028
Appending More than One Zero, 17, 1028
Finally Here is A Mixture of 4, 17, 1031–1033
For Single Numeric Digits, 17, 870, 1009, 1011
For Triple Numeric Digits, 17, 1010–1011
Here is a three numeric digit sample with <dot> characters, 17, 1019–1020
Here is a three numeric digit sample With or Without <dot> characters, 17, 1021–1022
Match Against First Two Numeric Groups, 17, 1016–1018
Match Against the First Match, 17, 1014–1015
Match Against the Last Match, 17, 1012–1013
Mix of Three, 17, 1026–1027
Mixed, 17, 41, 1023, 1028, 1429–1430, 1562
Multiple Digits within the Same String, 17, 1019, 1023
Remove the Two Numeric Digit Numbers from String, 17, 1029–1030
Pad Random Number Prefix with Zeros, 16, 862–864
Pad Single Digit Day with Zero, 16, 858–859
Pad Zeros to Single Numeric (In One Pass!), 18, 1037–1095
Double Numeric Digits, 18, 1011, 1042–1044
Prefix First four Numeric Groups in ANY File, 18, 1067–1073
Prefix First three Numeric Groups in ANY File Example #1, 18, 1057–1060
Prefix First three Numeric Groups in ANY File Example #2, 18, 1061–1066
Prefix First three Numeric Groups in ANY File Example #3, 18, 1074–1079
Prefix First two Numeric Groups in ANY File Example #1, 18, 1045–1048
Prefix First two Numeric Groups in ANY File Example #2, 18, 1053–1056
Prefix First two Numeric Groups in ANY File Example #3, 18, 1080–1083
Prefix First two Numeric Groups in String with Three or More, 18, 1078–1079
Prefix five Numeric Groups in a String, 18, 1089–1093
Prefix Last two Numeric Groups in a String, 18, 1084–1088
Prefix Numeric Digits in Strings that have Only 2 Numeric Groups, 18, 1049–1052
Single Numeric Digit as First numeric Group in String, 18, 1039
Single Numeric Digits, 17–18, 202, 450, 460, 471, 482, 870, 953, 963, 972, 975–980, 1009, 1011, 1023, 1028, 1038
Using a Negative Lookahead and a Negative Lookbehind to Determine A Single Numeric Digit at End of String, 18, 1094–1095
Pad Zeros to Single Numeric Digit MM, DD in Dates, 17, 963–968
Padding with Zeros, 16, 844–847

Prefix 0 if the Number is Between 1-9, 16, 955
Prefix Final Digit with Zero if Preceded by Word-Separator (or null) (Pad in SINGLE Pass!), 17, 461, 969–987
Renaming All Files with no Extension to .txt, 17, 1005–1006
Renaming based on the Contents of the Original File Name, 18, 1102–1103
Simple Pad of (Zero) to Suffix, 18, 1101

Regular Expressions - Date Manipulation

Date Manipulation, 5, 102, 113–192
Convert Date (M.D.Y) to (Y.M.D), 5, 115–120
Convert date from DD.MM.YY or DDMMYY to YYYY-MM-DD And Rearrange to Front, 5, 146–155
Convert Date from mm-dd-yy to yyyy-mm-dd, 5, 128–131
Convert DMY to Y-M-D, 5, 124–125
Convert M-D-Y to Y-M-D, 5, 121
Convert MDY into YMD so the Files will List Chronologically, 5, 126–127
dd_mmm_yyyy to yymmdd Convert Abbreviated Written Date to Numeric, 5, 140–145, 163
Insert Current 'Modified Date' and Move Date within Filename, 5, 132–135
Move Date M.D.Y to Front, 5, 114, 116–117
Move Date to Front, 5, 122–123
Rename DDMMYYYY to YYYYMMDD, 5, 136–139, 1601
Use Character Translation to Convert Written Month to Numeric Format, 5, 163–192
Using Custom date with BRC, 5, 156–162

