

BULK RENAME UTILITY
OPERATIONS MANUAL
2nd Edition



VOLUME I

Timothy R. Mongeon

[This Page Intentionally Blank]

Before I begin.

I am old school. Therefore I slip between using old terms like Directories and Sub-Directories instead of the more common term today of 'Folders'. I know the difference but for the average person, it tends to be confusing, so I will leave it at that. Personally, I think the old way best described them, especially when the directory structure is viewed in a tree formation. The tree is positioned with the root at the top with all of its branches (the directories) and limbs branching off from that (the sub-directories).

Also, since this is an English – the country, not the language – program, you will find words like Behaviour, Colour, and Favourite. These are not typos. In USA lingo the equivalents are Behavior, Color, and Favorite (Bloody Yanks!) so don't run amok with your spell checkers.

The Bulk Rename Utility (BRU) can find *any* pattern in a filename or folder name and make changes – a real search and replace function. It can be a bit daunting to use at first because it can do so much more than search and replace. Many people find the program interface hard to use at first and I was no different, so I decided a book was necessary. I also wanted to highlight many of the features of this wonderful program and to pay homage to some of the contributors of the BRU Forum – see Volume 2 of this series - Expert's Corner. All of the information from the original BRU manual from Jim Willsher and TGRMN Software has been incorporated in one form or another as well.

I should also mention that in my style of writing I use capitalization, bolding, italics and colour of certain words, topics and terminology as an *emphasis* which may or may not follow any proper protocol of writing.

Technical Notes:

Some photos were edited to fit the layout. I have preserved the essential information and this should be easily observable at 100% magnification in a PDF reader. However, all photos are high resolution, and if need be, you can enlarge (zoom) the page(s) in your reader to accommodate the detail without distortion.

This volume has been updated to reflect BRU v3.43 series and completely incorporates the BRU manual as of July 2021.

Disclaimer:

All information presented has been to the best of my knowledge excerpted or researched from public domain sources. Other uses of information fall under the Fair Use Act. *Most* references to sites other than BRU have had the hyperlinks removed to prevent future 'dead links' – you can't click on them.

Thanks to Matt of TGRMN Software.

His Parliament and our Congress could learn from our Bilateral collaboration.

and my wife, Patty of 40 years, without whom none of this is possible

Modification of this book is not permitted without express written consent of the author.

Original material by Tim Mongeon; Berkshire County, Pittsfield MA USA; Other sources where noted; Revision July 2021

Considerations:

1. Only **One Instance of BRU** can run at one time.
2. When you re-launch the utility, the application **remembers the screen position** from the previous launch.
3. **Command Line options:**
 - a. On the command line you can specify the name of a favourite file to open at start-up or a directory path to scan, i.e.:

"Bulk Rename Utility.exe" filename.bru

Or

"Bulk Rename Utility.exe" directory path

- b. Other command line options

Command Line Parameters for automatic license code registration:

"Bulk Rename Utility.exe" /writeregkey:"AAAA|BBBB" [/elevated]

Where:

/writeregkey:"AAAA BBBB"	AAAA is the registration text
	BBBB the registration key
	separation character

The whole text must be surrounded by quotes

[/elevated]	Optionally instructs to prompt for elevation to administrator if needed (to register for all users on computer).
-------------	--

Program will return 0 if the operation was successful.

Examples:

"C:\Program Files\Bulk Rename Utility\Bulk Rename Utility.exe" /writeregkey:"Paul|1234"

"C:\Program Files\Bulk Rename Utility\Bulk Rename Utility.exe" /writeregkey:"Paul|1234" /elevated

Note:

There is a separate Command Line version of Bulk Rename Utility called Bulk Rename Command or BRC. This version is not covered here.

Considerations:

4. Recursive Scans

It is possible to perform a recursive scan and rename from the current folder. This allows you to rename folders and files contained within any subdirectories from the current folder. Subdirectories of subdirectories are also scanned, right down to the lowest level.

To do this, enable the 'Subfolders' option of '[Section #12: Filters](#)'. This option needs to be treated with great care – if you scan a high-level folder such as C:\ or C:\Program Files, the program could have tens of thousands of files to scan. While the system should cope with in excess of 250,000 files, it will take a long time for the file list to be displayed. It is recommended that you only use the 'Subfolders' option only if you really need it.

Please note, renaming a folder using recursion will automatically refresh the File List in the Content Pane. This is to prevent problems with synchronization with the files on your hard drive (The File List no longer reflects the names of the actual files on your hard drive). This does not apply if just renaming files (with or without recursion).

For more information, refer to 'Subfolders (Recursion)' under '[Section #12: Filters](#)', 'Speeding up the Program'.

5. INI File

When you quit the application, your current settings (menu choices etc.) will be stored in a .INI file in the same folder as the executable program. As Bulk Rename Utility doesn't require an Installer this makes it quite useful as a utility on a "memory stick" or a "Tools CD".

If you need two sets of preferences with different values then create two copies of the executable with different names, and you'll get two INI files. Newer versions of BRU support Unicode.

6. Portability

There is also a portable version of Bulk Rename Utility (BRU_NoInstall.zip) available on the website. Note that the portable version does not include the built-in JavaScript libraries, sugar.js and date.js.

7. [Updates are available](#) from the application's website (www.bulkrenameutility.co.uk). Make sure you're using the latest version to obtain the most benefit.

8. [To unlock the JavaScript functionality, you need an inexpensive Commercial License](#) which also [shows your support](#) in the future development of both the freeware and commercial versions of BRU.

Even if you don't know JavaScript, you can copy and paste the examples found in Volume 2. Without a commercial license, you can paste the examples into the JavaScript Code Entry Form for testing ('[Section #14: JavaScript](#)'), but you won't be able to Rename the files. The JavaScript capability provides the full power of BRU, useful even if you are just a home user. Further assistance can be found in the JavaScript section of the BRU forums. There are also numerous books, publications, etc. on the web if you want to learn more.

Introduction to the 2nd Edition of Volume I

This volume has been updated to version 3.4.30 of BRU.

This is a complete rewrite of the first edition with over 400 pages of new and revised material that includes more photos, illustrations and diagrams. I have also added new elements, e.g., lines, shapes and arrows, to assist with understanding the text. I have redone the fonts, added more colour, and made the text sharper.

Where applicable, supplemental text will be addressed for any program changes specific to newer versions of v3.4x. I have chosen to make note of these changes at the appropriate end of each section and where necessary incorporate them within the main text of the subject matter.

Also of note, some added material that appears in the first few chapters addresses topics that will be discussed in more detail later on in the book, so if you don't understand something in one section of the book, the discussion will be forthcoming.

You now have many avenues available to you to locate information on your own – newly added Bookmarks and a full hyperlinked Index, and of course the Table of Contents and don't forget the Search facility of your own PDF reader.

Every photo has been painstakingly recreated from the original volume under the newest version as of this revision date.

I have used my own computer network to document this volume. Therefore, where I feel it necessary, photos have been altered and text redacted for my personal privacy concerns.

The section entitled, '**Section #1: RegEx**', now includes support for many of the improvements of PCRE v2. The RegEx Manual in the Appendix *has also* undergone a full rewrite – still based on PCRE v1 for the most part, though, but applicable just the same.

I have also included extensive analysis of the sample RegEx - something that was previously only available in Volume II. In doing so, I must state that my analysis is not fact, nor for that matter simply conjecture – it is my opinion based on how I view the logical progression of an evaluation of a RegEx to its conclusion. I will repeat this disclaimer at various places as a reminder.

Additionally, you will find that this volume includes samples of testing data that I used for validation surrounding a Metadata discussion in the heading titled, '**Section #8: Auto Date**'. I typically do not include my testing methods but in this case, I felt it was important to illustrate my findings rather than just state, 'this' or 'that', as absolute without showing proof.

Most page references that appeared in the first edition have been removed because of the complexity of tracking and updating them all. Instead, I have used the names of headings and sections. Any needed information can be searched within the content of this volume.

Exceptions were made with certain references to pages in Volume II. These, however will be outdated as soon as a new revision of Volume II is released.

[This Page Intentionally Blank]



Contents

Bulk Rename Utility Operations Manual

Before I Begin 3

Considerations 4

Introduction to the 2nd Edition of Volume I 6

Drag and Drop from Explorer 15

The Program Screen 17

User Interface 21

The 14 Sections Used to Specify the Criteria 25

Order of Expression Evaluation 26

Program Notes 32

Understanding Favourites 39

Save on Exit 39

Store Pathname 39

Section # 1 - RegEx 46

Match 48

Replace 49

v3.4 New Additions 51

Numbered Backreferences 51

Named Capture Groups 53

Using Named Backreferences with Capture Groups 54

Size Limitations in PCRE and PCRE2 60

Specifying Multiple Regular Expressions Using The (?X) Separator 62

v2 vs v1 & Simple 66

The Global Switch /g 88

Case Insensitive /i 92

Case Conversion 93

v3.43 New Additions 95

Added ability to use \E \L \l \U \u modifiers in the Replace field of the 'Simple' RegEx 95

Section # 2 - Filename 100

Filename 101

CONTENTS**Section # 3 - Replace 102**

Replace 103

v3.4 New Additions 104

Multiple Replacements 105

Match Case enabled – Perform case-sensitive replacement with Replace Only on First Match 107

Position Modifier \<Position value>\ 108

Section # 4 - Case 112

Case 113

v3.43 New Additions 114

The New York Times Title Case is now used in the Title option, Enhanced Title 114

New Changes in the Except(ion) field 116

Section # 5 - Remove 122

Remove 123

Section # 6 – Move/Copy Parts 136

Move/Copy Parts 137

Section # 7 – Add 144

Add 145

Using Substitution Tags 148

MP3 148

JPEG 150

Other Tags Available in BRU 154

Removed 154

File Size 155

Hash Value 156

Using Windows File Properties 159

File Properties as Dates and Numbers 162

Using Windows Clipboard Data 164

Using EXIF tags 165

EXIF Properties as Dates and Numbers 168

Section # 8 – Auto Date 170

Auto Date 171

Adding a New Date & Timestamp 176

Using the EXIF property ‘Taken (Original)’ 178

v3.4 New Additions 187

Item Date 187

Section # 9 – Append Folder Name 216

Append Folder Name 217

v3.42 New Additions 218

Append Negative Value for Specific Directory Name 218

Section # 10 – Numbering 222

Numbering 223

v3.43 New Additions 228

Roman Numerals section removed and placed under ‘Type’ section 228

Case 231

Section # 11 – Extension 247

Extension 248

Section # 12 – Filters 250

Filters 251

Subfolders (Recursion) 253

BRU’s Sub Dir Column vs Full Path 256

v3.42 New Additions 260

Set Subfolder Level to Control Recursive Scanning 260

CONTENTS**Section # 13 – Copy/Move to Location 262**

Copy/Move to Location 263

Moving Files from one Directory to Another Location with Files content! 268

Section # 14 – Special 271

Special 272

Menus 274**File Menu 275**

New (Ctrl + N) 276

Open (Ctrl + O) 276

Save (Ctrl + S) 276

Save As 276

Recent 277

Favourites 277

Save on Exit 277

Store Pathname 277

Actions Menu 279

Selection 280

Select All (Ctrl + A) 281

Deselect All (Ctrl + D) 281

Invert Selection (Ctrl + I) 281

Select From Clipboard 282

Jump to Path (Ctrl + J) 283

v3.4 New Additions 283

Network/UNC Paths are Supported 283

Troubleshooting UNC Under Jump to Path 299

Rename Object Manually (F2) 302

Refresh Files (F5) 302

Refresh Tree (Ctrl + F5) 302

Show/Hide Tree (F11) 302

Zoom (F8) 303

List 303

Reposition 303

Apply Random Sort to Current List 305

Show Only Items Affected by Renaming Criteria 307

Clear All Items from Current List 308

Auto-Select All Items After Listing a Folder 308

Clear All Non-Selected Items from Current List (Ctrl + O) 309

Import Rename-Pairs (Rename from a Text File) 310

Import Rename-Pairs 310

View Imported Rename-Pairs 314

Clear Imported Rename-Pairs 314

v3.4 New Additions 315

Full Path Support 315

Debug New Name 318

Reset All Renaming Criteria (Ctrl + T) 327

Revert All Criteria to Last Saved (Ctrl + E) 327

Preview (Ctrl + P) 328

Rename (Ctrl + R) 329

Undo Rename (Ctrl + Z) 329

Create Undo Batch File (Ctrl + B) 330

C O N T E N T S**Display Options Menu 331**

Always on Top 332

List 332

Show Gridlines 332

Show Icons 333

Show File Sizes as... 333

Show Picture Viewer (Ctrl + W) 334

Select Columns 335

Autofit All Columns (Ctrl + Alt + +) 342

Expand File List (Ctrl + F9) 343

Maximize File List (F9) 345

Colours 347

New Name OK 347

New Name Invalid 347

Active Criteria 348

Highlight Active Criteria 348

Changing the Colours 349

Font 350

Use Larger Font 350

Use Smaller Font 350

Reset 350

Sorting 351

Logical Sorting 351

Group Affected Files 354

Sort Files and Folders Together 355

v3.4 New Additions 356

Custom Column 356

Item Date 363

A Brief primer on EXIF Date Taken accuracy- GIGO 366

Renaming Options Menu 367

Retain Autonumber 368

Rename in Reverse Order 370

Prevent Duplicates 372

Advanced Options 373

Allow Using '\ ' in Renaming Criteria for Creation of New Folders 373

Allow Overwrite / Delete Existing Files During Renaming If Needed 376

ID3 /EXIF Data /File Properties 377

Extract ID3 Data (MP3) 377

Extract EXIF Data (Photos) 378

Extract Windows File Properties 378

File/Folder Extensions 379

Rename File Extensions as Being Part of File Name 379

Rename Folder Extensions as Being Part of Folder Name 379

Log Renaming Activity to File 380

Show Warning Message Before Renaming 380

Show Confirmation Message After Renaming 381

CONTENTS

Special Menu 382

Change File Attributes 383

Change File Timestamps 385

 Understanding Delta 389

 v3.4 New Additions 390

 Change File Timestamps to Allow Item Date 390

Character Translations 391

JavaScript Renaming 396

 Using the Test Facility 397

 Handling Syntax Errors 398

 About Conditional Renaming 401

JavaScript Libraries 405

 Include sugar.js 405

 Include date.js 405

JavaScript Filter Condition 406

Context Menu 409

BRU Context Menu 410

 Clipboard Copy 410

 Open Containing Folder 412

 Show List of File Properties 412

 Show List of EXIF Info (.JPG Files) 412

 v3.4 New Additions 413

 Copy all available column data for highlighted files 413

Windows Explorer Context Menu 415

 Bulk Rename Here 415

CONTENTS**Appendix 417****Speeding up the Program 418**

Display Options Menu 419

List - Show Icons 419

Renaming Options Menu 419

ID3 / EXIF Data / File Properties 419

Section #12: Filters 419

Subfolders Option (Recursive Scan) 419

v3.4 New Additions 419

Enabling v2 (PCRE v2 with Boost) 419

Regular Expressions (RegEx) Manual 420

Applying example to BRU - Review Time 421

v3.4 New Additions 422

BRU Supports PCRE v2 with Boost 422

Literal 423

Special Characters (aka Metacharacters) 423

Metacharacters in Depth 427

Greed and Lazy 566

Backtracking 588

Non-marking Groups 590

Using Options with RegEx 591

Case Insensitive 591

Free-Space Option 596

Comments 598

If Then Else 599

Using multiple Capture Groups to locate Partial Words 600

Lookarounds: Lookahead and Lookbehind 602

Using Alternates with Capture Groups 642

Personal Examples 661

JavaScript Bulk Rename Utility Constants and Variables 667

JavaScript BRU Constants and Variables 668

JavaScript BRU Utility Functions 669

EXIF Metadata Reference 671**ASCII Character Codes 697**

ASCII Character Chart (Version 1) 698

ASCII Character Chart (Version 2) 700

Extended ASCII Character Chart 709

PC Extended ASCII Character Chart 710

Last Word

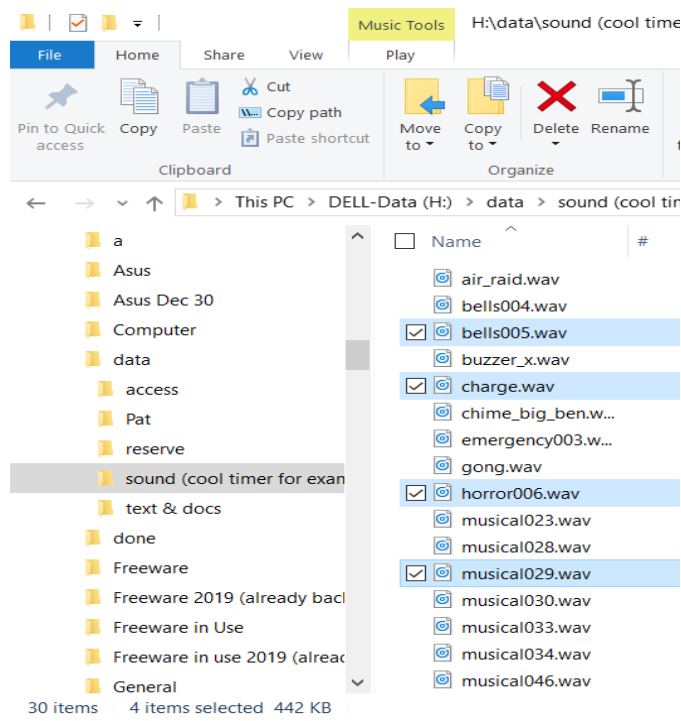
[This Page Intentionally Blank]

Drag and Drop from Explorer

You can now drag files and folders directly from within Windows Explorer. This means you can select files from anywhere on your computer and aggregate them all together, allowing you to rename them in a single operation.

To perform this task:

1. Launch Bulk Rename Utility in the normal way.
2. Launch Windows Explorer.
3. Find the files or folders that you wish to process.
4. Select the files/folders. If you select a folder, the contents of that folder will be aggregated together with any other selected files.

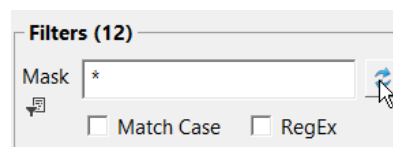


5. Drag them to the Bulk Rename Utility "file list" window.
 - a. You do not have to use ALT or Shift when dragging because the files are not actually moved or copied.

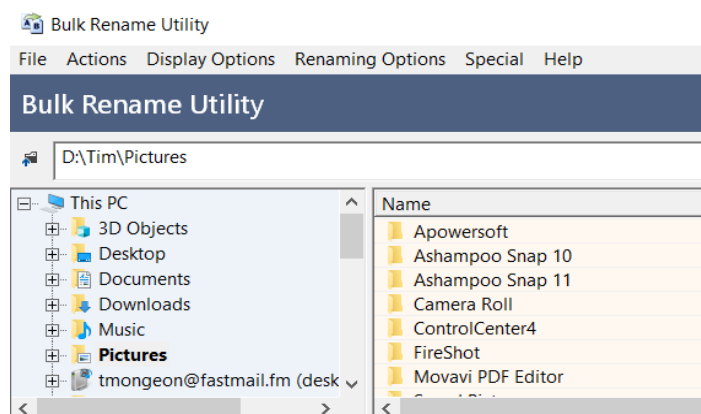
6. Release the mouse, and the files will be listed in the Content Pane.

Note:

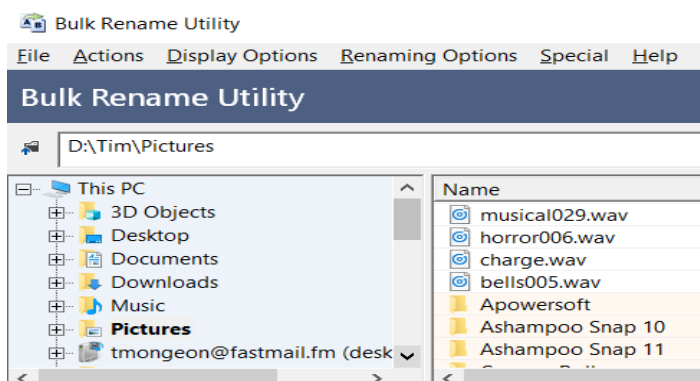
You can use the 'Filter Refresh' button to refresh the file listing and remove your selections.



File List prior to Drag and Drop:



File List after Drag and Drop:




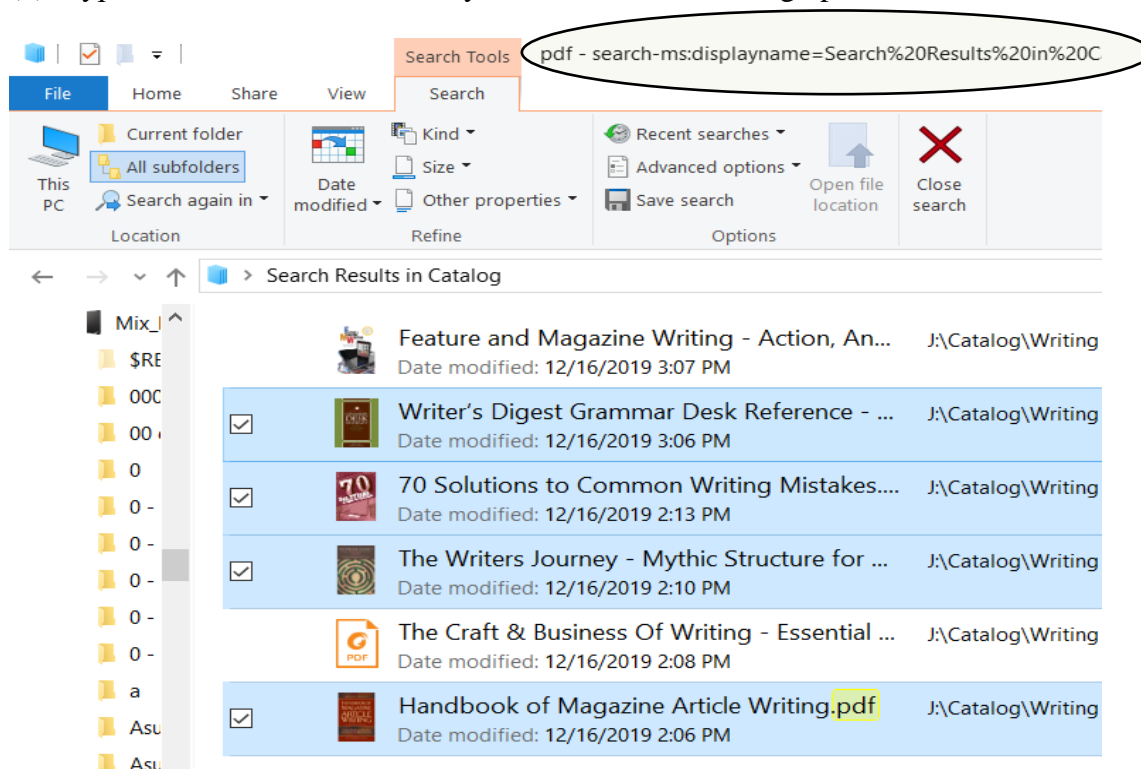
Drag and Drop from Explorer

Notes:

1. Any files/folders you drag on the window will be added to the list of files/folders already there.
2. To display ONLY those files in the Content Pane, hold down the CTRL key when you let go of the mouse button.
3. The files are not being actually Copied or Moved. They are being displayed in Bulk Rename Utility for subsequent processing.
4. This function is available for/from ANY Explorer window to BRU.
5. Windows "Search" facility can be used to locate files on your computer (for example, all your Word documents) and drag the files directly from the Search Results window. This also allows for files to be selected from **different directories** because the search results are aggregated together. The Program, 'Everything' from Void Tools is also a very good source for Searching and supports Drag and Drop into BRU as well.

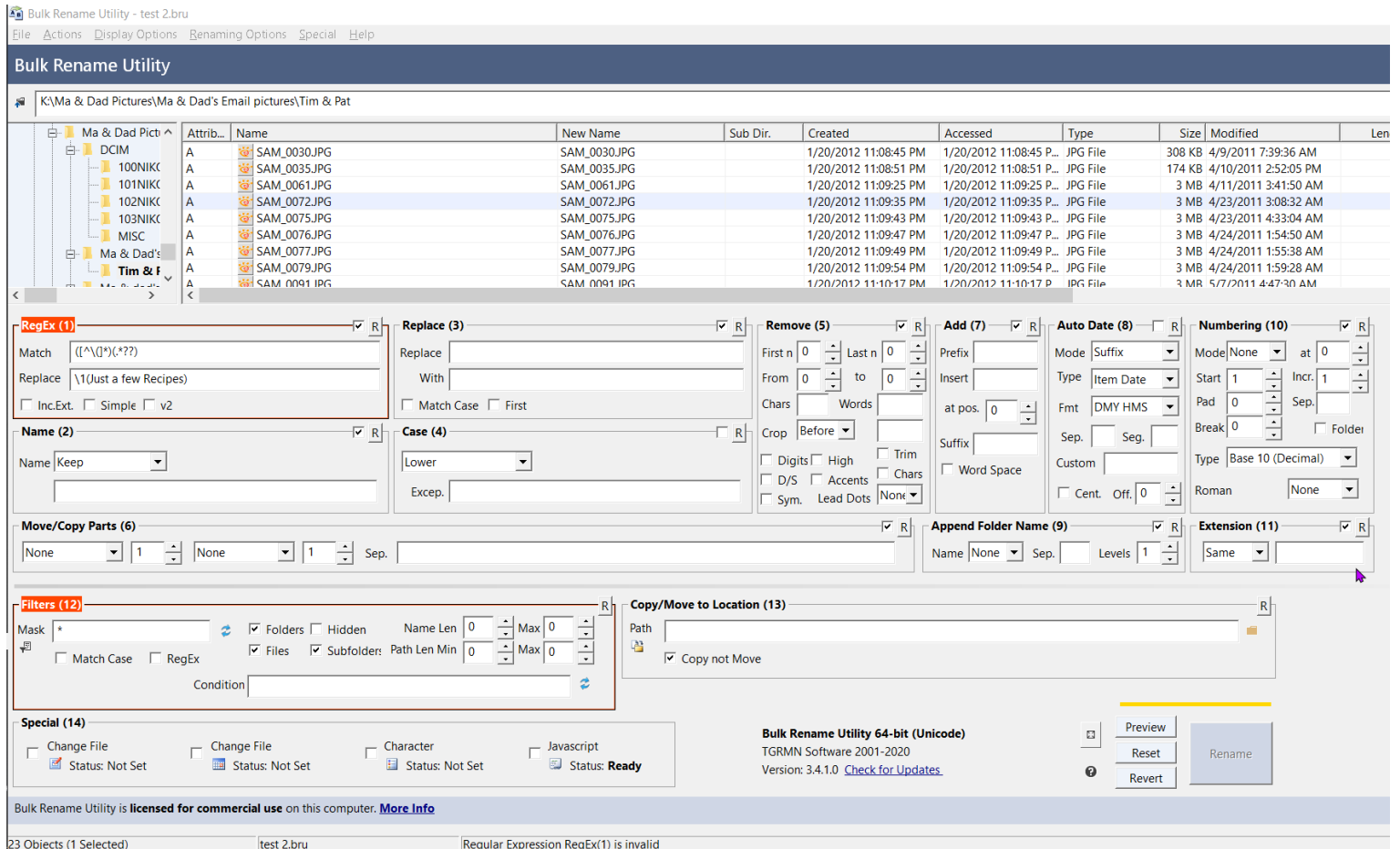
Example:

- (1). In Windows Explorer, navigate (using the Navigation Pane) to the directory you want to begin the search.
 - a. Windows search is limited to searching from within the selected Library or within the selected root directory down to the subdirectories below.
- (2). Type in the 'search' field what you want to search for e.g., pdf. 



- (3). Now you can select the files from the search results and drag them to BRU.

The Program Screen

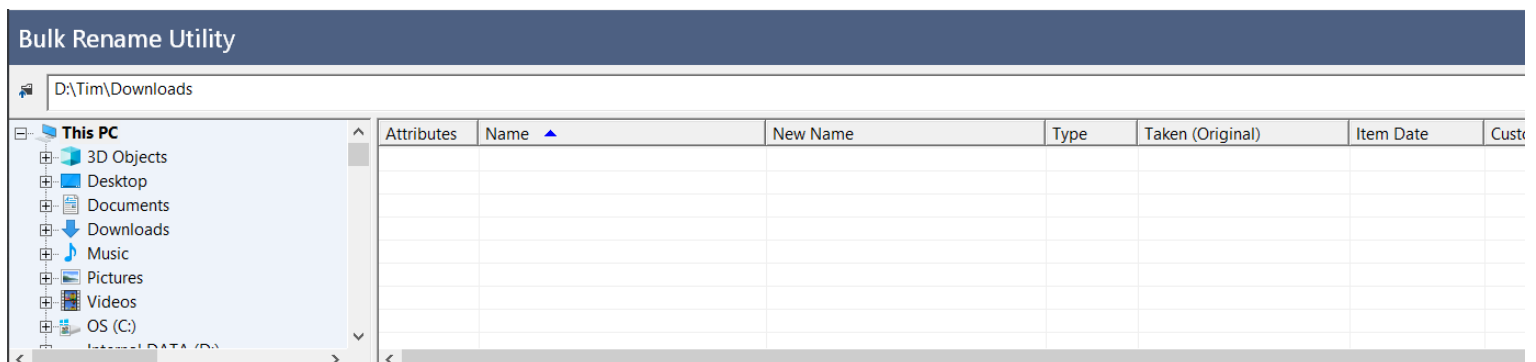


Let's take this one section at a time.

This is the Windows Explorer section. Nothing new here. You navigate the Hierarchy Tree list in the left pane (the Navigation Pane) and browse through the files content in the right pane (the Content Pane).

Navigation Pane

Content Pane



In the Navigation Pane containing the Tree, you select the folder containing files you want to rename. The Content Pane is where the file list appears. If the selected folder contains subdirectories, these folders will also appear here.

The Program Screen

I have maximized the window in this photo and expanded the columns so I can see what I am doing:

Bulk Rename Utility		Attributes	Name	New Name	Type	Tak
H:\Music\Flight of the Conchords			[LQ] Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords HQ.mp3	[LQ] Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords HQ.mp3	MP3 Audi...	
			[LQ] Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords.mp3	[LQ] Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords.mp3	MP3 Audi...	
			A Kiss Is Not a Contract.mp3	A Kiss Is Not a Contract.mp3	MP3 Audi...	
			Albi the Racist Dragon (Live).mp3	Albi the Racist Dragon (Live).mp3	MP3 Audi...	
			Albi the Racist Dragon.mp3	Albi the Racist Dragon.mp3	MP3 Audi...	
			Angels.mp3	Angels.mp3	MP3 Audi...	
			Au revoir.mp3	Au revoir.mp3	MP3 Audi...	
			Boom (Live).mp3	Boom (Live).mp3	MP3 Audi...	
			Bret, You've Got It Going On.mp3	Bret, You've Got It Going On.mp3	MP3 Audi...	
			Bus Driver s Song.mp3	Bus Driver s Song.mp3	MP3 Audi...	
			Business Time.mp3	Business Time.mp3	MP3 Audi...	
			Cheer Up Murray.mp3	Cheer Up Murray.mp3	MP3 Audi...	
			Demon Woman.mp3	Demon Woman.mp3	MP3 Audi...	
			Devil Woman.mp3	Devil Woman.mp3	MP3 Audi...	
			Do Australians Feel Love - Flight Of The Conchords.mp3	Do Australians Feel Love - Flight Of The Conchords.mp3	MP3 Audi...	
			Doggie Bounce.mp3	Doggie Bounce.mp3	MP3 Audi...	
			Fashion Is Danger.mp3	Fashion Is Danger.mp3	MP3 Audi...	
			Feel Inside (And stuff like that) - Flight of the Conchords.mp3	Feel Inside (And stuff like that) - Flight of the Conchords.mp3	MP3 Audi...	
			Femident Toothpaste - Flight Of The Conchords (Lyrics).mp3	Femident Toothpaste - Flight Of The Conchords (Lyrics).mp3	MP3 Audi...	
			Flight of the Conchords (Live).mp3	Flight of the Conchords (Live).mp3	MP3 Audi...	
			Flight of the Conchords (Think About It Live).mp3	Flight of the Conchords (Think About It Live).mp3	MP3 Audi...	
			Flight of the Conchords - Hiphopotamus vs Rhymenoceros (Live).mp3	Flight of the Conchords - Hiphopotamus vs Rhymenoceros (Live).mp3	MP3 Audi...	
			Flight Of The Conchords - Jenny.mp3	Flight Of The Conchords - Jenny.mp3	MP3 Audi...	
			Flight of the Conchords Ep 3 Hiphopotamus vs- Rhymenoceros.mp3	Flight of the Conchords Ep 3 Hiphopotamus vs- Rhymenoceros.mp3	MP3 Audi...	
			Flight of the Conchords Ep 3 Think About it 3.mp3	Flight of the Conchords Ep 3 Think About it 3.mp3	MP3 Audi...	
			Flight of the Conchords Ep 4 Sello Tape.mp3	Flight of the Conchords Ep 4 Sello Tape.mp3	MP3 Audi...	
			Flight of the Conchords Ep 7 'Mutha Uckers'.mp3	Flight of the Conchords Ep 7 'Mutha Uckers'.mp3	MP3 Audi...	
			Flight of the Conchords Ep2 Inner City Pressure 3.mp3	Flight of the Conchords Ep2 Inner City Pressure 3.mp3	MP3 Audi...	
			Flight of the Conchords Ep2 She's So Hot - Boom.mp3	Flight of the Conchords Ep2 She's So Hot - Boom.mp3	MP3 Audi...	
			Flight of The Conchords on Letterman.mp3	Flight of The Conchords on Letterman.mp3	MP3 Audi...	
			Flight of the Conchords- Business Time.mp3	Flight of the Conchords- Business Time.mp3	MP3 Audi...	
			Flight of the Conchords-Robots.mp3	Flight of the Conchords-Robots.mp3	MP3 Audi...	
			Footloose Parody.mp3	Footloose Parody.mp3	MP3 Audi...	
			Foux du Fafa.mp3	Foux du Fafa.mp3	MP3 Audi...	

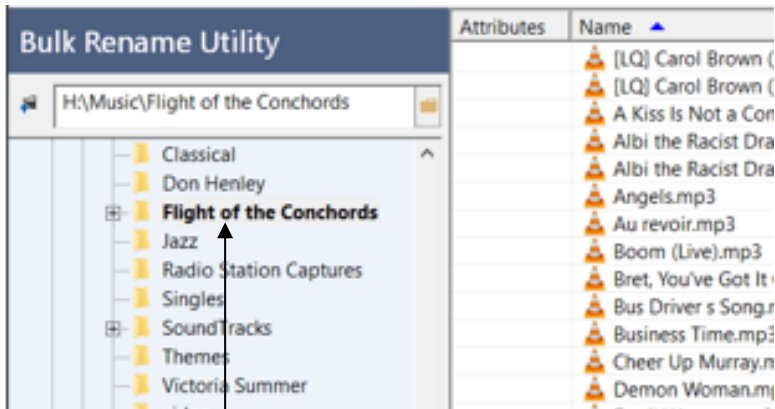
There are 14 sections evaluated from left to right in the order 1 through 14 (see photo previous page).

Sections 1 – 11, and two functions from Section 14, Character Translation and the JavaScript function, are used to set up a criteria of an inclusion or absence that build a final expression applied to each selected file when you click on **RENAME**. JavaScript requires an inexpensive license to activate.

‘**Section 12: Filters**’ is a filter that can be used to filter out what files and folders are visible in the Content Pane.

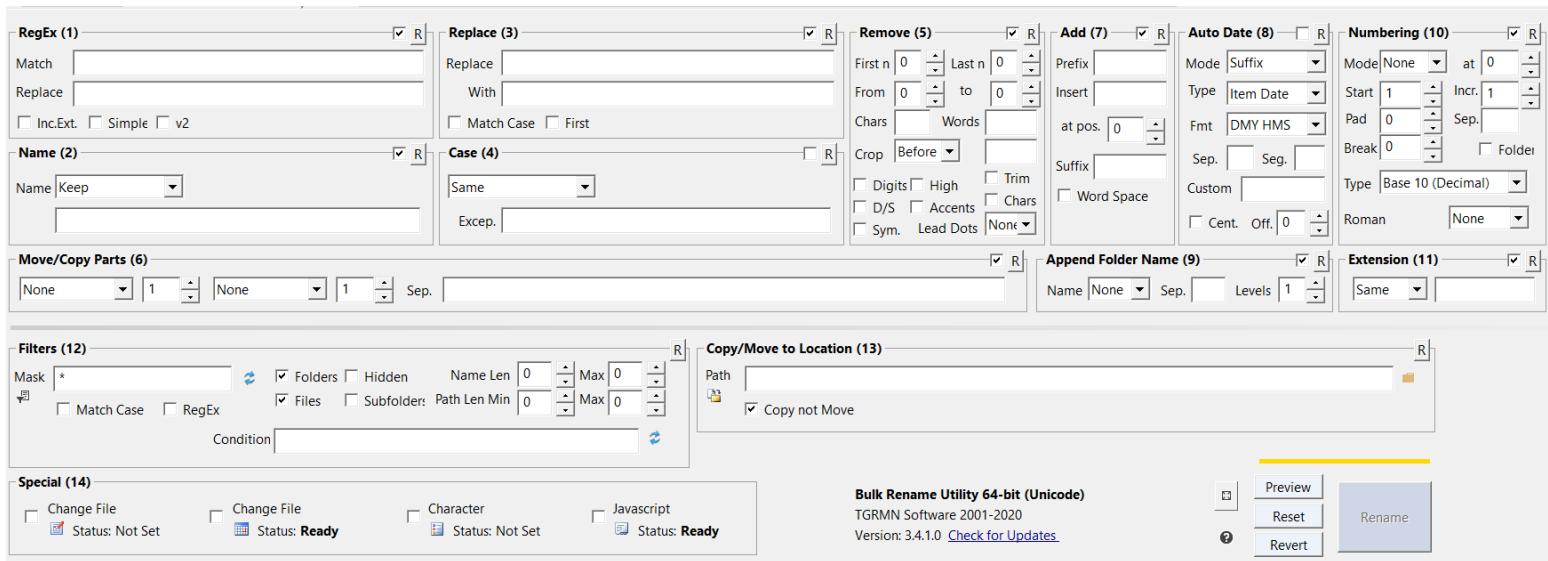
‘**Section 13: Copy/Move to Location**’, along with two functions from Section 14, File Attributes and File Timestamps, are used to affect the physical file after the renaming process.

The Program Screen



In a nutshell -

- (1). Select which directory contains the files to be processed in the Navigation Pane.
- (2). Select your criteria and file options using any one or more of the 14 sections.



The Program Screen

(3). In the Content Pane, select which files are to be processed (Ctrl + A to select all files, Ctrl + Mouse Click to individually select files, Shift + Down arrow to select files consecutively, and Ctrl + D to deselect all files).



Attributes	Name	New Name	Type	Task
[LQ]	Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords HQ.mp3	[LQ] Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords HQ.mp3	MP3 Audi...	Tak
[LQ]	Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords.mp3	[LQ] Carol Brown (Choir of Ex Girlfriends) - Flight of the Conchords.mp3	MP3 Audi...	
	A Kiss Is Not a Contract.mp3	A Kiss Is Not a Contract.mp3	MP3 Audi...	
	Albi the Racist Dragon (Live).mp3	Albi the Racist Dragon (Live).mp3	MP3 Audi...	
	Albi the Racist Dragon.mp3	Albi the Racist Dragon.mp3	MP3 Audi...	
	Angels.mp3	Angels.mp3	MP3 Audi...	
	Au revoir.mp3	Au revoir.mp3	MP3 Audi...	
	Boom (Live).mp3	Boom (Live).mp3	MP3 Audi...	
	Bret, You've Got It Going On.mp3	Bret, You've Got It Going On.mp3	MP3 Audi...	
	Bus Driver s Song.mp3	Bus Driver s Song.mp3	MP3 Audi...	
	Business Time.mp3	Business Time.mp3	MP3 Audi...	
	Cheer Up Murray.mp3	Cheer Up Murray.mp3	MP3 Audi...	
	Demon Woman.mp3	Demon Woman.mp3	MP3 Audi...	
	Devil Woman.mp3	Devil Woman.mp3	MP3 Audi...	
	Do Australians Feel Love - Flight Of The Conchords.mp3	Do Australians Feel Love - Flight Of The Conchords.mp3	MP3 Audi...	
	Doggie Bounce.mp3	Doggie Bounce.mp3	MP3 Audi...	
	Fashion Is Danger.mp3	Fashion Is Danger.mp3	MP3 Audi...	
	Feel Inside (And stuff like that) - Flight of the Conchords.mp3	Feel Inside (And stuff like that) - Flight of the Conchords.mp3	MP3 Audi...	
	Femident Toothpaste - Flight Of The Conchords (Lyrics).mp3	Femident Toothpaste - Flight Of The Conchords (Lyrics).mp3	MP3 Audi...	
	Flight of the Conchords (Live).mp3	Flight of the Conchords (Live).mp3	MP3 Audi...	
	Flight of the Conchords (Think About It Live).mp3	Flight of the Conchords (Think About It Live).mp3	MP3 Audi...	
	Flight of the Conchords - Hiphopotamus vs Rhymenoceros (Live).mp3	Flight of the Conchords - Hiphopotamus vs Rhymenoceros (Live).mp3	MP3 Audi...	
	Flight Of The Conchords - Jenny.mp3	Flight Of The Conchords - Jenny.mp3	MP3 Audi...	
	Flight of the Conchords Ep 3 Hiphopotamus vs Rhymenoceros.mp3	Flight of the Conchords Ep 3 Hiphopotamus vs Rhymenoceros.mp3	MP3 Audi...	
	Flight of the Conchords Ep 3 Think About it 3.mp3	Flight of the Conchords Ep 3 Think About it 3.mp3	MP3 Audi...	
	Flight of the Conchords Ep 4 Sello Tape.mp3	Flight of the Conchords Ep 4 Sello Tape.mp3	MP3 Audi...	
	Flight of the Conchords Ep 7 'Mutha Uckers'.mp3	Flight of the Conchords Ep 7 'Mutha Uckers'.mp3	MP3 Audi...	
	Flight of the Conchords Ep2 Inner City Pressure 3.mp3	Flight of the Conchords Ep2 Inner City Pressure 3.mp3	MP3 Audi...	
	Flight of the Conchords Ep2 She's So Hot - Boom.mp3	Flight of the Conchords Ep2 She's So Hot - Boom.mp3	MP3 Audi...	
	Flight of The Conchords on Letterman.mp3	Flight of The Conchords on Letterman.mp3	MP3 Audi...	
	Flight of the Conchords- Business Time.mp3	Flight of the Conchords- Business Time.mp3	MP3 Audi...	
	Flight of the Conchords-Robots.mp3	Flight of the Conchords-Robots.mp3	MP3 Audi...	
	Footloose Parody.mp3	Footloose Parody.mp3	MP3 Audi...	
	Foux du Fafa.mp3	Foux du Fafa.mp3	MP3 Audi...	

(4). Immediately see the results in the Content Pane under the New Name column as a preview BEFORE you process (one of the best features of this program)! Valid files to be renamed will be highlighted in green.

Attributes	Name	New Name
A	Belvedere Plantation 6490.jpg	test5.jpg
A	do.txt	do.txt

- Select All (Ctrl+A)
- Deselect All (Ctrl+D)
- Invert Selection (Ctrl+I)
- Select from Clipboard
- Jump to Path... (Ctrl+J)
- Rename Object Manually (F2)
- Refresh Files (F5)
- Refresh Tree (Ctrl+F5)
- Show/Hide Tree (F11)
- Zoom... (F8)
- List
- Import Rename-Pairs
- Debug New Name
- Reset All Renaming Criteria (Ctrl+T)
- Revert All Criteria to Last Saved (Ctrl+E)
- Preview (Ctrl+P)
- Rename (Ctrl+R)
- Undo Rename (Ctrl+Z)
- Create Undo Batchfile (Ctrl+B)

(5). Click on the RENAME button to begin processing.

Notes:

- To rename just a single file, highlight that file and press F2.
- There is even an UNDO feature under the Actions Menu.

3. The number of selected files is shown in the status bar.

Bulk Rename Utility is licensed for commercial use on this computer. [More Info](#)

2 Objects (1 Selected) | test 2.bru

User Interface

The bottom portion of the screen contains the 14 sections that are used to define the criteria and other tasks.

The screenshot displays the Bulk Rename Utility interface with 14 sections for configuration:

- RegEx (1)**: Match, Replace, Inc.Ext., Simple, v2.
- Replace (3)**: Replace, With, Match Case, First.
- Remove (5)**: First n, Last n, From, to, Chars, Words, Crop, Before, Digits, High, Trim, D/S, Accents, Chars, Sym., Lead Dots, None.
- Add (7)**: Prefix, Insert, at pos., Suffix, Word Space.
- Auto Date (8)**: Mode, Suffix, Type, Item Date, Fmt, DMY HMS, Sep., Seg., Custom, Cent., Off., 0.
- Numbering (10)**: Mode, None, at, 0, Start, 1, Incr., 1, Pad, 0, Sep., Break, 0, Folder, Type, Base 10 (Decimal), Roman, None.
- Move/Copy Parts (6)**: None, 1, None, 1, Sep.
- Append Folder Name (9)**: Name, None, Sep., Levels, 1.
- Extension (11)**: Same, .
- Filters (12)**: Mask, *, Folders, Hidden, Name Len, 0, Max, 0, Match Case, RegEx, Files, Subfolder, Path Len Min, 0, Max, 0, Condition.
- Copy/Move to Location (13)**: Path, Copy not Move.
- Special (14)**: Change File, Status: Not Set, Character, Status: Not Set, Javascript, Status: Ready.

At the bottom right, there are buttons for **Preview**, **Reset**, **Revert**, and **Rename**. The **Rename** button is highlighted in blue. Below the buttons, the text reads: **Bulk Rename Utility 64-bit (Unicode)**, TGRMN Software 2001-2020, Version: 3.4.1.0 [Check for Updates](#).

First Glance Features:

Zoom Feature –for most of the blank data fields in a section, press F8 to temporarily provide a larger view to make it easier to enter data.

The Zoom dialog box is a simple window with a title bar 'Zoom' and a close button 'X'. It contains a large text input field for entering data. At the bottom right, there are 'OK' and 'Cancel' buttons.

For each section there appears at the top right corner, a checkmark data field and an R next to it. **Remove (5)** R

The **checkmark** indicates to the program to include this section and its criteria in the final evaluation of the expression. It is okay to leave them checked (the default), because if there is no criteria specified by a section, it will be included but processed with null data. The result will have no effect and have no impact on performance.

The **'R'** will reset the values and settings for that single section back to its default state. Just click on it to reset. It will affect only that section.

At the far right lower corner are three buttons:

Three buttons are shown: **Preview**, **Reset**, and **Revert**.

Preview – Preview all the renaming actions before actually renaming (name change, timestamp change, attribute change, etc.). This is the equivalence of the 'Preview' menu item of the 'Actions' menu. For more information, refer to the 'Actions Menu' section.

Reset – Master reset to reset all values and settings for all 14 sections back to their default state.

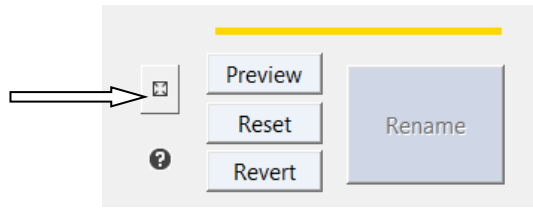
Revert – Loads the Favourites settings for all criteria previously saved using 'Save' under the File Menu. This is used to restore a Favourites file that had been loaded but had been changed during the current session.

The **Rename** button is shown, which is enabled when criteria has been set and qualified files have been selected in the Content Pane.

Rename – This will begin the Rename action. The button becomes enabled when criteria has been set and qualified files have been selected in the Content Pane.

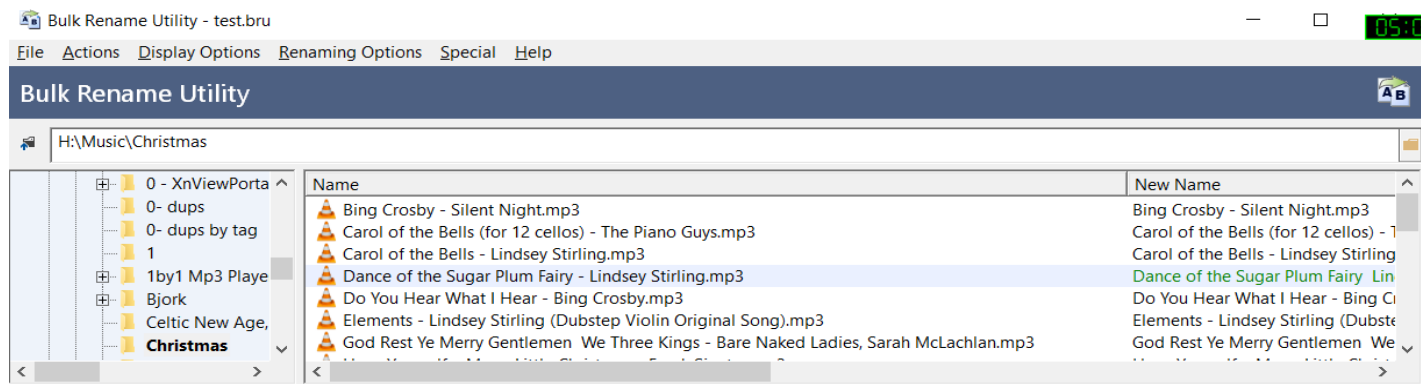
User Interface

In addition there is the Expansion button toggle for the Navigation and Content Panes:

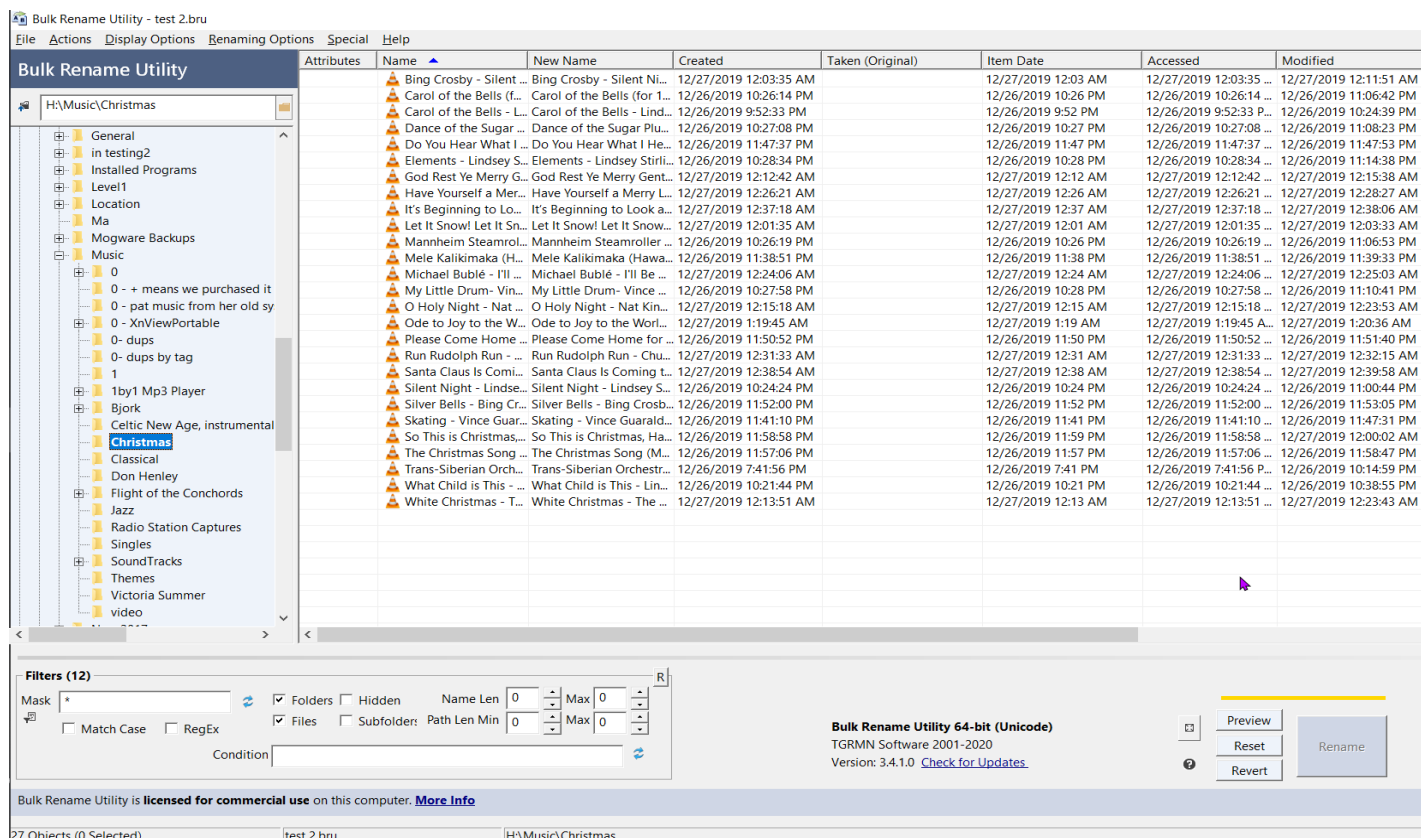


When you click on this, it expands allowing you to see more of the Tree List in the Navigation Pane and more of the File List in the Content Pane. This is the equivalence of the 'Maximize File List' selection in the 'List' sub-menu of the Display Options Menu.

From This:



To This:



With Bulk Rename Utility aka BRU, You can:

- Use Regular Expressions (**updated to v2** as of v3.4.1.0 – can use v1 (default) or v2 or the New **Simple** RegEx)
- Rename or replace using a Fixed name
- Perform text Manipulation including:
 - Replace Text with other Text (Do a Search and Replace by Pattern)
 - Add a Fixed Prefix or Suffix
 - Modify Text in the Middle of a Name
 - Move Text to the Beginning, End or within the Middle of the Name
 - Remove First or Last n characters of the Name
 - Remove Characters using Character Positions From.. To..
 - Remove a List of Characters
 - Remove All Digits or Symbols from a Name
 - Remove Double Spaces
- Change the Extension (Replace, Remove, Change Case or Add a Secondary Extension)
- Crop Text Before or After a Specified Text or Character (or from anywhere within the Name using a Wildcard)
- Add a Numbering sequence (Autonumber) to Prefix or Suffix using a pre-defined Minimum Length if Required)
- Append (Prefix or Suffix) or manipulate a Date (in various forms and formats)
 - Date Modified, Date Accessed, Date Created, (**Item Date**), or today's date
- Change the Case (with specified Exceptions)
- Extract and Use Metadata from Files Including:
 - EXIF Data from JPG Images
 - ID3 Tags (v1 and v1.1) from MP3 Sound Files
 - Windows File Properties from any File
 - EXIF v2 Metadata including Date Taken (EXIF Photo.DateTimeOriginal)
 - **Item Date** (For certain Image Files this would map to the Windows Property, System.PhotoDateTaken)
- Insert new sections within the filename
- Append the Folder Name and Path with various Directory Levels
- Create and move files into folders based on file names or file dates ('folderize')
- – see Advanced Options under Renaming Options Menu
- Perform Recursive Renaming on files located in subfolders in one action
- File Functions
 - Change the Attributes
 - Change the Timestamp
 - Move or Copy the renamed files
 - Drag and Drop support of Files from Windows Explorer and the Everything Search program by Void Tools
- Import a list of files to be renamed
- Use JavaScript (Commercial version ONLY- buy an inexpensive license today)
- Export/Import all criteria settings referred to as favourite

In addition, new features or improvements (some are in **red** type above) have been added to this version These will be explained in detail under the sub-heading, ‘ v3.4x New Additions ’, where appropriate.

[This Page Intentionally Blank]

The 14 Sections Used to Specify the Criteria

The screenshot displays the Bulk Rename Utility interface, organized into 14 numbered sections for specifying criteria:

- RegEx (1)**: Includes fields for Match and Replace, with checkboxes for Inc.Ext., Simple, and v2.
- Replace (3)**: Includes fields for Replace and With, with checkboxes for Match Case and First.
- Remove (5)**: Includes fields for First n, Last n, From, to, Chars, Words, and Crop, with checkboxes for Digits, High, Trim, D/S, Accents, Chars, Sym., and Lead Dots.
- Add (7)**: Includes fields for Prefix, Insert, at pos., Suffix, and Word Space.
- Auto Date (8)**: Includes fields for Mode, Type, Fmt, Sep., Seg., and Custom, with a Cent. Off. field.
- Numbering (10)**: Includes fields for Mode, at, Start, Incr., Pad, Sep., Break, Folder, Type, and Roman.
- Name (2)**: Includes a Name dropdown menu.
- Case (4)**: Includes a Case dropdown menu and an Excep. field.
- Move/Copy Parts (6)**: Includes dropdowns for Name and Sep., and numeric fields for 1 and 2.
- Append Folder Name (9)**: Includes a Name dropdown menu, a Sep. field, and a Levels numeric field.
- Extension (11)**: Includes a dropdown menu and a field.
- Filters (12)**: Includes a Mask field, checkboxes for Match Case and RegEx, and fields for Name Len, Max, Path Len, Min, and Max.
- Copy/Move to Location (13)**: Includes a Path field and a checked Copy not Move checkbox.
- Special (14)**: Includes checkboxes for Change File, Change File, Character, and Javascript, with status indicators (Not Set or Ready).

At the bottom right, there is a section for **Bulk Rename Utility 64-bit (Unicode)** with version information (TGRMN Software 2001-2020, Version: 3.4.1.0) and a [Check for Updates](#) link. Below this are buttons for Preview, Reset, Revert, and a large Rename button.

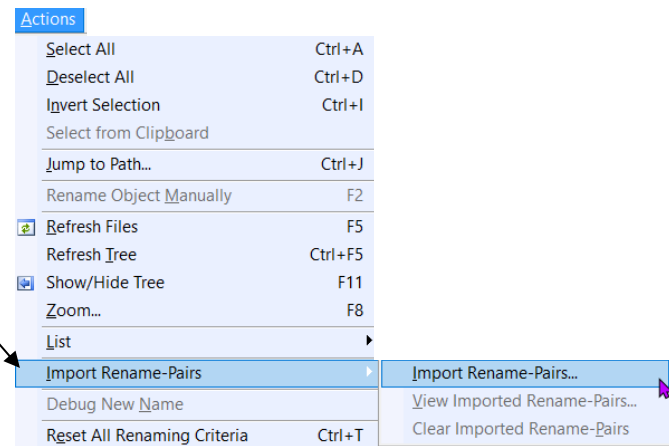
Order of Expression Evaluation

BRU has 14 sections that make up criteria you can specify to build an expression. As you build this expression, BRU is constantly working, providing a preview of the evaluation (one of the best features) allowing you to see problems before they happen. Once all of the desired criteria has been entered (you only need to select what you want to include), click on the 'Rename' button and BRU applies the finished evaluation to each of the selected files or folders.

The evaluation is processed from left to right and follows the same order as the criteria, Section #1 through Section #14 with certain exceptions noted. So for example, a *text replace* 'Section #3: Replace' will always be performed before a *change of case* 'Section #6: Case'.

The specific order of evaluation is as follows:

- 1) Apply any fixed name changes from an imported text file (Import Renamed-Pairs)
- 2) Apply Regular Expression reformatting – Section #1: RegEx

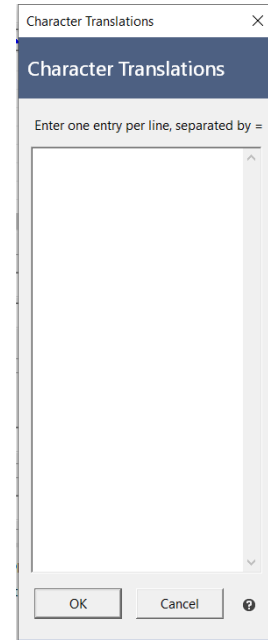
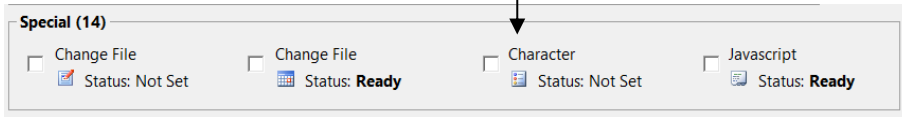


- 3) Remove any file name, or use a fixed name – Section #2: File

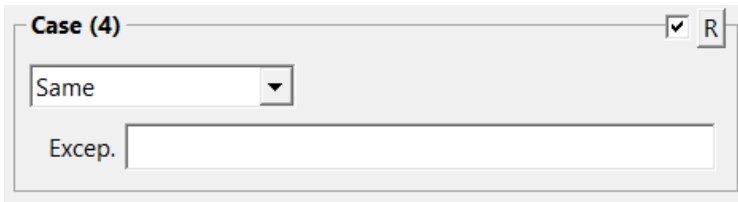
- 4) Perform any text substitutions – Section #3: Replace

Order of Expression Evaluation

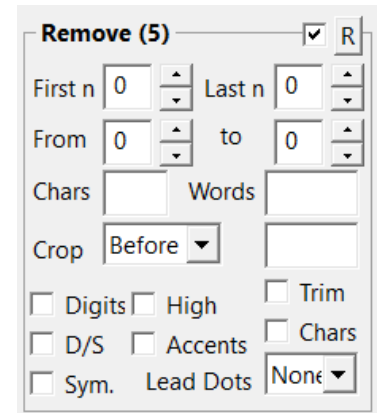
5) Perform Character Translations – [Section #14: Special – Character](#)



6) Perform any changes of case – [Section #4: Case](#)



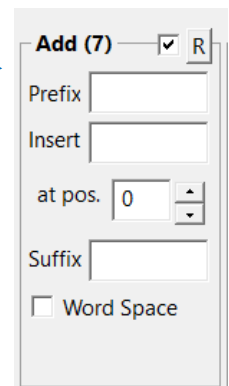
7) Remove n digits from the start, middle or end of the filename, and optionally remove certain characters, and/or all characters, and/or all digits, and/or all symbols, and/or all high-ASCII characters (128 to 256) – [Section #5: Remove](#)



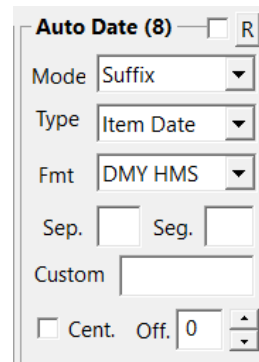
8) Move any text from/to the start, middle or end of the filename – [Section #6: Move/Copy](#)



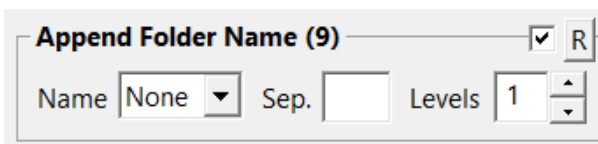
9) Add any prefixes or suffixes, or apply text to the middle of the filename. Suffixes are added at the end of the filename before any file extension – [Section #7: Add](#)



10) Apply any "Auto-Date" text as a prefix or suffix. "Sep" is the text to insert between the filename and the date; "Seg" is the separator between the day, month, year, hour, minute and second segments. Or you can use a custom date format – [Section #8: Auto Date](#)



11) Add the containing folder name as a prefix or a suffix, with a user-defined separator – [Section #9: Append Folder Name](#)




Order of Expression Evaluation

Notes:

1. Although part of **Section #14: Special**, any Character Translation will be performed (step 5 on previous page) **prior** to **Section #4: Case** (step 6 on previous page).


Be aware the Debug New Name function found in the Actions Menu does not show a distinction between the two operations, even though there are:

For example, below is the outcome from Character Translation translating the uppercase N to an uppercase P, followed by changing the CASE to Title, resulting in:

Name	New Name ▲
 DSCN0032.JPG	Dscp0032.JPG

When we view the Debug New Name output, we can see that Group 4 contains both the criteria for Character Translation, which is performed first, and Case, because the filename changes from DSCN0032.jpg to Dscp00232.jpg with no distinction between the two evaluations visible.

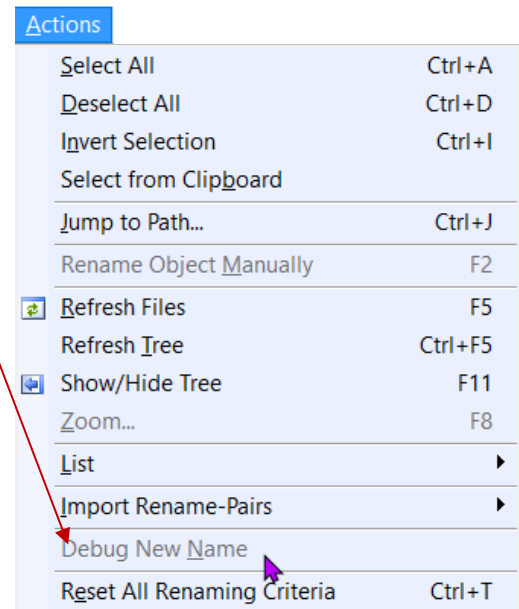
Filename Generation Debug Results

 Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1. Name=DSCN0032, Extension=.JPG
Finished Group 2. Name=DSCN0032, Extension=.JPG
Finished Group 3. Name=DSCN0032, Extension=.JPG
Finished Group 4. Name=Dscp0032, Extension=.JPG
Finished Group 5. Name=Dscp0032, Extension=.JPG
Finished Group 6. Name=Dscp0032, Extension=.JPG
Finished Group 7. Name=Dscp0032, Extension=.JPG
Finished Group 8. Name=Dscp0032, Extension=.JPG
Finished Group 9. Name=Dscp0032, Extension=.JPG
Finished Group 10. Name=Dscp0032, Extension=.JPG
Finished Group 11. Name=Dscp0032, Extension=.JPG

Final Name: Dscp0032.JPG

In Group 4 Character Translation changed the DSCN to DSCP, while Case changed the original case to Title Case, changing DSCP to Dscp.



Also note that although there are no other specified criteria other than for those two sections, the Debug New Name output still displays all of the criteria sections regardless. The reason there are only 11 sections listed is because JavaScript, if used, will display last in its own named section.

For more information, see 'Debug New Name' from 'Actions Menu' section.

Order of Expression Evaluation

Notes cont.:

2. **Section #12: Filter**, creates a mask of what files you see (the file list) in the Content Pane of BRU and is not a part of the expression built from the previous criteria.

Filters (12)

Mask: *

Match Case RegEx

Folders Hidden Files Subfolder:

Name Len: 0 Max: 0

Path Len Min: 0 Max: 0

Condition:

3. **Section #13: Copy/Move Location** will optionally copy and move the files AFTER they have been renamed and is not a part of the expression built from the previous criteria.

Copy/Move to Location (13)

Path:

Copy not Move

4. **Section #14: Special – Change File (Attributes)**, changes the selected files' attributes AFTER the renaming process and is not a part of the expression built from the previous criteria.

Special (14)

Change File Status: Not Set

Change File Status: Ready

Character Status: Ready

Javascript Status: Ready

Change Attributes

Change File Attributes

Read Only: No Change Clear Set

Archived: No Change Clear Set

System: No Change Clear Set

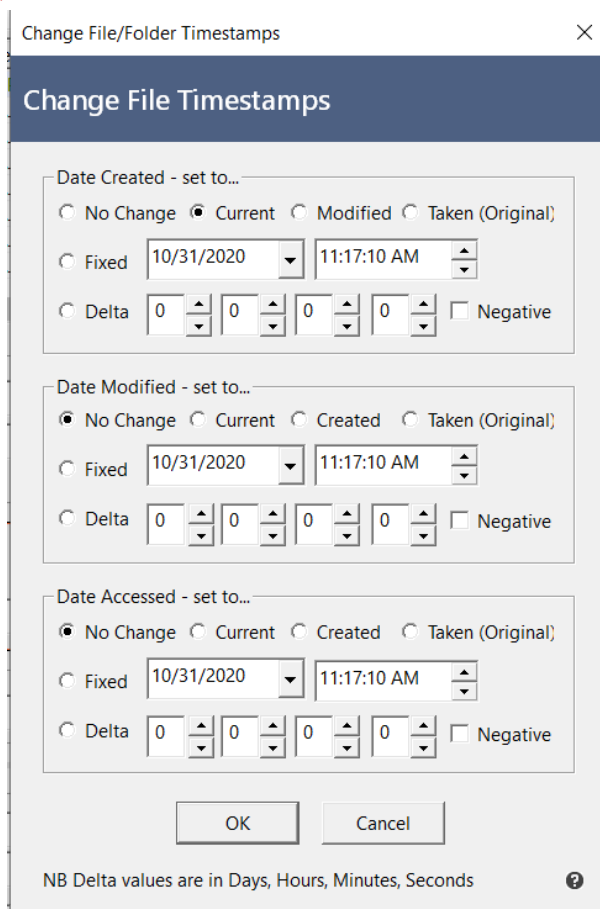
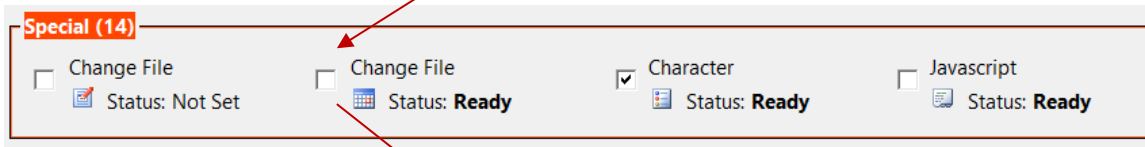
Hidden: No Change Clear Set

OK Cancel ?

Order of Expression Evaluation

Notes cont.:










5. **Section #14: Special – Change File (Timestamps)**, changes the selected files' timestamps AFTER the renaming process and is not a part of the expression built from the previous criteria.



Program Notes










- The selected files will be renamed according to your selection criteria. You can select multiple files by holding down the SHIFT for consecutive selection or the CTRL + Left Mouse for non-consecutive (individual) selection.

Consecutive Selection:

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date
A	 DSCN0032.JPG	DSCN0032.JPG	10/31/2009 11:22:48 PM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	 DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	 DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	 DSCN0055.JPG	DSCN0055.JPG	11/25/2009 8:47:30 PM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	 DSCN0056.JPG	DSCN0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	 DSCN0057.JPG	DSCN0057.JPG	11/25/2009 8:49:36 PM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	 DSCN0058.JPG	DSCN0058.JPG	11/26/2009 4:59:58 PM	11/26/2009 11:59:58 AM	11/26/2009 11:59 AM
A	 DSCN0059.JPG	DSCN0059.JPG	11/26/2009 12:00:58 PM	11/26/2009 12:00:58 PM	11/26/2009 12:00 PM
A	 DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM

Files are listed one right after the other. Click on the first file to select, then use the Shift + Down Arrow to select the files beneath. In the same manner you can use the Shift + Up Arrow to deselect files.

Non-Consecutive Selection:

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date
A	 DSCN0032.JPG	DSCN0032.JPG	10/31/2009 11:22:48 PM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	 DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	 DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	 DSCN0055.JPG	DSCN0055.JPG	11/25/2009 8:47:30 PM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	 DSCN0056.JPG	DSCN0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	 DSCN0057.JPG	DSCN0057.JPG	11/25/2009 8:49:36 PM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	 DSCN0058.JPG	DSCN0058.JPG	11/26/2009 4:59:58 PM	11/26/2009 11:59:58 AM	11/26/2009 11:59 AM
A	 DSCN0059.JPG	DSCN0059.JPG	11/26/2009 12:00:58 PM	11/26/2009 12:00:58 PM	11/26/2009 12:00 PM
A	 DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM

Individual files are selected using the Ctrl + Left Mouse Click.

Program Notes

2. If there are any problems with the rename operation, you have the option to roll back (undo) the operation.

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date
A	DSCN0032.JPG	DSCP0032.JPG	10/31/2009 11:22:48 PM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	DSCN0055.JPG	DSCP0055.JPG	11/25/2009 8:47:30 PM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	DSCN0056.JPG	DSCN0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	DSCN0057.JPG	DSCP0057.JPG	11/25/2009 8:49:36 PM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	DSCN0058.JPG	DSCN0058.JPG	11/26/2009 4:59:58 PM	11/26/2009 11:59:58 AM	11/26/2009 11:59 AM
A	DSCN0059.JPG	DSCP0059.JPG	11/26/2009 12:00:58 PM	11/26/2009 12:00:58 PM	11/26/2009 12:00 PM
A	DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM

Replace (3)

Replace

With

Match Case First

Remove (5)

First n Last n

From to

Chars Words

Add (7)

Prefix

Insert

at pos.

Actions

- Select All Ctrl+A
- Deselect All Ctrl+D
- Invert Selection Ctrl+I
- Select from Clipboard
- Jump to Path... Ctrl+J
- Rename Object Manually F2
- Refresh Files F5
- Refresh Tree Ctrl+F5
- Show/Hide Tree F11
- Zoom... F8
- List
- Import Rename-Pairs
- Debug New Name
- Reset All Renaming Criteria Ctrl+T
- Revert All Criteria to Last Saved Ctrl+E
- Preview Ctrl+P
- Rename Ctrl+R
- Undo Rename Ctrl+Z**
- Create Undo Batchfile Ctrl+B

Attributes	Name	New Name ▲
A	DSCP0032.JPG	DSCP0032.JPG
A	DSCN0001.JPG	DSCN0001.JPG
A	DSCN0029.JPG	DSCN0029.JPG
A	DSCP0055.JPG	DSCP0055.JPG
A	DSCN0056.JPG	DSCN0056.JPG
A	DSCP0057.JPG	DSCP0057.JPG
A	DSCN0058.JPG	DSCN0058.JPG
A	DSCP0059.JPG	DSCP0059.JPG
A	DSCN0064.JPG	DSCN0064.JPG

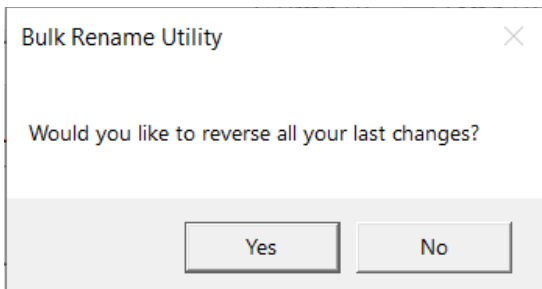
Replace (3)

Replace

With

Match Case First

Case (4)



You can also press Ctrl + Z

Program Notes

3. Remember - you can always preview the new name in the file list This allows you to refine your criteria before actually committing to the renaming process.

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date
A	DSCN0032.JPG	DSCP0032.JPG	10/31/2009 11:22:48 PM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	DSCN0055.JPG	DSCP0055.JPG	11/25/2009 8:47:30 PM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	DSCN0056.JPG	DSCN0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	DSCN0057.JPG	DSCP0057.JPG	11/25/2009 8:49:36 PM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	DSCN0058.JPG	DSCN0058.JPG	11/26/2009 4:59:58 PM	11/26/2009 11:59:58 AM	11/26/2009 11:59 AM
A	DSCN0059.JPG	DSCP0059.JPG	11/26/2009 12:00:58 PM	11/26/2009 12:00:58 PM	11/26/2009 12:00 PM
A	DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM

R **Replace (3)**

R **Remove (5)**

R **Add (7)**

Replace

With

Match Case First

First n Last n

From to

Chars Words

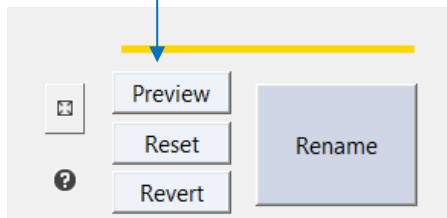
Prefix

Insert

at pos.

Select the 'Preview' option under the 'Actions Menu':

Or click on the 'Preview' Button:



or simply press Ctrl + P

Preview

Current Name	New Name	Action
DSCN0001.JPG	DSCP0001.JPG	Name Change
DSCN0032.JPG	DSCP0032.JPG	Name Change
DSCN0056.JPG	DSCP0056.JPG	Name Change
DSCN0058.JPG	DSCP0058.JPG	Name Change

Total Count: 4

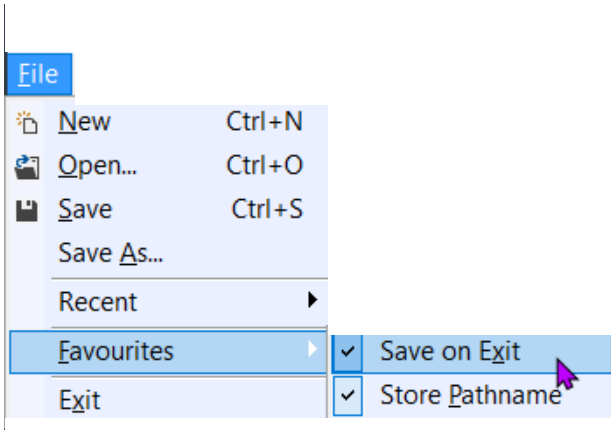
Actions

- Select All Ctrl+A
- Deselect All Ctrl+D
- Invert Selection Ctrl+I
- Select from Clipboard
- Jump to Path... Ctrl+J
- Rename Object Manually F2
- Refresh Files F5
- Refresh Tree Ctrl+F5
- Show/Hide Tree F11
- Zoom... F8
- List ▶
- Import Rename-Pairs ▶
- Debug New Name
- Reset All Renaming Criteria Ctrl+T
- Revert All Criteria to Last Saved Ctrl+E
- Preview Ctrl+P**
- Rename Ctrl+R
- Undo Rename Ctrl+Z
- Create Undo Batchfile Ctrl+B

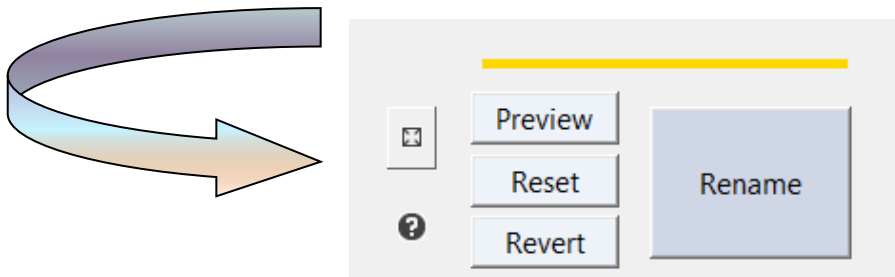
This is a read-only dialog box. You cannot make changes.


Program Notes

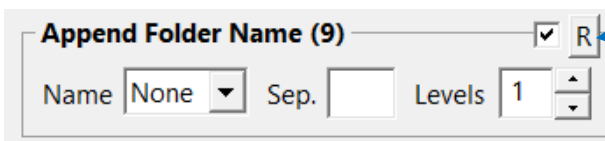
4. Favourites allow you to preserve your renaming criteria for the next time you use the utility.



5. You can use the 'Reset' button to reset all your renaming criteria back to their default values












... or the smaller 'R' buttons  located to the top right of each section to just reset one individual criteria group.



Program Notes

6. The files will always be processed in the order of the *displayed sequence* in the Content Pane.

What does this mean?

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date
A	 DSCN0001.JPG	DSCP0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	 DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	 DSCN0032.JPG	DSCP0032.JPG	11/1/2009 3:22:48 AM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	 DSCN0055.JPG	DSCN0055.JPG	11/26/2009 1:47:30 AM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	 DSCN0056.JPG	DSCP0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	 DSCN0057.JPG	DSCN0057.JPG	11/26/2009 1:49:36 AM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	 DSCN0058.JPG	DSCP0058.JPG	11/26/2009 4:59:58 PM	11/26/2009 11:59:58 AM	11/26/2009 11:59 AM
A	 DSCN0059.JPG	DSCN0059.JPG	11/26/2009 5:00:58 PM	11/26/2009 12:00:58 PM	11/26/2009 12:00 PM
A	 DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM

In the example above I have selected 4 files. The order in which they appear in the Content Pane will be the order in which each of these files is evaluated. I am not referring about the Order of Evaluation – that has to do with the specified criteria and has already been discussed.

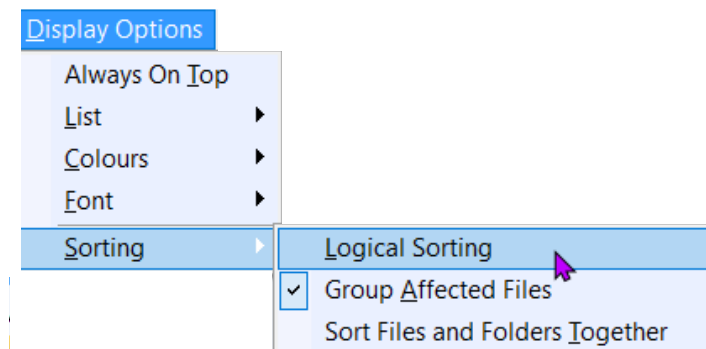
I am referring to which file will be processed first, second, third and fourth, in other words, the order of processing.

Currently the order of selection is:

DSCN0001.JPG
 DSCN0032.JPG
 DSCN0056.JPG
 DSCN0058.JPG

But there are several options that can be used to manipulate this order.

a. The sorting options available from the Display Options Menu:



Program Notes

b. The ‘Reposition’ sub-menu and ‘Apply Random Sort to Current List’ menu items from the List sub-menu option of the Actions Menu:

Actions

Select All	Ctrl+A
Deselect All	Ctrl+D
Invert Selection	Ctrl+I
Select from Clipboard	
Jump to Path...	Ctrl+J
Rename Object Manually	F2
Refresh Files	F5
Refresh Tree	Ctrl+F5
Show/Hide Tree	F11
Zoom...	F8

List

Reposition		Move Up Selected Item	Ctrl+Alt+Up
Apply Random Sort to Current List	Ctrl+8	Move Down Selected Item	Ctrl+Alt+Down
Show Only Items Affected by Renaming Criteria	Ctrl+9	Move Top Selected Item	Ctrl+Alt+PgUp
Clear All Items from Current List	F7	Move Bottom Selected Item	Ctrl+Alt+PgDn
Clear All Non-Selected Items from Current List	Ctrl+0	Swap Selected Two Items	Ctrl+Atl+S
Auto-Select All Items After Listing a Folder		Remove Selected Item(s) from List	Delete

c. The ‘Rename in Reverse Order’ menu item from the Renaming Options Menu:

Renaming Options

- Retain Autounumber
- Rename in Reverse Order**
- Prevent Duplicates

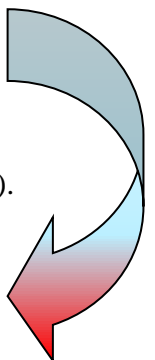
a. Sorting the Column Headers in the Content Pane ascending or descending by clicking on the Header Name:

Before – e.g., I will sort by the ‘Created’ Date column in ascending order by clicking on the ‘Created’ Date header:

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date
A	DSCN0001.JPG	DSCP0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	DSCN0032.JPG	DSCP0032.JPG	11/1/2009 3:22:48 AM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	DSCN0055.JPG	DSCN0055.JPG	11/26/2009 1:47:30 AM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	DSCN0056.JPG	DSCP0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	DSCN0057.JPG	DSCN0057.JPG	11/26/2009 1:49:36 AM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	DSCN0058.JPG	DSCP0058.JPG	11/26/2009 4:59:58 PM	11/26/2009 11:59:58 AM	11/26/2009 11:59 AM
A	DSCN0059.JPG	DSCN0059.JPG	11/26/2009 5:00:58 PM	11/26/2009 12:00:58 PM	11/26/2009 12:00 PM
A	DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM

After – (note that DSCN0058.JPG appears further down the file list and is currently not visible in the photo).