Regular Expressions - Element Substitution

Element Substitution, 20, 102, 1180–1196
Convert a numeric identification preceded with underscore to (#), 20, 1181–1182
Move String to Another Location within and Replace Dot with Hyphen, 20, 1184–1186
Replace 19 with 2019 within Filename, 20, 1183
Replace a Space Between Numbers with a Dot, 20, 1188–1193
Replace Parentheses ( ) with Brackets [ ], 20, 1195
Simple Substitution - Substitute 'M' for 4th Character in String, 20, 1196
Substitute 'QB' for First two Digits, 20, 1194
Substitute Dot for First Hyphen in String, 20, 1187

Regular Expressions - Filter RegEx

Filter RegEx, 11, 102, 528
Displaying Filenames with One or More Numeric Digits Anywhere in String, 11, 531–533
Ending with 3 but Not 4 Digits, 11, 529–530
How to Select Files with No Extension (to Rename), 11, 534

Regular Expressions - Find Occurrences

Find Occurrences, 9–10, 102, 448–527
Add a Space to Certain Files Before/After a Certain Word, 9, 505–506
Add Text after First or Last Space in String, 10, 521–523
Copy Characters to Start of Name (Good Explanation of \w), 9, 517–518
Find any First Occurrence of a Single Digit Number in a Filename, 9, 449
Find any Single Digit Number anywhere in a Filename and Add a Preceding Zero #1, 9, 454–460
Find any Single Digit Number anywhere in a Filename and Add a Preceding Zero #2, 9, 461–485

Insert 'Hyphen' after Second Word in String, 9, 513–516
Insert Text at Second Comma in String - i.e., 'brief', 9, 486–487
Move everything prior to '-' to the end of file name, 9, 498–500
Move Person's Name to end, Eliminate the Underscore after Initial and Add Underscore after #, 9, 501–502
Move the Word 'The' in first Instance Only to Front of Title and Remove Comma, 9, 507–508
Pad with Zero at Beginning, Middle and End of String, 9, 450–455, 459
Remove all Characters after x Occurrences of a Character (Remove All after 2nd Occurrence of Underscore), 9, 509–512
Remove all Text after the 'Last' Occurrence of T# when followed by Hyphen, 9, 519–520
Replace last Hyphen (in a 3 Hyphen String) with <space>, 9, 488–497
Replace the "." between Letters with a <space>, Ignore "." between Numbers, 10, 525–527
Replace Third Underscore with a Space, 9, 503–504

Regular Expressions - Remove

Remove, 2–3, 6–9, 12–17, 22–24, 26, 48–50, 52–55, 57–59, 61–62, 64, 69–70, 80, 83, 96, 102, 108, 115, 146, 150, 153, 156, 159, 175, 178, 213–237, 244, 277–278, 286, 289–291, 303, 316, 324, 337, 351, 365–366, 413, 419, 426, 446–447, 450, 453–454, 458, 463, 473–474, 485, 494, 507–512, 519–520, 524, 535–556, 559–838, 850–857, 868, 907, 919, 943, 981, 1028–1030, 1104, 1189, 1191, 1226, 1236, 1242, 1247, 1249, 1257–1258, 1349, 1351, 1362, 1370, 1375–1377, 1392–1393, 1417–1434, 1438, 1440–1441, 1446, 1454, 1456–1457, 1459–1464, 1466, 1469–1471, 1473–1474, 1480, 1483, 1487–1514, 1518–1519, 1552, 1554, 1559, 1568–1569, 1572, 1645, 1710, 1713, 1790
A Simple Lesson on RegEx Remove, 15, 774–788
A Word on Simple RegEx, 12, 557–558
Change First Name to Initials on Folders, 12, 581–582
Convert TV Series Season Information from s(##)e(##) to just Numeric, 14, 651–653
Delete Everything after SxxExx i.e., S07E05, 12, 536–537
Delete Everything in Brackets without Numbers, 14, 654–660
Delete the first word & <space> in string following the '-', 13, 583–586
Filename should have One <space> Preceding, 15, 813–819
From End… Remove last Underscore with any Following text at end of Filename, 15, 795–796
Get last x Characters, 13, 616–619
Get rid of the URL part of the Name, 13, 608–609
How to find 12 Digit Number and just Remove it, 13, 631–634
How to Keep just a 4 Digit or 5 Digit Number, 14, 663–665
Keep Last part of Filename (Preceded with Underscore) with Extension, 12, 571–579
Moving Articles (a, an, the) from the End of the Filename, 15, 820–823
Remove 'Devo* <space>' from Title, 13, 587–597
Another Method of Parsing the String, 13, 595–597
Remove all Words Before Last Word in String Retaining Characters At Beginning up to Hyphen, 13
Remove 'The' from Beginning to End of Title (Not End of Filename), 12,