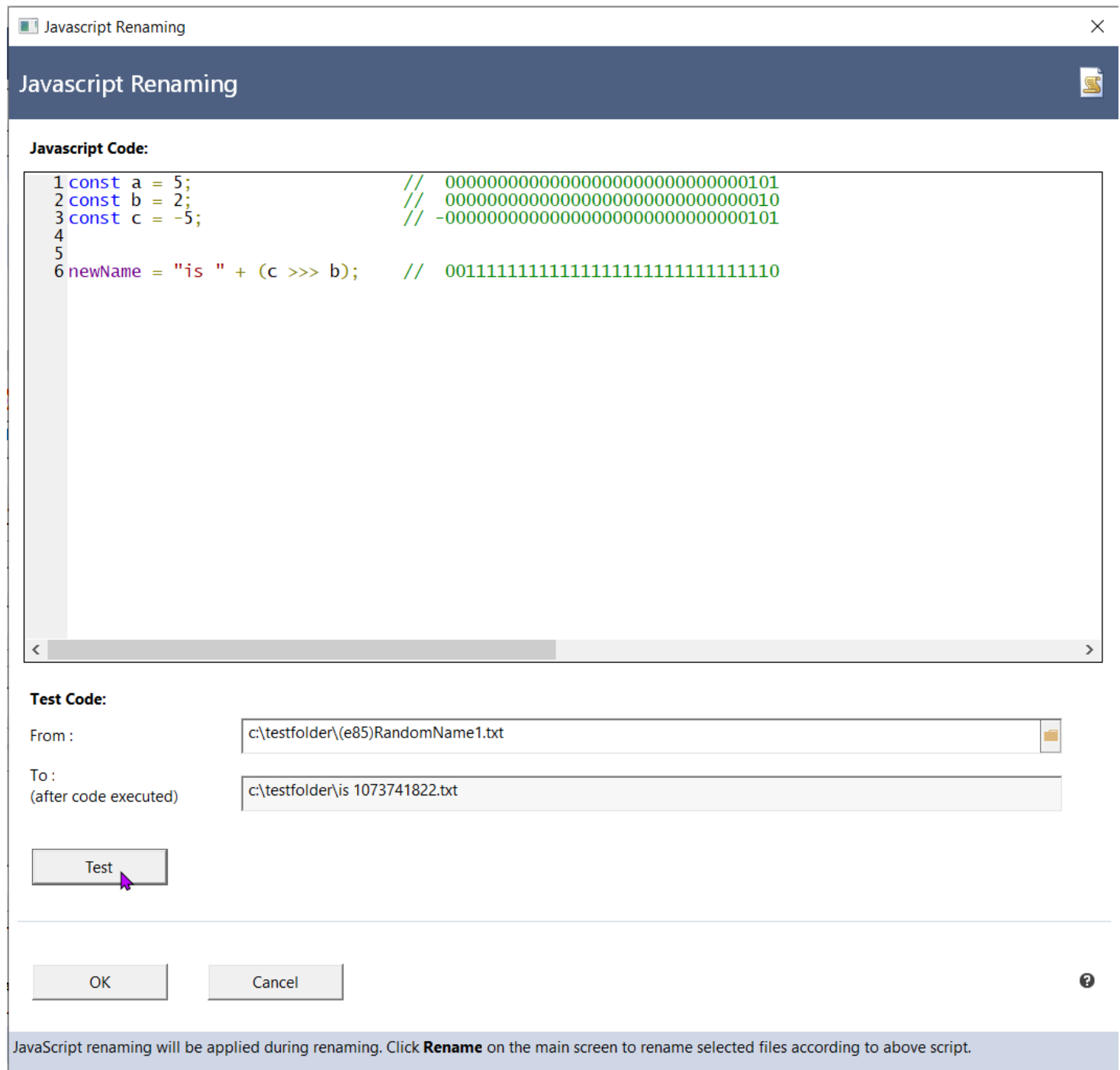
Attributes	Name	New Name	Created ▲	Taken (Original)	Item Date
A	DSCN0001.JPG	DSCP0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	FSCN0006.JPG	FSCN0006.JPG	9/10/2009 2:26:14 PM	9/7/2009 12:12:44 PM	9/7/2009 12:12 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	DSCN0032.JPG	DSCP0032.JPG	11/1/2009 3:22:48 AM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	DSCN0056.JPG	DSCP0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	DSCN0055.JPG	DSCN0055.JPG	11/26/2009 1:47:30 AM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	DSCN0057.JPG	DSCN0057.JPG	11/26/2009 1:49:36 AM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM
A	DSCN0065.JPG	DSCN0065.JPG	11/26/2009 1:30:48 PM	11/26/2009 1:30:48 PM	11/26/2009 1:30 PM



Program Notes

7. JavaScript function requires an inexpensive commercial license and supports future development. JavaScript is also beneficial for home users and unleashes the full power of BRU. Refer to Volume II for many useful examples.

JavaScript Code Entry Form:



Javascript Renaming

Javascript Code:

```
1 const a = 5;           // 0000000000000000000000000000000101
2 const b = 2;           // 00000000000000000000000000000010
3 const c = -5;          // -000000000000000000000000000000101
4
5
6 newName = "is " + (c >>> b); // 00111111111111111111111111111110
```

Test Code:

From :

To :
(after code executed)

JavaScript renaming will be applied during renaming. Click **Rename** on the main screen to rename selected files according to above script.

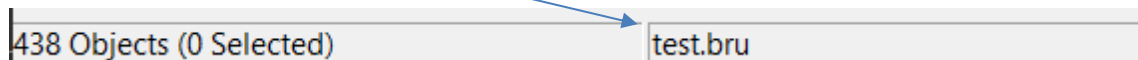
Even without a license, you can enter JavaScript into the form and click on Test to see if there are any syntax errors. With the license, this step is optional. Clicking OK will execute the script and apply the changes on the selected filenames along with any of the other renaming criteria. Remember that the actual renaming process does not take place until you click on the 'Rename' button.

Understanding Favourites

BRU allows you to save your current selection and rename criteria along with the current folder in a named file with a *.bru extension called a 'Favourite'. Using Favourites, you can define multiple renaming jobs and recall them quickly.

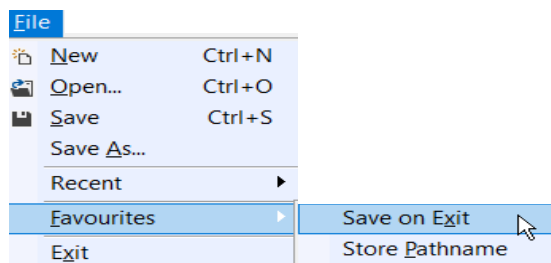
To create a Favourite file, simply set your criteria and other settings and use the 'Save' or 'Save as' menu items from the File Menu. Favourites can be recalled similarly using the 'Open' menu option from the File Menu or by double clicking on the file name in Windows Explorer.

Once loaded, all of the criteria and settings that were in place at the time the file was originally saved are automatically entered into BRU. The status bar at the bottom of the BRU program Interface will reflect the current Favourites file loaded. e.g.,



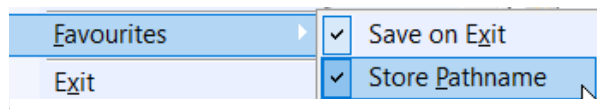
Save On Exit

You can have BRU always save any changes that you may have made during the session to the currently loaded Favourites file by setting the 'Save on Exit' option located in the File Menu:



If you do not have 'Save on Exit' enabled and changes are made, those changes will be lost.

Store Pathname



This is a misnomer. When the Favourites file is saved, it saves the **current pathname** regardless of this setting.

[General]

Path="H:\00"

[Regular Expressions]

Match="(.*)\s"

Replace="\1 \3.\5\6"

Include Extension=0

[Filename]

Mode=0

So what does this do?

Even though the pathname is saved, BRU will only use it if the 'Store Pathname' setting is enabled.

1. If the setting is disabled and you load a Favourite file, the program remains in the current directory.
2. If the setting is enabled and you load a Favourite file, the program reads in the pathname and changes to that directory.

Understanding Favourites

Notes:

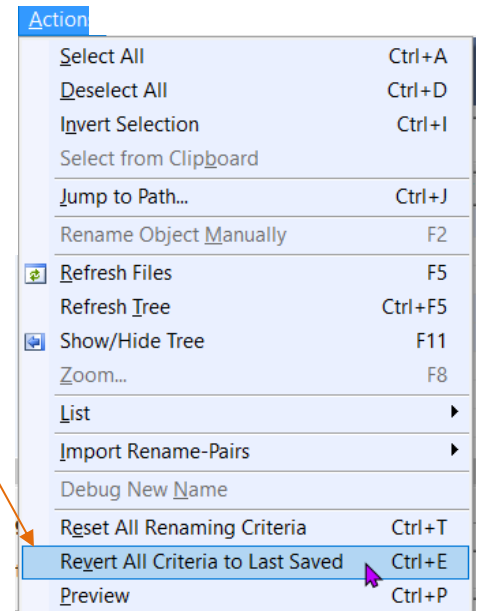
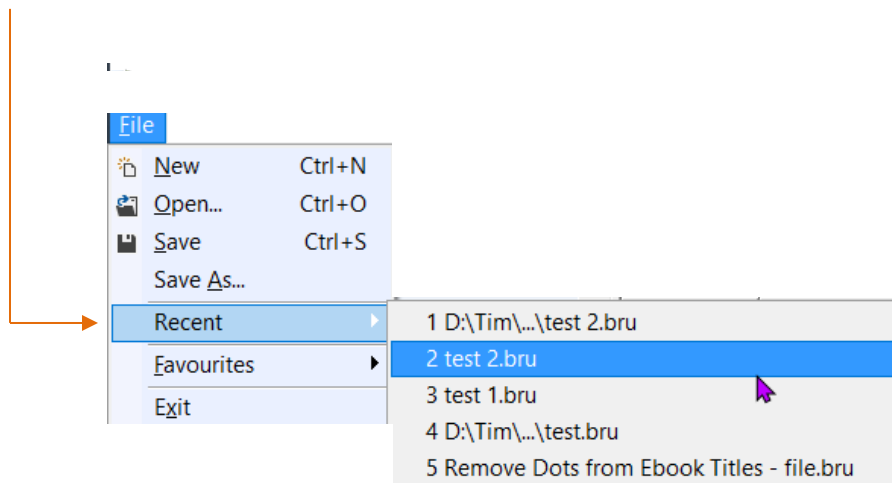
- BRU will always automatically load the last saved Favourites file when you start the program.
- If you have a Favourites file loaded and make changes, you can remove those changes by selecting the 'Revert All Criteria to Last Saved' menu option from the Actions Menu.

Note that this will not work if the changes have already been saved via –

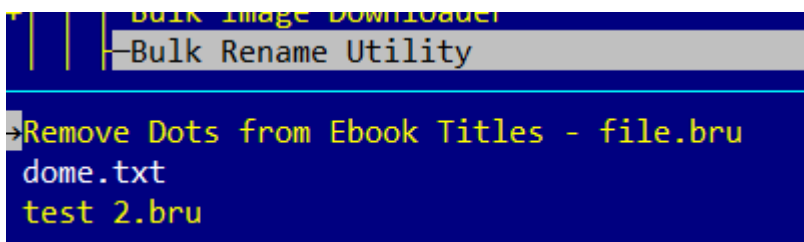
e.g.,

used 'Save', Ctrl + S, or exited the program with 'Save on Exit' set.

- The most recent loaded Favourites Files can be accessed for convenience using the 'Recent' menu option from the File Menu.



- Favourites files by default are stored in '..\Documents\Bulk Rename Utility\'.

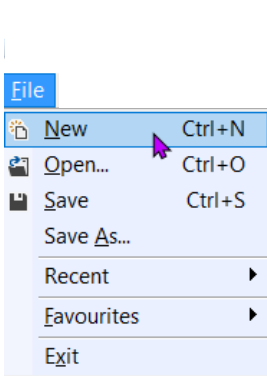


But you can place them anywhere and use the 'Open' menu item from the File Menu and recall them easily using the 'Recent' menu item. In the above example, the Favourites file, 'test 2.bru' is located in the path, D:\Tim\...\test 2.bru

Understanding Favourites

Notes cont.:

5. Selecting 'New' from the 'File Menu' will unload any previous Favourite file and 'Reset' all the criteria:



Before: The Favourites file, **test2.bru** is loaded and this displays in the status bar at the bottom of the screen. In addition, 'Section #3: Replace' is currently active (highlighted in Red) as well as the JavaScript option under 'Section #14: Special'.

Table: File List

Attributes	Name	New Name	Created	Taken (Original)	Item Date
A	DSCN0001.JPG	is 1073741822.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM
A	FSCN0006.JPG	FSCN0006.JPG	9/10/2009 2:26:14 PM	9/7/2009 12:12:44 PM	9/7/2009 12:12 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM	10/31/2009 5:20 PM
A	DSCN0032.JPG	DSCP0032.JPG	11/1/2009 3:22:48 AM	10/31/2009 5:22:48 PM	10/31/2009 5:22 PM
A	DSCN0056.JPG	DSCP0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM	11/25/2009 3:47 PM
A	DSCN0055.JPG	DSCN0055.JPG	11/26/2009 1:47:30 AM	11/25/2009 3:47:32 PM	11/25/2009 3:47 PM
A	DSCN0057.JPG	DSCN0057.JPG	11/26/2009 1:49:36 AM	11/25/2009 3:49:36 PM	11/25/2009 3:49 PM
A	DSCN0064.JPG	DSCN0064.JPG	11/26/2009 1:30:28 PM	11/26/2009 1:30:28 PM	11/26/2009 1:30 PM
A	DSCN0065.JPG	DSCN0065.JPG	11/26/2009 1:30:48 PM	11/26/2009 1:30:48 PM	11/26/2009 1:30 PM

Settings Summary:

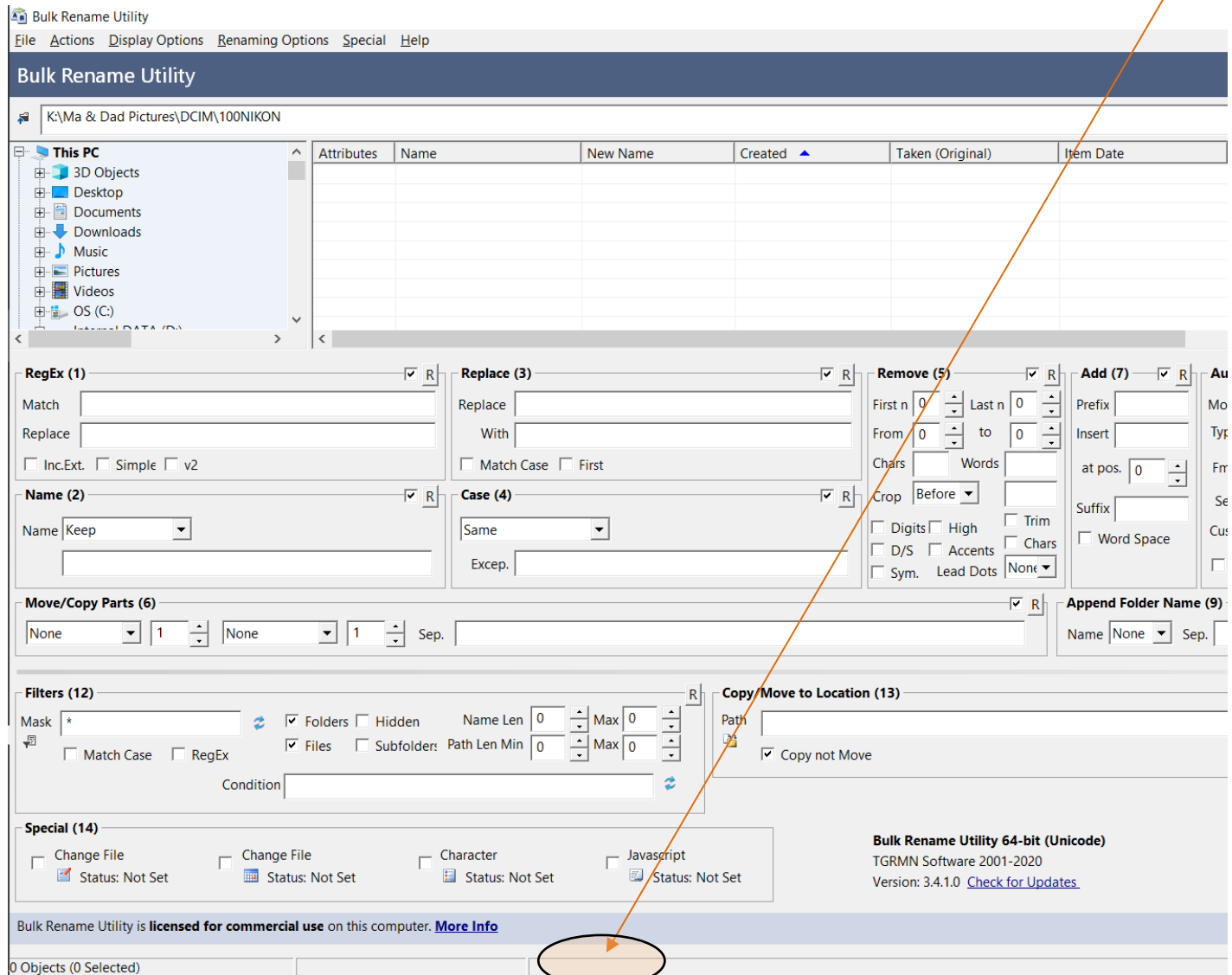
- RegEx (1):** Match, Replace, Inc.Ext., Simple, v2
- Name (2):** Name: Keep
- Case (4):** Same, Excep.
- Remove (5):** First n, Last n, From, to, Chars, Words, Crop, Digits, High, Trim, D/S, Accents, Chars, Sym., Lead Dots, Nonk
- Add (7):** Prefix, Insert, at pos., Suffix, Word Space
- Append Folder Name (9):** Name, Sep.
- Filters (12):** Mask, Match Case, RegEx, Folders, Hidden, Files, Subfolder, Name Len, Path Len, Min, Max
- Copy/Move to Location (13):** Path, Copy not Move
- Special (14):** Change File, Status: Not Set, Change File, Status: Ready, Character, Status: Ready, Javascript, Status: Ready

Status Bar: 74 Objects (4 Selected) | test2.bru

Understanding Favourites

Notes cont.:

After: All criteria has been reset (no longer highlighted in Red) and the Favourites file, test2.bru, is unloaded and no longer appears in the status bar.



The above indicates that any active criteria are now inactive and no longer highlighted and all of the data fields have been blanked, set to their default values, or set to **null** values depending on each section in question.

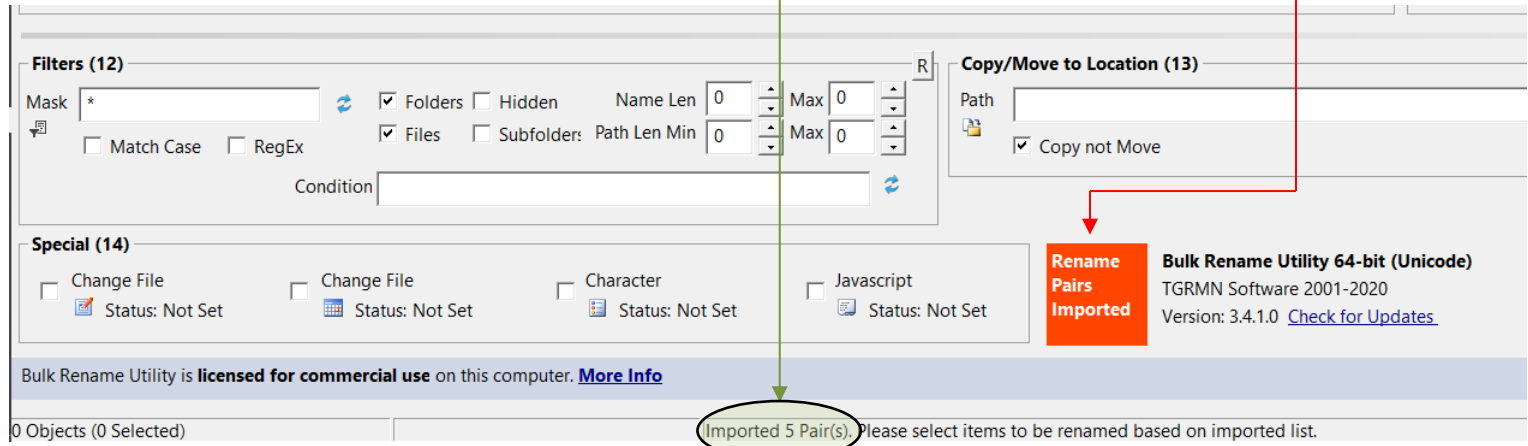
Important to note that in addition to resetting and un-enabling the JavaScript sub-section, this has also removed any JavaScript coding that was contained within the JavaScript Code Entry Form. So be careful if all you want to do is reset the Criteria, don't use 'New' but use the Master Reset instead. This will un-enable the JavaScript function without removing any current script.

Understanding Favourites

Notes cont.:

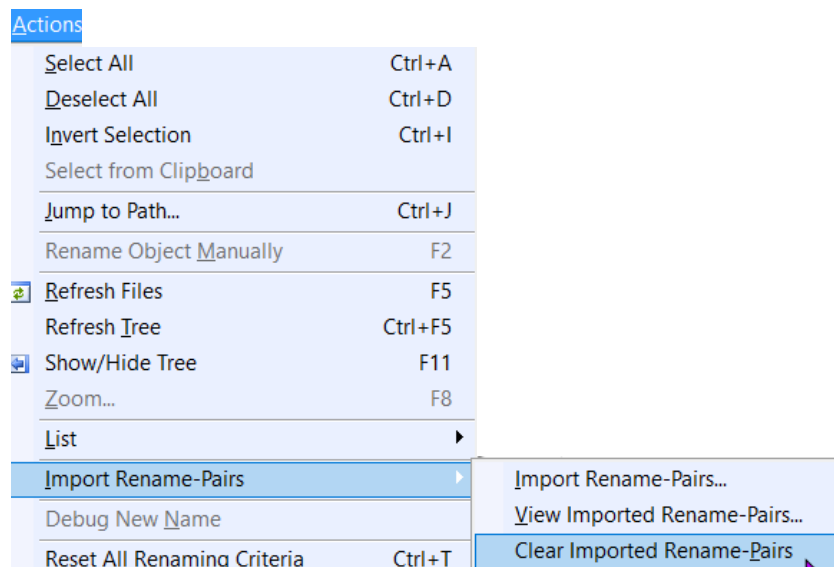
It is also important to note that 'New' has no effect on any loaded 'Imported Rename Pairs'.

In this example, a Rename Pairs file has just been loaded – this is reflected in the status bar. The Red Box indicates that the Rename Pairs Imported feature is currently active.



If you click on 'New' in the File Menu, the status of the Imported Rename Pairs will remain unchanged.

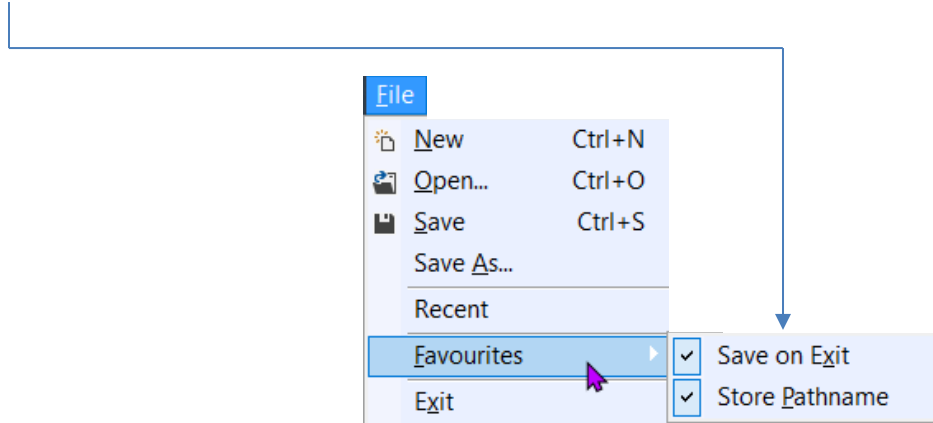
The only available option to unload the file is to use the 'Clear Imported Rename-Pairs' menu item from the 'Import Rename-Pairs' sub-menu of the Actions Menu.



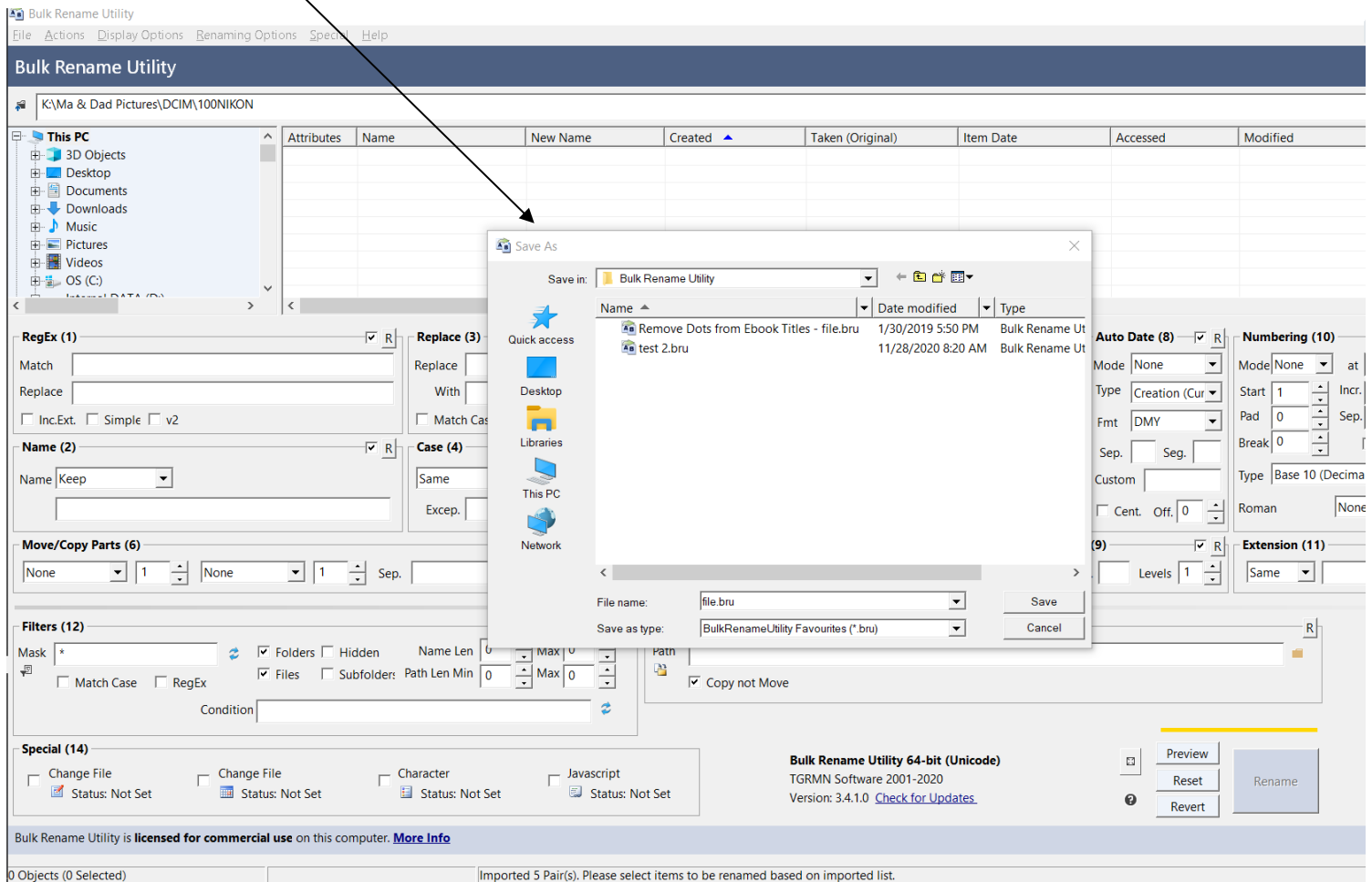
Understanding Favourites

Notes cont.:

In addition, if 'Save on Exit' is set and 'New' is still in effect (there is currently no Favourites file loaded)



... then you will be reminded to create a Favourites file upon exiting the BRU program:

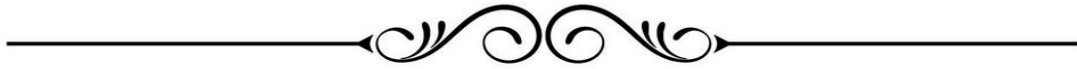


The 'Save As' dialog box pops up as a reminder that the Favourites File has not been saved.

Understanding Favourites

Notes cont.:

6. Selected files in the Content Pane are not saved as part of the Favourites file. You will need to select these again.
7. BRU will not change directories to the original directory upon loading in a Favourites file unless the 'Store Pathname' option is enabled.
8. The 'Store Pathname' is a misnomer. It should be labeled as 'Use Stored Pathname'. This has no bearing on *saving* a Favourite file, but in *opening* it. It can be activated at any time if you forget to set it beforehand. Just set it and reload the file to change directories.
9. 'New' menu item will also direct BRU to navigate to its default of 'This PC' in the Navigation Pane.
10. For more information, refer to the 'File Menu' and 'Actions Menu' in the 'Menus' section.

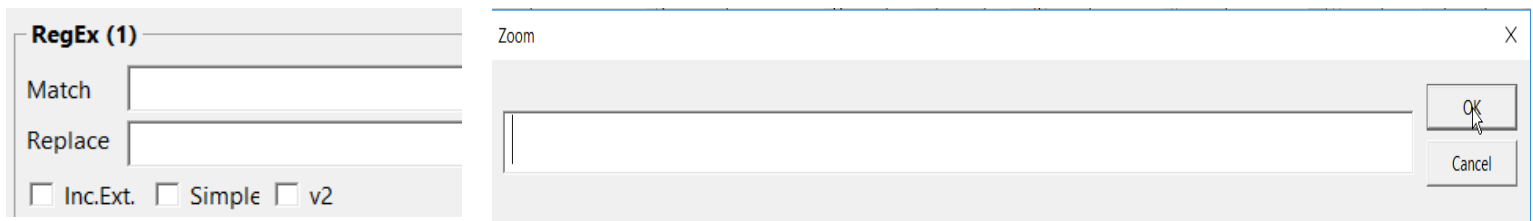


Section # 1 - RegEx

Section # 1: Regular Expressions (RegEx)

Regular Expressions - The first section is only used if you understand the Regular Expression language. If you don't, then you don't need to fill anything out. Use the other 13 sections that don't require any knowledge of Regular Expressions. If you want to learn, this section is a good start. You can further your education by looking through the RegEx manual in this volume's Appendix and go to the head of the class by studying Volume II.

RegEx can be combined with any of the other criteria. Remember to Press F8 to open a zoomed window of the data field to make data entry easier if desired. By the Order of Evaluation, RegEx is evaluated first, '**Section #1: RegEx**'.



F8 opens this window

What you are matching up against is called the Input String(s). In BRU, these will be the selected files.

A Regular Expression is a search string that uses special characters to match 'patterns' of string data. BRU uses a Match and Replace field for this purpose.

The Match data entry field will contain the Regular Expression used to test against the input string. If a match is found, then the replacement data is applied and a resulting filename is entered into the New Name column of the Content Pane.

The entire Regular Expression sequence can be summed up as:

If

Input String = pattern

Then

Replace

Where:

Input String – what is tested against the pattern, in this case, a directory or filename

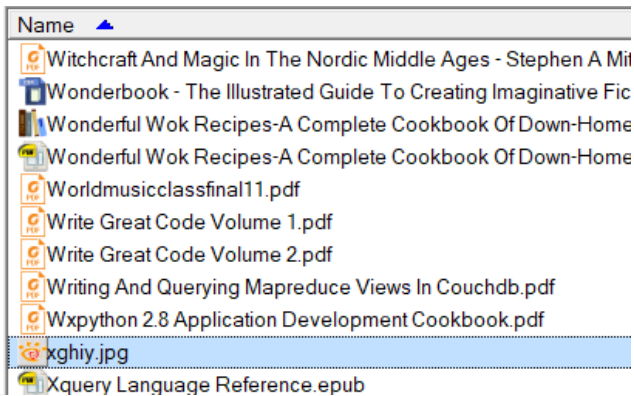
Pattern – the Regular Expression used for the match

Notes:

1. In these books, I refer to the file list of Input Strings as Sample Strings, dataset or something similar. An individual Input String is addressed as simply, String. The Regular Expression may be addressed as RegEx or Match String.

Section # 1: Regular Expressions (RegEx)

The string is the folder(s) or filename(s) displayed in the Content Pane:



In BRU, the RegEx section displays as:



Match

Match is limited to searching out what you want to replace.

1. Look at what you want to rename
2. Find the pattern (alphanumeric or whatever) of the filename and create an expression using the Regular Expression language based on that pattern.
3. Add Capture Groups by enclosing a segment of the expression in parentheses that will isolate what you want to rename from the rest of the filename.
 - a. Each Capture Group represents a sub-expression of the string. When the sub-expression is evaluated for a match, the resulting value of a successful match is saved or 'held' by that Capture Group.
 - b. Each Capture Group is given a numeric designation that can call up the substring value in the Replace data field called 'Backreferences'.
 - c. These Capture Groups are designated from the first to last with the first Capture Group designated 1, and so on.
 1. The notation used, e.g., to bring up **Capture Group 1**'s resulting value in the Replace data entry field is `\1`
 - d. The RegEx engine supports from `\1` to `\9` Capture Groups.
 - e. Not all of the expression have to be in Capture Groups. They can be outside (not enclosed within parentheses).
 - f. Resulting values of the evaluated sub-expressions outside of Capture Groups are not saved and discarded.
 - g. Using Regular Expressions allow you to save what you want to keep and discard what you don't. Saved values can be displayed in the New Name column of the Content Pane by using their Backreference designations.

Section # 1: Regular Expressions (RegEx)

Replace

Replace strings are created by indicating Capture Groups' values to include in the New Name column string. These are referenced by numeric designations based on the order they appear in the expression from left to right, called Backreferences. They use the syntax, '`\n`' where '`n`' = Capture Group number.

For example

`(..)(..)(..)`


Here are three Capture Group that would be referenced in the Replace String as `\1\2\3`.

You can include any text or punctuation in the Replace.. for example, `\1[\2]` would include Capture Group 1 and also Capture Group 2 but the string value held by Capture Group 2 would be bracketed with square brackets.

String = .. This is a test

Match: .. (This) is a (test)

Replace: \1 <space> [\2]

Name	New Name
 .. This is a test	This [test]


Another example of using RegEx in BRU:

String = xghiy.jpg

Match	<code>((x)?g(?:1)c h))</code>
Replace	<code>\1, \2</code>

a comma and <space> is added after Capture Group's 1 value

Name:

 xghiy.jpg

New Name:

xgh, x.jpg

Notes:

1. Text, punctuation, etc. are called 'literals'. Literals are any character that can be created by the keyboard.
2. I typically refer to certain characters in the RegEx sections, e.g., space, hyphen, by placing them within angular brackets. I do this especially with <space> so that you know what is there. So if you see <space>, it doesn't mean to enter it as '<space>', but enter it normally as " ". Be aware that angular brackets are part of the syntax of some RegEx, e.g., Named Capture Groups, and therefore need to be entered as is.
3. I typically highlight null in **green**. Null has an empty value.
4. Undefined refers to a value that was never set, or if it was set, it was taken away. This is different than null. Null is an assigned value of nothing. Undefined has no assigned value. Undefined appears in **red**.

Section # 1: Regular Expressions (RegEx)

Notes: cont.

5. Think of the Regular Expression language like parsing (for any of you programmers out there).
6. PCRE (Perl Compatible Regular Expressions) is a C library. The original author, Jim Willsher, points out that this implementation uses the Perl 5 Regular Expression syntax of PCRE. PCRE is considered the Regular Expression Engine. It is the PCRE Engine that performs the evaluation of the RegEx expression. BRU refers to this RegEx Engine as v1. Newer versions of BRU now support what is called v2. More on this later.
7. RegEx is the shorthand for Regular Expressions and not only references the topic but the scripts produced as well.
8. Limitations can be placed on the number of matches allowed for a sub-expression. There are several operators that fall under this called Repeat Quantifiers or just Quantifiers ...

* ? + {*min,max*} {max}

9. If a limitation on the number of matches is included as part of the pattern, then each character of the input string is tested against that pattern until the match limit has been reached or fails to find a match because the length of the string has been exceeded and exhausted.
 - a. A string that has been exceeded is at the end of the string or EOL (End of Line or End of File EOF).
 - b. A string is said to be exhausted when no more matches can be made. Evaluation ends.
 - c. A tested character is said to be ‘consumed’.
10. Regular Expressions can be simple as demonstrated by the first example, or complex as demonstrated by the second example.
11. If there is no limitation on the number of matches, then each character of the string is tested against the pattern until the length of the string has been exceeded (all the characters have been consumed).

If the EOL has been reached, any further evaluation of the next part of the expression causes the PCRE engine to backtrack – working backwards through the string to find a match. Each part of that sub-expression is evaluated against the input string in this manner.

See Backtracking section in the RegEx Manual located in the Appendix for more information.

12. If you want to learn Regular Expressions, start with the RegEx Manual that can be found in the Appendix.

My Message to those learning Regular Expressions ~

Personally, I struggle understanding Regular Expressions. By pouring over a lot of material out there, I hope to educate myself and in doing so write down my findings so perhaps it can help others. Using information obtained from the PCRE Man(ual) pages as well as from many other sources, (regular-expressions.info, autohotkey.com, etc.), I will attempt to explain in simple terms, Regular Expressions. I have also included my findings in a RegEx Manual that can be found in the Appendix of this volume as well as a continuation in Volume II.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Numbered Backreferences using the \$ Substitution Syntax

Remember all of the ‘tricks’ and ‘code optimizing’ that was used to keep the number of Capture Groups under ten because PCRE v1 only supports up to 9 Capture Groups? Ah.. the good old bad days.

Well no more.

Backreferences are either Numbered Backreferences or Named Backreferences.

Numbered Backreferences – these are the ones you are most familiar with using \1 to \9 to represent Capture Groups 1 through 9. In addition, however, using the ‘\$’ substitution syntax as a replacement for the escape character backslash, PCRE v2 supports up to 99(!) Capture Groups from \$0 through \$99.

\$0 is used when expressing all of the matches and not just the match string. A detailed discussion can be found in the RegEx manual located in the Appendix under subsection, ‘Parentheses when used with a Quantifier’, heading, ‘Metacharacters in Depth’.

1 2 3 4 5 6 7 8 9 10 11

Match: (H)(e)(l)(l)(o)() (W)(o)(r)(l)(d)/i

Replace: \$11

Produces:

New Name
D

I specified the 11th Capture Group, something not possible before.


If \$<Capture Group number>, e.g., \$6, refers to a Capture Group that doesn’t exist, it will display as a **null** but it won’t **Invalidate** the RegEx that would otherwise occur using \6 under PCRE v1. The same holds true if I were to use \6 under PCRE v2:

e.g., Under PCRE v2

String = DSCN0001-more.jpg

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

Replace: Capture Group 2 = \2 Capture Group 3 = \$3 Capture Group 6 = \$6

Name ▲	New Name
 DSCN0001-more.jpg	Capture Group 2 = SCN Capture Group 3 = 0001 Capture Group 6 = .jpg


Section # 1: Regular Expressions (RegEx)v3.4 New Additions

String = DSCN0001-more.jpg

Match: `(.*)(SCN)(\d+)(.*)`


If I change the Replacement String to have Capture Group 6 represented as a regular numbered Backreference, there is no difference under v2.

Replace: Capture Group 2 = \2 Capture Group 3 = \$3 Capture Group 6 = \6

Name ▲	New Name
 DSCN0001-more.jpg	Capture Group 2 = SCN Capture Group 3 = 0001 Capture Group 6 = .jpg

Under PCRE v1, a reference to a non-existent Capture Group would have **Invalidated** the RegEx:

Replace: Capture Group 2 = \2 Capture Group 6 = \6

Name ▲	New Name
 DSCN0001-more.jpg	Capture Group 2 = SCN Capture Group 6 = \6.jpg

This means that under PCRE v2, you may have to be more cautious because obvious errors like this can go unnoticed since BRU is not **red**-flagging it so to speak.

Notes:

1. Appears in **RED** because, if the Capture Group didn't exist but was referenced in the Replace String, then the backslash and the numeric digit would be interpreted as string literals rather than a backreference. This would display the backslash and the numeric digit in New Name. The backslash character is not a legal character in a filename, so BRU flags it as **Invalid**. The rename operation would not be able to be performed as long as the **Invalid** flag remains.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Named Capture Groups






Named Capture Group Syntax:

(?<name of Capture Group> expression to be evaluated to 'name')

Named Capture Groups help to better document sub-expressions. They also provide the means of exceeding the limitation of 99 Capture Groups imposed by the substitution syntax. But clearly, this is dependent on any limitation BRU imposes on the number of characters allowed in the RegEx data entry field, and, if it can sustain enough characters to create expressions that can handle 99 Capture Groups or more.. It's a good bet that it does. I lost track at 1,947 characters during my testing.

Example:

Sample data:

Name ▲
 DSCN0001.JPG
 DSCN0029.JPG
 DSCN0032.JPG
 DSCN0055.JPG
 DSCN0056.JPG

Match: `(.*)SCN(?<This is a test>.*)`

Replace: `$+{This is a test}`

Analysis:

- | | | |
|--------------------------|---|-----------------------------------|
| 1. (.*) | Capture to end of string.
Capture Group 1 is a normal Numbered Capture Group and if referenced in the Replace String, would be designated as \1 | Capture Group 1 = <entire string> |
| 2. SCN | Backtrack to match string 'SCN' (Not captured). | Changes Capture Group 1 = D |
| 3. (?<This is a test>.*) | Assign the name, 'This is a test', to the second Capture Group made up of the sub-expression, .*
Captures from the position after the 'N' to <u>EOL</u> .
This includes the numeric portion of each filename. | This is a test = 0001 |

Where:

(?	Begin the syntax of the Named Capture Group.
<This is a test>	Define the name that will be assigned to the Capture.
.*	Define the sub expression to be evaluated for Capture Group2.

Notes:

1. This example uses Named Capture Groups in the Match String and Named Backreferences in the Replace String.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Using Named Backreferences with Capture Groups






Named Backreference Syntax:

`${name}`

Named Backreferences are used in the Replacement String to directly reference Named Capture Groups in the RegEx.

`${Name}` is the syntax used to tell BRU to replace the text with the Named Capture Group.

In this example, ‘ `${This is a test}` ’ will replace the filename with the evaluation of the sub expression `.*` assigned to the Capture Group named, ‘This is a test’. In the photo below you can see that for each filename, the ‘DSCN’ has been removed from New Name.

Name ▲	New Name
 DSCN0001.JPG	0001.JPG
 DSCN0029.JPG	0029.JPG
 DSCN0032.JPG	0032.JPG
 DSCN0055.JPG	0055.JPG
 DSCN0056.JPG	0056.JPG

Note:






Capture Group 1 contains ‘D’. It was not used and not really needed, but I wanted to illustrate that Numbered Capture Groups and Named Capture Groups can be used together within the same RegEx.

Here’s another example using the same dataset:

Using Numbered Capture Groups in the Match String and Numbered Backreferences in the Replace String, I isolate the alpha portion of the filename from the numeric portion of the filename -

Match: `(.*?)([d]+)`

Replace: `\1 - \2`

Name ▲	New Name
 DSCN0001.JPG	DSCN - 0001.JPG
 DSCN0029.JPG	DSCN - 0029.JPG
 DSCN0032.JPG	DSCN - 0032.JPG
 DSCN0055.JPG	DSCN - 0055.JPG
 DSCN0056.JPG	DSCN - 0056.JPG

Analysis:

1. `(.*)` Capture to end of string.
2. `[d]+` Backtrack to gather up the numeric portion of string, i.e. ‘DSCN0001.jpg’ using a Class made up of `\d` along with a Greedy Quantifier, +






Capture Group 1 = <entire string>
 Capture Group 2 = 0001
 Changes Capture Group 1 = DSCN

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

The same expression using Named Capture Groups in the Match String along with Named Backreferences in the Replace String and switching the filename so that the numeric portion comes first -

Match: `(?<Alpha>.*?)(?<Numeric>[\d]+)`

Replace: `${Numeric} - ${Alpha}`

Name ▲	New Name
 DSCN0001.JPG	0001 - DSCN.JPG
 DSCN0029.JPG	0029 - DSCN.JPG
 DSCN0032.JPG	0032 - DSCN.JPG
 DSCN0055.JPG	0055 - DSCN.JPG
 DSCN0056.JPG	0056 - DSCN.JPG

You can see that using Named Capture Groups easily documents what each sub-expression is evaluating and the Named Reference Groups easily document the end result obtained.

Analysis:

1. `(?<Alpha>.*?)` Assign the name, 'Alpha', to the first Capture Group made up of `.*?`

Where:

`.*` Captures the entire string using a Greedy Quantifier Alpha = DSCN0001.JPG

`?` Makes the Quantifier lazy resulting in the RegEx Engine repositioning back to the start of the string. This creates a Zero Occurrence Match for Capture Group 1

Alpha = **null**

2. `(?<Numeric>[\d]+)` Assign the name 'Numeric' to the second Capture Group made up of `[\d]+`

Where:

`[\d]` Move forward to Capture the first numeric digit in the string, e.g., 'DSCN00001.JPG', using a class of `[\d]`. Numeric = 0
Changes Alpha = DSCN

`+` Make it Greedy (captures remainder of numeric portion of string). Numeric = 0001

BTW, If you didn't quite understand my analysis of the RegEx used with the sample data, I would please refer you to Volume II where by reading and studying the many examples, you will get a better understanding of RegEx and how it works and how you can make it work for you. In both this volume and Volume II you will also find a RegEx Manual that will help you to understand the syntax of the Regular Expression language.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Mixing Named and Unnamed Capture Groups


This next example shows how the PCRE Engine designates Capture Groups under PCRE v2

Capture Groups are determined by the PCRE Engine **by counting the number of open parenthesis** (left parenthesis) from left to right moving forward through the evaluation of the RegEx. This is true of both PCRE v1 and PCRE v2.

String - DSCN0001-more.jpg

Match: `(.*)SCN(?<This is a test>\d+)(.*)`

Replace: `Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3`


Name ▲	New Name
 DSCN0001-more.jpg	Capture Group 1 = D Capture Group 2 = 0001 Capture Group 3 = -more.jpg

The above displays both Numbered and Named Capture Groups referenced by Numbered Backreferences.

I have changed the Replace String so that it references both a Named Capture Group and a Numbered Capture Group.

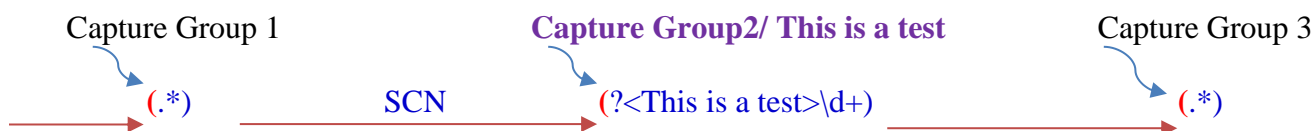
Match: `(.*)SCN(?<This is a test>\d+)(.*)`

Replace: `Named Capture Group = ${This is a test} - Capture Group 2 = \2`

Name ▲	New Name
 DSCN0001-more.jpg	Named Capture Group = 0001 - Capture Group 2 = 0001.jpg

The Numbered Capture Group, Capture Group 2 and the Named Capture Group, 'This is a test', hold the same value, '0001'. This is because the PCRE Engine designates Numbered Capture Groups first and then designates Named Capture Groups second.

To the RegEx Engine, Capture Group 2 is the same as the Named Capture Group in the RegEx because in both instances, the Named Capture Group and the Numbered Capture Group 2 appear in the **same position**, the **second** Capture Group in the RegEx.



Where:


- `(.*)` = Numbered Capture Group 1, encountered as first left parenthesis.
- `(?<This is a test>\d+)` = Numbered Capture Group 2 as well as the Named Capture Group, 'This is a test' encountered as second left parenthesis.
- `(.*)` = Numbered Capture Group 3, encountered as third left parenthesis.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

So by assigning a Named Capture Group to a Numbered Capture Group, what this does is to allow a Numbered Backreference to reference a Named Capture Group in the Replacement String without having to use a Named Backreference.

Match: `(.*)SCN(?<This is a test>\d+)(.*)`

Replace: `Capture Group 2 = \2`


Name ▲	New Name
 DSCN0001-more.jpg	Capture Group 2 = 0001.jpg

The Numbered Backreference in the Replace String was able to reference the value held in the Named Capture Group, 'This is a test'.

But can this cause confusion to the PCRE Engine? Let's find out. I will change the positioning of the Named Capture Group, by adding a Capture Group around 'SCN' in the second position in the RegEx.

Match: `(.*)(SCN)(?<This is a test>\d+)(.*)`


Replace: `Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4`

Name ▲	New Name
 DSCN0001-more.jpg	Capture Group 1 = D Capture Group 2 = SCN Capture Group 3 = 0001 Capture Group 4 = -more.jpg

So far so good...

Changing the Replacement String...

Replace: `Named Capture Group = ${This is a test} - Capture Group 2 = \2 - Capture Group 3 = \3 - Capture Group 4 = \4`

Name ▲	New Name
 DSCN0001-more.jpg	Named Capture Group = 0001 - Capture Group 2 = SCN - Capture Group 3 = 0001 - Capture Group 4 = -more.jpg

Worked fine. The RegEx Engine designated the Named Capture Group, 'This is a test', to the same value as Capture Group 3 based on the new positioning of the Capture Group in the RegEx. In other words, changing around the position made no difference. The RegEx Engine was still able to ascertain the proper order and designate accordingly.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Under PCRE v2, there are many more methods available to accomplish the same thing.

<https://www.regular-expressions.info/refext.html>

This link provides the different syntax that can be used to express a Named Capture Group. Some work and some don't – it depends on the RegEx Engine used. You can determine for yourself through trial and error. The link also allows you to select the RegEx engine that you use so it can better filter those methods for each function and allows you to compare them as well.

When making your selections, remember that BRU uses PCRE (v1), PCRE2 (v2) and Boost. Simple, though, is a proprietary flavor restricted to BRU alone so it is not represented anywhere else.


For example, here I can define a Named Capture Group using the alternate syntax:

(?'name of Capture Group')

Instead of angular brackets, the name is between single quotes.

Match: `(.*)EM('sample'.*)`

Replace: `${sample}`

Name ▲	New Name
 This is a &EM&first&EM& test.jpg	& test.jpg

Evidently, when all is said and done, you only need one method that works for what you want to accomplish. But isn't it nice to know? PCRE v1 was so limiting.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions**Size limitations in PCRE and PCRE2**

According to the documentation I have found the following (taken from online sources):

PCRE and PCRE2 have the same limits:

- (1) All values in repeating Quantifiers are limited to 65,535.
- (2) 65,535 Capture Groups (named and unnamed inclusive).

This is dependent on IF that large a number of Capture Groups can be referenced in the Replacement string using an equally large number of Backreferences. Is this supported under BRU? Under BRU PCRE v1 – definitely not, but under v2 – who knows? Untested.

- (3) 10,000 Named Capture Groups. (untested in BRU)
- (4) The default maximum depth of Nested Capture Groups is 250
- (5) The maximum length of names for Named Capture Groups is 32 code units.

where:

A char is represented by a length of code units (depending on encoding). For example –

in Unicode UTF-8 "Ç" has 2 code units: 0xC3 0x87, thus the character "Ç" has a length of 2.

Symbol	Ç	
Name	Latin capital letter c with cedilla	
Unicode number	U+00C7	
Latin-1 Supplement		
Encoding		
UTF-8	0xC3 0x87	(2 code units)
UTF-16	0x00C7	(1 code unit)
UTF-32	0x000000C7	(1 code unit)

Your milage may vary.

Multiple code units for a character only applies to Unicode because ASCII characters all have a length of one code unit (16 bit number).

For sake of simplicity, if you don't want to count code units, then just keep your names pithy. ☺

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

In BRU,

Max

(1) Numbered Backreferences = \1 - \9

This would suggest that even under PCRE v2 the maximum number of Numbered Capture Groups allowed under BRU would still be limited to 9 unless referenced by a Substituted Backreference using the syntax, \$<number> as below.

(2) Substituted Backreferences = \$1 - \$99 (\$100 not tested but I don't think it would work)

(3) Named Backreferences = In BRU - ? but could this be equal to 10,000, to match up with the maximum number of Named Capture Groups allowed under PCRE?

Section # 1: Regular Expressions (RegEx)v3.4 New Additions**Specifying Multiple Regular Expressions Using The (?X) Separator**

Until now, BRU Regular Expressions were limited to running only a single RegEx at one time. BRC, the Command Line version, was the only source for running multiple RegEx. This has changed with the release of v3.4.

Bulk Rename Utility 3.4 introduces the ability to specify multiple Regular Expressions in '**Section # 1: RegEx**'. Only v2 Regular Expressions and Simple Regular Expressions are supported. Multiple Regular Expressions can be specified with a delimiter, **(?X)**. Multiple Replacement Strings can also be specified in this same manner.

For example:

```
Match:      (S)(?X)(P)
Replace:    A(?X)R
```

will run two Regular Expressions:

first ...

```
Match:      (S)
Replace:    A
```

.. and then:

```
Match:      (P)
Replace:    R
```

If only one expression is used in the Replace field, it will apply to both matches:

```
Match:      (S)(?X)(P)
Replace:    A
```

will run two matches, first ...

```
Match:      (S)
Replace:    A
```

and then:

```
Match:      (P)
Replace:    A
```

Notes:

1. If you need to use (?X) in the regular expression itself and not as a delimiter, you can escape it with `\(?X)`.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

The following is taken from Bulk Rename Utility Operations Manual Volume II pages 224 to page 233:


String Manipulation (Parsing)**Separate Run on Names, Remove extraneous hyphens and use 'and' to join last person's name**

Match: (Modified from original)

```
^[A-Z]{2,})-([A-Z][^A-Z-]*?)([A-Z].*)$(?X)^(.*[{}][^-*])-(A-Z)[^A-Z-]*?)([A-Z].*)$(?X)^(.*[{}][^-*])-(A-Z)[^A-Z-]*?)([A-Z].*)$(?X)(.*[{}].*), (.*)(?X)(.*[{}].*), (.*)(?X)(.*[{}].*)
```

Replace: (Modified from original)

```
\1}\2 \3(?X)\1, \2 \3(?X)\1, \2 \3(?X)\1 and \2(?X)\1 and \2(?X)\1 - \2
```

Name	New Name
 UK-PeterFord-JeremyJordan-LisaWendt.pdf	UK - Peter Ford and Jeremy Jordan and Lisa Wendt.pdf

The following are the original RegEx, meant to run separately in as many iterations as required to complete the cycle. In this case, it was 6 times for this particular string. This is just an example and would not work if applied to other strings of varying lengths and run on sequences. But it does show how, what would have taken multiple runs, can be accomplished in a single run using the new capability. Note that each of the RegEx below requires a different Replacement String.

RegEx #1 – runs once

Match: `^[A-Z]{2,})-([A-Z][^A-Z-]*?)([A-Z].*)$`
Replace: `\1}\2 \3`

RegEx #2 – runs two times

Match: `^(.*[{}][^-*])-(A-Z)[^A-Z-]*?)([A-Z].*)$`
Replace: `\1, \2 \3`

Match: `^(.*[{}][^-*])-(A-Z)[^A-Z-]*?)([A-Z].*)$`
Replace: `\1, \2 \3`

RegEx #3 – runs two times

Match: `(.*[{}].*), (.*)`
Replace: `\1 and \2`

RegEx #4 – runs one time

Match: `(.*[{}])(.*)`
Replace: `\1 - \2`

Match: `(.*[{}].*), (.*)`
Replace: `\1 and \2`

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Here is another one. In this example it is removing extraneous information from the end of the string from different document titles - .pdf. .chm. .azw, .epub and .mobi. Unlike the last example, this doesn't require different Replacement Strings for each RegEx string. Instead only one Replacement String is specified and this will be used for each.

Match: `(.+)(.epub)({.*})(?X)(.+)(.pdf)({.*})(?X)(.+)(.azw)({.*})(?X)(.+)(.mobi)({.*})(?X)(.+)(.chm)({.*})`
Replace: `\1\2`

Requirements: Include extension, uses v2

Name
<input type="checkbox"/> Advanced Myofascial Techniques by Til Luchau. (z-lib.org).mobi.DocumentInfo
<input checked="" type="checkbox"/> Advanced Myofascial Techniques by Til Luchau. (z-lib.org).mobi{Rev 2020;10;05 10;32;46 AM}.mobi
<input type="checkbox"/> Advanced Myofascial Techniques Neck, Head, Spine and Ribs by Til Luchau (z-lib.org).epub.DocumentInfo
<input checked="" type="checkbox"/> Advanced Myofascial Techniques Neck, Head, Spine and Ribs by Til Luchau (z-lib.org).epub{Rev 2020;10;05 10;30;25 AM}.epub
<input type="checkbox"/> Advanced Phonics Patterns - Sound City Reading (PDFDrive).pdf.DocumentInfo
<input type="checkbox"/> Advanced Photography, Sixth Edition (PDFDrive.com).pdf.DocumentInfo
<input checked="" type="checkbox"/> Advanced Phonics Patterns - Sound City Reading (PDFDrive).pdf{Rev 2020;10;09 05;12;31 AM}.pdf
<input checked="" type="checkbox"/> Advanced Photography, Sixth Edition (PDFDrive.com).pdf{Rev 2020;09;13 10;22;46 AM}.pdf
<input type="checkbox"/> Advanced planetary magic by Jason Miller (z-lib.org) pdf DocumentInfo

New Name
Advanced Myofascial Techniques by Til Luchau. (z-lib.org).mobi.DocumentInfo
Advanced Myofascial Techniques by Til Luchau. (z-lib.org).mobi
Advanced Myofascial Techniques Neck, Head, Spine and Ribs by Til Luchau (z-lib.org).epub.DocumentInfo
Advanced Myofascial Techniques Neck, Head, Spine and Ribs by Til Luchau (z-lib.org).epub
Advanced Phonics Patterns - Sound City Reading (PDFDrive).pdf.DocumentInfo
Advanced Photography, Sixth Edition (PDFDrive.com).pdf.DocumentInfo
Advanced Phonics Patterns - Sound City Reading (PDFDrive).pdf
Advanced Photography, Sixth Edition (PDFDrive.com).pdf
Advanced planetary magic by Jason Miller (z-lib.org) pdf DocumentInfo

Same example applied to more strings:

Name
<input checked="" type="checkbox"/> 700 Victorian Ornamental Designs (PDFDrive.com).pdf{Rev 2020;09;04 19;36;34 PM}.pdf
<input checked="" type="checkbox"/> 701 Money Saving Tips - A Huge List For Frugal Living by Ron Tomby [Tomby, Ron] (z-lib.org).azw{Rev 2020;09;21 20;17;13 PM}.azw
<input type="checkbox"/> 701 Money Saving Tips - A Huge List For Frugal Living by Ron Tomby [Tomby, Ron] (z-lib.org).azw.DocumentInfo
<input checked="" type="checkbox"/> 701 Money Saving Tips - A Huge List For Frugal Living by Ron Tomby [Tomby, Ron] (z-lib.org).azw{Rev 2020;10;01 16;10;13 PM}.azw
<input type="checkbox"/> 75 Floral Blocks to Knit_ Beautiful Patterns to Mix & Match for Throws, Accessories, Baby Blankets & More (PDFDrive.com).pdf{Rev 2020;08;11 19;51;34 PM}.part
<input type="checkbox"/> 75 Floral Blocks to Knit_ Beautiful Patterns to Mix & Match for Throws, Accessories, Baby Blankets & More (PDFDrive.com).pdf.DocumentInfo
<input checked="" type="checkbox"/> 75 Floral Blocks to Knit_ Beautiful Patterns to Mix & Match for Throws, Accessories, Baby Blankets & More (PDFDrive.com).pdf{Rev 2020;08;29 10;09;14 AM}.pdf
<input checked="" type="checkbox"/> 75 Lace Crochet Motifs Traditional Designs with a Contemporary Twist, for Clothing, Accessories, and Homeware by Sainio, Caitlin (z-lib.org).pdf{Rev 2020;10;11 10;18;...}
<input type="checkbox"/> 75 Lace Crochet Motifs Traditional Designs with a Contemporary Twist, for Clothing, Accessories, and Homeware by Sainio, Caitlin (z-lib.org) pdf DocumentInfo

New Name
700 Victorian Ornamental Designs (PDFDrive.com).pdf
701 Money Saving Tips - A Huge List For Frugal Living by Ron Tomby [Tomby, Ron] (z-lib.org).azw
701 Money Saving Tips - A Huge List For Frugal Living by Ron Tomby [Tomby, Ron] (z-lib.org).azw.DocumentInfo
701 Money Saving Tips - A Huge List For Frugal Living by Ron Tomby [Tomby, Ron] (z-lib.org).azw
75 Floral Blocks to Knit_ Beautiful Patterns to Mix & Match for Throws, Accessories, Baby Blankets & More (PDFDrive.com).pdf
75 Floral Blocks to Knit_ Beautiful Patterns to Mix & Match for Throws, Accessories, Baby Blankets & More (PDFDrive.com).pdf.DocumentInfo
75 Floral Blocks to Knit_ Beautiful Patterns to Mix & Match for Throws, Accessories, Baby Blankets & More (PDFDrive.com).pdf
75 Lace Crochet Motifs Traditional Designs with a Contemporary Twist, for Clothing, Accessories, and Homeware by Sainio, Caitlin (z-lib.org).pdf
75 Lace Crochet Motifs Traditional Designs with a Contemporary Twist, for Clothing, Accessories, and Homeware by Sainio, Caitlin (z-lib.org) pdf Docu

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

And as always – is there a better way to do this? Yes, of course:

This one removes anything past the left parentheses from the end of the string.

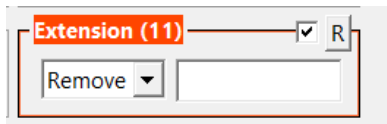
Match: `(.+)\(\(.*)`
 Replace: `\1`

Requirements: Don't include extension, Uses v1

This one removes anything past the left curly bracket from the end of the string.

Match: `(.+)\{\(.*)`
 Replace: `\1`

Requirements: Don't include extension, Uses v1 and in addition uses the 'Section #11: Extension' Criteria.



Here is a sample:

Name
1 The GIMP Toolbox How to Use GIMP Tools To Create Stunning Photos.pdf{Rev 2020;08;22 21;50;04 PM}.pdf
1 The GIMP Toolbox How to Use GIMP Tools To Create Stunning Photos.pdf{Rev 2020;08;29 10;08;50 AM}.pdf
New Name
1 The GIMP Toolbox How to Use GIMP Tools To Create Stunning Photos.pdf
1 The GIMP Toolbox How to Use GIMP Tools To Create Stunning Photos.pdf

Notes:

1. When I state 'uses v1', I am referencing that v2 is not required. If, however, v2 was enabled, it typically wouldn't disrupt the evaluation of the RegEx, although Simple may (untested).
2. I believe, and this is not verified, that just as with enabling certain features – JavaScript libraries, Extraction of Metadata, etc., performance could take a hit with v2 enabled. So my thinking is if it is not warranted, don't bother with enabling it.

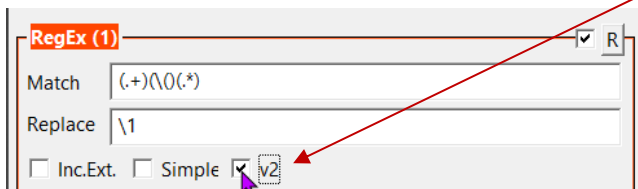
Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

v2 vs v1 & Simple

The Regular Expressions Engine you have been using up until now has been PCRE v5 Perl Compatible otherwise known as Version 1 and this engine configuration went as high to 8.x. When the Newer Library came out it started at v10.0. This was referred to as PCRE2 or PCRE v2 along with the Boost RegEx library that currently fully supports the Perl Regular Expression to the latest version, ECMAScript and JavaScript. A little confusing but as far as you are concerned, v2 provides a lot more capability than v1. Several new options in BRU, including one just discussed, the ability to use multiple RegEx on the same line, require v2 to be enabled. Other features are related to the Boost Library functionality (part of v2). At this time TGRMN hasn't updated the JavaScript library which remains at es5.

Telling BRU to use v2 instead of v1 is as simple as it gets:

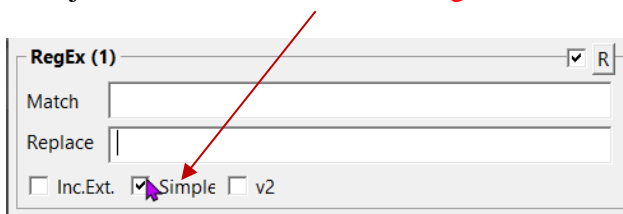


Now you can use many of the extended capabilities offered by the PCRE v2 Perl Compatible RegEx library in accordance with BRU. As to what you can or cannot use of the library functions, just as with the JavaScript capability offered through the purchase of the licensed version, it is up to you through trial and error to determine what works and what doesn't. Basically, though, TGRMN states that all of PCRE2 is supported.

Simple: Optional function of [Section #1: RegEx](#)

Speaking of Simple, there is a BRU Only Proprietary version of Regular Expressions called 'Simple'. Simple uses tags instead of the somewhat complicated Regular Expression syntax. I have attempted to make the process of learning RegEx as painless as possible with my inclusion of supplemental manuals in the Appendix of both Volume I and Volume II. In addition to this, the majority of Volume II is dedicated to learning RegEx through the use of 'programming by example', with full analysis of almost everything in the book provided.

Still, though, it can be difficult for some. Enter 'Simple'. You enable it in the same manner as you did for using v2, with just a click in [Section #1: RegEx](#).



Simple Regular Expressions work by matching text and then removing or rearranging the matched text. The syntax is very 'simple'. Up to 9 matching tags, %1, through %9 are provided which match a string of text. Perhaps uncoincidentally, this is the same limit imposed on Regular Expression matched groups under v1. It may be limited to maintain compatibility, or it may be the Simple function is a front end that still relies on the Regular Expression PCRE Engine v1. This is probably the better of the two explanations.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

You can't do anything too involved or complicated – that still requires learning Regular Expressions, but it suffices for many easy renaming operations.

Examples:

Match: %1-%2

Replace: %2-%1

This will match a string of text with %1 followed by the <hyphen> character, '-', followed by another string of text.

The Replacement String will replace the original string with the second string of matched text, the <hyphen> character, and the first string of text matched.

Effectively, it will switch the text around the literal character '-' <hyphen>.

Attrib...	Name	New Name
A	Begin-SAIL	SAIL-Begin

In the above example, the first part of the string is matched and 'gathered' (captured) as (into) %1 until the <hyphen> is encountered. The <hyphen> is acting as a delimiter. When this occurs, the text value of %1 is saved (matched successfully). The <hyphen> is not. Only the tags referenced by %1 through %9 are allowed to save values. Any characters outside of these tags are not saved. This is why the <hyphen> has to be specified in the Replacement String if the user wants to preserve the <hyphen> in the finished renamed string.

What happens if there are multiple <hyphens> in the string? (the following refers to version of BRU prior to v3.4.20):

Name	New Name
ABC-123-45	45-ABC-123

In that case, it acts more like the Regular Expressions of v1.

%1 is matched to the first <hyphen> that follows after 'ABC' but because there is a second <hyphen>, it continues to match %1 to the second <hyphen>. This results in %1 having a value of 'ABC-123' and not 'ABC'.

%2 then takes the remainder of the string to the end (in the Regular Expression language, the end is referred to as EOF or End of File, but it is also called EOL or End of Line). %2 has a resulting value of '45'. The Replacement String ends up as:

%1 = **ABC-123** %2 = **45**

Ending String = %2-%1 or 45-ABC-123

Why did the first <hyphen> save when I just got done stating that anything outside of the tags are not? Because the <hyphen> is *not* outside of the %1 tag but *included* as part of it. It became included when the match extended to the second <hyphen>.



Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

In February 2021, TGRMN changed the Simple RegEx in v3.4.20 by making the default ‘short’ instead of ‘long’.

Taken from Change log: ‘Changed ‘Simple’ Regex behaviour from longest match to shortest match.’

Now what do they mean by that? This has to do with Greedy vs Lazy. I have numerous examples throughout Volume II and a full explanation can also be found in the RegEx Manual in the Appendix of this volume of what Greedy and Lazy are, but that has to do with Regular Expressions and the whole idea of the Simple functionality was to not bother with learning Regular Expressions. I can see their point.

When v3.4 came out, Simple used Greedy for all tag matches.

To best illustrate what I mean, I will take the example from the previous page and once more take up the question:

What happens if there are multiple <hyphens> in the string?

String = ABC-123-45

Match: %1-%2

Replace: %2-%1

Name	New Name
<input type="checkbox"/> ABC-123-45	45-ABC-123

1. %1 gathers up to the second <hyphen> = ABC-123
2. %2 gathers up the remainder of the string = 45

The <hyphen> is ‘kept’ as part of ‘ABC-123’ because it was ‘gathered’ up along with the ‘ABC’. It extended past the first <hyphen> and ‘gathered up’ until the second <hyphen>.

This was referred to by TGRMN as the ‘longest match’. In Regular Expressions it is called ‘Greedy’, meaning it gathers up as much as possible to match. In this case it means capture everything until the last <hyphen> encountered in the string.

Match String %1 - %2

String A B C - 1 2 3 - 4 5

Because the <hyphen> was specified in the Match String of %1 - %2 it remains outside of the tags:

 ↓ ↙

 %1 - %2

 Match 1 Match 2

.. it would not be included when it was encountered as the **last** <hyphen> in the sample string, ‘ABC-123-45’; therefore, %2 = 45 and not -45.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

The Replacement String put the name back together but reversed the two elements with %2' s value in the first placement of the new filename and %1's value in the second placement within the new filename. The <hyphen> was added back as a literal character, a character that can be created from the keyboard, to act as a delimiter (separator) between the two values.

%2 - %1

New Name = 45 - ABC123

TGRMN felt it was defeating the purpose of Simple by having all of the tags gather up as much as possible. Simple is only supposed to deal with those parts of the string that are implicitly expressed by the tags with the exception of the **last tag** in the Match String that **remains Greedy** (long). Any tags prior to the last will be **Lazy** (short).

In Regular Expression, or RegEx, the equivalent would be:

Match: (.*)-(.*)

Replace: \2-\1

Name ▲	New Name
ABC-123-45	45-ABC-123

1. The .* means to gather up or 'capture' everything in sight and store in (Capture Group 1) until the last encounter of the <hyphen>.
2. Do not include the <hyphen> because it is outside of the Capture Groups indicated by the parentheses.
3. The final .* means to gather the rest of the string and store it in a second Capture Group, (Capture Group 2).
4. Finally, the Replace String is similar to the one used in the Simple example, except that Capture Groups are designated as \1 and \2 instead of the tags %1 and %2.

But that is exactly what Simple is not supposed to do – require you to learn Regular Expressions unless you want to. Simple is supposed to be... well, Simple.

TGRMN instead wants:

%1 - ←

To mean gather up until the first encounter of **any** <hyphen> and stop – the Shortest match, or what is called Lazy. Lazy is just what it is – do as little work as possible to get the job done.

while,

%2

Is not changed in its purpose, only the value has changed. %2 will still gather up any remainder of the string including the second <hyphen>. The last tag in this example, %2, remains Greedy or in Simple terminology, the 'Longest Match'.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Breaking down the Simple syntax, each tag represents one part of the string. It could be a single character or more depending on:

1. The number of tags to match against the number of characters in the string.
2. If a match is made outside of the tag, that would limit how many characters of the string could be gathered.
3. The **last tag** of the Match String or 'equation' if you will allow, remains **Greedy**, the **Longest Match**, **all tags prior** to this are **Lazy**, the **Shortest Match**, as of v3.4.20.

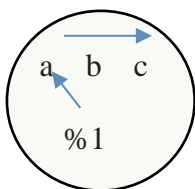
What do I mean by this? Let's take it step by step.

1. The number of tags to match against the number of characters in the string.

String = abc

Match %1
 Replace Group 1 = %1

Name ▲	New Name
abc	Group 1 = abc



Where:

%1 = abc

There is only one tag so %1 = entire string (last tag is always Greedy). It starts by gathering the 'a', and because there are no other tags, it continues to gather until the end of the string sample (in Regular Expressions, this is called the End of Line or EOL).

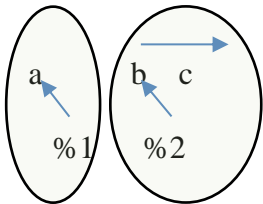
Section # 1: Regular Expressions (RegEx)v3.4 New Additions

String = abc

Match %1%2

Replace: Group 1 = %1 Group 2 = %2

Name ▲	New Name
abc	Group 1= a Group 2 = bc



Where:

%1 = a %2 = bc

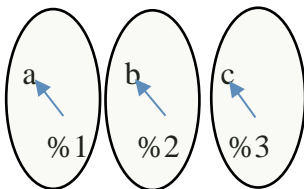
This time %1 tag matches the first character and %2 matches the second character to the end of the string. Each tag will match to the first match encountered and with the exception of the last tag, limit that match to as few characters possible. That makes it the shortest match or 'Lazy'. There are three characters that make up the sample string and two tags. That means that each tag will gather at least one character, the first encountered, leaving any additional character cleanup for the next tag. Tag %1, gathers up the 'a', leaving tag %2 to cleanup the rest.

String = abc

Match %1%2%3

Replace: Group 1= %1 Group 2 = %2 Group 3 = %3

Name ▲	New Name
abc	Group 1= a Group 2 = b Group 3 = c



Where:

%1 = a %2 = b %3 = c

Because there is an equal number of tags to an equal number of characters in the string, each tag is matched to a single occurrence of each character. The last tag, %3 would be left to cleanup any remainder of the string. Since there are no other characters beyond the 'c' of 'abc', %3 matches to a singular value.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

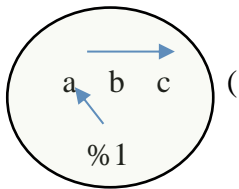
2. If a match is made outside of the tag that would limit how many characters of the string could be gathered.

Next I will introduce a character that will be outside of the tags and observe how Simple behaves.

String = abc(

Match: %1(
Replace: Group 1= %1

Name ▲	New Name
abc(Group 1= abc



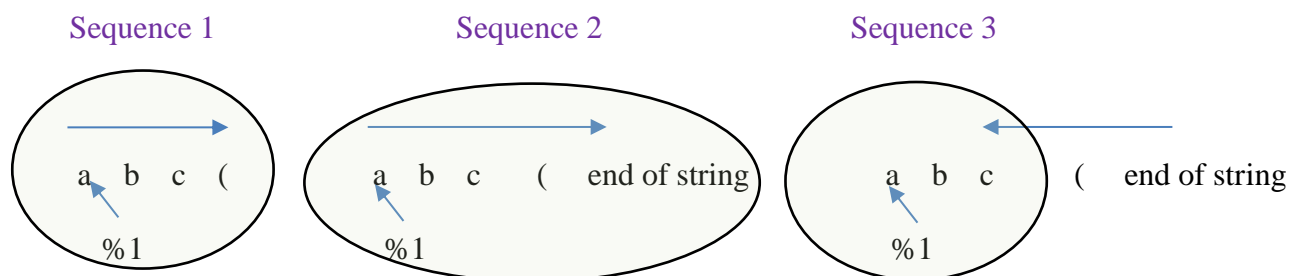
Where:

%1 = abc

Because the left parenthesis is outside of the tag, it is not included as part of the tag's returned value. The only tag in the Match String becomes both the first and last tag, so it is Greedy and will gather up as much of the string as allowed. This value will not include the left parenthesis for the reasons already stated.

What is really happening in the background that you are not aware and probably don't care when talking about Simple will become important later on if you select to use Simple for more than just reversing or moving string portions around.

In [Sequence 1](#) in the diagram below, %1 is the only tag in the Match String and thereby Greedy. It gathers up any characters that match, and this INCLUDES the left parenthesis. In [Sequence 2](#) It continues to move through the string looking for any additional matches until it finds no more and reaches the end of the sample string. In [Sequence 3](#) it looks for a left parenthesis and moves backwards to search (Backtracking). When it matches, it will 'spit out' the left parenthesis previously gathered as being outside of the scope of the %1 tag. In Regular Expressions terminology this is called giving up the match or giving up characters of the match.



This will become clearer later. For now let's proceed as before.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

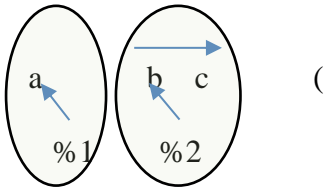
Now I will introduce a second tag.

String = abc(

Match: %1%2(

Replace: Group 1= %1 Group2 = %2

Name ▲	New Name
abc(Group 1= a Group2 = bc



Where:

% 1 = a % 2 = bc

Tag %1 has a singular value of the first match, 'a', Lazy, and %2, Greedy gathers up the remainder of the string up to the specified left parenthesis character that is outside of %2 and therefore not included as part of the value.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

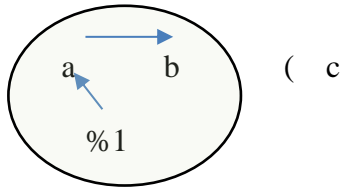
I will change the sample, by moving the left parenthesis after the 'b' of 'abc' and begin with a single tag.

String = ab(c

Match: %1(

Replace: Group 1= %1

Name ▲	New Name
ab(c	Group 1= ab



Where:

%1 = ab

The value of the %1 tag matches to the 'a' and because there are no other tags, it is Greedy and proceeds to gather up the string sample characters until the match of the left parenthesis. Because the left parenthesis is outside of the %1 tag, it is not included as part of the value. In addition, however, the 'c' of 'abc' is never evaluated (considered) because it is also outside of the purview of %1, resulting in %1's value of 'ab'.

FYI Notes:

1. The left parenthesis is also called the Open Parenthesis and the right parenthesis is also called the Closed Parenthesis.
2. Gather in Regular Expression terminology is 'Capture'
3. The phrase used to 'spit out' a literal character is in reference to Regular Expression terminology where the Match is given up or characters from the match are given back.
4. If interested in learning more about Regular Expressions, please refer to the sections on RegEx with additional material that can also be found in the RegEx Manuals found in both the appendix of this volume and in Volume II.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

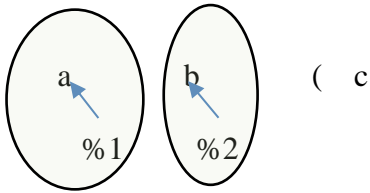
Using the same string sample, I will add the second tag, %2.

String = ab(c

Match: %1%2(

Replace: Group 1= %1 Group2 = %2

Name ▲	New Name
ab(c	Group 1= a Group2 = b



Where:

% 1 = a % 2 = b

The first tag, %1, lazily gathers up the first character because it will always match against the first character and not have to take on any extra work if it can be helped. Since there is a second tag in the Match String, the %2 tag will take on the second character, and seeing that there are no other tags, it Greedily has no choice but to continue gathering up the string, of which there are no more characters because it is halted by the match of the left parenthesis character that is outside of the %2 tag. Once more the 'c' is never considered because it is also outside of the %2 tag.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

3. The last tag of the Match String remains Greedy, all tags prior to this are Lazy.

String = ab(c

Match: %1(%2

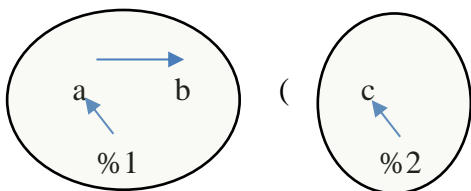
Replace: Group 1= %1 Group2 = %2

Name ▲	New Name
ab(c	Group 1= ab Group2 = c

Making a slight change, moving the left parenthesis outside of the %1 tag instead of the %2 tag, changes the returned value of %1 to 'ab' because now it has to do more work and gather up to the first (and only thus far) match of the left parenthesis. The second tag, %2, will gather up any remainder of the string, resulting in the returned value of 'c'.

It should not be inferred that Lazy means that ONLY the first character matched in the string is gathered. No, it means that it will always match against the first character matched then NOT have to gather up any additional characters if it can be helped, especially if there are other tags in the Match String to carry the load.

But here you can see that Lazy sometimes has to do a little more work than just match against a singular value.



Where:

%1 = ab

% 2 = c

%1 matches against the first character of the sample string, the 'a' but grudgingly has to continue gathering until the encounter of the match of the left parenthesis not included as part of %1's value because it is outside of the match.


%2 greedily gathers up any remainder of the string after the left parenthesis. There is only the 'c' of 'abc' left, thus %2's value is the 'c'.

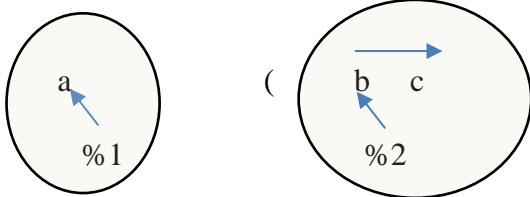
Section # 1: Regular Expressions (RegEx)v3.4 New Additions

String = a(bc)

Match: %1(%2

Replace: Group 1= %1 Group2 = %2

Name ▲	New Name
 a(bc)	Group 1= a Group2 = bc



Where:

%1 = a %2 = bc


If we change the sample string slightly moving the left parenthesis after the 'a', it changes the returned values but not the theory. The first tag Lazily gathers up to the match of the left parenthesis. The second tag Greedily gathers up any remainder of the string.

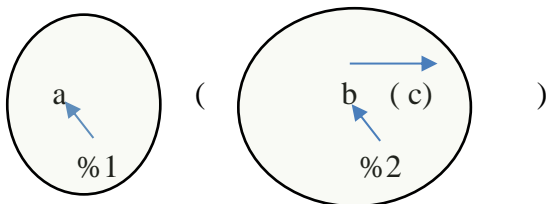
Next, I'll introduce a second literal character into the Match String and change the sample string.

String = a(b(c))

Match : %1(%2)

Replace: %2

Name ▲	New Name
 a(b(c))	b(c)



Where:

%1 = a %2 = b(c)

%2 = b(c) and not b(c)... how?

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

By now you know where %1 gets its value. But to review - Simple as of v3.4.20 will match against the first occurrence for any tags, Lazy, or the ‘shortest match’, other than the last tag that remains Greedy, the ‘longest match’.

%1(in the Match String will gather until the first occurrence of the left parenthesis. Since the Match String’s specification for the literal left parenthesis character is outside of the scope for %1, it is precluded from %1’s value, so %1 = ‘a’ and not ‘a(’

Because the Replace String only specifies %2, the value displayed in New Name is %2’s value of ‘b(c)’

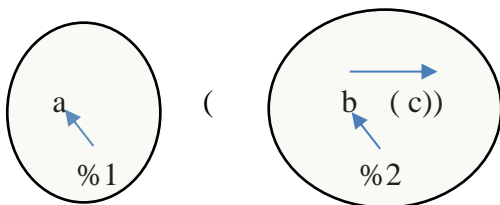
This still doesn’t fully explain how %2 got its value. Let’s investigate.

I’ll start by simplifying %2 to gathering up the remainder of the string and leave off the ending right parenthesis in the Match String.

String = a(b(c))

Match: %1(%2
Replace: %2

Name ▲	New Name
a(b(c))	b(c)



Where:

%1 = a %2 = b(c))

%1 matches against the first character, ‘a’, limited by the match of the left parenthesis that occurs outside the scope of the %1 tag.

%2, as expected, matches the ‘b’ character, and greedily gathers up the remainder of the string. Because the Replace String only specifies %2, the value displayed in New Name is that of %2’s value of ‘b(c))’ including the second occurrence of the right parenthesis.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

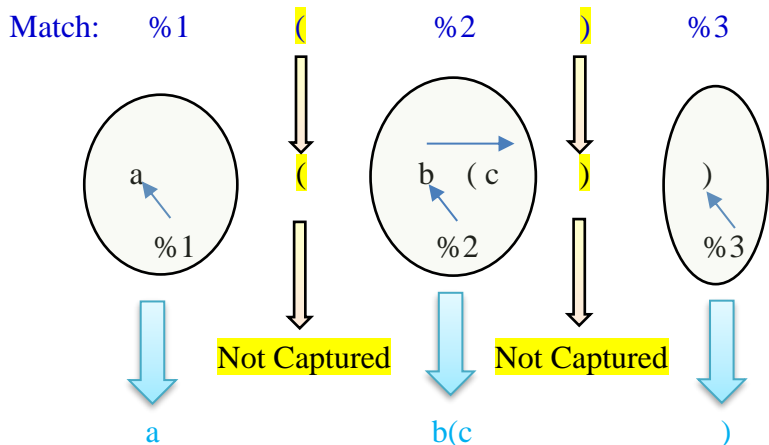
Now I will add a single right parenthesis into the equation after the %2 tag along with a %3 tag and see what happens.

String = a(b(c))

Match: %1(%2)%3

Replace: Group 1= %1 Group 2= %2 Group 3 = %3

Name ▲	New Name
a(b(c))	Group 1= a Group 2= b(c Group 3 =)



Where:

%1 = a %2 = b(c %3 =)

%1 matches to the first character of the string, the 'a', and is limited by the match of the left parenthesis that occurs outside the scope of the %1 tag and is precluded from %1's value.

%2 matches against the 'b' and lazily gathers until the first occurrence of the right parenthesis that matches outside the scope of the %2 tag and is precluded from %2's value.

%3 greedily gathers up any remaining part of the string, the last right parenthesis that is within the purview of the tag.

It becomes clear that the first occurrence of the right parenthesis is matched but not gathered. This leaves %3's value as the LAST right parenthesis.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

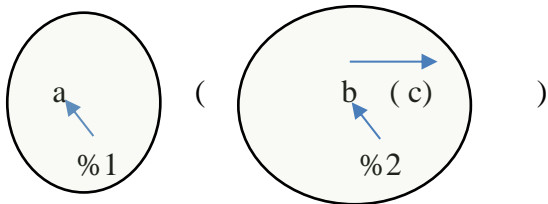
Returning to the original example,

String = a(b(c))

Match : %1(%2)

Replace: %2

Name ▲	New Name
a(b(c))	b(c)



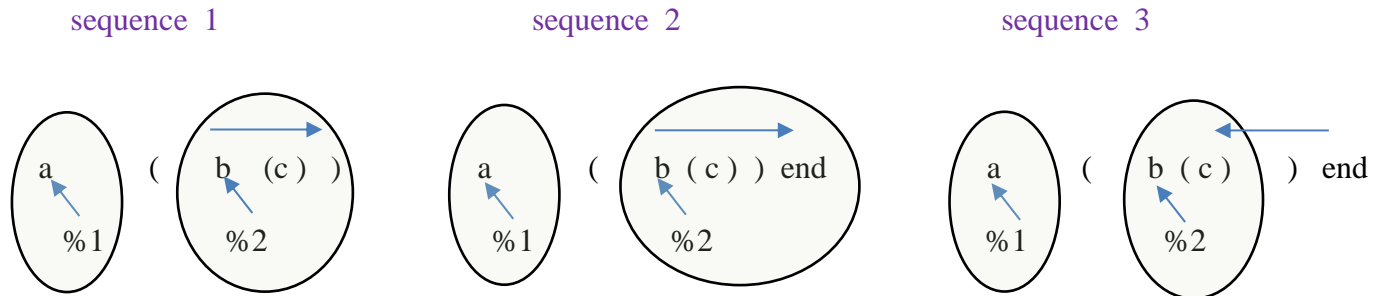
Where:

%1 = a %2 = b(c)

Looking only at %2, it matches against the 'b' as the second character of the string. The first character has already been gathered by the %1 tag. There are only two tags that make up the Match String, therefore %2 is Greedy while %1 is Lazy.