544–545, 729
Remove 'The' only at the Start of a Filename, 15, 729–730
Remove All after Year and Convert Dots to Spaces and Add Parentheses Where Noted, 13, 620–630
Remove All before last Bracket, 14, 646–648
Remove All Except Last Numeric Digits Greater than Zero, 14, 696–725
Remove All Except Preserve Filename and Date in Parentheses, 15, 810–812
Remove All Except the 'pattern in bold' part of Filename, 13, 605–607
Remove all Extraneous information after First group of Digits, 12, 538–539
Remove all Extraneous junk after Extension, 12, 542–543
Remove All Text after the Dot and keep the Second Underscore, 15, 793–794
Remove all text before and after format -$-$-, 12, 552–556
Remove All Text Between { and), 14, 726–728
Remove All Text prior to Song Title at Beginning of String, 15, 802–807
Remove Duplicate Title from Multiple Music Filenames, 14, 674, 685–688
Remove Email Addresses from String, 15, 742–767
Remove Everything after (USA), 13, 640–645
Remove Everything after Brackets at End of Filename and Replace Brackets with Parentheses, 14, 671–675
Remove Everything after First Occurrence of ')', 12, 540–541
Remove Everything after Last Space, 15, 808–809
Remove Everything after Second Hyphen. No Change if Only One Hyphen, 12, 561–562
Remove Everything Between the First _ and last _, 12, 563–564
Remove Everything within First set of Square Brackets, 12, 579–580
Remove Extraneous Junk at End of Filename, 13, 610–611
Remove First 3 Characters if ##_, 12, 549
Remove Front end of Filename using Pattern <space> <hyphen> <space>, 14, 666–667
Remove Hyphen as First Character in String, 14, 661–662
Remove hyphens between date at Beginning of String, 15, 824–830
Remove Hyphens from String (Demonstrate Greedy vs Lazy), 15, 738–741
Remove Numeric Digits for any Files that Have Three or More, 13, 601–604, 631
Remove Numeric Digits, Spaces and Hyphen at the Beginning of Filename, 14, 669–670
Remove Numeric Dollar Amount from String, 14, 649–650
Remove parts 'a' and 'b' from Filename (Demonstrate the \w), 15, 734–737
Remove second + and replace with Underscore (capture parts of filename and put back together), 12, 546–548
Remove Single Numeric Digit (and Preceding Underscore) from Files that End in Single Digit, 13, 612–613
Remove Space from End of Filename (to Demonstrate the $ Anchor), 14, 676
Remove Text Between a Word and a Date, 15, 801
Remove Text between Two Dots, 14, 677–684
Remove Text in Parentheses or Brackets, 15, 831–838
Remove the Space Before the Number while Ignoring the Files that do Not Include a Number, 12, 550–551
Remove the Timestamp and Change Dot to Hyphen, 13, 635–639
Remove Trailing Dots, 13, 614–615

Remove Underscore from Title, Replace with <space> and Remove Brackets, 13, 598–600
Removing Multiple Instances of the Same Character, 15, 731–733
Removing the Numeric string '_#####' from the Middle of File Names, 14, 689–692
Rename to Remove UTC (Coordinated Universal Time) Date/Time/Zone, 15, 789–792
Renaming Files to only Initials, 15, 768–773
Similar but Not the Same - Remove Front end of Filename using Pattern, 14, 668
Simple Removes - Keep Last 5 Characters of String, Keep First 6 Characters of String, 14, 693–695
Strip Away all Prefix of Numbers and Space Character, 12, 559–560
Switch Around String and Remove Digits, 12, 565–570
To Beginning… Remove blank space in front of file name, 15, 797–800

Regular Expressions - String manipulation (Parsing)