%2, being Greedy, begins to gather the string until it encounters the first right parenthesis. What does it do? It **is** Greedy so it will continue to try and match UNTIL it reaches the end of the sample string, gathering everything it can along the way.

In the sample string there is a second occurrence of the right parenthesis, and this is matched as well. Remember the little discussion earlier about what is happening in the background and how it spits out matches made outside of the tag? This is significant if you are to understand how 'b(c)' is the value of %2.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions**Sequence 1**

%2 greedily matches from the 'b' on including the match against the second right parenthesis.

Sequence 2

%2 reaches the end of the string.

Sequence 3

%2 spits out the second right parenthesis because it is outside the scope of %2.

%2's value is 'b(c)'

If %2 was not Greedy, it would have stopped at the match of the first right parenthesis making %2's value 'b(c)'. The second right parenthesis would not even have been considered.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Understanding the Importance of Delimiters in Simple

To make the use of Simple practical, it's best to work with filenames that have delimiters or characters that separate one word or a portion of the filename from the other. The Simple syntax does not provide for the recognition of text from numeric or other advanced pattern expressions. For that you have to turn to Regular Expressions or JavaScript.

The tags in the Match String are lazy with the exception of the last tag which is Greedy and can gather up the remaining characters of the string. This means that unless you want the lazy tags to capture only singular values, there must be delimiters that can be matched against, that will allow these tags to gather more than one character.

I'll start 'simple' and build from there.

String = ABC

Match: %1%2%3
 Replace: Group 1 = %1 Group 2 = %2 Group 3 = %3

Name ▲	New Name
ABC	Group 1 = A Group 2 = B Group 3 = C

A	B	C
↑	↑	↑
%1	%2	%3

Equal number of tags for each character in the string. Each tag represents a singular value. %3 is the only tag that is Greedy, but there are no other characters in the string leaving tag %3 to gather the 'C'.

String = ABC123DEF

Match: %1%2%3%4%5%6%7
 Replace: Groups 1 – 6 = %1, %2, %3, %4, %5, %6 and Group 7 = %7

Name ▲	New Name
ABC123DEF	Groups 1 – 6 = A, B, C, 1, 2, 3 and Group 7 = DEF

A	B	C	1	2	3	D E F
↑	↑	↑	↑	↑	↑	└─┬─┘
%1	%2	%3	%4	%5	%6	%7

With no delimiter, tags %1 through %6 will match against singular values of the first 6 characters in the sample string. The last tag %7 is Greedy and will gather up the remainder of the string.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

String = ABC-123

Match: %1-%2

Replace:: Group 1 = %1 Group 2 = %2

Name ▲	New Name
ABC-123	Group 1 = ABC Group 2 = 123

A B C - 1 2 3
 { } { }
 %1 %2

The introduction of a <hyphen> delimiter allows tag %1 to gather more than one singular value. In a sense it is told to gather up to the first <hyphen> encountered. In the example, %1 gathers 'ABC', precluding the <hyphen> because it is outside of the match. If there were two <hyphens> in the string sample, it would make no difference. Only the first set of characters would still be matched to tag %1 because it is lazy and will stop gathering at the first occurrence.

String = ABC-123-DEF

Match: %1-%2-%3

Replace: Group 1 = %1 Group 2 = %2 Group 3 = %3

Name ▲	New Name
ABC-123-DEF	Group 1 = ABC Group 2 = 123 Group 3 = DEF

A B C - 1 2 3 - D E F
 { } { } { }
 %1 %2 %3

The introduction of a second <hyphen> delimiter in the Match String allows tag %2 to gather to the second <hyphen> leaving tag %3 to cleanup the rest.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

String = ABC-123-DEF#456

Match: %1-%2-%3#%4

Replace: Group 1 = %1 Group 2 = %2 Group 3 = %3 Group 4 = %4

Name ▲	New Name
ABC-123-DEF#456	Group 1 = ABC Group 2 = 123 Group 3 = DEF Group 4 = 456

A B C - 1 2 3 - D E F # 4 5 6
 { } { } { } { }
 %1 %2 %3 %4

This is to demonstrate that multiple delimiters do not have to be the same character. In this example, both <hyphens> and a numeric sign are used as delimiters. It doesn't matter. It also doesn't matter what character a delimiter is. It could be any literal character. If you recall, a literal character is any character that can be reproduced by the keyboard.

Notes:

1. The specified literal characters in the Match String are outside of the scope of the tags and therefore precluded.

In the first part of this section I also showed you that delimiters are not required. I could match to the '2' if I wanted:

Match: %1-%2%3

Replace: Group 1 = %1 Group 2 = %2 Group 3 = %3

Name ▲	New Name
ABC-123-DEF#456	Group 1 = ABC Group 2 = 1 Group 3 = 3-DEF#456

Tag %2 matches only up to the '2' of '123' leaving tag %3. The specified '2' in the Match String is outside of the match and therefore precluded in %2's value. Tag %3 consequently starts its match at the '3' of '123'.

String	ABC	-	1	2	3-DEF#456
Match:	%1	-	%2	2	%3
Tags	%1	Not Captured	%2	Not Captured	%3
Value	ABC		1		3-DEF#456

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

Back to delimiters.

Using the delimiters, the tags allow you to isolate certain portions of the sample string. This is called 'parsing' where the string can be broken up into individual pieces.

Why would you want to do that?

Because the entire purpose of Simple is to have a very easy method by which to manipulate those parts of the string. You can take the pieces and rearrange them in any order using the Replacement String.

If I have:

%1 = A, %2 = B and %3 = C

I could use a Replacement String of:

%2%1%3 = BAC

Do you have to bother with knowing Greedy, the Longest Match, vs Lazy, the Shortest Match? No, but if you did enjoy the discussion, and it is my hope that you now understand the difference between Greedy vs Lazy, at least in this context, then maybe after you get done playing around using Simple, you may be ready to tackle learning Regular Expressions. Let's call Simple the Gateway language to learning RegEx. Regardless, using Simple will provide even more options for those new to the amazing program, Bulk Rename Utility.

And that, in conclusion, is the long and short of it ☺

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

My Final Observation on Simple:

So now you should have a full understanding of Simple and how it works. I have also shown you the differences between the original v.3.4 of Simple that used Greedy, or ‘the longest match’, for all matches except the first tag that used Lazy, or the ‘shortest match’, and v3.4.20 that uses Greedy only on the last tag and all other subsequent tags are Lazy.

There is actually much more that is happening behind the scenes, because it is obviously using the RegEx Engine to appropriate these values, but because this is Simple, I am not going to get into that other than what has been discussed.

When multiple tags are involved in the Match String, the ‘Lazy’ or Shortest Match really comes into play. So, did TGRMN do the right thing by changing the Simple functionality to use the ‘shortest match’ aka ‘Lazy’ vs the ‘longest match’ aka ‘Greedy’ in v3.4.20? Yes, I believe it really is better for first learners.

v2 creates a copy of the original string in New Name

In v1, the RegEx was used to parse the string keeping only what you wanted by capturing those elements and any elements that were not captured were discarded in New Name. Under v2, the entire string is copied over to New Name and specific elements using the RegEx, can be substituted for those same elements in New Name. New Name will always be a new string.

e.g., `String = My Cat has milk to drink`

Match: `\bmilk\b(.*)\bdrink\b`

Replace: `tuna\1eat`

Name ▲	New Name
My Cat has milk to drink	My Cat has tuna to eat

New Name is a new string where the words, ‘tuna’ and ‘eat’ have been substituted for ‘milk’ and ‘drink’. The ‘to’ is captured in Capture Group 1 and backreferenced in the Replace String. All other elements of the string, ‘My Cat has’ remains unchanged in New Name.

There will be more on this under the Global Switch discussion next.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

The Global Switch /g

Ladies and Gentlemen, we have been waiting for this for some time now. My involvement with TGRMN has provided a special privilege of ‘having their ear’. I have been working for *you* getting many of these long standing requests done and this is just another one of those requests that many users have expressed their desire to have. I want to thank TGRMN for their extensive cooperation in listening not just to me but to the other voices out there. Of course the only problem is now I have to redo some of this manual to reflect the new features. What’s that expression? oh, yes, be careful what you wish for ☺

Anyway back to the news of the day. Under PCRE v2, BRU supports The Global Switch!!

What is the Global Switch? Up until now, BRU Regular Expressions could only match against the first match. Any other matches found were ignored by New Name or never evaluated. The Global Switch, also called a modifier, allows all matches within the string, not just the first match that BRU v1 supports.

To best understand this feature, I again take you to an excerpted example, this time presented from the JavaScript section of Volume II.

from page 1498..

JavaScript

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

Nitro.Pro.13.15.1.282.Ent_x64

Match: ([^d])[\.]*/g
Replace: \1

The idea is to remove the dot characters from in between the Nitro and Pro as well as between Pro and 13 but ignore any dot characters between numbers, so you end up with:

NitroPro13.15.1.282.Ent_x64

Nitro . Pro . 13.15.1.282.Ent_x64

Not Captured Not Captured

Nitro Pro 13.15.1.282.Ent_x64

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

In the Regex Buddy program, the Global Switch is always used. This can be clearly seen -

Match: `([^\d])[\.]+`

Nitro.o.Pro.13.15.1.282.!Ent_x64

Match	Group	Start	Length	CL	RF
Match 1	o.	4	4	2	CL RF
Group 1	o	4	4	1	CL RF
Match 2	o.	8	8	2	CL RF
Group 1	o	8	8	1	CL RF

Two matches are performed at one time.

Beginning match attempt at character 4

1	o
2	o
3	o.

Match found in 3 steps

Beginning match attempt at character 8

1	o
2	o
3	o.

Match found in 3 steps

What it did is Match against the <dot> character from 'Nitro' as well as the <dot> character from 'Pro'.

Without the Global Switch, under PCRE v1 only the first match would be evaluated in BRU.

Name	New Name
Nitro.Pro.13.15.1.282.Ent_x64	o.Ent_x64

With the Global Switch under PCRE v2 you have:

Name	New Name
Nitro.Pro.13.15.1.282.Ent_x64	NitroPro13.15.1.282.Ent_x64

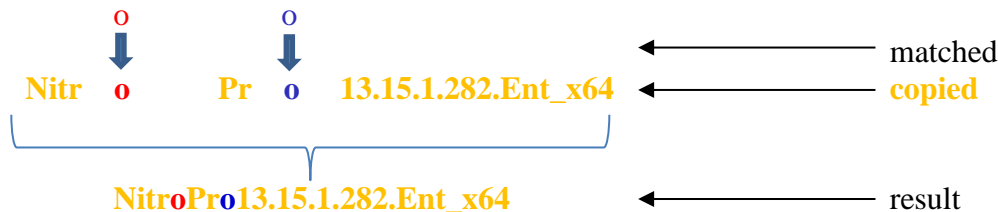
Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Notice though something else...

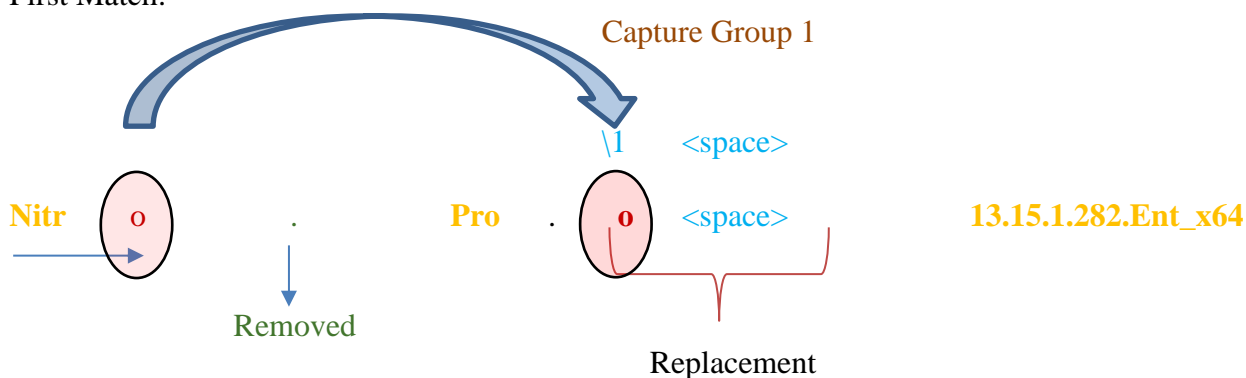
The New Name *without* the Global Switch is the isolated 'o' of 'Nitro', (along with the extension, '.Ent_x64'). This is the expected behavior because in v1, BRU has to match everything you want to keep. In other words, only the captured text is retained.

The New Name *with* the Global Switch is made up of the entire string along with the matched 'o' of 'Nitro' and the matched 'o' of 'Pro', excluding the (ignored) dot characters removed using the greedy class of `[\.]+`. These were all changed *within* the string. This is because using the Global Switch allows all matches to be retained, so the new match values do not 'isolate' the current values, but instead are 'replaced' or copied into the new string without disturbing the rest of the string value. Previously, this could only be accomplished through JavaScript. The help in BRU states it: 'In v2 the **unmatched** text is **copied** to the output, unlike in the default regular expressions.'

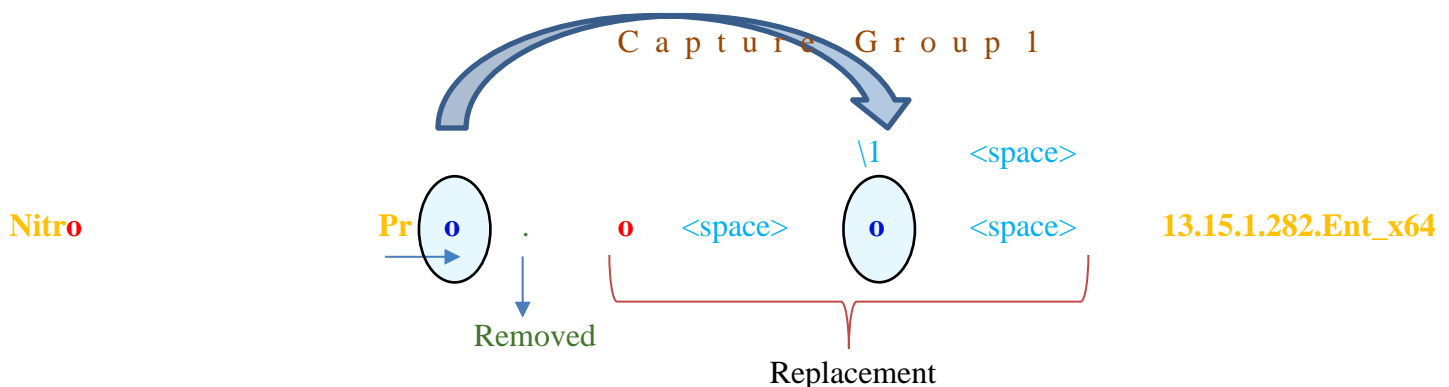


To visualize this a bit more clearly, if the Replacement String was instead `/1 <space>`, the expression would look like:

First Match:



Second Match:



Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Here is one more example to tie it all up:

String = is this all there is

Match: (is)
Replace: \1 <space>

This example without global enabled will match only against the first 'is' in the string. This is the 'is' of the first occurrence of the word, 'is' --- **is** this all there is

By placing the <space> in the Replace String, the match is easily identifiable in New Name:

Name ▲	New Name
is this all there is	is this all there is

Adding the Global Switch:

String = is this all there is

Match: (is)/g
Replace: \1 <space>

Name ▲	New Name
is this all there is	is this all there is

You can see at least one other match is included and that is the 'is' of '**this**'. But did it also match against the second occurrence of the word, 'is', at the end of the string? Hard to tell if there is an added <space> after, '**there is**' in New Name.

It can be verified if I change the string slightly to:

String = is this all there is or more

Match: (is)/g
Replace: \1 <space>

Name ▲	New Name
is this all there is or more	is this all there is or more

The <space> is definitely after, '**there is**'. The answer is yes, it matched all occurrences of the string characters 'is'.

So there you have it. The Global Switch.

Section # 1: Regular Expressions (RegEx)

v3.4 New Additions

Case Insensitive /i

Although I provided work-arounds to use in v1 in both Volume I and Volume II, BRU now fully supports the Case Insensitive modifier in v2. Prior to this, BRU only supported Case Sensitive. This also makes it more compatible with the RegEx Buddy program which also supports Case Insensitive.

/i makes the regular expression Case Insensitive.

Here is an example taken from the JavaScript section of Volume II page 1461 that illustrates a RegEx:

JavaScript

Remove the second repetition of filename from the end

The example presented in the book uses the work-around, (?i), for using Case Insensitivity in the previous version of BRU:

Match: `^(?i)(.*? *({8,}).*?)\2(.*?)$`
 Replace: `\1\3`

Name	New Name
AAA any repeated phrase BBB Any Repeated Phrase CCC	AAA any repeated phrase BBB CCC

Using that same example with the Case Insensitivity modifier – the /i at the end of the RegEx:

Match: `^(.*? *({8,}).*?)\2(.*?)$/i` ←
 Replace: `$1$3`

Name	New Name
AAA any repeated phrase BBB Any Repeated Phrase CCC	AAA any repeated phrase BBB CCC

What the RegEx does and how it works is not of significance here but the outcome is the same for both and that is significant.

Also I threw in the use of the \$ + **Capture Group designator** to illustrate how, e.g., **\$1** can be used in place of **\1** in the Replacement String under v2.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions**Case Conversion**

\E = turn off conversion
 \L = Lowercase until next \l (lowercase L) or \E
 \l = Lowercase next character (hard to tell but this is a lowercase L)
 \U = Uppercase until next \L or \E
 \u = Uppercase next character

string = HeLIO WoRID

Match: (Hello) (World)/i

Replace: \U\$1 \L\$2

Name	New Name
HeLIO WoRID	HELLO world

Note:

1. Don't get confused between the lowercase letter 'l' and the number one '1', used in these examples.
2. Any usage of Case Conversion Metacharacters turn off any preceding Case Conversion. This means you can't have, for example:

Replace: \l \U\$2

\l = lowercase next character – this is not the numeral one, followed by
 U = uppercase next character, followed by Group 2 Backreference

and expect: wORLD

instead you get

Name ▲	New Name
HeLIO WoRID	WORLD

Because although \l will lowercase next character, which it does in Capture Group 2, it is superseded by \U which converts the entire \$2 (Capture Group 2) to Uppercase.

Here's another one, just the opposite where Case Conversion is left on:

Replace: \U\$1 \2

Produces:

New Name
HELLO WORLD

Although Conversion to uppercase was specified for Capture Group 1 only, Case Conversion is still enabled when it encounters Capture Group 2.

Section # 1: Regular Expressions (RegEx)v3.4 New Additions

You could use:

Replace: `\U$1 \l\2` (`\l` = lowercase next character – this is not the numeral one)

to produce:

New Name
HELLO wORLD

Turn on Uppercase conversion until `\l` where it converts the next character, ‘W’ to ‘w’, but Uppercase remains enabled so ‘WoRlD’ becomes ‘wORLD’.

Where:

`$1` = HELLO `\2` = wORLD

If you used:

Replace: `\U$1 \E\l\2` (`\l` = lowercase next character – this is not the numeral one)

This produces:

New Name
HELLO woRID

Turn on Uppercase until the `\E` which turns off case conversion at that point. Continuing, `\l` converts the next character only to lowercase and the remainder of Capture Group 2 is left unchanged.

Where:

`$1` = HELLO `\2` = woRID

Match : `(\w)/g`

Replace: `\L$1`

Name	New Name
<input type="checkbox"/> AA11BB22CC33DD44	aa11bb22cc33dd44

This will replace ALL upper-case to lower-case with v2

Section # 1: Regular Expressions (RegEx)v3.43 New Additions**Added ability to use \E \L \I \U \u modifiers in the Replace field of the ‘Simple’ RegEx.**

To review, these are:

\E = turn off conversion
 \L = Lowercase until next \I or \E
 \I = Lowercase next character
 \U = Uppercase until next \L or \E
 \u = Uppercase next character

Example #1:

String = HeLIO WoRID

Match: %1
 Replace: \U%1

Name ▲	New Name
HeLIO WoRID	HELLO WORLD

Converts the entire string to uppercase because %1 matches the string and \U%1 converts it to uppercase.

Where:

%1 = HeLIO WoRID

Analysis:

The Match:

1. %1 Matches against first character H, and is Greedy.
Review – last tag under Simple is always Greedy.

The Replace String:

1. \U Enable Uppercase conversion.
Will convert all following characters until EOL or \E whichever comes first.
2. %1 Backreference to %1 tag.

Notes:

1. I am not going to repeat all of my findings concerning Greedy, Lazy, Short vs Long that I have already discussed at length (pun not intended). Therefore, these analysis' will be 'simplified'.

Section # 1: Regular Expressions (RegEx)v3.43 New Additions

Example #2:

String = This is_ version 2010 test

Match: %1_%2
Replace: \U%1\E_%2

Name ▲	New Name
 This is_ version 2010 test	THIS IS_ version 2010 test

Converts the first part of the string before the ‘_’ to uppercase.

Where:

%1 = This is %2 = version 2010 test

Analysis:

The Match:

1. %1_%2 Match against the ‘T’.
%1 will continue to match against characters until the underscore delimiter used in the Match String encounters the underscore in the sample string. The underscore is not part of %1 because it is outside the scope. %2 is Greedy and gathers up the remainder of the string.

The Replace String:

1. \U Enable Uppercase conversion.
Will convert all following characters until EOL or \E whichever comes first.
2. %1 Backreference to %1 tag.
3. \E Turn off conversion.
4. _ Replaces the underscore character that was not part of the match.
Review – the underscore character was outside of the %1 tag in the Match String.
5. %2 Backreference to %2 tag.

Section # 1: Regular Expressions (RegEx)v3.43 New Additions

Example #3:

String = abc_123-DEF

Match: %1_%2
 Replace: \U%1\E_\L%2

Name ▲	New Name
abc_123-DEF	ABC_123-def

Converts the first part of the string before the ‘_’ to uppercase and converts the remainder to lowercase.

Where:

%1 = abc %2 = 123-DEF

Analysis:

The Match:

1. %1_%2 Match against the ‘a’.
%1 will continue to match against characters until the underscore delimiter used in the Match String encounters the underscore in the sample string. The underscore is not part of %1 because it is outside the scope. %2 is Greedy and gathers up the remainder of the string.

The Replace String:

1. \U Enable Uppercase conversion.
Will convert all following characters until EOL or \E whichever comes first.
2. %1 Backreference to %1 tag.
3. \E Turn off conversion.
4. _ Replaces the underscore character that was not part of the match.
Review – the underscore character was outside of the %1 tag in the Match String.
5. \L Enable Lowercase conversion.
Will convert all following characters until EOL or \E whichever comes first.
5. %2 Backreference to %2 tag.

Section # 1: Regular Expressions (RegEx)v3.43 New Additions

Example #4:

String = TESTING the Words TITLE

Match: %1%2%3%4
Replace: \L%1%2%3\E%4

Name ▲	New Name
TESTING the Words TITLE	tesTing the Words TITLE

Convert the first 3 characters of the string to Lowercase.

Where:

%1 = T %2 = E %3 = S %4 = Ting the Words TITLE

Analysis:

The Match:

1. %1 Match against the 'T'.
Lazy, so no other characters captured and no delimiter characters that would force additional matches.
2. %2 Match against the 'E'
Lazy.
3. %3 Match against the 'S'
Lazy.
4. %4 Match against remainder of string.
Greedy. Last tag is always Greedy.

The Replace String:

1. \L Enable Lowercase conversion.
Will convert all following characters until EOL or \E whichever comes first.
2. %1 Backreference to %1 tag.
3. %2 Backreference to %2 tag.
4. %3 Backreference to %3 tag.
5. \E Turn off conversion.
6. %4 Backreference to %4 tag.

Notes:

1. In all of these examples when conversion is turned off via the \E modifier, any remaining string referenced by a tag in the Replacement String will display in New Name with the Case as it was originally in the string.

Section # 1: Regular Expressions (RegEx)v3.43 New Additions**Summary:**

v2 also known as PCRE2, offers a lot of improvements. Further information about the PCRE and PCRE2 Engine with the Boost Library can be found:

General information:

<https://perldoc.perl.org/perlre>

Information specific to the Match Expression:

https://www.boost.org/doc/libs/1_74_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html

Information specific to the Replace String:

https://www.boost.org/doc/libs/1_74_0/libs/regex/doc/html/boost_regex/format/boost_format_syntax.html

Information on the new Substitution Markers

https://www.boost.org/doc/libs/1_75_0/libs/regex/doc/html/boost_regex/format/boost_format_syntax.html

Additional resources for using some of these new capabilities can be found at:

<https://www.rexegg.com/>

<https://www.regular-expressions.info>

<https://regexlib.com>

My Regular Expression manuals remain a good resource for familiarizing yourself with RegEx for beginners and the experienced alike. Volume II covers some of the more advanced features. Best of all v2 removes most of the limitations imposed by v1 and likewise those limitations that have already been documented and illustrated in both of these volumes.

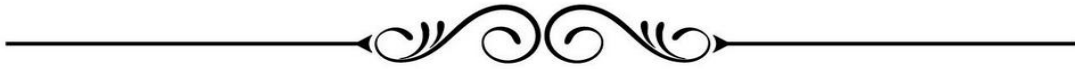
Final Conclusion:

This is just a sample of the new possibilities opened up by BRU's support of the PCRE2 RegEx Engine with the Boost Library.

Note:

Volume II was written before the newer version of BRU. Therefore, although Volume II only covers RegEx v1 (PCRE v1 5.x) and not RegEx v2 (PCRE2 with Boost) or Simple RegEx, it is still a valuable learning resource. With all of the numerous examples presented, you are always encouraged to try them yourself and make improvements and with the new PCRE v2, this is a certainty. Volume II, however, is more than just examples of RegEx. It teaches you the language through the study of the written analysis in formats similar to the ones briefly presented here.

Enjoy.



Section # 2 – (File) Name

Section # 2: (File) Name

Filename - This manages the current filename.

Your options in the drop down list are:

1. **Keep** – the current filename will not change (default)
2. **Remove** – completely removes the current file name. Must be used in conjunction with other renaming criteria that will replace the current name with a New Name. If no other criteria is specified, New Name will not reflect any change unless the filename contains an extension. See notation #1.

Name ▲	New Name
catamaran	catamaran

By itself it displays no change to New Name (sample string does not have an extension)¹:

But if I include a prefix under the '**Section #7: Add**' ...

Name ▲	New Name
catamaran	test

New Name reflects the change of both '**Section #2: Name**' and '**Section #7: Add**':

The original filename has been removed and replaced with 'test'.

Notes:

1. If no other criteria is specified except to remove the filename and the filename contains an extension, then this will display in New Name as:

Name ▲	New Name
Portishead-Roads a.mp3	.mp3

.. where only the extension will remain.

2. The '**Section #1: RegEx**' will not work in conjunction with '**Section #2: Name**' because the Order of Evaluation will preclude any criteria section numbered lower than '**Section #2: Name**'. I could not have the name removed in '**Section #2: Name**' and added back using RegEx in '**Section #1: RegEx**'.

Section # 2: (File) Name

Filename - This manages the current filename.

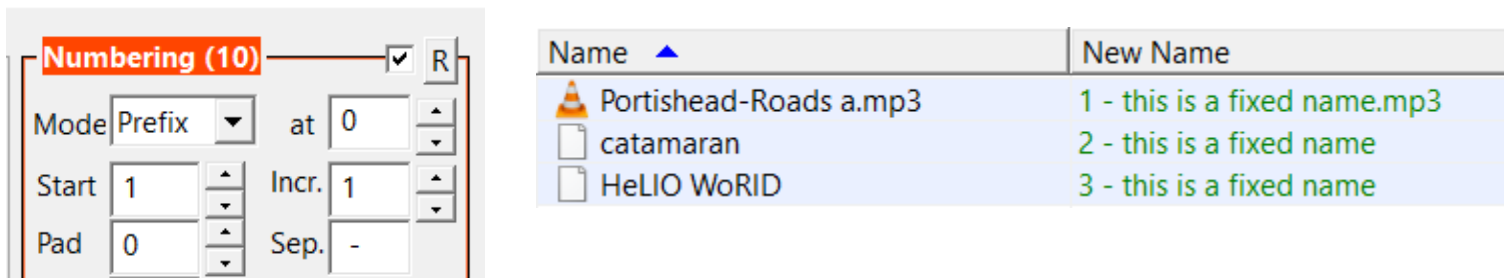
3. **Fixed** – A new specified filename (in the Name data field) will be applied for all selected files. Only useful in conjunction with the ‘[Section #10: Numbering](#)’, because otherwise if more than one file is selected, they will have the same name and generate an error when you try to apply the Rename function. By appending an incrementing number, the files will continue to be unique.



results in:

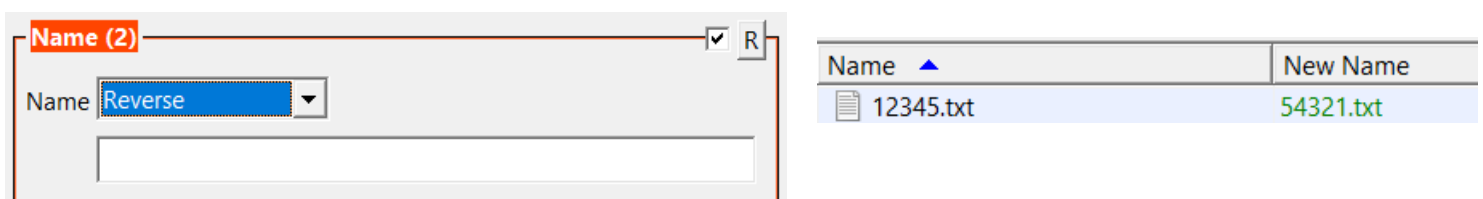
Name ▲	New Name
Portishead-Roads a.mp3	this is a fixed name.mp3
catamaran	this is a fixed name
HeLIO WoRID	this is a fixed name

Adding ‘[Section #10: Numbering](#)’, makes the filenames unique.

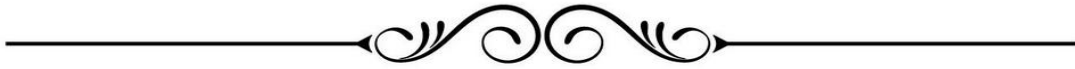


4. **Reverse** – completely reverses (backwards) the filename –

e.g. 12345.txt



... becomes 54321.txt.



Section # 3 - Replace

Section # 3: Replace

Replace - This is a Search and Replace by pattern.

It can find any pattern.

For example if I want to search for the pattern “- <space>” and replace with “<space> - <space>” I would enter:

Spaces can't clearly be seen in the data fields. That is why I typically enclose them in angular brackets for the purposes of this document. Do not, however, enter the, ‘angular bracket space angular bracket’, when entering <spaces> into any of the data fields. Just enter the character produced from the spacebar on the keyboard.

Remember, that once your criteria has been entered, select the files from the File List in the Content Pane you want to include through the normal selection process of **Select all (Ctrl + A)**, **non-consecutively (Ctrl + left mouse click)**, or **consecutively (Shift + down arrow)**.

Unicode and High ASCII can also be entered:

For example,

2000-07-23 name █.pdf

I Created the symbol in the filename by using Alt + 221 (from Numeric keyboard, hold down Alt key and hit 2,2,1).

Enter the symbol by keying in the same Alt + 221 into the Replace field (numlock must be ON).

Results in:

Name	New Name
2000-07-23 name █.pdf	2000-07-23 name hello.pdf

Section # 3: Replace

v3.4 New Additions

Multiple Replacements

Multiple replacements can be specified using the | (Pipe character) delimiter.

Examples:

Multiple specifications in the 'Replace' search field are replaced 'With' multiple replacement values.

Using:

Replace: H|L
With: T|J

Name ▲	New Name
HeLIO WoRID	TeJJO WoRID

Replaces all instances of 'H' in the string with 'T' and subsequently all instances of 'L' with 'J'.

Specifying just a single replacement for 'With' and multiple specifications in the 'Replace' search field results in all instances of the multiple characters in the string replaced by the single 'With' replacement value.

Name ▲	New Name
HeLIO WoRID	TeTTO WoRTD

All instances of both the 'H' and 'L' are replaced by the 'T'.

Section # 3: Replace

v3.4 New Additions

You are not limited to two replacements.

Here is another example:

Three changes are specified for the 'Replace' search but only two 'With' replacements are provided.

Replace: H|L|O
With: T|Y

Name ▲	New Name
HeLIO WoRID	TeYYY WYRYD

All instances of 'H' are replaced with 'T'. The remaining two searches of 'L' and 'O' have all instances replaced with 'Y'. Because I did not enable 'Case', case remains Insensitive, resulting in both upper and lowercase characters replaced.

The only 'H' in the substring, 'HeLIO', is replaced with 'T'.

Both the uppercase 'L' and lowercase 'l' of 'HeLIO' and the lowercase 'l' in 'WoRID' are replaced with 'Y'. The final search specification of the 'O' of both the uppercase 'O' of 'HeLIO' and the lowercase 'o' found in 'WoRID' are also replaced with 'Y'.

Notes:

1. If you wish to include the pipe character in the Replacement, then precede it with an 'Escape' character (a backslash \). The escape character tells BRU to ignore the character as a command or as part of the syntax of a command and instead treat it as a literal. e.g., \|
2. The pipe character is not a legal character in a Windows filename, meaning, BRU would not allow you to rename a file containing this character normally. However with the added improvements in the '[Section #1: RegEx](#)' allowing multiple RegEx to be run, the Global switch, etc., the pipe character could be used temporarily within the renaming process (there are examples of this that can be found in Volume II) and be removed here in '[Section #3: Replace](#)' before the final renaming operation.
3. The reason that the Escape character is required in this section, '[Section #3: Replace](#)', is because the pipe character is now a part of the syntax for that section. In other sections, if the pipe character is used and is not a part of the syntax of that function, the escape would not be required.

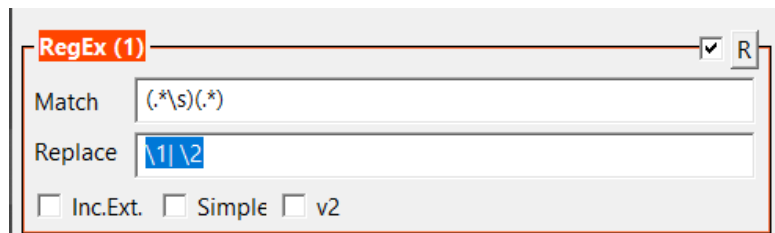
Section # 3: Replace

v3.4 New Additions

The following example is in two parts.

First, a RegEx is used to place the pipe character in the string because, again, I cannot do this directly since Windows won't allow a filename containing the character. For more information on literals, Metacharacters and Special Characters, see Volume II. Some information can also be found in the RegEx manual in the Appendix of this volume as well.

Match: `(.*\s)(.*)`
 Replace: `\1| <space> \2`



It should be noted that I have not included the expanded version of a <space> character using the angular brackets in all the examples in this book and Volume II. Therefore I recommend that you use Copy and Paste to enter them in your own BRU program if you don't see '<space>' explicitly.

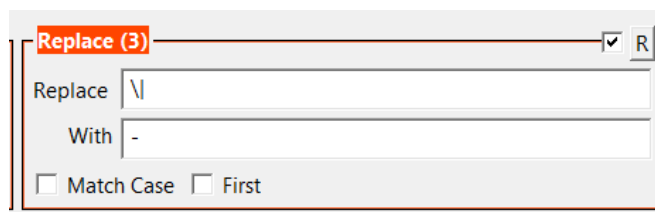
The first part of the Match string, `.*\s`, will capture the entire string to EOL, then backtrack to search out the <space> character which it finds after the word, "HeLIo", and isolate it in Capture Group 1. The second part of the RegEx, `.*`, isolates and captures the remaining string into Capture Group 2. The Replace String will display Capture Group 1 that contains the word 'HeLIo <space>'.

Next, it places the pipe character as a literal (literals in the Replace String of '[Section #1: RegEx](#)', do not require a preceding escape character) into the filename. Windows doesn't complain because the actual renaming has not been initiated at this stage. So far all changes are in memory only. BRU on the other hand *is* complaining. New Name is in **red** meaning that this is a warning the filename is **Invalid** and will not be renamed if you proceed.

Name ▲	New Name
HeLIo WoRID	HeLIo WoRID

\1 = Capture Group 1 = HeLIo <space>
 \2 = Capture Group 2 = WoRID

The final part will use '[Section #3: Replace](#)' to remove the pipe character and replace it with a <hyphen> which *is* a legal character in a Windows Filename. New Name turns from **red** (**Invalid**) to **green** (**valid** and good to go):



Because the pipe character is now a part of the syntax used in '[Section #3: Replace](#)', it must be escaped to be recognized as a literal, whereas when used in the Replace String of '[Section #1: RegEx](#)' above, it is not part of the syntax and does not need to be escaped.

Name ▲	New Name
HeLIo WoRID	HeLIo - WoRID

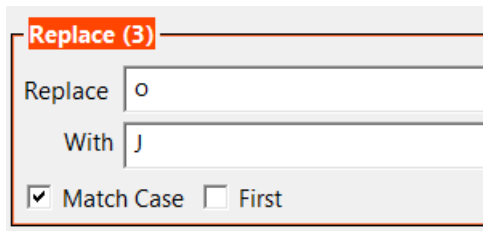
Section # 3: Replace

v3.4 New Additions

Match Case enabled - Perform case-sensitive replacement with Replace Only on First Match

Case Sensitive matches were allowed previously but a quick example is necessary to explain the new option, 'First':

Example:

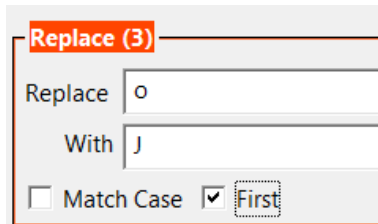


The screenshot shows a dialog box titled "Replace (3)". It has two input fields: "Replace" containing the lowercase letter 'o' and "With" containing the uppercase letter 'J'. Below the input fields are two checkboxes: "Match Case" which is checked, and "First" which is unchecked.

Name ▲	New Name
HeLIO WoRID	HeLIO WJRID

Only the lowercase 'o' is affected and replaced with 'J' in the word, 'WoRID' and not the uppercase 'O' found in 'HeLIO'.

First enabled - Replace only first match.



The screenshot shows a dialog box titled "Replace (3)". It has two input fields: "Replace" containing the lowercase letter 'o' and "With" containing the uppercase letter 'J'. Below the input fields are two checkboxes: "Match Case" which is unchecked, and "First" which is checked.

Name ▲	New Name
HeLIO WoRID	HeLIJ WoRID

Only the first occurrence found in the string, the uppercase 'O' of 'HeLIO' is affected and replaced with 'J' leaving the second occurrence, the lowercase 'o' in the word, 'WoRID' intact.

Section # 3: Replace

v3.4 New Additions

Position Modifier \<Position value>\

Position values are:

first	\first\	Replace only the first match (equivalent to enabling 'First')
last	\last\	Replace only the last match
start	\start\	Replace at start of string
end	\end\	Replace at end of string
second	\second\	Replaces only the second match (second occurrence) found in the string
third	\third\	Replaces only the third match (third occurrence) found in the string
fourth	\fourth\	Replaces only the fourth match (fourth occurrence) found in the string
fifth	\fifth\	Replaces only the fifth match (fifth occurrence) found in the string
sixth	\sixth\	Replaces only the sixth match (sixth occurrence) found in the string
seventh	\seventh\	Replaces only the seventh match (seventh occurrence) found in the string
eighth	\eighth\	Replaces only the eighth match (eighth occurrence) found in the string
ninth	\ninth\	Replaces only the ninth match (ninth occurrence) found in the string

The position modifier has to be specified at the **start** of the Replace String.

Examples:

The screenshot shows a dialog box titled "Replace (3)". It has two input fields: "Replace" containing "\first\e" and "With" containing "TRM". Below these fields are two checkboxes: "Match Case" (unchecked) and "First" (checked). A green arrow points from the "First" checkbox to the "First" option in the table below.

Name ▲	New Name
HeLIO WoRID	HTRMLIO WoRID

Replace first 'e' in string with 'TRM'.

As already stated, this position modifier, \first\, is equivalent to enabling the 'First' option –

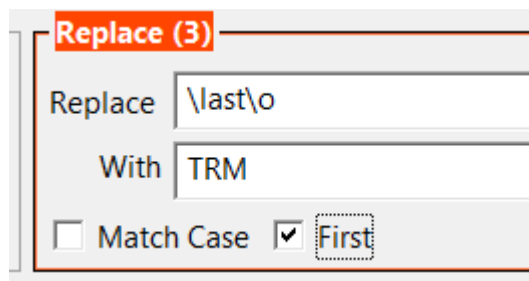
The screenshot shows a dialog box titled "Replace (3)". It has two input fields: "Replace" containing "e" and "With" containing "TRM". Below these fields are two checkboxes: "Match Case" (unchecked) and "First" (checked). A green arrow points from the "First" checkbox to the "First" option in the table below.

Name ▲	New Name
HeLIO WoRID	HTRMLIO WoRID

Section # 3: Replace

v3.4 New Additions

It should be noted that enabling 'First' overrides any position modifier in the Replace field:



The screenshot shows the 'Replace (3)' dialog box. The 'Replace' field contains the text '\\last\\o'. The 'With' field contains the text 'TRM'. Below the fields, there are two checkboxes: 'Match Case' (unchecked) and 'First' (checked). The 'First' checkbox is highlighted with a dashed border.

Name ▲	New Name
HeLIO WoRID	HeLITRM WoRID

Although I wanted the second 'o' in the string, the 'o' in 'WoRID', it is the first 'o', the 'O' in 'HeLIO' that is changed. This is because 'First' is enabled and supersedes the Position Modifier, \\last\\.