String manipulation (Parsing), 6–8, 102, 193–447
A Parsing Lesson, 8, 338–343
Another Real Simple Example of Re-Arrangement, 7, 284–285
Another Switch around, 6, 243–244
Change 'Date Taken' attribute of the photo ((YYYYMMDD_HH.MM.SS) to More Legible, 7, 298–300
Change around Names at End of Filename, 8, 352–360
Change around String but Add Zeros to Single Digits, 6, 202–204
Change Format to both Folders and the Files Within, 6, 205–206
Change Second Hyphen to Underscore, 7, 279–280
Convert <first name> <space> <surname> ~to~ <first Initial> <surname>, 6, 266
Demonstration of 3 Methods to Rearrange a not-so-simple String, 8, 346–350
Example of Parsing a Group of Numbers with no Delimiters, 8, 365–370, 1614
Exercise in Parsing String Containing Numeric Digits - Restrictive to Universal, 7, 325–337
Ignore files with parentheses, otherwise append (Original Mix), 6, 260
Insertion or Deletion from End of Filename, 8, 373–382
More Run on Sentences, 8, 383–408
Move 'The' to End of Artist up to Hyphen, 6, 261–262
Move 'The, 'A', 'An' (insignificant words) to End of Filename, 6, 263–265
Move 4-digit year (within Filename) to Beginning of File, 6, 269–270
Move all Text after "keyword" to Front of File Name, 8, 409–411
The Case for Case Insensitivity, 8, 410–411
Move End to Beginning (Numeric Digits), 6, 267–268
Move Underscored Date to beginning of String, 6, 237–238
Moving a Date (within Filename) to the Front of the File, 7, 271–272
Moving around Digits in date for Filename, 7, 275–276
Moving Part of a Filename to Another Location, 6, 197
Moving the last 2 Words of any Phrase, 7, 286–288
Not as Simple a Rearrange, 6, 245–259
Parse and Rearrange Complex String, 7, 292–296
Parsing at its Simplest - Reformat items with Comma, 7, 281–283, 285
Place Parentheses Around Year in String, 6, 207–210

Re-arrange String Parts, 6, 195–196
Re-arranging Year Placement w Variable Lengths, 7, 319–324
Rearrange - Take First Initial, Append to End and Lowercase All, 8, 344–345
Rearrange MP3 Title, 7, 301–304
Rearrange Name in Brackets at End of Filename, 7, 297
Rearrange Names and Remove Extraneous, 8, 351
Rearrange Names to LastNameFirstInitial, 7, 309–310
Rearrange Removing First Name and Parentheses, 7, 273–274
Rearrange Text in Parentheses, Remove Parentheses, Move Hyphen, 7, 289–291
Rearrange Word and Remove first Underscore, 7, 277–278
Rearrange Word following Comma to Beginning of String, 8, 363–364, 729
Rename Camera Image Files, 6, 198–200
Replace all the Spaces and Brackets in a Filename by Underscores, 7, 311–316
Separate Run on Names, Remove Extraneous Hyphens, 6, 213–236, 1236
Simple Rearrange, 6–8, 239–242, 305–308, 317–318, 361–362
Simple Rearrange #1, 7, 305
Simple Rearrange #2, 7, 306–308
Simple Rearrange #3, 7, 317–318
Simple Rearrange #4, 8, 361–362
Simple Rearrange - More difficult, 6, 241–242
Suffix Year and Add Parentheses, 8, 412–447
Switch Artist - Title to Title - Artist, 6, 201
Switch Portions of Filename strings, 8, 371–372
Tim Notes: Parsing, 6, 194
Within string - Title first , date second, 6, 211–212

Regular Expressions - Uncategorized

Uncategorized, 21–22, 102, 1197–1367, 1385
Add Left, Right Suffix for Even/Odd Numbered Files, 21, 1284–1351
Downloaded TV Shows, 21, 1199–1203, 1227–1235, 1257–1260
Even More Downloaded TV Shows, 21, 1257–1260
Example of a Very Specific Rename of a Run-On Word Group, 21, 1261–1262
Filtering Out Filenames/Foldernames Containing Spaces, 21, 1279
Identify Every 'Nth' File from a Folder Containing Hundreds of Files, 21, 1273–1276
Isolate non-ASCII Characters, 21, 1204–1226
Match Groupings, 21, 1354–1367
More Downloaded TV Shows, 21, 1227–1235, 1257–1260
Move Files based on Creation Date to another Directory, 21, 1280–1283
Rename TV Shows from 1x01 to s01e01, 21, 1277–1278
Renaming Long Filenames, 21, 1263–1272
Separate Run-On Sentences with Spacing #1, 21, 1236–1256
Specifying a Filename Containing a <space> in a BRC /Pattern Switch, 21, 1352–1353

Root Directory

Root Directory, 26, 63, 71, 1534, 1560

S

Section #8: Auto Date

Section #8: Auto Date, 61, 82, 159, 1539–1542, 1783, 1785
Custom Format, 61, 160–161, 1540
List of Codes, 162

Section #9: Append Folder name

Section #9: Append Folder name, 52, 57, 72, 1411, 1413–1415, 1545

Section #10: Numbering

Break, 67–68, 103, 108, 134, 167–171, 187–188, 199, 212, 225–226, 270, 294, 297, 365, 393, 415, 429–432, 474, 482, 492, 518, 653, 678, 722, 750, 809, 838, 886, 896, 907, 910, 925, 943, 958, 964, 978, 987, 993, 1004, 1017, 1035, 1040, 1047, 1051, 1055, 1059, 1076, 1082, 1086, 1091, 1100, 1118, 1123, 1127, 1130, 1133, 1136, 1139, 1149–1150, 1158, 1164, 1185, 1192, 1200, 1202, 1218, 1224, 1237, 1242, 1249, 1273–1275, 1321, 1332, 1340, 1355, 1455–1456, 1462, 1476, 1514, 1608, 1620–1621, 1683, 1687, 1701–1702, 1704, 1708, 1726–1727, 1774–1775
Highest number you can enter in the 'Start' field of 'Section #10: Numbering', 51
Insert At, 19, 98, 1104–1107
pad, 3, 9, 16–18, 26, 57, 62, 64, 67, 70, 80, 450–455, 458–459, 461, 484, 844, 858–859, 862–871, 938–946, 963–987, 1007–1095, 1101, 1111, 1273–1275, 1561–1566
Roman Numerals, 74–75
Section #10: Numbering, 19, 48, 51–52, 57, 62, 64–65, 67, 70, 75, 80, 98, 988, 1104–1107, 1181, 1275–1276, 1453, 1562, 1566, 1779, 1783
Sep., 57, 159, 964
Sequential Counter, 65
Type, 48, 61, 70, 80, 82, 91–92, 132, 159–161, 169, 248, 321, 325, 689–692, 1096, 1102, 1275, 1486, 1526, 1528, 1535, 1539–1542, 1547, 1552, 1601, 1690–1697, 1784, 1787, 1794, 1815, 1818

Section #11: Extension

Section #11: Extension, 58, 159, 1005

Section #12: Filters

Filter, 11, 23, 37, 56, 64, 71–72, 78, 102, 141, 156–157, 159, 164, 179–181, 418, 425, 528–534, 661, 797, 803, 813, 830, 907, 943, 945, 1190, 1211, 1226, 1261, 1279–1280, 1348, 1351, 1364, 1384, 1408, 1434–1435, 1437, 1464, 1467, 1481–1485, 1524, 1536, 1544, 1575, 1780, 1802–1804
Recursion, 37, 56, 64, 71, 78, 205, 1280, 1435, 1524, 1536, 1544–1545, 1780
Refresh Filter Button, 945, 1190
Refresh, 35, 37, 54, 71, 81, 164, 531, 534, 830, 945, 1189–1190, 1211, 1226, 1280, 1349–1350, 1408, 1435, 1464, 1482
Section #12: Filters, 37, 56, 64, 71–72, 78, 102, 141, 156, 159, 164, 529, 531, 534, 661, 797, 803, 813, 830, 1190, 1211, 1226, 1279–1280, 1408, 1435, 1437, 1464, 1467, 1524, 1536, 1544, 1780, 1802–1803