To correct this, disable 'First' and you will obtain the expected results.



The screenshot shows the 'Replace (3)' dialog box. The 'Replace' field contains the text '\\last\\o'. The 'With' field contains the text 'TRM'. Below the fields, there are two checkboxes: 'Match Case' (unchecked) and 'First' (unchecked). The 'First' checkbox is highlighted with a dashed border.

Name ▲	New Name
HeLIO WoRID	HeLIO WTRMRID

Section # 3: Replace

v3.4 New Additions

If a **null** value or 'empty' string is specified for the 'With' replacement string, the 'Replace' value is removed.

Replace (3)

Replace

With

Match Case First

A **null** value is specified (no entry), not to be confused with a <space> character that would also not display.

Results in the removal of the last 'o' in the string, the 'o' in 'WoRID'.

Name ▲	New Name
<input type="checkbox"/> HeLIO WoRID	HeLIO WRID

More Examples:

Example:

Replace (3)

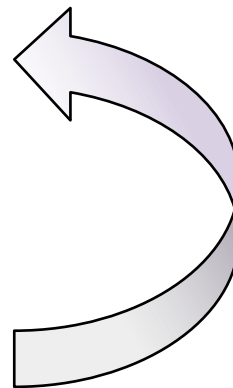
Replace

With

Match Case First

Name ▲	New Name
<input type="checkbox"/> HeLIO WoRID	BaLIO WoRID

The 'HeL' of 'HeLIO' is replaced with 'Bal'.



The specified 'Replace' string must be text found at the **start of the string only**. The following will not work:

Replace (3)

Replace

With

Match Case First

Name ▲	New Name
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID

Section # 3: Replace

v3.4 New Additions


Example:

Replace (3)

Replace

With

Match Case First

Name ▲	New Name
 HeLIO WoRID	HeLIO WKRID

This replaces the second Match or occurrence of the 'o' with 'K' in the string. The first match is found at the 'O' of 'HeLIO' and the second match is found at the 'o' of 'WoRID'. It is the second match, the 'o' of 'WoRID' that is changed to a 'K'.


If there is no second match, the evaluation fails and no change occurs:

Replace (3)

Replace

With

Match Case First

Name ▲	New Name
 HeLIO WoRID	HeLIO WoRID

Note: It is also possible to use the tag, **<clip>**, in both the Replace and With fields. **<clip>** will be substituted with the **current textual content of the Windows Clipboard**.

Placed in Windows Clipboard:


This is a Test

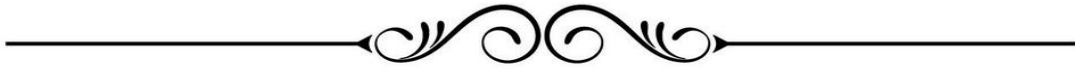
Replace (3)

Replace

With

Match Case First

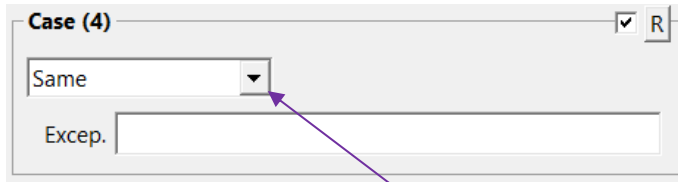
Name ▲	New Name
 HeLIO WoRID	HeLIO This is a TestoRID



Section # 4 - Case

Section # 4: Case

Case- Allows you to change the Case of the selected Filenames.



Under the drop-down menu your options are:

Same – leave as is (default)

Lower - converts all letters in filename to lowercase (small letters)

Upper - converts all letters in filename to uppercase (capitals)

Title - converts the first letter of each word to uppercase

Sentence - converts only the first letter of each word to uppercase.

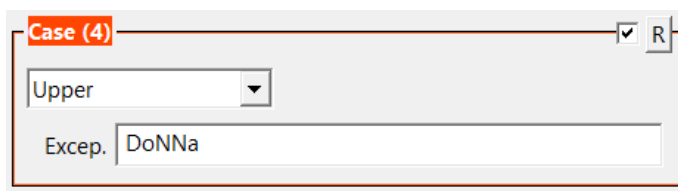
- * In addition you can specify words in the ‘Except.’ field that are excepted from having the above case action performed.
- * The case used for the words in this field will appear in the selected files *as they appear* in the ‘Except.’ field.
- * Multiple exceptions are specified using a colon to separate the words (called a delimiter character).

So for example,

Original filename:

Donna Summer- Mac Arthur Park Suite (original extended version).mp3

The Specified Case will be Upper for all letters of the selected filename with the exception of Donna. The word Donna is entered into the ‘Except. Field, specified as **DoNNa** for fun.



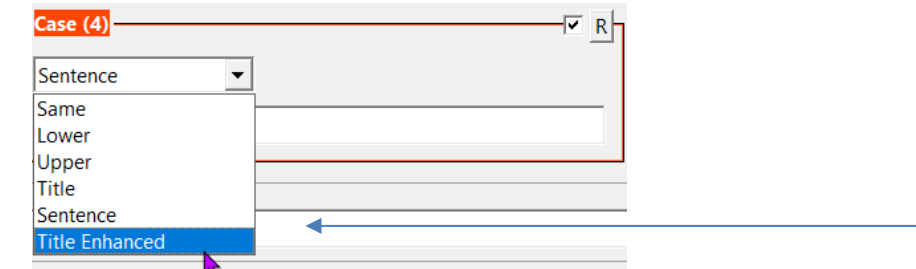
becomes:

→ **DoNNa** SUMMER- MAC ARTHUR PARK SUITE (ORIGINAL EXTENDED VERSION).MP3

Section # 4: Case

v3.43 New Additions

The New York Times Title Case is now used in Title option, 'Title Enhanced', of the drop down Case menu.



So what is New York Times Title Case? **The RULES:**

The following are Capitalized:

1. Nouns, pronouns, and verbs.
2. All words of four or more letters.

Bru, however, doesn't distinguish between the quantity of letters:

Name ▲	New Name
<input type="checkbox"/> test j my cat	Test J My Cat

Single letter (**j**), two letter (**my**) and three letter (**cat**) words are all converted to uppercase.

3. no, nor, not, off, out, so, up

Name	New Name
<input type="checkbox"/> no, nor, not, off, out, so, up	No, nor, Not, Off, Out, So, Up

Bru, however, doesn't capitalize 'nor'

4. The following are not capitalized unless they appear at the **beginning** or **end** of a string:

a, and, as, at, but, by, en, for, if, in, of, on, or, the, to

Name ▲	New Name
<input type="checkbox"/> a, and, as, at, but, by, en, for, if, in, of, on	A, and, as, at, but, by, en, for, if, in, of, On

5. The following **are** Capitalized:

v., vs., via

Name ▲	New Name
<input type="checkbox"/> testing the words v., vs., via for title...	Testing the Words V., Vs., Via for Title Enhanced

Section # 4: Casev3.43 New Additions

BRU doesn't use the full New York Times Style. For instance the following are **not** true.

6. Capitalize when used as adverbs.

Name ▲	New Name
<input type="checkbox"/> We stayed in to watch a movie	We Stayed in to Watch a Movie

'in' is an adverb in the above sentence, but is not capitalized. Understandable because I don't believe that BRU is grammatically sophisticated enough to tell the difference. It is most likely programmed with a list of words and what to do with them.

7. Capitalize 'for' if it takes the place of a verb meaning "support" or "advocate"

Name ▲	New Name
<input type="checkbox"/> Hooray for Hollywood	Hooray for Hollywood

BRU leaves 'for' unchanged, following rule #1 instead. Understandable because I don't believe that BRU is grammatically sophisticated enough to tell the difference. It is most likely programmed with a list of words and what to do with them.

8. In hyphenated compounds, do not capitalize the second part if it follows a prefix of two or three letters, and if the <hyphen> separates doubled vowels.

Name ▲	New Name
<input type="checkbox"/> Hoo-ray for Holly-wood	Hoo-Ray for Holly-Wood

BRU converted both sides of the compound to uppercase, regardless. Same reason. BRU is not able to distinguish what should be capitalized and what shouldn't when it comes to hyphenated words.

Notes:

1. Other considerations of why BRU doesn't support the full New York Times Case, is because that style, as you can ascertain from the rules, is meant for styling paragraphs and not one line folder or filenames.

Section # 4: Casev3.43 New Additions**New Changes in the Except(ion) field:**

The following special tags are available in the Exception data field under Title Enhanced Case:

<clear>

for Title Enhanced case, it clears all default words that are not to be capitalized unless they are at the start or at the end of a string..

The default words are:

a and as at but by en for if in of on or the to

Example #1

Under Title Enhanced:

<clear>

.. will clear the default exception list

Using a string with the default exception list in uppercase, the default is to lowercase all of these items.

Name	New Name ▲
<input type="checkbox"/> testing - A, And, As, At, But, By, En, For, If, In, Of, On, Or, The, To - end Test	Testing - a, and, as, at, but, by, en, for, if, in, of, on, or, the, to - End Test

When I add the tag,

Case (4) [checked] [R]

Title Enhanced ▼

Except. <clear>

... using a string with the default exception list in lowercase results in:

Name ▲	New Name
<input type="checkbox"/> testing - a, and, as, at, but, by, en, for, if, in, of, on, or, the, to - end test	Testing - A, And, As, At, But, By, En, For, If, In, Of, On, Or, The, To - End Test

Items in lowercase by default under Title Enhanced, will be in uppercase, the same behaviour as under Title Case.

<clear> tag has no effect under the Title Case.

Section # 4: Casev3.43 New Additions

The following special tags are available in the Exception data field under Title Enhanced Case: cont.

<clear> cont.

Example #2

Under Title Enhanced:

<clear>:and:or

.. will clear the default exception list but maintain the items 'and' and 'or' as lowercase.

Under Title Enhanced, the default behaviour is to lowercase all of these items.

Name ▲	New Name
<input type="checkbox"/> testing - a, and, as, at, but, by, en, for, if, in, of, on, or, the, to - end test	Testing - a, and, as, at, but, by, en, for, if, in, of, on, or, the, to - End Test

When I add the tag,

Case (4) [checked] [R]

Title Enhanced ▼

Excep. <clear>:and:or

results:

New Name
Testing - A, and, As, At, But, By, En, For, If, In, Of, On, or, The, To - End Test

All items of the string are now Capitalized regardless of setting of Title Enhanced, because the default list that normally would have lowercased these items has been cleared with the exceptions of 'and' and 'or'.

Section # 4: Casev3.43 New Additions**Under Title:**

Name ▲	New Name
<input type="checkbox"/> testing - a, and, as, at, but, by, en, for, if, in, of, on, or, the, to - end test	Testing - A, And, As, At, But, By, En, For, If, In, Of, On, Or, The, To - End Test

Normally, the default is to uppercase all of these items.

When I add the tag,

Results:

New Name
Testing - A, and, As, At, But, By, En, For, If, In, Of, On, or, The, To - End Test

The 'and' and 'or' items have been converted to lowercase.

Of course this is the default behaviour of the Exception list anyway.

New Name
Testing - A, and, As, At, But, By, En, For, If, In, Of, On, or, The, To - End Test

No difference. This is because `<clear>` does not seem to do anything under the Title Case, but only works under the Title Enhanced Case.

It would be interesting if BRU used the Exception list to convert the opposite of what was the norm. For example, an item that is normally lowercase would be case converted to uppercase and vice versa using an `<Opposite>` tag, but that will have to be reserved for a future discussion. I have put in this request to TGRMN.

Section # 4: Casev3.43 New Additions



<ic>

For Title Enhanced case, ignore words that are all caps and do not change capitalization for them.

Sample Strings:

Graham the aB ab abc abC ABC Miller.jpg
ABC-123-DEF

Normally,

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Graham the Ab Ab Abc Abc Abc Miller.Jpg
 ABC-123-DEF	Abc-123-Def



But applying the Special tag, <ic>

Case (4) **R**

Title Enhanced ▼

Excep.

Results:

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Graham the Ab Ab Abc Abc ABC Miller.Jpg
 ABC-123-DEF	ABC-123-DEF

The first string, Graham the aB ab abc abC ABC Miller.jpg, the 'ABC' item in uppercase is unaffected. In the second string made up of all uppercase letters, there is no change and New Name reflects this.

Has no effect under the Title Case.

Section # 4: Casev3.43 New Additions

The following tags work under both Title Case and Title Enhanced Case –

<rnlo> Roman Numeral Lowercase

Lowercase all Roman Numerals regardless. Note that the code is **r n l o** - **not mlo** if you look too quick.

String = beethoven's ninth symphony part III

Applying the Special tag, **<rnlo>**

Results:

Name ▲	New Name
beethoven's ninth symphony part III	Beethoven's Ninth Symphony Part iii

<rnup> Roman Numeral Uppercase

Upper case all Roman Numerals regardless. Note that the code is **r n u p** - **not mup** if you look too quick.

String = beethoven's ninth symphony part iii

Applying the Special tag, **<rnup>**

Results:

Name ▲	New Name
Beethoven's niNTH syMPHONY part iii	Beethoven's Ninth Symphony Part III

Section # 4: Casev3.43 New Additions

The following work under both Title Case and Title Enhanced Case –

You can also specify words that you always want full caps:

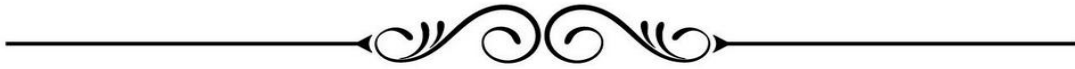
String = beethoven mozart liszt

BEETHOVEN:MOZART:LISZT

Results:

Name ▲	New Name
beethoven mozart liszt	BEETHOVEN MOZART LISZT

This goes along with the original rules of the Exception List. Remember DoNNa?



Section # 5 - Remove

Section # 5: Remove

Remove- Removes parts within a filename (excluding the extension).

Your options are:

First n chars - Remove the first n characters from the **beginning** of the filename.

Last n chars - Remove the last n characters from the **end** of the filename.

Example:

String = Andrew Gregory Macintyre.jpg

Remove the first 2 and the last two characters from the string.

Results:

Name ▲	New Name
Andrew Gregory Macintyre.jpg	drew Gregory Macinty.jpg

Section # 5: Remove

From n to n – This allows you to remove a consecutive string of characters by specifying their exact numeric position, first (starting from the left) and last, as they appear in the filename.

From to

For example, a filename:

This is a test.txt

The filename has a total of 14 characters.

T	h	i	s		i	s		a		t	e	s	t
1	2	3	4	5	6	7	8	9	10	11	12	13	14

all characters are counted including spaces.

If the word, 'is' was to be removed, it would be specified as - From 6 to 7

resulting in:

This a test.txt There are now two spaces after 'This'. The original <space> following, 'This', and the <space> previously following, 'is'.

In BRU:

Remove (5) R

First n Last n

From to

Name ▲	New Name
This is a test.txt	This a test.txt

Notes:

Positioning in BRU, other than RegEx, generally starts with position 1. In RegEx, the position starts with 0.

Section # 5: Remove

Chars Words

Chars

Chars – Removes any occurrences of characters entered here wherever they appear in the filename. There is no delimiter between characters because the entire character set can be used including symbols and even high ASCII. For example, entering Alt + 128 using the numeric keypad enters Ç.

Okay, so ‘[Section #3: Replace](#)’ can do that. What’s the big deal?

Well, smarty, can Replace do this?

Chars Words

Name ▲	New Name
 This is a test.txt	This s es.txt

I specified three lowercase characters to be removed using a comma delimiter. As a result, the ‘i’ of ‘This’, the ‘a’, and the ‘t’ of ‘test’ have all been removed at one time. The uppercase ‘T’ of ‘This’ is not affected because of Case Sensitivity.

Words - Remove any occurrences of words entered here. Multiple words are separated by a <space> delimiter.

Example:

Chars Words

Name ▲	New Name
 This is a test.txt	is a test.txt

Removes the word, ‘This’, from the string. As with Chars, the real difference with ‘[Section #3: Replace](#)’ is that multiple words can be entered via a <space> delimiter.

Section # 5: Remove
 D/S Accents Chars

 D/S

Double Spaces – convert any occurrences of double spaces to single spaces


Remember that example where we ended up with a double <space>? Here it is again.

Remove (5)

 R

 First n Last n

 From to

Name ▲	New Name
 This is a test.txt	This a test.txt

This time we enable the D/S option –

Remove (5)

 R

 First n Last n

 From to

 Chars Words

 Crop

 Digits High Trim

 D/S Accents Chars

 Sym.

.. no more double <space>

Name ▲	New Name
 This is a test.txt	This a test.txt

Section # 5: Remove
 D/S Accents Chars

 Accents

Accents – Replace Accented characters with their non-accented equivalent.

Example:

String = Bahá í Muñoz

Name ▲	New Name
Bahá í Muñoz	Baha i Munoz

If you wanted to remove all of the High ASCII characters then you would use the ‘High’ option and not the ‘Accents’ option.

 Chars

Chars – remove all alpha-numeric characters

String = DSCN0001-more.jpg

Name ▲	New Name
DSCN0001-more.jpg	0001-.jpg

Section # 5: Remove

Sym. Lead Dots None ▾

Sym.

Symbols - remove all non alpha-numeric characters

Example:

String = DSCN0001-more.jpg

Remove (5) R

First n Last n

From to

Chars Words

Crop

Digits High Trim

D/S Accents Chars

Sym. Lead Dots None ▾

Removes the <hyphen>

Name ▲	New Name
 DSCN0001-more.jpg	DSCN0001more.jpg

Section # 5: Remove
 Sym. Lead Dots **None** ▼

 Lead Dots **None** ▼

Lead Dots – Removes a selection of a leading single, double dot or both in the filename. None is default. The documentation states that this is useful if you copied files from a Linux or Unix type environment.

You can under the drop-down menu:

- None** – leave as is (default)
- .** – Remove leading single dot character in the filename
- ..** – Remove leading double dot characters in the filename
- Both** – Remove either a leading single or a leading two dot characters from a filename but not both in the same filename. It will be one or the other that is removed based on which comes first.

Strings = . DSCN0001-more.jpg .. DSCN0001-more.jpg ... DSCN0001-more.jpg ... DSCN0001-more.jpg

Name ▲	New Name
. DSCN0001-more.jpg	DSCN0001-more.jpg
.. DSCN0001-more.jpg	.. DSCN0001-more.jpg

With the single dot enabled, only the first file is affected.

Name ▲	New Name
. DSCN0001-more.jpg	. DSCN0001-more.jpg
.. DSCN0001-more.jpg	DSCN0001-more.jpg

With the double dot enabled, only the second file is affected.

Name ▲	New Name
. DSCN0001-more.jpg	DSCN0001-more.jpg
.. DSCN0001-more.jpg	DSCN0001-more.jpg

With 'Both enabled, files that contain a leading single dot or two dots are affected. The first and second file are affected.

'Both' will not remove both leading single dots and two dots from the same filename:

... DSCN0001-more.jpg

Name ▲	New Name
... DSCN0001-more.jpg	.. DSCN0001-more.jpg

one ...

... DSCN0001-more.jpg

Name ▲	New Name
... DSCN0001-more.jpg	. DSCN0001-more.jpg

or ... the other... but not both

Section # 5: Remove

Digits High Trim

Digits

Digits – Remove any numeric characters

String = DSCN0001-more.jpg


Remove (5) R
First n 0 Last n 0
From 0 to 0
Chars Words
Crop Before
 Digits High Trim
 D/S Accents Chars
 Sym. Lead Dots None

Trim

Trim – Remove leading and trailing spaces

String = DSCN0001-more .jpg

Remove (5) R
First n 0 Last n 0
From 0 to 0
Chars Words
Crop Before
 Digits High Trim
 D/S Accents Chars
 Sym. Lead Dots None

Name ▲	New Name
 DSCN0001-more .jpg	DSCN0001-more.jpg

All leading and trailing <space> characters have been removed.

Section # 5: Remove

Digits High Trim

High

High ASCII – removes any High ASCII characters (ASCII 128 to ASCII 255)

String = Bahá í Muñoz

Name ▲	New Name
Bahá í Muñoz	Bah Muoz

The High ASCII characters have been removed in the string. If you wanted only to remove the accents themselves, you would use the ‘Accents’ option and not the ‘High’ option.

Extended ASCII Chart (character codes 128 - 255)

128 Ç	143 Å	158 Æ	172 ¼	186	200 ℒ	214 ¶	228 Σ	242 ≥
129 ù	144 É	159 f	173 ;	187]]	201 ¶¶	215 ¶¶	229 σ	243 ≤
130 é	145 æ	160 á	174 «	188]]	202 ¶¶	216 ¶¶	230 μ	244 ∫
131 â	146 Æ	161 í	175 »	189 ∟	203 ¶¶	217 ¶	231 τ	245 ∫
132 ä	147 ô	162 ó	176 ▯	190 ∟	204 ¶¶	218 ¶	232 Φ	246 ÷
133 à	148 ö	163 ú	177 ▯	191 ∟	205 =	219 ▯	233 Θ	247 ≈
134 å	149 ò	164 ñ	178 ▯	192 ∟	206 ¶¶	220 ▯	234 Ω	248 °
135 ç	150 û	165 Ñ	179 ∟	193 ∟	207 ∟	221 ▯	235 δ	249 ·
136 ê	151 ù	166 ª	180 ∟	194 ∟	208 ∟	222 ▯	236 ∞	250 ·
137 ë	152 ŷ	167 °	181 ∟	195 ∟	209 ∟	223 ▯	237 φ	251 √
138 è	153 Ö	168 ¿	182 ∟	196 ∟	210 ∟	224 α	238 ε	252 n
139 ÿ	154 Ü	169 ¬	183 ∟	197 ∟	211 ∟	225 β	239 ∩	253 ²
140 î	155 ç	170 ¬	184 ∟	198 ∟	212 ∟	226 Γ	240 ≡	254 ▯
141 ï	156 £	171 ½	185 ∟	199 ∟	213 ∟	227 ∟	241 ±	255
142 Ä	157 ¥							

Note – There are full ASCII charts available in the Appendix of both volumes.

Section # 5: Remove

Crop Before

Crop – Removes any text either before or after a character or word in the filename as specified in the data entry field that appears to the right.

String = Andrew Gregory Macintyre.jpg


Remove (5) R

First n Last n

From to

Chars Words

Crop Before

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Gregory Macintyre.jpg

All characters in the string before the word ‘Greg’ have been cropped. In the example, ‘Andrew <space>’ has been removed.


Remove (5) R

First n Last n

From to

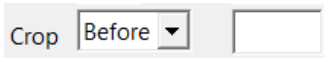
Chars Words

Crop After

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Andrew Greg.jpg

All characters after the word ‘Greg’ have been cropped. In the example, ‘ory Macintyre’ has been removed.

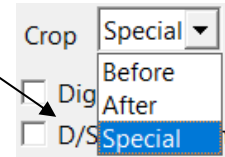
Section # 5: Remove



* The drop down list also provides for a 'Special' option that can be applied using the **wildcard * character**. This will not only remove the specified string but any characters before or after dependent on the position of the wildcard character (for this to work, the 'Special' option must be selected in the drop down list).

Normally a filename would be represented by:

<filename> . <ext> or *.*



This does not affect the filename extension. Therefore only one asterisk is used to represent the filename portion.

(1). Removing all characters *after* and including the specified string (wildcard positioned after):

Summer* when applied to...

Donna Summer- Mac Arthur Park Suite (original extended version).mp3

results in...

Donna <space>.mp3

This removed the string 'Summer' and all characters following.

Name	New Name
Donna Summer- Mac Arthur Park Suite (original extended version).mp3	Donna .mp3

(2). Removing all characters *before* and including the specified string (wildcard positioned before):

*Mac when applied to...

Donna Summer- Mac Arthur Park Suite (original extended version).mp3

results in...

<space> Arthur Park Suite (original extended version).mp3

This removed the string 'Mac' along with all characters prior.

Name	New Name
Donna Summer- Mac Arthur Park Suite (original extended version).mp3	Arthur Park Suite (original extended version).mp3

Section # 5: Remove

- (3). Specified string appears in multiple occurrences in filename:
 a. If specified string appears more than once, the action will be performed using only the **first** occurrence.

Using the filename...

Donna Summer- Love to Love You Baby (extended original version).mp3

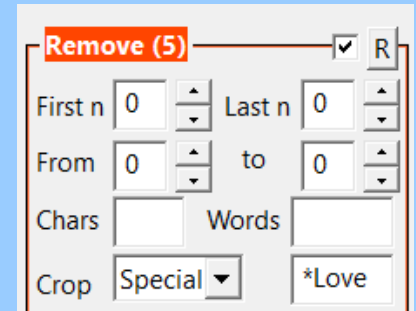
And the string...

*Love

results in ...

<space> to Love You Baby (extended original version).mp3

... because it has removed the first occurrence of the string, 'Love' as well as all characters prior.



Name	New Name
🔊 Donna Summer- Love to Love You Baby (extended original version).mp3	to Love You Baby (extended original version).mp3

- (4). Removing a section

In addition, you can remove a section in the middle using the format:

<starting position, left character or string> * <ending position, right character or string>

If the words 'Love to Love You' were to be removed...

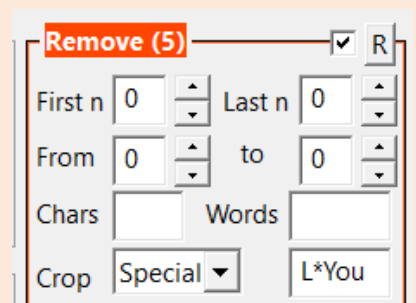
Donna Summer- Love to Love You Baby (extended original version).mp3

Use this...

L*You

results in...

Donna Summer- Baby (extended original version).mp3



This removes the first occurrence of the specified character 'L' as well as all characters leading up to and including the word 'You'

Name	New Name
🔊 Donna Summer- Love to Love You Baby (extended original version).mp3	Donna Summer- Baby (extended original version).mp3

Section # 5: Remove

Since this is complicated, here is another example.

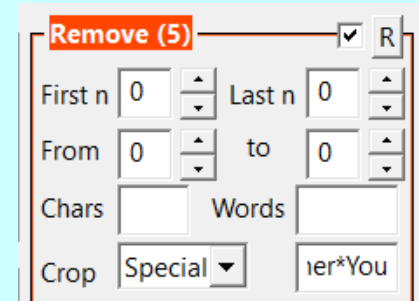
Summer*You

When applied to the filename,

Donna (Summer- Love to Love You) Baby (extended original version).mp3

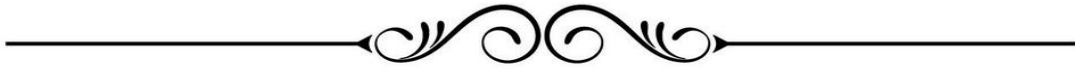
Results in ...

Donna Baby (extended original version).mp3



.. because it has removed the specified string 'Summer' unaffected any characters prior, and removing all characters leading up to and including the specified string 'You'.

Name ▲	New Name
Donna Summer- Love to Love You Baby (extended original version).mp3	Donna Baby (extended original version).mp3



Section # 6 - Move/Copy Parts

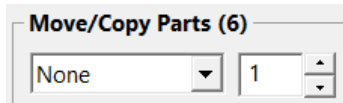
Section # 6: Move/Copy Parts

Move/Copy Parts – Move or copy a selected set of characters to another part of the filename.

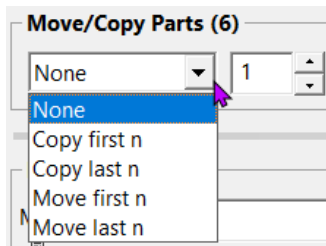


* This is useful when you have some code or date that appears at e.g., the end of the filename and you'd rather have it in the beginning.

a. Select what you want to do



Your options in the drop down list are:



None (default) – do nothing

Copy first n characters as specified by,  indicating character position in the filename

Copy last n characters as specified by,  indicating character position in the filename

Move first n characters as specified by,  indicating character position in the filename

Move last n characters as specified by,  indicating character position in the filename

Section # 6: Move/Copy Parts

b. Select the characters to copy

You do this by increasing or decreasing the number in

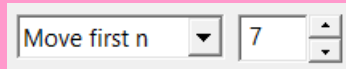


For example,

Using the filename, 130646_How to Avoid.png

filename	1	3	0	6	4	6	_	H	o	w	t	o	A	v	o	i	d		
position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Move first n characters

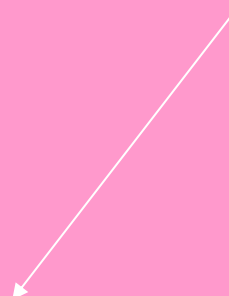


where n = 7 would create a string of 7 characters

results in the preliminary filename:

How to Avoid.png

Name ▲	New Name
130646_How to Avoid.png	How to Avoid.png



In this example, the string takes from position 1 through position 7 = '130646_'

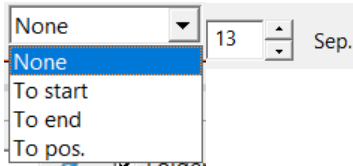
Note:

This is preliminary. The movement has not yet taken place.

Section # 6: Move/Copy Parts

c. Specify where you want to place the string

Your Options in the drop down list are:



None (default) – do nothing

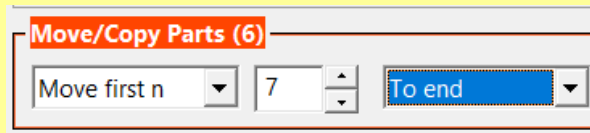
To end – place selected characters at the end of the filename

Example,

Using the filename, [130646_How to Avoid.png](#)

Having set the ‘Move first n characters where n= 7’, the string is ‘**130646_**’, ...

Current preliminary filename = [How to Avoid.png](#)



Results in:

How to Avoid130646_.png

Name ▲	New Name
130646_How to Avoid.png	How to Avoid130646_.png

The string, ‘130646_’, specified by the, ‘Move first n’ function, is placed at the end of the file, ‘How to Avoid’.png

Note:

Using, ‘**Move last n To End**’, would not change the filename, because all you would be doing is moving characters that are already present at the end to their same positions in the string. In order to use the option, ‘**To End**’, it would need to be used with the other options, ‘**Move first n**’, ‘**Copy first n**’, or ‘**Copy Last n**’.

Section # 6: Move/Copy Parts

To start – place selected characters at the beginning of the filename

Example,

using the filename, [How to Avoid_130646.png](#)

Set the ‘Move last n characters where n= 7’, the string is ‘[_130646](#)’, ...

Current preliminary filename = [How to Avoid.png](#)

results in:



[_130646How to Avoid.png](#)

Name ▲	New Name
How to Avoid_130646.png	_130646How to Avoid.png

The string, “[_130646](#)”, specified by the, ‘Move last n’ function, is placed at the start of the file, ‘130646_’.png

Note:

Using, **Move first n to Start**, would not change the filename, because all you would be doing is moving characters that are already present at the start to their same positions in the string. In order to use the option, ‘**To Start**’ it would need to be used with the other options, ‘**Copy first n**’, ‘**Copy last n**’, or ‘**Move last n**’.

Section # 6: Move/Copy Parts

The options, 'Copy last n', and 'Copy first n', are similar in how they function with their 'Move' counterparts, but I will provide one example for your complete understanding.

For example,

Using the filename, [130646_How to Avoid.png](#)

filename 1 3 0 6 4 6 _ H o w t o A v o i d
 position 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

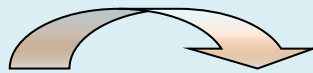
Copy first n characters where n = 7 would create a string of 7 characters

Since this is a 'Copy' function, no preliminary filename is generated

Select the placement of the string, '130646_', e.g., **To end**:

Move/Copy Parts (6)

results in:



Name ▲	New Name
130646_How to Avoid.png	130646_How to Avoid130646_.png

In this example, the string takes from position 1 through position 7 = '130646_' and copies it to the end of the filename, '130646_'How to Avoid'.

Section # 6: Move/Copy Parts

To pos(ition) – place selected characters to a position within the filename.

The position is specified by increasing or decreasing the number in

Move/Copy Parts (6)

Move first n

7

To pos.

13

For example,

130646_How to Avoid.png

filename	1	3	0	6	4	6	_	H	o	w		t	o		A	v	o	i	d
position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

To end up with (using the last example):

How to Avoid130646_.png

You would use:

Move first n

7

First, this takes the first seven characters of the filename = '130646_' and preliminary removes them.

You now have:

filename	H	o	w		t	o		A	v	o	i	d
position	1	2	3	4	5	6	7	8	9	10	11	12

Second, specify the Position in the string to where the string will be moved.

To pos.

13

This moves the string, '130646_', to position 13 in the filename. Position 13 is now at the end of the filename.

Results in:

How to Avoid130646_.png

Name ▲	New Name
130646_How to Avoid.png	How to Avoid130646_.png

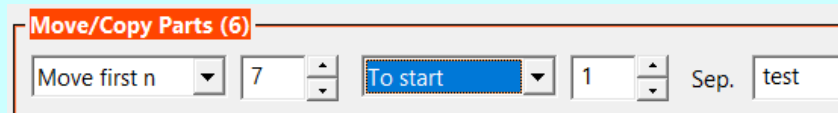
Section # 6: Move/Copy Parts

Sep.

There is one final section titled, ‘Sep.’ which stands for Separator. Any characters entered here will separate the string that was moved or copied from the original text remaining, , depending on the Start, End or Pos(ition) selection.

‘To Start’ selected, the characters appear immediately following the string at the beginning of the filename.

For example,

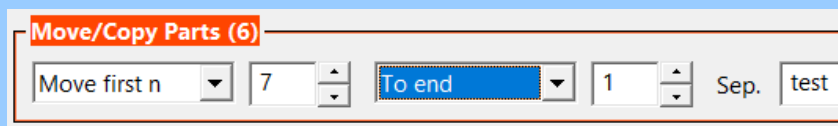


This will result in:

130646_testHow to Avoid.png

‘To End’ selected, the characters appear immediately following the string at the end of the filename.

For example,

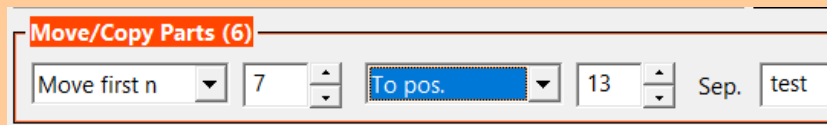


This will result in:

How to Avoidtest130646_.png

‘Pos(ition)’ selected, the characters appear on either side of the string.

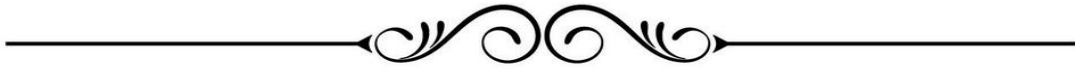
For example,



This will result in:

How to Avoidtest130646_test.png

The Sep. is typically a single character like a space rather than a string because it is intended to avoid run-together text as is seen in these examples, but multiple characters are supported.



Section # 7 – Add

Section # 7: Add

Add – add a string at the beginning, end or anywhere within the filename.

Prefix - Add the specified string at the **Beginning of the filename**. String is entered in the Prefix field.

Example,

results in:

cat130646_How to Avoid.png

Name ▲	New Name
130646_How to Avoid.png	cat130646_How to Avoid.png

Suffix – Add the specified string at the **End of the filename**. String is entered in the Suffix field.

Example,

results in:

130646_How to AvoidCat.png

Name ▲	New Name
130646_How to Avoid.png	130646_How to AvoidCat.png

Insert – Place the specified string **at the position 'Pos' within the filename**. String is entered in the Insert field.

Example,

results in:

130646_How toCat Avoid.png

Name ▲	New Name
130646_How to Avoid.png	130646_How toCat Avoid.png

Notes:

1. You can enter a space before 'Cat' or after or both as needed in the data fields so text does not run together.
2. Negative numbers can be entered directly into the 'at Pos' data field, or you can use the Up/Down indicators:



Section # 7: Add

In the final example under ‘Pos’, I will elaborate how the result was obtained:

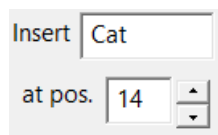
‘Pos’ refers to a position in the string where characters can be inserted.

In that example I used the filename,

130646_How to Avoid.png

```
filename  1 3 0 6 4 6 _ H o w   t   o       A v o i d
position  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

Thus when the Add options were set to:



‘Cat’ was inserted at position 14 which in the string is the <space> after ‘to’.

```
filename  1 3 0 6 4 6 _ H o w   t   o
position  1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Cat

resulting in:

```
filename  1 3 0 6 4 6 _ H o w   t   o   C A T   A v o i d
position  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
```

Name ▲	New Name
130646_How to Avoid.png	130646_How toCat Avoid.png

Notes:

- No, I did not include a <space> after ‘Cat’. The <space> character at position 17 is the <space> character originally at position 14 that was displaced.

Section # 7: Add

Word Space – this places a <space> character before any Capitalized word if there is not one already present. The first word of the filename is exempted from this action.

Example,

Add (7) R
 Prefix
 Insert
 at pos.
 Suffix
 Word Space

With Word Space active, and using the same example of the 'Insert' at 'Pos', the new result is:

130646_How to Cat Avoid.png

This example is simple. It could just as easily been done using a space character in the data entry field, but more useful if applied to filenames that already have run together text (PascalCase or camelCase).

e.g.,

[ThisIsAnExampleOfPascalCase.pdf](#)

[thisIsAnExampleOfCamelCase.pdf](#)

PascalCase is defined by the words all run together with each word in uppercase. Word Space uses the uppercase as a delimiter to separate the words. This will also work with camelCase where only the first word is lowercase and the rest are in uppercase. Even though the first word is lowercase, the delimiter on the second word will result in the correct separation. This will not work with a lowercase word anywhere after the first word.

Add (7) R
 Prefix
 Insert
 at pos.
 Suffix
 Word Space

Name	New Name
ThisIsAnExampleOfPascalCase.pdf	This Is An Example Of Pascal Case.pdf
Name	New Name
thisIsAnExampleOfCamelCase.pdf	this Is An Example Of Camel Case.pdf
Name	New Name
ThiswillNotWorkInusingWordSpace.pdf	Thiswill Not Work Inusing Word Space.pdf

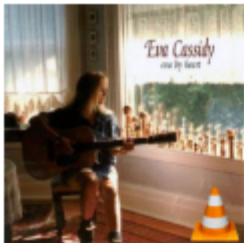
Section # 7: Add

Although BRU only supports three tags for Music ID3, I think it is important for your own information to know how the ID3 Metadata relates to the Metadata stored by Windows.

Here is how some of the ID3 tags in the Header of the file presented on the previous page, match up to Windows Explorer Preview of the Metadata:

Songbird.mp3

MP3 Audio File (VLC)



Contributing artists: Eva Cassidy
 Album: Eva by Heart
 Genre: Folk (0)(98)(99)Christ...
 Length: 00:03:47
 Rating: ☆☆☆☆☆
 Year: 1997
 Size: 8.68 MB
 #: 6
 Album artist: Eva Cassidy
 Title: Songbird

...image/jpeg...art\art..2..jpg.....JFIF.....

.....E·v·a· ·C·a·s·s·i·d·y·
E·v·a· ·b·y· ·H·e·a·r·t·
 ...·(·8·0·)·(·0·)·(·9·8·)·(·9·9·)·C·h·r·i·s·t·m·a·s·/·A·d·u·l·t· ·C·o·n·t·e·m·

.....1·9· 9·7·

TRCK.....6·
E·v·a· ·C·a·s·s·i·d·y·
S·o·n·g·b·i·r·d·

And this is how the ID3 Metadata matches up with the Windows File Properties (BRU tag support is in blue):

Artist

Album artist	Eva Cassidy	System.Music.AlbumArtist
--------------	-------------	--------------------------

Album

Album	Eva by Heart	System.Music.AlbumTitle
-------	--------------	-------------------------

Genre

Genre	Folk (0)(98)(99)Christ	System.Music.Genre
-------	------------------------	--------------------

Length:

Length	00:03:47	System.Media.Duration
--------	----------	-----------------------

Title

Title	Songbird	System.Title
-------	----------	--------------

Size

Size	8.68 MB	System.Size
------	---------	-------------

Year

Year	1997	System.Media.Year
------	------	-------------------

Track

#	6	System.Music.TrackNumber
---	---	--------------------------

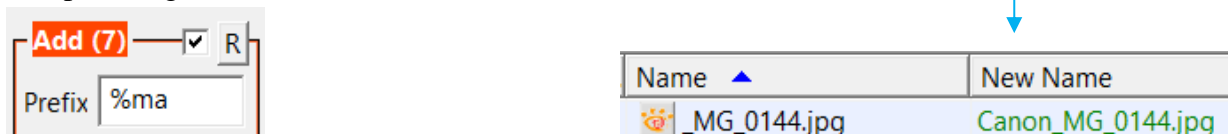
Section # 7: Add

Substitution Tags available for JPEG image files:

%a	- Aperture	%ma	- Camera Make
%c	- Comments	%mo	- Camera Model
%e	- Exposure		
%f	- Focal Length		
%xb	- Exposure Bias		

String = _MG_0144.jpg

Example using %ma



In this example, the value of Camera Make has been prefixed to the file, '_MG_0144.jpg'.

1. For this to work:

- You have to enable, 'Extract EXIF Data (Photos)', from the ID3/ EXIF Data / File Properties submenu of the Renaming Options Menu for MP3 Music files.
- The EXIF Data has to be present within the file, e.g., _MG_0144.jpg. Part of the information below is cut off – sorry ☹



```
H:\Test\_MG_0144.jpg
00000000 .....JFIF.....Exif..MM.*.....1.....2.....;.....i
000000C8 .....
00000190 .....
00000258 .....
00000320 .....
000003E8 .....
000004B0 .....
00000578 .....
00000640 .....
00000708 .....
000007D0 .....
00000898 .....Canon·Canon EOS DIGITAL REBEL XTi·Adobe Photoshop Lightroom·2010:04:28 20:54:51·unknown·
00000960 .....P.....X.....h.....00.....00.....
00000A28 .....
.....<.....2010:03:26 20:41:34·2010:03:26 20:41:34·Z!·
·XICC_PROFILE.....HLino··mnrRGB XYZ .....1·acspMSFT··IEC sRGB.....HP .....
ny·desc.....sRGB IEC61966-2.1.....sRGB IEC61966-2.1.....XYZ
·$.·desc.....IEC http://www.iec.ch.....IEC http://www.iec.ch.....
66-2.1 Default RGB colour space - sRGB.....desc.....,Reference Viewing Condition in IEC61966-2.1
....._·.....\.....XYZ .....L·V·P··W·meas.....sig .....CRT curv.....
.....%·+·2·8·>·E·L·R·Y·~·g·n·u·|.....&·/·8·A·K·T·]·g·q·z·
```

Section # 7: Add

Here is how some of the ID3 tags in the Header of the file presented on the previous page, match up to Windows Explorer Preview of the Metadata:

_MG_0144.jpg

JPG File



Date taken: 3/26/2010 8:41 PM
 Tags: PHT135
 Rating: ☆ ☆ ☆ ☆ ☆
 Dimensions: 453 x 640
 Size: 162 KB
 Title: Add a title
 Authors: unknown
 Comments: Add comments
 Camera maker: Canon
 Camera model: Canon EOS DIGITAL REBEL...