Section #13: Copy/Move to Location

Create Empty Directories, 1545
Section #13: Copy/Move to Location, 54, 1283, 1545, 1779, 1782

Slow

Slow, 25, 1240, 1464, 1536, 1800

String Position

String Position, 4, 97–101, 645, 664–665

Suffix

Suffix, 2, 8, 18–19, 21, 23, 52–53, 57, 61,

66–67, 97, 159–161, 412–447, 1101, 1104–1107, 1179, 1181–1182, 1284–1351, 1353, 1417–1433, 1468, 1518–1519, 1539, 1541–1542, 1562

T

T A B L E Of C O N T E N T S

T A B L E Of C O N T E N T S, 2–30

Testing the Waters

Testing the Waters, 4, 108–112

Tim

A tale of Biblical Proportions, 1347
A Word from Our Sponsor, 912
coloring, 45, 87, 1370
My Philosophy, 4, 105–107
Tim Analysis, 103–104, 114, 116, 119–121, 123, 125, 127, 129, 131, 135, 137, 139–140, 143, 147, 149, 151–152, 154, 164, 172–178, 189–192, 196–197, 200–201, 203–204, 206–207, 209–210, 212, 215, 220–221, 224–233, 236, 238, 240, 242, 244, 246, 249–251, 255–257, 260, 262–263, 265–266, 268, 270, 272, 274, 276, 278, 280, 282–283, 287–288, 291, 295–296, 300–301, 304–305, 307, 310, 312, 314, 318, 322, 324, 327, 329, 331, 334, 337, 341–345, 347, 350, 354–357, 359–360, 362, 364, 367–370, 372, 375–377, 380–382, 390–392, 396–411, 416–417, 422–424, 433–446, 449, 451–453, 457–459, 468, 475–477, 483, 487, 489, 493, 500, 502, 504, 506, 508, 511–512, 514, 516, 518, 520, 523–524, 526–527, 530, 532, 534, 536, 539, 541, 543–545, 547–549, 551, 554, 556, 560, 562, 564, 567, 570, 572, 574, 576, 578, 580, 582, 584, 586, 589, 593–594, 597, 600, 602, 604, 606–607, 609, 611, 613, 615, 617, 619, 621, 623–625, 627, 630, 632, 634, 637, 639, 641, 643, 645, 647–648, 650, 653, 655, 657, 659–660, 662, 664–665, 668, 670, 672, 675–676, 678–680, 682, 684, 687–688, 690, 692, 694–695, 697–698, 701, 706, 710–711, 714, 723–725, 730, 733, 735, 737, 740–741, 744, 747–750, 754, 757, 760, 763, 767, 772–773, 777, 779–781, 784–788, 791–792, 794, 796, 800–801, 804, 809, 811–812, 815, 817, 819, 822–823, 825, 827, 829, 832, 834, 838, 841, 843, 845, 847, 849, 852, 854, 856–857, 859, 861, 864, 866, 869, 871, 874, 877–879, 881, 887–889, 893–894, 896–906, 913–916, 920–921, 926–932, 934, 936–937, 940, 942, 944, 947–948, 950, 952, 954–955, 959–960, 962, 965, 967–968, 971, 974, 976, 979–980, 987, 990, 994, 996, 998–999, 1001, 1004, 1006, 1008, 1011, 1013, 1015, 1018, 1020, 1022, 1025, 1027, 1030, 1033, 1036–1038, 1040, 1044, 1048, 1052, 1056, 1060, 1065–1066, 1071–1073, 1077, 1083, 1087–1088, 1092–1093, 1095, 1097, 1100–1101, 1103, 1106, 1118, 1121, 1124, 1128, 1131, 1134, 1137, 1140, 1145, 1151, 1154–1155, 1158–1159, 1162, 1165, 1167, 1169, 1171, 1173, 1175, 1178, 1182–1183, 1186–1187, 1193–1195, 1200, 1203, 1205, 1208, 1210, 1219, 1228, 1231, 1233, 1235, 1238–1240, 1243–1246, 1249–1256, 1259–1260, 1262, 1264–1265, 1267–1268, 1270, 1272, 1278, 1286, 1290, 1293, 1296, 1299, 1302–1303, 1306–1312, 1315–1316, 1322–1323, 1327, 1333–1335, 1342–1346, 1354, 1356, 1360, 1365–1367, 1462–1463, 1474, 1483, 1487–1488, 1496, 1503, 1506–1508, 1579–1583, 1585, 1589–1590, 1594, 1598, 1604–1605, 1608–1609, 1611–1612, 1616–1617, 1620–1621, 1624, 1626, 1628–1629, 1632–1633,