.....2010:03:26 20:41:34.

..P.H.T.1.3.5..

..unknown...

.....Canon.

..Canon EOS DIGITAL REBEL XTi.

The information that makes up this Metadata is recorded and stored in the header of the file. Much of it can be seen in the photos. However, not all Metadata is recorded for all files. The Metadata that is written is dependent on the device that created the file. Windows can also write Metadata and some programs will also add their own Metadata. There are also programs out there that the user can use to alter existing Metadata. The program, 'EXIFtool', comes to immediate mind, for example.

It should be noted, because I have seen this asked many times by BRU users – **No, BRU cannot alter existing EXIF or ID3 Metadata nor can it write this Metadata.** All it can do is to read this existing Metadata and extract that information for purposes of adding it as criteria for the renaming process. BRU **can** alter the Windows Properties, Date Created, Date Modified and Date Accessed through the Change File Timestamps function available in '**Section 14: Special**', but cannot change e.g., Taken (Original), which is a value taken from EXIF Photo.DateTimeOriginal.

I hope that puts the matter to bed once and for all.

Section # 7: Add

And this is how the EXIF Metadata matches up with the Windows File Properties (BRU tag support is in blue):

Exposure	Exposure time	1/60 sec.	System.Photo.ExposureTime
----------	---------------	-----------	---------------------------

Focal Length	Focal length	39 mm	System.Photo.FocalLength
--------------	--------------	-------	--------------------------

Exposure Bias	Exposure bias	-0.7 step	System.Photo.ExposureBias
---------------	---------------	-----------	---------------------------

Aperture	Aperture	f/5	System.Photo.Aperture
----------	----------	-----	-----------------------

Date taken:	Date taken	3/26/2010 8:41 PM	System.Photo.DateTaken
-------------	------------	-------------------	------------------------

PHT Tags:	Tags	PHT135	System.Keywords
-----------	------	--------	-----------------

Rating: Dimensions:	Dimensions	453 x 640	System.Image.Dimensions
------------------------	------------	-----------	-------------------------

Size:	Size	162 KB	System.Size
-------	------	--------	-------------

Authors:	Authors	unknown	System.Author
----------	---------	---------	---------------

Camera maker:	Camera maker	Canon	System.Photo.CameraManufacturer
---------------	--------------	-------	---------------------------------

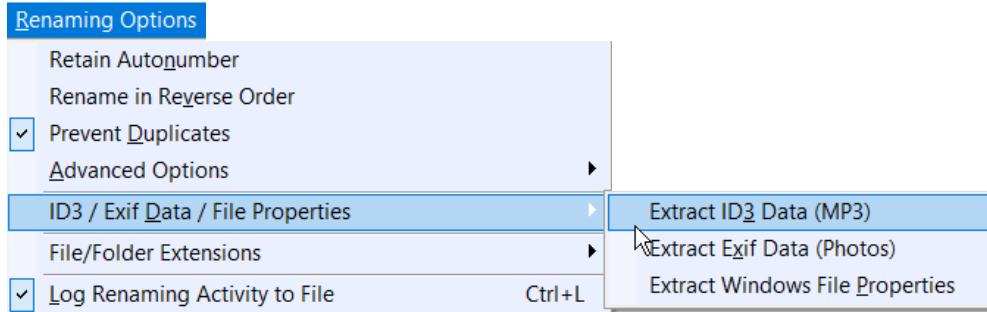
Camera model:	Camera model	Canon EOS DIGITAL F	System.Photo.CameraModel
---------------	--------------	---------------------	--------------------------

Comments: No Windows Property available for 'Comments'

Section # 7: Add

Notes:

1. Metadata in BRU includes EXIF, ID3 and Windows File Properties.
2. For EXIF data, the option, 'Extract-EXIF data (Photos)' must be enabled.
– see 'ID3 / EXIF Data / File Properties' selection under the Renaming Options Menu.
3. For MP3 files' Id3 data, the option, 'Extract-ID3 Data (Mp3)' must be enabled.
– see 'ID3 / EXIF Data / File Properties' selection under the Renaming Options Menu.
4. For Windows Properties data, the option, 'Extract Windows File Properties' must be enabled.
– see 'ID3 / EXIF Data / File Properties' selection under the Renaming Options Menu.



5. ID1 and ID3 refer to a Metadata container most often used in conjunction with the MP3 audio file format. It allows information such as the title, artist, album, track number, and other information about the file to be stored in the file itself. ID tags are identified in the header information of the MP3 file. For example:

ZTreeWin v2.4.172 - H: Carol of the Bells - Lindsey Stirling.mp3

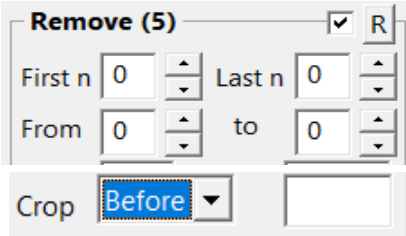
```
H:\Music\Christmas\Carol of the Bells - Lindsey Stirling.mp3
00000000 ID3.....TXXX...%.....m.a.j.o.r._b.r.a.n.d.....i.s.c
000000C8 ..C.a.r.o.l. .o.f. .t.h.e. .B.e.l.l.s.TXXX.....c.c
00000190 .....L.a.v.f.5.7..4.8..1.0.0.UFID...;..http://mus
00000258 8.1.9.c.--7.b.b.e.4.b.6.f.0.a.1.c.TPE1..#....L.i.n.d
00000270 2.8.b.7.TXXX...G...M.u.s.i.c.R.e.n.a.m.i.n.g..A.l.b.u.m
```

6. Enabling these options can slow down the processing. If you don't need these fields, then leave them unchecked.
7. ID3 v2 is currently not supported at the time of this writing.

Section # 7: Add**Other Tags Available in BRU****Removed Tag <removed>**

The <removed> tag is used to **add back** what was removed by ‘Section #5: Remove’ and **replace** it in another position..

It is limited and will only be effective on the following Remove functions:



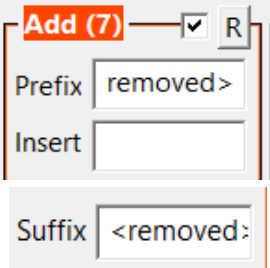
‘First – Last’ function

‘Position – From, To’ function

and the ‘Crop’ function.

Has no effect on any other functions, e.g., Chars, Words, etc.

The <removed> tag can be used in the following Add functions:




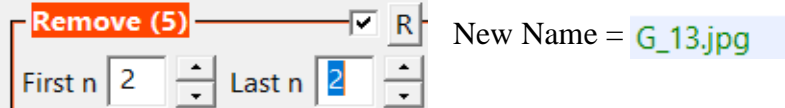
‘Prefix’ function

‘Insert’ function

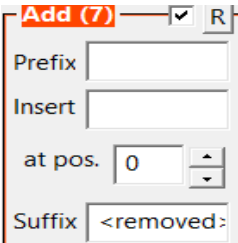
And ‘Suffix’ function

All at the same time if desired.

Example: Name =  _MG_1368.jpg



New Name = G_13.jpg

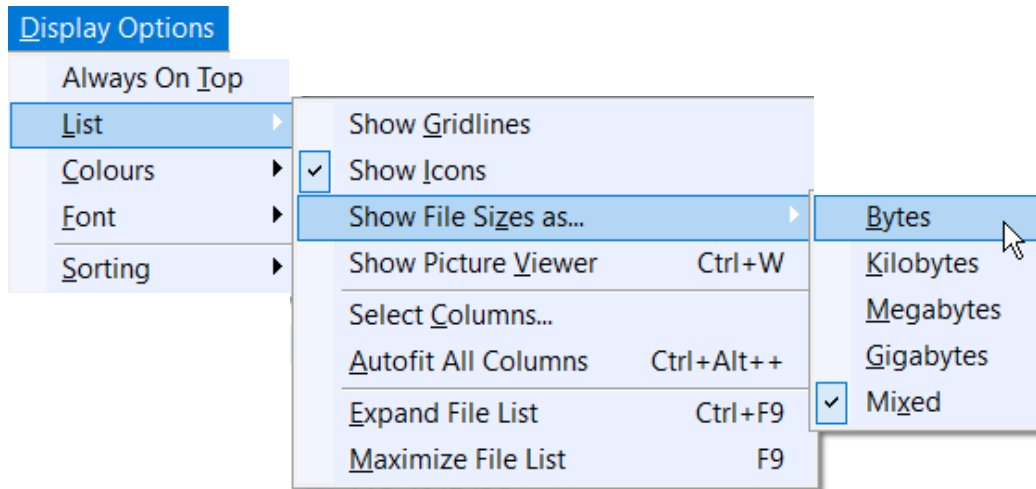


New Name = G_13_M68.jpg

The Removed characters of ‘_M, 68’ have been replaced in the filename’s Suffix

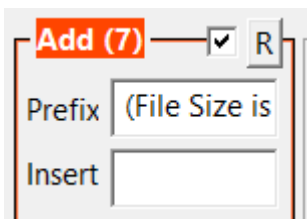
Section # 7: Add***File Size %z***


%z can be entered in the data entry fields, **Prefix**, **Suffix** and **Insert**, to append the file size. The format is determined under the 'List' menu item of the Display Options Menu by:



Example:

Using the string, '(File Size is %z)' in the Prefix (and the format either as 'Mixed; or 'Kilobytes')




Name =  _MG_1368.jpg

New Name = (File Size is 192 KB)_MG_1368.jpg

Section # 7: Add**Hash Value Tags****Hash Values**

A Hash Value or 'Checksum' is produced by an algorithm calculated from a file. This value is then attached to the file and can then be used to check the file's integrity at a later time. This is because the checksum changes as the file is altered. Cryptographic Hash Codes, such as the ones illustrated here, are by definition a one-way hash and can't be "decrypted" or reversed - all you can do is find a value which hashes to the expected value.

Example file used =  _MG_1368.jpg added spaces at end for clarity (i.e. '<(hash:crc32)> ')

The Hash Values that BRU generates are:

Cyclic Redundancy Check (CRC)

CRC is an error-detecting code commonly used to detect accidental changes to raw data. When a file gets entered into a system, blocks of data are checked with a calculated value attached (a 'Checksum'). On retrieval of the file, the calculation is repeated (Cyclic Redundancy) and, in the event the check values do not match, a CRC error is generated. Because the check value has a fixed length, the algorithm that generates it is occasionally used as a hash function.

CRC-32 is one of those algorithms.

BRU tag- <(hash:crc32)>

New Name =

918b5944 _MG_1368.jpg

Keccak

Keccak is another cryptographic hash function. The algorithm uses what is referred to as 'sponge construction'. Sponge construction takes an input bit stream ('absorbing') of any length and produces an output bit stream ('squeezing') of any desired length.

BRU tag- <(hash:keccak)>

New Name =

63d46c0bc68435895d192eed3eeb67864928bc8fd30a88fd76d3155249f802a7 _MG_1368.jpg

Section # 7: Add**Secure Hash Algorithm (SHA)**

SHA-1, SHA-2 and SHA-3 are cryptographic hash functions all designed by the United States National Security Agency, and is a U.S. Federal Information Processing Standard. They take input and produce a fixed-length hash value known as a message digest – typically rendered as a hexadecimal number.

SHA-1 has security concerns (code has been broken and compromised) and has been replaced by SHA-2. SHA-256 is a SHA-2 hash function computed with a 32-bit word. SHA-3 has not replaced SHA-2 but is used alongside and is based on the Keccak algorithm.

BRU tags - <(hash:sha1)> <(hash:sha256)> <(hash:sha3)>

SHA-1

New Name =

2cfc4e57e54b17c1b372f3930cdd56cca3e53319 _MG_1368.jpg

SHA-2

New Name =

82347c1da98e8697805a2bdd73a862d81e6ffb36f8c71cce67f3daf2dd869f7d _MG_1368.jpg

SHA-3

New Name =

f88d93c0fe7efb7a24554130d38f59a46182c66ca559a331ce37f8095c4339ec _MG_1368.jpg

Section # 7: Add

MD5 (Message-Digest)

This algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it proved too vulnerable, but remains as an excellent checksum to verify data integrity, but only against unintentional corruption. MD5 is commonly in practice by the typical computer user for file integrity checks. There are many freeware programs out there that can both generate and verify MD5 Hash Values.

BRU tag- `<(hash:md5)>`

New Name = `9cc130c0cdb565a1a922c930c3ef22e _MG_1368.jpg`

Notes:

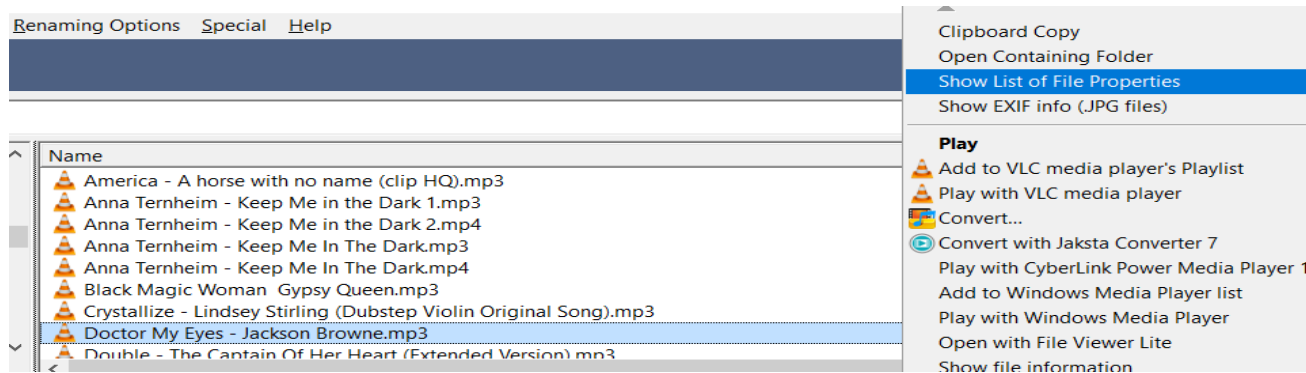
1. The angular brackets are part of the syntax for any of these tags and must be entered.

Section # 7: Add

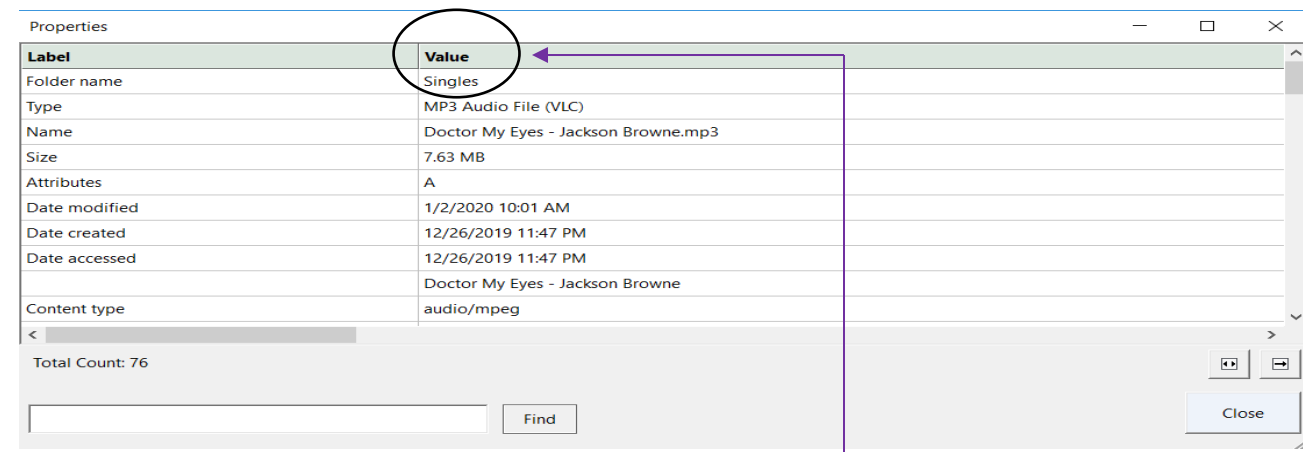
Using Windows File Properties (Windows Vista or Newer)

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. BRU supports using Windows Property values in ‘Section #7: Add’ in ‘Prefix’, ‘Suffix’ and ‘Insert’ data entry fields as well as under ‘Section #14: JavaScript’.

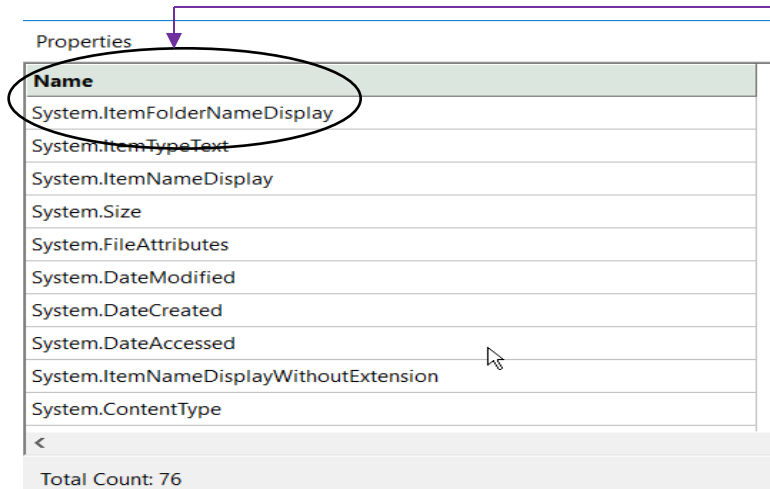
To see the available properties for a file, right click and select, ‘Show List of File Properties’ from the context menu:



This displays the Windows Properties for the selected file. This shows you the ‘Label’ property and the current Value.



If you scroll over, you can see the ‘Name’ property, e.g., for ‘Singles’, this is ‘System.ItemFolderNameDisplay’.



Alternatively, you can also see the file properties of a file in Windows Explorer: right-click on a file and select ‘Properties’ and then ‘Details’.

(No photo available)

Section # 7: Add

BRU can use either 'Label' or 'Name' Windows Properties.

Format: <(label | name)>

e.g., <(Folder name)> or <(System.ItemFolderNameDisplay)>

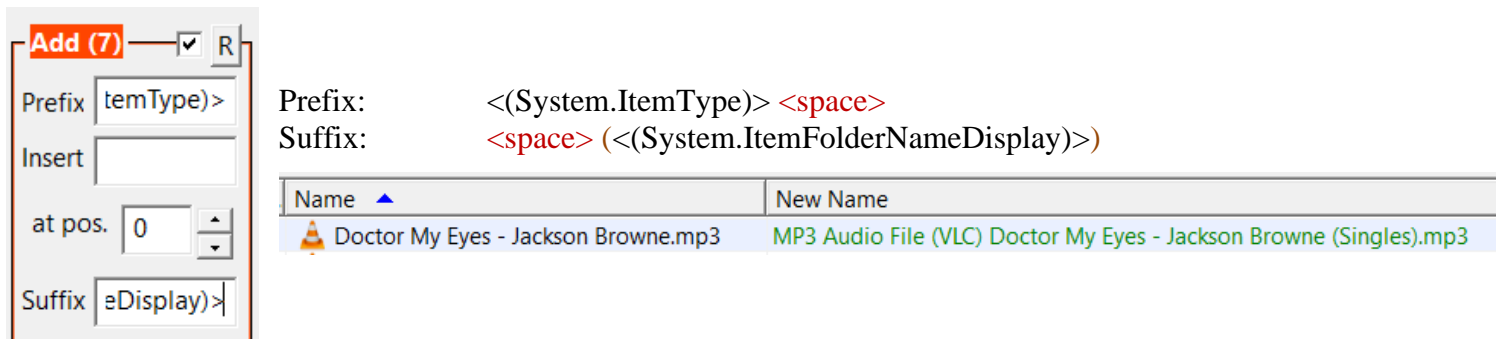
Where:

Folder name is a 'Label' property

System.ItemFolderNameDisplay is a 'Name' property

Both refer to the same current value - 'Singles'. This value is what is used to rename the file.

Example using <(System.ItemType)> as Prefix and <(System.ItemFolderNameDisplay)> as Suffix:



Prefix: <(System.ItemType)> <space>
 Suffix: <space> <(System.ItemFolderNameDisplay)>

Name	New Name
Doctor My Eyes - Jackson Browne.mp3	MP3 Audio File (VLC) Doctor My Eyes - Jackson Browne (Singles).mp3

Notes:

1. In the photo for 'Section #7: Add', you can't see the entire values for the Prefix and Suffix.

a. Prefix is Windows Property 'Name' using –

'<(System.ItemType)>' and for clarity, followed by a space = 'MP3 Audio File (VLC)' <space>

b. Suffix is Windows Property 'name' to which I added the preceding space and the outer parentheses for clarity using –

'<(System.ItemFolderNameDisplay)>' = <space> '(Singles)'

2. You are not limited to just JPEG and MP3 files. Windows Properties are available for any file.

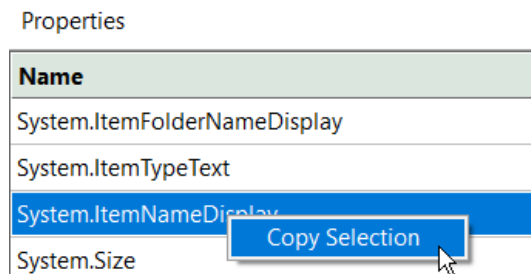
3. More information on the Windows Property System that can be set on Windows files can be found here:
<https://docs.microsoft.com/en-us/windows/win32/properties/props>

Section # 7: Add

Tip:

If you have a problem getting BRU to recognize the tag, (most likely a 'Name' or syntax error) –

1. Bring up the Windows Properties list.
2. Locate the property you want to include (in either the 'Label' or 'Name' column)
3. Right click on that property and select 'Copy Selection'



4. Paste it into the data entry field.

Prefix: System.ItemNameDisplay

5. Add the appropriate syntax, parentheses and angular brackets.

Prefix : <(System.ItemNameDisplay)>

Now you know the tag is correct and BRU should recognize it.

IMPORTANT:

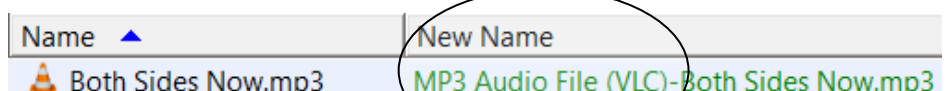
You can also specify some extra characters '*within*' the property <> tag and they will display in the tag as long as the specified property is not empty (the property has no value assigned). If the property is empty, the whole tag <> will be empty.

e.g., Adding <hyphen> character *within* the 'Subject' property tag **is dependent on if that property is empty** or not –

<(System.ItemType)->

Note that this is not the same as –

<(System.ItemType)>-



.. where the <hyphen> character is added *outside* of the property tag and **will always display regardless** of the current status of the property.

Section # 7: Add**File Properties as Dates and Numbers**

If you want to add a file property to a file name as a date or as a number, use the **Property Formatting Markers**.

'#' format a property as a number.

Note:

If a file property value has numbers and letters, the letters will be ignored.

File name = 

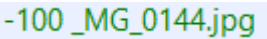
Example using the 'ISO speed' 'Label' property:

Properties

Label	Value
Exposure program	Normal
ISO speed	ISO-100


Suffix : < - (#ISO speed)>

New Name would add a suffix of ' - 100'. The file property is added as a number only removing the ASCII characters 'ISO-'.

New Name = 

Suffix : < - (ISO speed)>

New name would add a suffix of ' - ISO100'. The file property is added as a text string.

New name = 

Section # 7: Add

'\$' format a property as a date. Format is determined by the **date and time Fmt options** in '**Section #8: Auto-Date**'.

Example using the 'System.Photo.DateTaken' 'name' property:

Label	Value	Name
Exposure program	Normal	System.Photo.ExposureProgram
ISO speed	ISO-100	System.Photo.ISOSpeed
Date taken	3/26/2010 8:41 PM	System.Photo.DateTaken
Shutter speed	1/60 sec.	System.Photo.ShutterSpeed

If –

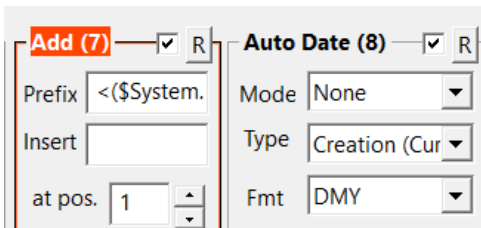
Prefix : <(\$System.Photo.DateTaken) - >

This will add the 'date taken' followed by a ' – ', formatted according to values in '**Section #8: Auto-Date**'.

File name = _MG_0144.jpg

Settings for Add = Prefix of <(\$System.Photo.DateTaken) - > followed by an added space character for clarity.

Settings for Auto Date Fmt = DMY



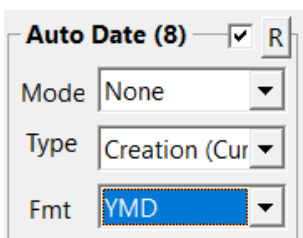
New Name = 260310_MG_0144.jpg

where:

26 = Day 03 = Month 10 = 2010 Year

.. If I change the format of Auto Date -

Settings for Auto Date Fmt = YMD



New Name = 100326_MG_0144.jpg

where:

10 = 2010 Year 03 = Month 26 = Day

Section # 7: Add

Using Windows Clipboard Data

BRU supports using the data contained within the Windows Clipboard in the ‘Add’ renaming process – using the **Prefix**, **Suffix** and **Insert** functions.

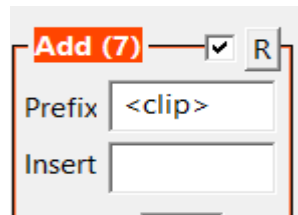
Format:

<clip>


Example:

I copied the text, ‘This is a Test -’ into the Windows Clipboard (Ctrl + C).

Using a Prefix of <clip>



If –

Name =  _MG_0144.jpg

New Name = This is a Test - _MG_0144.jpg

This will also work with multiple lines of text.

Clipboard contents =

‘This is a Test *
This is another Test –’

New Name = This is a Test *This is another Test – _MG_0144.jpg

Notes:

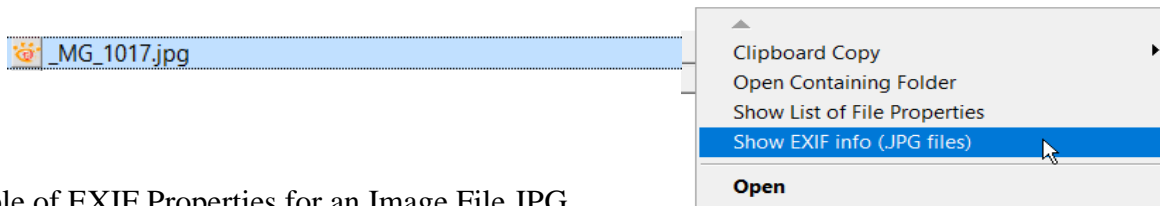
1. Only works with text data. Graphic data will not be displayed.
2. If clipboard contents change, the new data will not be reflected in New Name until you change the focus (e.g., moving the mouse to another file and back again). This will refresh the file name (or press F5 to refresh all files).
3. If you want to edit the Windows Clipboard data before committing to Rename, use a Clipboard Manager, e.g., Clipmate.by Thornsoft.com.
4. Displays in red because the asterisk is not a legal character in a filename.

Section # 7: Add

Using EXIF Tags

In addition to Substitute tags, BRU fully supports the extended version 2.2 EXIF Tags. They can be added as text, number or as a formatted date, using the methods illustrated previously. Tags can also be used in JavaScript.

You can show a list of all available EXIF attributes for a file by right-clicking on the file in the main file list and select 'Show EXIF info (.JPG files)' from the context menu.



Example of EXIF Properties for an Image File JPG

EXIF Property	Value
exif:ImageResolution	640x480
exif:ImageWidth	640
exif:ImageHeight	480
exif:Make	NIKON
exif:Model	COOLPIX P6000
exif:Orientation	1
exif:XResolution	300
exif:YResolution	300
exif:ResolutionUnit	2
exif:Software	Nikon Transfer 1.1 W
exif:DateTime	2008:11:01 21:15:09
exif:DateTimeOriginal	2008:10:22 16:46:53
exif:DateTimeDigitized	2008:10:22 16:46:53
exif:ExposureTime	0.015432
exif:FNumber	5.6
exif:ExposureProgram	2
exif:ISOSpeedRatings	64
exif:FocalLength	22.1
exif:Flash	16
exif:MeteringMode	5
exif:LensInfo.FocalLengthIn35mm	103
exif:GeoLocation.Latitude	43.468243
exif:GeoLocation.Longitude	11.880172
exif:GeoLocation.GPSMapDatum	WGS-84
exif:GeoLocation.GPSTimeStamp	14 45 20.91
exif:GeoLocation.GPSDateStamp	2008:10:23

Section # 7: Add

Format:

<(EXIFproperty)>

Where:


EXIFproperty is made up of-

‘EXIF’ followed by a colon, ‘:’ followed by the property **name**

i.e.

<(EXIF:ImageWidth)>
 <(EXIF:ImageResolution)>
 <(EXIF:Software)>

Example:

Name =  _MG_4051.JPG

If the image has the following EXIF data available →

Using a Prefix of <(EXIF:Model)>,

New Name = **Canon EOS 50D_MG_4051.JPG**

EXIF properties for _MG_4051.JPG

EXIF Property	Value
exif:ImageResolution	4752x3168
exif:ImageWidth	4752
exif:ImageHeight	3168
exif:Make	Canon
exif:Model	Canon EOS 50D
exif:Orientation	6
exif:DateTime	2011:07:19 19:24:34
exif:DateTimeOriginal	2011:07:19 19:24:34
exif:DateTimeDigitized	2011:07:19 19:24:34
exif:SubSecTimeOriginal	54
exif:ExposureTime	0.010000
Total Count: 23	

Notes:


1. If you have a problem getting BRU to recognize the tag, (most likely a syntax or ‘Name’ error) – right click on the desired EXIF property and select, ‘Copy Section’, then you can paste into the ‘Add’ data entry field enclosed with the appropriate syntax ‘<()>’. This way you can be assured the tag is correct and BRU should now recognize it.

Section # 7: Add**IMPORTANT:**

You can also specify some extra characters '*within*' the EXIF < > tag and they will display in the tag as long as the specified property is not empty (the property has no value assigned). If the property is empty, the whole tag < > will be empty.

For example, adding the <hyphen> character **within** the 'EXIF:DateTimeOriginal' property tag,

<- (#EXIF:DateTimeOriginal)>

Name ▲	New Name
 DSCN0001.JPG	- 20090901 DSCN0001.JPG

Note that this is not the same as –

-< (#EXIF:DateTimeOriginal)>

.. where the <hyphen> character is added **outside** of the EXIF tag and will always display regardless of the current status of the property.

IMPORTANT:

Using EXIF tags requires enabling 'ID3 /EXIF Data/ File Properties' in the Renaming Options Menu.

Notes:

1. A <space> was added to the Prefix of '**Section #7: Add**', for clarity in the above New Name photo.

Section # 7: Add**EXIF Properties as Dates and Numbers**


If you want to add an EXIF property to a file name as a date or as a number, use the **Property Formatting Markers**.

'#' format a property as a number.

Note:

If a file property value has numbers and letters, the letters will be ignored.

Example using the 'EXIF:DateTimeOriginal' property:

Name:  Ducky 5003.jpg

EXIF properties for Ducky 5003.jpg	
EXIF Property	Value
exif:DateTime	2009:06:27 23:19:49
exif:DateTimeOriginal	2009:06:27 15:40:12

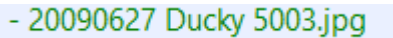
Suffix : <- (#EXIF:DateTimeOriginal)> followed by a <space> character for clarity.

New Name will add a suffix of ' - 20090627'. The file property is added as a text representation of the numeric portion only, stopping at the first non-numeric character it encounters. This results in removing any data **after and including** the <space> character (including the Hour:Minute:Seconds data).

The colon character is ignored because it is an **Invalid** character in a filename designation (BRU automatically strips it away), otherwise you would obtain '2009'. This is why the end result is '20090602' and not '20090627154012'.

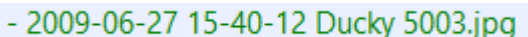
It should be noted that I used the EXIF:DateTimeOriginal Metadata to show a distinction between the '#' and '\$' markers. You would typically want to use the '\$' marker for Timestamps and reserve using the '#' for EXIF tags that hold alpha-numeric textual data.

With the '#' Property Formatting Marker -

New Name = 

Without the '#' Property Formatting marker, the file property is added as a text string in its entirety. New name will add a suffix of ' - 2009-06-27 15-40-12' -

Suffix : <- (EXIF:DateTimeOriginal)>

New Name = 

Section # 7: Add

'\$' format a property as a date. Format is determined by the **date and time Fmt options** in '**Section #8: Auto-Date**'.

Example using the 'EXIF:DateTimeOriginal' property:

File name =  Ducky 5003.jpg

EXIF properties for Ducky 5003.jpg

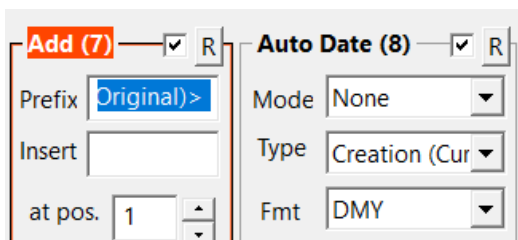
EXIF Property	Value
exif.DateTime	2009:06:27 23:19:49
exif.DateTimeOriginal	2009:06:27 15:40:12

If –

Prefix : <- (#EXIF:DateTimeOriginal)> followed by a space character for clarity.

This will add the 'date and time originally taken' preceded by a <hyphen> and followed by a space, formatted according to values in '**Section #8: Auto-Date**'.

Settings for Auto Date Fmt = DMY



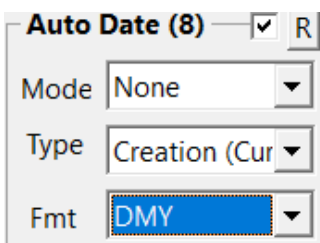
New Name = - 270609 Ducky 5003.jpg

where:

27 = Day 06 = Month 09 = 2009 Year

.. If I change the format of Auto Date -

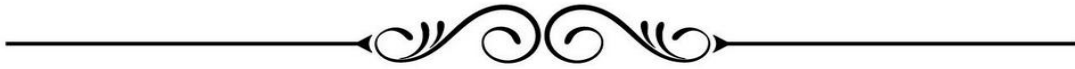
Settings for Auto Date Fmt = DMY HMS



New Name = - 270609154012 Ducky 5003.jpg

where:

27 = Day 06 = Month 09 = 2009 Year
15 = hour (3:00 pm) 40 = minutes 12 = seconds



Section # 8 - Auto Date

Section # 8: Auto Date

Auto Date – append a date to the beginning or end of the filename.

Mode

Must be set to other than ‘None’ for Auto Date to be applied to the New Name.

Mode lets you choose between:

- None (default) – no additions
- Prefix – add date at beginning of filename
- Suffix – add date at end of filename

DSCN0001.JPG	201217 DSCN0001.JPG
DSCN0001.JPG	DSCN0001 201217.JPG

I added a <space> character as a ‘Sep.’(arator) for clarification in photos.

Type

Type selects the type of date data to be used. You can select among (Windows File properties):

- Creation date (Current) – the date used will be the current creation date
- Modified date (Current) – the date used will be the date the file was last modified
- Accessed (Current) – the date used will be the date the file was last accessed
- Current – the date used will be today’s date

There are also additional types available dependent on other settings.

For more information on these refer to:

‘Adding a New Date & Timestamp’, ‘ Using the EXIF Property “Taken (Original)” ’ in this section.

Section # 8: Auto Date

Fmt

Format – this is the format used for the Date. Select among –

DMY-	Day Month Year	17 12 20	
DMY HM-	Day Month Year Hour Minute	17 12 20 22 34	24 hour military time
DMY HMS-	Day Month Year Hour Minute Seconds	17 12 20 22 34 30	24 hour military time
MDY-	Month Day Year	12 17 20	
MDY HM-	Month Day Year Hour Minute	12 17 20 22 35	24 hour military time
MDY HMS-	Month Day Year Hour Minute Seconds	12 17 20 22 35 27	24 hour military time
YMD-	Year Month Day	20 12 17	
YMD HM-	Year Month Day Hour Minute	20 12 17 22 35	24 hour military time
YMD HMS-	Year Month Day Hour Minute Seconds	20 12 17 22 36 12	24 hour military time

I added a <space> character as a ‘Seg.’ for clarification in photos.

Sep.

Separator character – any character(s) entered here will be placed between the date data and the original filename.

Example with Mode = Suffix and Fmt DMY, using the semi-colon as a separator will change New Name to:

New Name = Michael 5661;200112.jpg

Seg.

Segment character – this character(s) is used to distinguish between the different sections that make up the date.

Example, using the <hyphen> as a segment character, the date changes from:

To: 200112
20-01-12

New Name = Michael 566120-01-12.jpg

Section # 8: Auto Date

Cent.

 Cent

Century – normally the year is expressed as two digits but if this is checked, it will be expressed as 4 digits.

e.g. 02-11-2011 instead of 02-11-11

Custom Format

In addition there is a CUSTOM format that can be selected. This allows a custom format to be entered in the Custom data entry field:

The format codes are:

Code	Meaning	Using
%a	Abbreviated weekday name	lowercase a
%A	Full Weekday name	uppercase A
%b	Abbreviated month name	lowercase b
%B	Full month name	uppercase B
%d	Day of Month (01-31)	lowercase d
%H	Hour in 24-hour format (00-23)	uppercase H
%I	Hour in 12-hour format (1-12)	uppercase I
%j	Day of Year (01-366)	lowercase j
%m	Month number (01-12)	lowercase m
%M	Minute (00-59)	uppercase M
%p	AM/PM Indicator	lowercase p
%S	Seconds (00-59)	uppercase S
%U	Week number of year (00-53), with Sunday as the first day of the week	uppercase U
%w	Weekday (0-6), with Sunday=0.	lowercase w
%W	Week number of year (00-53), with Monday as the first day of the week	uppercase W
%y	Year, with no century indicator (00-99)	lowercase y
%Y	Year, with century indicator (e.g. 2004)	uppercase Y
%z	Time zone name	lowercase z
%%	Percentage sign	percentage sign %

These, as well as any characters or text you may want to include can be entered.

Example,

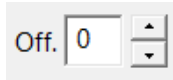
"Created on %a, %d %B, %Y"

Results in:

"Created on Tue, 25th March 2004"

Section # 8: Auto Date

Off.



Offset – can be expressed as **positive** or **negative** value. This can be used to alter the date by setting a **future (positive)** or **past (negative)** date. The offset value is applied to **time** and is in the form of **hours**. So an offset of +1 is an increment of 1 hour. An offset of –1 is a decrement of 1 hour.

For example to set back the date by 24 hours to yesterday, enter –24 in the Offset field using a Type as Current Date.

Or... setting a file timestamp to account for daylight savings time using an offset value of –1 (Fall) or +1 (Spring).

If the offset's value is in hours and 1 Day = 24 hours, as the offset value increases it will affect first the day, then the month and then the year.

So, to increment by one ...

day is.. + 24 hours (1x24)

month is.. + 720 hours (30 x 24)

year is .. + 8760 hours (365 x 24)

Notes:

1. Month calculation can also be based on 31 (744 hours), 28 (672 hours) and sometimes 29 (696 hours).

2. Also refer to 'Special Menu – Change Timestamps – Delta'.

3. Negative numbers can be entered directly into 'Offset' data field or you can use the Up/Down indicators:



4. Time is entered in 12 hour format but New Name appended Timestamps are expressed in 24 hour Military Time.

Hour: (the word, 'hours', is optional written or spoken)

Tim Tidbit –

12:00 am = 00 (zero zero hundred hours)

1:00 am = 01 (zero one hundred hours)

2:00 am = 02 (zero two hundred hours)

3:00 am = 03 .

4:00 am = 04 .

5:00 am = 05 etc.

6:00 am = 06

7:00 am = 07

8:00 am = 08

9:00 am = 09

10:00 am = 10 (ten hundred)

11:00 am = 11 (eleven hundred)

12:00 pm = 12 (twelve hundred)

1:00 pm = 13

2:00 pm = 14

3:00 pm = 15

4:00 pm = 16

5:00 pm = 17

6:00 pm = 18

7:00 pm = 19

8:00 pm = 20

9:00 pm = 21

10:00 pm = 22

11:00 pm = 23

Verbally properly expressed as:

12 hr. Military

11:59 pm = twenty three fifty nine hundred

12:06 am = zero zero six hundred (hours)

6:30 pm = eighteen thirty hundred (hours)

6:35 pm = eighteen thirty five hundred

3:01 am = zero three one hundred (hours)

2:15 am = zero two fifteen hundred

2:37 am = zero two thirty seven hundred

6:00 am = zero six hundred (hours)

not .. 'oh six hundred'

e.g., **7:30 pm** is written as **1930** expressed as **nineteen thirty hundred hours**

Section # 8: Auto Date

There is one more consideration. When applied against a file date, the offset value is taken **in conjunction** with the **timestamp of the file** to obtain the final value.

Example using ‘Taken (Original) EXIF’ as type,

The filename

SAM_0096.JPG

Using these settings:

Mode is Prefix.

The Taken (Original Date) is May 07, 2011

The date format is Month Day Year

The Separator character is a single <space> (can’t tell from illustration but it is there)

The Segment character is a single <hyphen>

The Century is enabled so the year is expressed as 4 digits

Custom format is not used.

Offset is currently set at zero.

results in:

Name	New Name
SAM_0096.JPG	05-07-2011SAM_0096.JPG

What If I needed to use an offset to change the date forward to May 8th ? I would need to use a positive offset value, but which value? Look at the time stamp...

If you look to the right in the Content Pane, you can see the timestamp of the Taken (Original) date as 4:57 AM. The hour value is 4. The offset must move to midnight of the next day, hour 24.

Type	Size	C..	Modi...	Acce...	Length	Taken (Original)
JPG ...	2 MB	1/20...	5/7/2...	1/20/...	22	5/7/2011 4:57:34 AM

If the Offset is increased to **20**, $20 + 4 \text{ hours} = 24 \text{ hours}$, the day will change from May 7 to May 8.

Name	New Name
SAM_0096.JPG	05-08-2011SAM_0096.JPG

For yesterday, use a value of **-5** because to get back to yesterday you have to get past midnight of the current day.