1635–1640, 1642–1646, 1648, 1650–1651, 1653, 1655–1658, 1660–1669, 1671, 1673–1675, 1677, 1679–1681, 1683–1685, 1701, 1703–1705, 1707–1711, 1713–1714, 1718, 1720–1722, 1724, 1727–1728, 1730–1733, 1744–1745, 1749–1753, 1758–1762, 1764
Tim Contribution, 51, 69, 103, 117–120, 142–145, 156, 192, 204, 208–210, 213–221, 285, 302–304, 313–314, 316, 325–337, 345, 378–382, 393–408, 490–497, 515–516, 525–527, 531–533, 553–556, 569–570, 573–578, 585–597, 603–604, 612–613, 618–619, 622–630, 633–634, 638–639, 673–675, 698–725, 736–737, 745–767, 782–788, 805–807, 814–819, 833–838, 855–857, 866–871, 873–874, 884–889, 943–946, 971–980, 997–999, 1009–1036, 1038–1044, 1061–1093, 1105–1106, 1116–1118, 1122–1178, 1207–1211, 1229–1240, 1258, 1261–1262, 1351, 1353, 1390–1391, 1404–1405, 1415, 1417–1419, 1425–1430, 1438, 1448–1453, 1475, 1541
Tim Notes, 6, 47, 52, 54, 57–58, 61, 63, 65, 70, 80, 90, 100–101, 103–104, 112, 114, 116, 118, 129, 134–136, 139, 141, 144–145, 148, 151, 154–158, 161, 163–164, 169, 172–180, 182, 189–192, 194, 198, 202, 204–205, 207, 213–214, 216, 220, 224, 232–233, 240, 242–243, 258–259, 264, 266, 271, 273, 275, 281–283, 285, 287–290, 293, 297–299, 303, 308, 311–313, 321–322, 324, 326, 329, 334, 342, 347–348, 353, 355, 357, 360, 365, 368–369, 377, 382, 385, 387–388, 390–394, 396, 398, 400, 402, 404, 406, 408, 411, 416, 419, 422, 424–426, 433, 435–436, 438–441, 443–446, 450, 454–456, 458–460, 463–467, 472, 475–477, 481, 485, 489, 491, 497, 499, 501, 510–511, 513, 520, 530, 532, 534, 540, 542, 548, 552–553, 555, 564, 568, 579, 581, 587, 591, 595, 601–602, 604–605, 612, 618–619, 623, 625, 628–631, 633, 636, 641–642, 645, 648–649, 652–656, 658–660, 664–665, 668, 670–672, 674–676, 678–679, 682, 684, 688, 691, 698–700, 702, 704–705, 708–710, 714, 718, 724, 727, 729, 735, 737, 741–742, 745, 748, 750, 752, 754–756, 760–761, 768–770, 772–773, 777, 779–783, 785, 787–788, 790–791, 795, 798, 801–802, 804, 809, 815, 819, 822–824, 827, 829, 831, 847, 852, 855, 858, 860–861, 864, 870, 872, 878–879, 883, 891, 893, 895, 897, 907–909, 911, 917, 919–921, 923, 926, 933, 935–936, 942, 946, 948, 951, 953, 957–958, 963, 969, 973, 981–982, 986, 988, 992–995, 997–998, 1000, 1005, 1007–1008, 1011–1013, 1015, 1018, 1020, 1022–1023, 1028–1029, 1031, 1035, 1037, 1039, 1041–1042, 1044, 1046, 1048, 1051–1052, 1058, 1061–1062, 1064, 1066–1068, 1073–1074, 1076, 1078–1080, 1082–1084, 1086, 1089, 1093–1095, 1102–1104, 1106, 1110, 1120, 1127, 1134, 1141–1142, 1149–1150, 1154, 1159–1161, 1165–1167, 1179, 1181, 1189–1190, 1193, 1202, 1204–1206, 1219, 1223, 1225, 1231, 1233, 1235, 1238–1239, 1242–1243, 1247, 1250–1251, 1255, 1260–1261, 1263, 1266, 1268–1269, 1271, 1276, 1283, 1285–1287, 1289–1290, 1292–1296, 1298–1299, 1301–1303, 1305–1311, 1313–1316, 1320–1322, 1324, 1326–1327, 1329, 1333–1335, 1337–1338, 1340–1346, 1348–1349, 1351, 1362, 1364–1366, 1372, 1383, 1398–1400, 1403, 1412, 1414, 1419–1420, 1422, 1424, 1431, 1433, 1442, 1446, 1448, 1453, 1455–1456, 1459, 1461, 1463, 1468, 1470–1471, 1478, 1483–1484, 1486, 1488–1490, 1493, 1498, 1501–1503, 1506, 1509–1510, 1528–1529, 1532,

1536–1537, 1540–1541, 1549–1550, 1559–1560, 1562, 1569, 1574, 1577, 1583, 1585–1587, 1592–1593, 1603–1604, 1606, 1609–1610, 1614–1615, 1617–1620, 1622–1625, 1627–1629, 1633–1635, 1638–1639, 1641–1645, 1647–1648, 1654, 1657, 1661–1664, 1668–1670, 1674, 1682, 1685, 1687, 1696–1697, 1701, 1703, 1706–1707, 1709–1710, 1712–1714, 1716, 1724, 1726, 1730, 1746, 1763, 1770, 1774, 1800, 1814

timestamp

timestamp, 13, 23, 635–639, 789, 1435, 1539–1542, 1784, 1801, 1803–1804

U

User Account Access

UAC, 1537, 1556–1558

User Interface

'Reset' button, 35, 46, 86, 1370
Content Pane, 26, 35, 37–38, 57, 72, 79, 81, 1279–1281, 1347, 1434, 1444, 1464, 1522, 1524, 1539, 1544–1545, 1567–1572, 1575, 1780, 1784, 1802
Columns, 1545, 1784, 1794
Column, 36, 51, 57, 76–78, 81–82, 108, 1281, 1444, 1536, 1539–1540, 1542, 1545, 1576, 1630, 1780, 1784
Select Columns, 1545
Sub. Dir, 1545
Taken (Original), 36, 81–82, 1539–1541
Title Bar, 57, 81, 1281, 1545
F5, 35, 37, 54, 71, 81, 164, 534, 830, 1226, 1280, 1482
Navigation Pane, 35, 56, 71, 1280, 1536
Folders, 2–3, 6, 12, 25, 37, 54, 58, 71–72, 205–206, 470, 529, 581–582, 649, 892, 1005–1006, 1206, 1279, 1434, 1543–1545, 1802–1803
Directories, 25, 54, 63, 363, 1280, 1282, 1534, 1540, 1543–1545, 1547, 1554, 1556
Tree Hierarchy, 71, 1280, 1536
Preview, 40, 43, 57, 73, 224, 227, 463, 891–892, 982, 1242, 1454, 1456, 1521
Selection, 26, 105, 141, 164, 418, 425, 909, 928, 972, 1012, 1108, 1110, 1113, 1261, 1273, 1282–1283, 1348, 1445, 1524, 1567–1572, 1575, 1802
Consecutive Selection, 1445, 1568–1572
Drag and Drop, 38, 1348, 1568
Non-Consecutive Selection, 1568, 1570–1572
Select All, 54, 57, 80, 1434, 1444

UTF-16

UTF-16, 3, 73, 1769

V

verification

verification, 1299, 1322–1323, 1327, 1336, 1341
verify, 100, 572, 673, 993, 1341, 1344, 1346, 1685, 1799

Visualization

Railroad Diagram, 4, 90–95, 326
Visualization, 4, 90–95, 184, 298, 308, 323, 501, 540, 629, 1023, 1079, 1287, 1461, 1483, 1498, 1642, 1763

[This Page Intentionally Blank]