Name	New Name
SAM_0096.JPG	05-06-2011SAM_0096.JPG

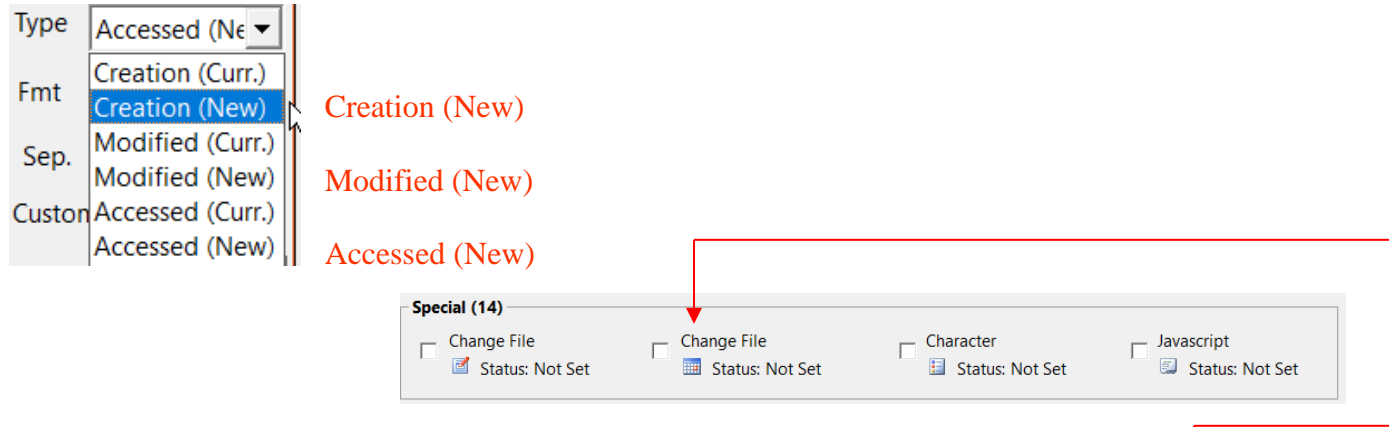
Therefore, $4\text{am} - 5 \text{ hours} = -1$, or hour 23 of the previous day if you think in terms of ...

0 = hour 24 (midnight of the current day) , and -1 = hour 23 (11 pm of the previous day)

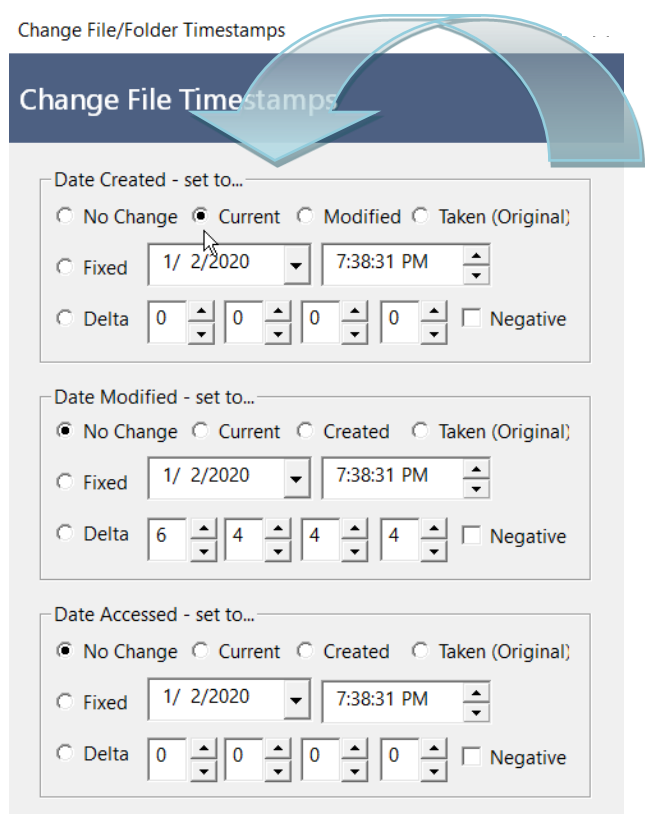
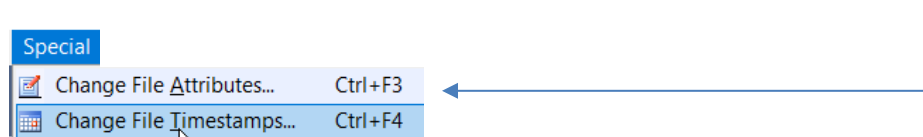
Section # 8: Auto Date

Adding a New Date & Timestamp

In the 'Type' of date, there are three additional options for using the changed file date and timestamp as part of the new file name. This directs BRU to use the **Creation, Modified** or **Accessed Dates** that were entered in the 'Change File Timestamps' function of 'Section #14: Special' as the source data for 'Type' instead of the current Windows Properties values for those timestamps.



This works in conjunction with 'Change File Timestamps' of the Special Menu or 'Section #14: Special'.



Data entered here will modify the file's Timestamp.

In this example, the Date Created is set to 'Current', e.g., 1\3\2020. This will change the file's creation date from it's original date of 1\20\2012 to the current date:

From this -

Created
1/20/2012 6:59:33 PM

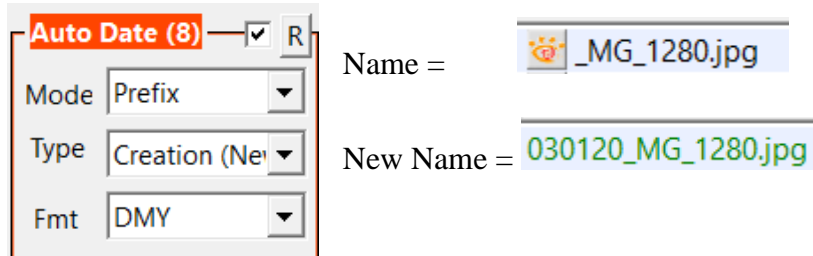
To this -

Created
1/3/2020 3:46:28 AM

Selecting one of the three, 'Creation (New)', Modified (New)' or 'Accessed (New)' Type(s), makes this data available to be included as part of the new file name.

Section # 8: Auto Date

Using the same example on this filename,



When you click the 'Rename' button, not only will it have changed the file's creation date, but this information will also be assigned to the file name as well.

Notes:

1. The 'Change File Stamps' option from the Special Menu or '[Section #14: Special](#)' must contain data in order for Auto Date's (New) change to occur.
2. For more information on setting options in the 'Change File Stamps', refer to the Special Menu under the Menus Section.

3. **Tim Tidbit - Military Time Conversion:**

If Military Time is PM, Subtract 12 to convert Military 24 hour time to Standard 12 hour time:

$$1800 - 12 = 6 \text{ or } 6 \text{ pm}$$

If Standard Time is PM, Add 12 to convert Standard 12 hour time to Military 24 hour time:

$$6:00 \text{ pm} = 6 + 12 = 18 \text{ or } 1800 \text{ (18 hundred hours)}$$

4. **Tim Tidbit - Military Time Conversion trick:**

Drop the one and minus two –

e.g.,

$$18 \text{ hundred hours} = 1800 \quad \text{drop the one} = \quad 8 \quad \text{minus two} = \quad 6 = 6:00 \text{ pm}$$

5. If a Timestamp is appended in New Name it uses Military Time unless Custom Format 12 hour time is specified.
 - a. 24 hour time is similar to Military Time except that Military Time does not use colons and expresses 8 as 08.
 - b. BRU uses Military Time because of the numeric notation of 08 vs 8. Colons can be added as Seg(ment) characters.

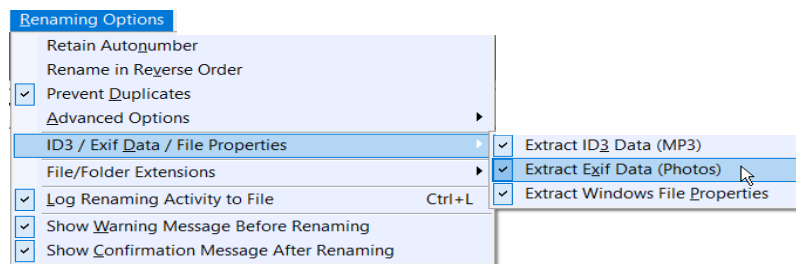
Section # 8: Auto Date

Using the EXIF property 'Taken (Original)'

Image files that were created with a digital device will have embedded Metadata information called EXIF data. One of these bits of information is called 'Taken (Original)', or the 'original date' the image was taken.

Name	New Name	Size	Created	Modified	Accessed	Track	Status
Michael 5675.jpg	Michael 5675.jpg	108 KB	1/20/2012 6:56:06 P...	1/20/2012 3:46:42 P...	1/20/2012 6:56:06 P...	0	
Michael 5705.jpg	Michael 5705.jpg	119 KB	1/20/2012 6:56:06 P...	1/20/2012 3:46:42 P...	1/20/2012 6:56:06 P...	0	
Michael and Poppa Mike.jpg	Michael and Poppa Mike.jpg	52 KB	1/20/2012 6:56:06 P...	1/20/2012 3:45:20 P...	1/20/2012 6:56:06 P...	0	
Michael and Tom 5685.jpg	Michael and Tom 5685.jpg	97 KB	1/20/2012 6:56:06 P...	1/20/2012 3:46:42 P...	1/20/2012 6:56:06 P...	0	
Michael Christmas Aged Photo 01.jpg	Michael Christmas Aged Photo 01.jpg	49 KB	1/20/2012 6:56:09 P...	1/20/2012 3:38:40 P...	1/20/2012 6:56:09 P...	0	
Michael Christmas Aged Photo 01.tif	Michael Christmas Aged Photo 01.tif	456 KB	1/20/2012 6:56:09 P...	1/20/2012 3:38:40 P...	1/20/2012 6:56:09 P...	0	
Michael Face 5.jpg	Michael Face 5.jpg	19 KB	1/20/2012 6:55:52 P...	1/20/2012 3:33:58 P...	1/20/2012 6:55:52 P...	0	
Michael IV.tif	Michael IV.tif	633 KB	1/20/2012 6:55:57 P...	1/20/2012 3:35:00 P...	1/20/2012 6:55:57 P...	0	
Michael MG.jpg	Michael MG.jpg	71 KB	1/20/2012 6:56:06 P...	1/20/2012 3:45:20 P...	1/20/2012 6:56:06 P...	0	
Michael VI.tif	Michael VI.tif	668 KB	1/20/2012 6:55:57 P...	1/20/2012 3:35:00 P...	1/20/2012 6:55:57 P...	0	

In order to use this, you must first enable, 'Extract EXIF Data (Photos)', of the Renaming Options Menu –



The column, 'Taken (Original)' is visible. It contains a value for the EXIF data extracted from an image file. ↓

Name	New Name	Size	Created	Modified	Accessed	Taken (Original)
Michael 5675.jpg	Michael 5675.jpg	108 KB	1/20/2012 6:56:06 P...	1/20/2012 3:46:42 P...	1/20/2012 6:56:06 P...	8/4/2009 12:47:28 PM
Michael 5705.jpg	Michael 5705.jpg	119 KB	1/20/2012 6:56:06 P...	1/20/2012 3:46:42 P...	1/20/2012 6:56:06 P...	8/4/2009 3:44:26 PM
Michael and Poppa Mike.jpg	Michael and Poppa Mike.jpg	52 KB	1/20/2012 6:56:06 P...	1/20/2012 3:45:20 P...	1/20/2012 6:56:06 P...	8/15/2009 10:07:51 AM
Michael and Tom 5685.jpg	Michael and Tom 5685.jpg	97 KB	1/20/2012 6:56:06 P...	1/20/2012 3:46:42 P...	1/20/2012 6:56:06 P...	8/4/2009 3:38:57 PM
Michael Christmas Aged Photo 01.jpg	Michael Christmas Aged Photo 01.jpg	49 KB	1/20/2012 6:56:09 P...	1/20/2012 3:38:40 P...	1/20/2012 6:56:09 P...	
Michael Christmas Aged Photo 01.tif	Michael Christmas Aged Photo 01.tif	456 KB	1/20/2012 6:56:09 P...	1/20/2012 3:38:40 P...	1/20/2012 6:56:09 P...	1/1/1970 11:59:59 PM
Michael Face 5.jpg	Michael Face 5.jpg	19 KB	1/20/2012 6:55:52 P...	1/20/2012 3:33:58 P...	1/20/2012 6:55:52 P...	
Michael IV.tif	Michael IV.tif	633 KB	1/20/2012 6:55:57 P...	1/20/2012 3:35:00 P...	1/20/2012 6:55:57 P...	4/25/2009 7:04:40 PM
Michael MG.jpg	Michael MG.jpg	71 KB	1/20/2012 6:56:06 P...	1/20/2012 3:45:20 P...	1/20/2012 6:56:06 P...	8/15/2009 10:08:22 AM

e.g., Michael 5705.jpg holds EXIF data. The data is displayed in the 'Taken (Original)' column.

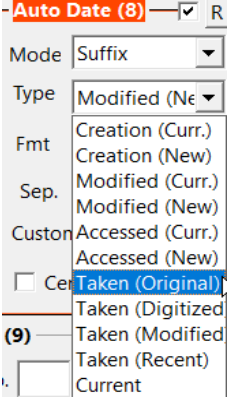
Name	New Name	Size	Created	Modified	Accessed	Taken (Original)
Michael 5705.jpg	Michael 5705.jpg	119 KB	1/20/2012 6:56:06 PM	1/20/2012 3:46:42 PM	1/20/2012 6:56:06 PM	8/4/2009 3:44:26 PM

Notes:

1. The EXIF Metadata value for Taken (Original) maps from EXIF DateTimeOriginal. This is the EXIF data being referred to in the above examples.
2. The term, 'mapping' means in this context to extract a value from the EXIF property source.
3. Metadata is also generally known as 'tags'

Section # 8: Auto Date

In the 'Type' item of Auto-Date (8), you can select among the following additional entries:

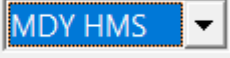
	<p>Taken (Original) - The date when the image was first captured. The original timestamp which should never change. Maps to EXIF DateTimeOriginal.</p> <p>Taken (Digitized)- The date the image was stored as digital data. This would normally be the same as the original timestamp, but if the RAW file was edited, e.g., Canon Digital Photo Professional, then the new date is reflected. Maps to EXIF DateTimeDigitized.</p> <p>Taken (Modified)- Reflects any last modification. Maps to EXIF DateTime.</p> <p>Taken (Recent)- The most "recent" of all three. By recent, this means the last timestamp in the file, not necessarily the most recent in time. This is purely to retain the same behaviour as previous versions of Bulk Rename Utility.</p>
--	---

With the exception of Taken (Recent), the other two items, Taken (Original) and Taken (Digitized) get their values directly from EXIF data items. Typically, these two values are the same but they can vary by a few seconds.

Using the example from before:

Name =  Michael 5705.jpg

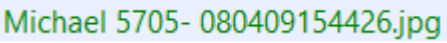
Created	Modified	Accessed	Taken (Original)
1/20/2012 6:56:06 PM	1/20/2012 3:46:42 PM	1/20/2012 6:56:06 PM	8/4/2009 3:44:26 PM

Auto Date Fmt = Fmt  using a Sep of '- <space>'

Mode = Suffix


Where Type =

Taken (Original)

New Name =  where: 08 = Month, 04 = Day, 09 = 2009 (Year)
15 = (3:00 pm) Hour, 44 = Minutes, 26 = Seconds

Taken (Date Digitized) = no change

Taken (Modified)

New Name =  where: the date changed to Aug 16 2009
Time changed to 7:53:49 pm

Taken (Recent) = back to using original timestamp

Section # 8: Auto Date

Notes:

1. BRU will only get the flags from JPEG images (.JPG or .JPEG extension), TIFFs (.TIF, .TIFF), Nikon (.NEF) and Canon (.CR2) files.
2. If there is no value in the Taken (Original) column (the EXIF property is empty) for a particular file, then selecting the Taken field as the 'Type' will append nothing to the New Name field.
3. The EXIF properties sheet will be blank if you select a file type other than a JPEG file. You can bring up the Property sheet by right clicking on the file and selecting from the Context Menu, 'Show EXIF Info (.JPEG Files)'.
4. You need to enable 'Extract EXIF Data (Photos)' from the Renaming Options Menu if there are no values in the Taken (original) column for any image files containing Metadata or if the message 'Set menu option to extract EXIF data' is visible in the column display. This may also require a 'Refresh Files' either through the Actions Menu or by pressing F5 to see the changes take effect.
5. 'Taken (Modified)' is not the Windows Property 'Modified' date. It is the EXIF property, DateTime, so don't get this confused in the example on the previous page.

Other Considerations:

1. Taken (Original) does not support RAW files. You can set a DSLR (Digital Single Lens Reflex) camera to create a RAW image instead of a JPG when taking an image. A RAW image is just that, RAW – unprocessed. JPG is not only processed, but uses compression that may diminish the original quality that might otherwise be obtained.
2. If you have a RAW image, you can use software that can process the image the way you want. It offers higher quality with more options available to modify the image than if it was just in a JPG format. After editing, the file can be saved as a JPG and read by BRU.
3. The main disadvantage with RAW is that it takes more time because it has to be individually processed. The format is also proprietary to the camera manufacturer and requires software that is licensed to that manufacturer's format in order to process the file. If the RAW file is stored away, you may not be able to read the file at a future date unless you have the proper software capable of decoding it.
4. Under what circumstances would there not be a value for Taken (Original) for an image file?
 - a. The file was created in Windows or was created in a device not capable of creating the Metadata.
 - b. The EXIF Metadata was removed by a user or a third party utility.
 - c. The file was transferred or copied over from another File System, FTP, USB Stick or other source and the Metadata was lost in the transfer.
 - d. The filetype is not supported by EXIF DateTimeOriginal. See Notation #1 at the top of this page.

Section # 8: Auto Date**Understanding EXIF and Windows Properties as they apply to BRU**

When it comes to Date Metadata there are two property types:

EXIF (Exchangeable Image File Format)

The Exchangeable image file format, officially EXIF, in accordance with JEIDA/JEITA/CIPA specifications, is a standard that specifies the Metadata formats for images, sound, and ancillary tags used by digital cameras (including smartphones), scanners and other systems handling images. Bulk Rename Utility supports the EXIF Metadata and attributes version 2.2. A full listing can be found in the appendix.

When you take a picture with your digital device, an image file (usually a JPEG or JPG) is written that not only contains the information that makes up the image, but also what is called Metadata. This Metadata can consist of many different pieces of additional information including date, time, camera settings, as well as geolocation data. Supplemental Metadata can also be added by the photo processing software, third party software, and from the user.

These dates are assessed at the point in which the file is created by the device and may consist of –

`DateTimeOriginal`, or `CaptureDate` depending on the device.

`DateTimeDigitized` or `CreateDate` depending on the device.

This Metadata will always be the most accurate and is recorded into the file's header. Exactly which Metadata items are recorded is dependent on the device that created the file. For example, a file may have the EXIF `CreateDate` but not EXIF `DateTimeOriginal`. These timestamps are not easily edited but there are programs that can, e.g., EXIFTool.

Here are the EXIF items that BRU uses in '**Section #8: Auto Date**' –

EXIF:DateTimeOriginal: When the shutter was clicked The date and time when the original image data was generated. For a digital still camera, this is the date and time the picture was taken and recorded.

EXIF:DateTimeDigitized: When the image was converted to digital form. For digital cameras, `DateTimeDigitized` will be the same as `DateTimeOriginal`; for scans of analog pics, `DateTimeDigitized` is the date of the scan, while `DateTimeOriginal` was when the shutter was clicked on the film camera.

EXIF.DateTime: This is the date and time the file was changed (Modified).

In the 'Type' option these are referred to as:

Taken (Original) ~	corresponds directly to EXIF <code>DateTimeOriginal</code>
Taken (Digitized) ~	corresponds directly to EXIF <code>DateTimeDigitized</code> , i.e. Scanning an image file
Taken (Modified) ~	corresponds directly to EXIF <code>DateTime</code>

Section # 8: Auto Date

Windows File Properties

These dates are assessed as Windows Properties. This information may not be as accurate as the EXIF Metadata but is representative of the timestamps that Windows applies **after** the files are **imported from the device** and assigned as the item passes through the Windows OS. These properties are also applied to files that are **created within Windows** directly. If, however, a Windows Property maps from an EXIF item, it would assume that value.

The Windows File Properties for Date Metadata (Date Created, Date Modified, and Date Accessed) do not map from any EXIF Metadata. That information is obtained from the File System.

All of these properties are easily modified.

For example, these are Windows File Properties:

Date Created – Assigned either at the moment the file is entered into the Windows OS or assigned as the file is created within Windows. This timestamp is not necessarily the same value as the EXIF item DateTimeOriginal. Date Created is not carved in stone. It can be changed by Windows or the user. Where **Date Created is the Label** of the Windows Property, **System.DateCreated is the Name** (From right-click Context Menu, select ‘Show List of File Properties).

Date Modified – This timestamp is assigned whenever the file undergoes an edit, modification or if the file was moved. Where **Date Modified is the Label** of the Windows Property, **System.DateModified. is the Name**.

Date Accessed – This timestamp is assigned whenever the file is accessed by a user. Where **Date Accessed is the Label** of the Windows Property, **System.DateAccessed is the Name**.

In ‘**Section #8: Auto Date**’ these are referred to as:

Created (Current) ~	corresponds to Date Created
Modified (Current) ~	corresponds to Date Modified
Accessed (Current) ~	corresponds to Date Accessed

Notes:

1. These are not EXIF Metadata and do not map from EXIF Metadata. They originate from the File System.
2. BRU doesn’t do any mapping, it only reads whatever data is available. When I refer to mapping, I refer to mapping performed outside of BRU to obtain the values that BRU ultimately extracts.
3. Certain Windows Properties are stored as part of the file system, FAT32, exFAT, UDFS and NTFS4. These include **Date Created**, **Date Modified** and **Date Accessed**.
4. EXIF and other Metadata, including ID3, are stored within the physical file, commonly in the File Header.

Section # 8: Auto Date

Understanding Metadata

Before continuing, I feel a further discussion about Metadata is warranted. This concept has caused so much confusion, but it is important to understand if we want to determine where BRU gets its values.

There are three general types of Metadata. That which is stored as part of the File System or somewhere on the Volume, that which is stored as part of the file itself and that which is stored in a separate file as an ancillary.

The Windows File System Metadata

The File System is an area on the storage media or 'Volume', that has an MFT (Master File Table) in a FAT32, FAT or NTFS system. In other systems, like UNIX based, it may be referred to as something else. It is represented on the, e.g., hard drive as \$MFT, and not generally accessible. The MFT acts like an index to the information stored on the volume.

Each volume has its own MFT or equivalent, and therefore its own File System. Logical drives are on the same volume and share one File System. Not all of the File System Metadata is stored in the MFT. Some of it is stored in a completely different structure, e.g., inode, Streams (Microsoft), or Forks (Apple), but within the same volume the file resides.

It holds the following information:

1. It keeps records of all files in a volume
2. The files' location in the directory
3. The physical location of the files on the drive
4. **The file Metadata**

The last one is important for our consideration.

Some of this Metadata is defined as Resident Attributes. The File Metadata falls under the 'Standard Attribute' category under Resident Attributes. But don't worry, I'm not going to discuss Resident vs Non-Resident or get tangled up in Sparse files. This is just to give you an idea of the complexity involved. I am going to make it as simple as I can hopefully without misinforming.

File Metadata Stored Within the Windows File System

The File Metadata consists of:

1. **Date Created**
2. **Date Modified**
3. **Date Accessed**
4. Last Written Date
5. Physical Size (the space the file actually occupies on the hard drive)
6. Logical Size (the size of the information in the file – number of characters, etc.)
7. ACL (Access Control List which are the Permissions)

Section # 8: Auto Date

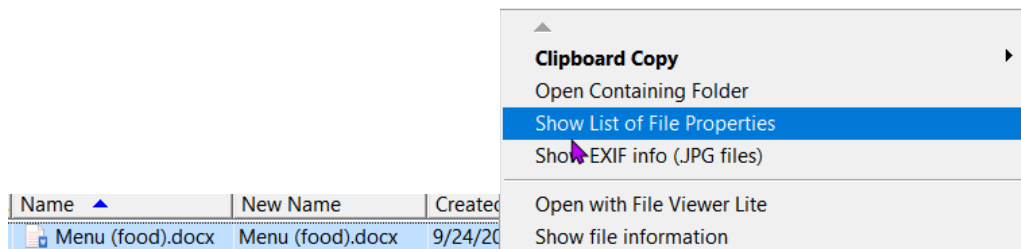
BRU extracts the File Metadata and displays it in the columns for **Created Date**, **Modified Date** and **Date Accessed**. These, together with other Metadata, make up the Windows Property System. Most of the Metadata is stored not as part of the Windows File System, but within the file itself, or in a separate ancillary file created by certain proprietary software. EXIF and XDM and ID3 Metadata are also stored in the file but are not Windows Properties.

File Metadata Stored Within the File Itself

I have already discussed EXIF Timestamps Metadata and how it is different from the Windows Properties of Date Created and Date Modified. EXIF Metadata as well as ID3 are also stored as part of the file, usually in the Header. EXIF Metadata only applies to certain filetypes; image, video and sound. ID3 only applies to MP3 files. EXIF properties are recorded by the device creating the file while Windows Properties are assigned by the OS as the file passes through the system. Other Metadata, like ID3, may be added through other sources but again this would be applied within the OS. EXIF and XDM (Extensible Device Metadata) are the only Metadata that are not OS dependent (to my knowledge). XDM is relatively new and not currently supported by BRU.

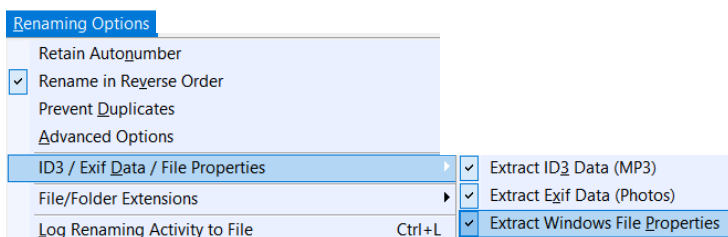
Much of the other Metadata that make up the Windows Properties including Metadata applied by various sources, e.g., third party programs, user, etc. are stored within the file itself. Certain other Metadata may also be stored in a separate ancillary file generated by proprietary software.

For a list of a file's available Windows Properties, use the 'Show List of File Properties' selection from the Context Menu of a file by right clicking on the filename. In newer versions of BRU, these Windows Properties can be used in the renaming process.



Notes:

1. In order for BRU to extract the information, you first have to enable, 'Extract Windows File Properties' from the 'ID3 / EXIF Data / File Properties' submenu of the Renaming Options Menu.



BRU can use any of these Windows Properties in renaming. For more information, please refer to 'Using Windows Properties' and 'Using EXIF Tags' under the section heading of Section #7: Add.

Section # 8: Auto Date

EXIF and XDM Metadata doesn't change easily unless you use a third party utility. Windows Properties can easily be changed by Windows, by the user or by other software. Metadata can also be added through same means.

Metadata when Moving or Copying Files and Folders

Because the Standard File Attributes, including the Date Created, Date Modified and Date Accessed, are stored as part of the File System, whether or not this Metadata is preserved during a Copy or Move operation is first dependent on the target location.

1. **If the target location is a different volume**, and this could be a server or other storage device, e.g., USB Flash Drive, external Hard Drive, etc. – in short – any volume that has its own File System, some of the File System Metadata will not be preserved.
 - a. If a file is copied, the new File System will assign its own Date Created and other attributes using the current timestamp. To the File System, this is a newly created file, however Date Modified is preserved because this has to do with the contents. If the contents of the file are not changed, then the Modified Date won't change. Date Created, on the other hand, relates to the file created on the new File System and that does change.
 - b. If a file is moved, it is essentially a Copy and Delete operation, thus the new file will have the current timestamp for both the Date Created and Date Modified. The previous Metadata for these items are not preserved.
 - c. Metadata stored within the file itself should transfer over because it is part of the file itself. Ancillary files typically do not transfer over.
 - d. Date Accessed - Always set with the current timestamp. Previous Metadata for this item is not preserved.
2. **If target location is on the same volume** (this also applies to logical partitions on the same volume), then the File System is the same. The Standard File Attributes Metadata that are preserved is dependent on whether it is a Copy or a Move operation.
 - a. If a file is copied, it retains the Date Modified but the Date Created is not preserved and changes to the current timestamp unless the file already exists at the target location. Under that circumstance, the Date Created Metadata would be preserved.
 - b. If a file is moved, both the Date Created and Date Modified are retained.
 - c. Date Accessed - Always set with the current timestamp. Previous Metadata item is not preserved.
 - d. Copying Folders that contain files will change the Folder timestamp to the current timestamp, but the files themselves should retain their original Metadata. If you want to preserve the Timestamps of the folders, then you must first zip them before performing the copy operation. This behavior is File System dependent and whether or not a directory sub-structure is created in the copy/move operation. This will be discussed next.

Section # 8: Auto Date

Metadata when Copying and Moving Folders in a Directory Sub-Structure

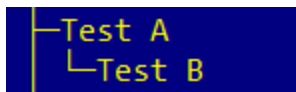
Under NTFS:

1. A directory is created on a volume with the Date Created and Date Modified set with the current timestamp.
2. Date Modified will only change if the **contents** of the file changes.

Scenario –

Two directories, Test A and Test B on the same volume.

- a. The Test B directory is **moved** as a sub-directory under Test A.



Test A becomes the root or Main directory of the new sub-directory structure.

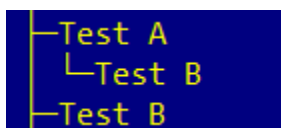
1. Test A:

Date Created is retained, but the **Date Modified changes to current timestamp**. The File System views the added sub-directory of Test B as a modification to Test A's contents.

2. Test B Subdirectory:

Both the Date Created and Date Modified are retained. The File System already has the information on the Test B directory in the MFT. No new entry is required, so the Metadata does not change.

- b. The Test B directory is **copied** as a sub-directory under Test A. The original Test B directory remains.



Test A becomes the root or Main directory of the new sub-directory structure.

The Test B original directory does not change.

1. Test A:

Date Created is retained, but the **Date Modified changes to current timestamp**. The File System views the added sub-directory of Test B as a modification to Test A's contents.

2. Test B Subdirectory:

Both the Date Created and Date Modified changes to current timestamp. Although Test B was copied, the File System has to create a new entry in the MFT to accommodate the new subdirectory with new Metadata.

Under FAT or FAT32:

1. The Date Modified does not change even if the contents of the folder change. Thus, if Test B was copied, both the Date Created and Date Modified are retained.

Section # 8: Auto Date

v3.4 New Additions

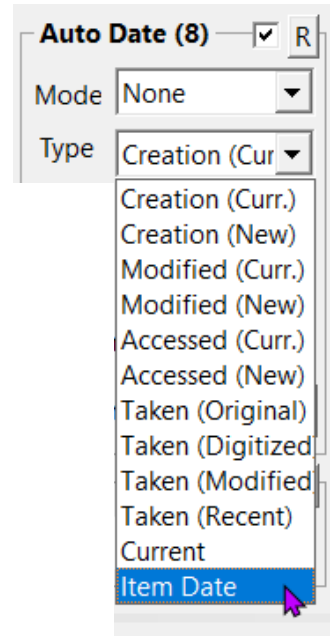
Item Date –

‘Item Date comes from the Windows Property, System.ItemDate and it is the primary date of interest for an item.’

That statement is taken directly from the Microsoft docs.

The statement continues –

‘In the case of photos, for example, this property maps to another Windows Property, System.Photo.DateTaken’



Now my take –

For other video or media file types, supplementary mapping may take place, but more importantly, for certain image types that are supported by System.Photo.DateTaken, it corresponds to an EXIF property depending on the file type. Why is this important? Because EXIF values are more accurate than Windows Properties. The available EXIF Metadata is dependent on the device that created the image. Windows will map from the EXIF value from any of these Metadata for System.Photo.DateTaken:


EXIF:DateTimeOriginal	This is the date and time an image was taken at the moment of the shutter actuation.
XMP:DateTimeOriginal	This is the XMP equivalent of the EXIF value. Not currently supported in BRU.
EXIF:CreateDate	This is the timestamp that the file was written to the memory card in a device at the moment of creation. Typically there should only be a few seconds difference between this item and DateTimeOriginal with DateTimeOriginal being the more accurate of the two <i>if</i> both are available; typically, not. Most devices will only create one or the other. This is dependent on the type of device creating the file and the filetype. It should also be noted that DateTimeOriginal is listed in the EXIF v2.2 and CreateDate is not. I do not know why. CreateDate may be created for an e.g., video filetype instead of DateTimeOriginal while DateTimeOriginal is reserved for supported image filetypes of JPEG, TIFFs, NEF and CR2 files.
EXIF:Photo.DateTimeDigitized	This is the timestamp the image was stored as digital information. Any difference in time between this item, CreateDate and DateTimeOriginal will also be minimal but in most cases, the same. Some devices may create the CreateDate item while others would create only the DateTimeDigitized item. CreateDate, I believe, is the equivalent of DateTimeDigitized.

Section # 8: Auto Date

v3.4 New Additions

BRU's Taken (Original) corresponds with the EXIF:DateTimeOriginal value and will only display if there is a value. This value would be unavailable for images originating in a device not capable of generating the Metadata.

This is an image file that was created in a camera and later imported into the computer.

Name ▲	New Name	Created	Accessed	Modified	Taken (Original)	Item Date
 FSCN0006.JPG	FSCN0006.JPG	9/10/2009 2:26:14 PM	8/7/2017 1:00:00 AM	9/10/2009 2:26:14 PM	9/7/2009 12:12:44 PM	9/7/2009 12:12 PM

Look at a closeup of these three dates:

Created	Taken (Original)	Item Date
9/10/2009 2:26:14 PM	9/7/2009 12:12:44 PM	9/7/2009 12:12 PM

The Item Date for this filetype mirrors the Taken (Original) value, both of which in turn, map directly from the EXIF Metadata DateTimeOriginal, so the timestamp **is** accurate. This date reflects the creation of the image in the camera. On the other hand, the Windows Property, Date Created, is when the image was created in Windows. In this example, it is the point at which the image file was transferred from the camera into the computer. You can see the discrepancy of three days in the dates between the other two. Typically, though, with most filetypes supported by the EXIF Metadata DateTimeOriginal, there won't be that much of a difference, if any. This is dependent, of course, on when the file is imported into Windows from the device.

If Item Date and Taken (Original) are basically the same value, why do we need it? They are not.

1. Taken (Original) only supports limited filetypes – as noted previously, JPEG images (.JPG or .JPEG), TIFFs (.TIF, .TIFF), Nikon (.NEF) and Canon (.CR2) files and **only** if there is an EXIF DateTakenOriginal Metadata value available for those files that was created by the originating device.
2. Item Date applies to most media, pictures, and video filetypes including HEIC files, RAW camera files, etc., This date can be mapped from other Metadata as well as EXIF DateTakenOriginal, meaning, that even if BRU has no value for Taken (Original), Item Date most likely will. The Item Date Metadata is also available for most other file types and not just media and images.
3. Taken (Original) is restricted to the epoch date that limits the earliest recognized date to 01/01/1970 (see 'Renaming with Dates Prior to January 1, 1971' in the Appendix of Volume II for further information on the epoch dates). Item Date does not have this restriction. Update – I *believe* this restriction has been lifted in newest version.

HEIC stands for High Efficiency Image Format, a graphic format adopted by Apple for its newer devices. It allows photos to be created in smaller file sizes while retaining a higher image quality as compared with JPEG/JPG.

Other Considerations:

1. Taken (Original) is accurate to the seconds.
2. Item Date does not take in consideration seconds. It is accurate to the minute (Microsoft dropped the Seconds).

3. Certain exceptions of .exe files exhibit Metadata but Item Date returns no value. I am at present working on the problem as to why. This seems to be an abnormality and not typical of most .exe filetypes.

Section # 8: Auto Date

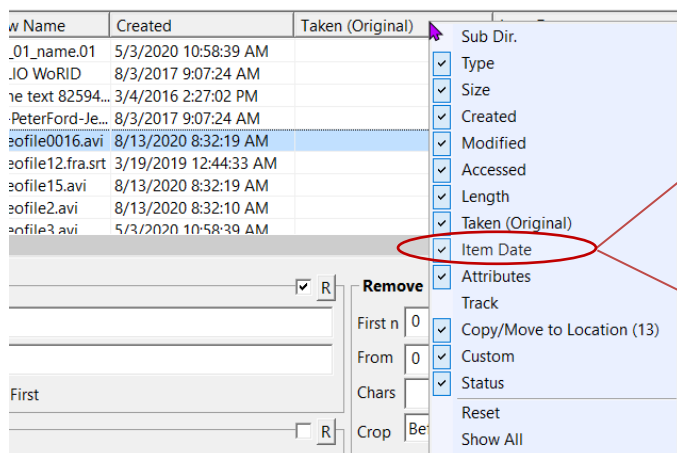
v3.4 New Additions

Notes:

1. EXIF, Exchangeable Image File format is fully supported by BRU.
2. XMP, Extensible Metadata Platform, is relatively new and not currently supported by BRU.
3. The Windows Property, Date Created, is NOT the EXIF property CreateDate as already explained above, so do not get confused between the two. Date Created is a timestamp assigned by the File System as the file is first entered into the Windows OS. In the case of the example cited previously, a photo was imported from a camera, thus, the two values may not be identical.
4. There are utilities that can alter EXIF data affecting the reliability of the original timestamps. One such popular utility is EXIFTool.
5. Item Date does not only apply to video and media files but most any file type, but **only** if the Metadata exists. If not, then the Item Date will be blank for that file.

Attributes	Name ▲	New Name	Created	Taken (Original)	Item Date
A	ooRexx-4.2.0.windows.x86_6...	ooRexx-4.2.0.windows.x...	11/27/2020 6:21:46 PM		
A	Package or Product.html	Package or Product.html	11/27/2020 6:24:43 PM		11/27/2020 6:24 PM
A	passwordmanager.exe	passwordmanager.exe	11/17/2020 6:48:18 PM		11/17/2020 6:48 PM
A	PBIDesktopSetup_x64.exe	PBIDesktopSetup_x64.e...	11/12/2020 4:32:00 PM		11/12/2020 4:32 PM

6. The Item Date can be shown as a column in the Content Pane. It does require enabling the option, 'Extract Windows File Properties' from the Renaming Options Menu. To view the column, just right click in any of the Column Headings and select Item Date from the list:



Now, the Item Date column is visible:

Name ▲	New Name	Created	Taken (Original)	Item Date
videofile12.fra.srt	videofile12.fra.srt	3/19/2019 12:44:33 AM		3/19/2019 12:44 AM

Section # 8: Auto Date


v3.4 New Additions

My Conclusions arrived at for Item Date.


Item Date is the Windows Metadata property, System.ItemDate. In BRU's List of File Properties, the Label is 'Date'. System.ItemDate's value can not only map from the EXIF Metadata DateTimeOriginal for supported image filetypes, but also maps from other Windows Metadata, primarily, Date Created and Date Modified. This provides an Item Date value for most any filetype. I wanted to find out how.

I tested a number of different filetypes. Here is just a sample.


.txt file:

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Annet Mahendru told this joke - funny.txt	Annet Mahendr...	8/4/2017 8:58:29 AM	5/2/2015 4:30:19 PM	8/4/2017 8:58:29 AM	5/2/2015 4:30 PM

.pdf file

Name ▲	New Name	Created	Modified	Accessed	Item Date
 office problems solved.pdf	office problems...	8/4/2017 8:59:57 AM	1/18/2008 11:30:15 AM	8/4/2017 8:59:57 AM	1/18/2008 11:30 A...

.exe file

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Faronics_IGS.exe	Faronics_IGS.exe	10/11/2018 11:49:57 P...	10/11/2018 11:50:25 PM	10/11/2018 11:49:57 PM	10/11/2018 11:49 PM

You can see that Item Date assumes a value for all of these different filetypes.

My findings:

- (1) In the event of a filetype that **is** supported by EXIF DateTimeOriginal where the Metadata exists and holds a value, both Taken (original) and Item Date will assume this value.
- (2) In the event of a filetype that is:
 - a. Unsupported by EXIF DateTimeOriginal ...
 - b. Neither an image or media type file ...
 - c. Supported by EXIF DateTimeOriginal (JPEG, TIFFs, NEF and CR2 files) but has no value or the EXIF Metadata for DateTimeOriginal is unavailable ...

... then the following takes place:

1. Item Date will look at the Windows Properties of Date Created and Date Modified, and assume the value of the earliest date with the exception of file types that are recognized by Windows as documents.
2. Item Date will assume the Windows Metadata, Date Last Saved, if available, for 'recognized' document file types.

Section # 8: Auto Date

v3.4 New Additions

So where does System.ItemDate get its value for these type of files and how does it determine which value to use?

Back to the statement, ‘... it is the primary date of interest for an item.’ Vague. Can’t you do better Microsoft? Okay, so it is up to me and a lot of theories and testing.

The docs also state that System.ItemDate can map to System.Photo.DateTaken from among others. Logically, if EXIF DateTimeOriginal holds a value then it is reasonable to presume that System.Photo.DateTaken will assume that value and in turn, System.ItemDate will hold that same value.

Why do I say this? Because Microsoft Docs support this.

Microsoft documentation references are in dark sky blue—

System.ItemDate –

<https://docs.microsoft.com/en-us/windows/win32/properties/props-system-itemdate>

‘The primary date of interest for an item. In the case of photos, for example, this property maps to System.Photo.DateTaken.’

System.Photo.DateTaken –

<https://docs.microsoft.com/en-us/windows/win32/properties/props-system-photo-datetaken>

‘The date when the photo was taken, as read from the camera in the file’s Exchangeable Image File (EXIF) tag.’

The EXIF tag in question can only be one of **DateTimeOriginal** or **DateTimeDigitized**.

This concludes that the Windows Metadata property, System.ItemDate, **IF** it maps from the Windows Metadata property, System.Photo.DateTaken, assumes the value of the EXIF value of DateTakenOriginal **IF** this tag holds a value and it would only hold a value for the supported filetypes. Those filetypes are JPEG, TIFFs NEF and CR2 files.

Other research garnered the following explanation, also taken from some Microsoft Docs stating:

‘Windows determines the most relevant date based on the type of the item. For example, if the item is a photo, System.ItemDate (a Windows property) maps to System.Photo.DateTaken (a Windows property of which the value is assumed from EXIF DateTimeOriginal). Or, if the item is an MP3, then System.ItemDate (may also map(s) to, System.Media.DateReleased (another Windows Metadata property).’

Section # 8: Auto Datev3.4 New Additions

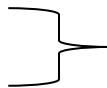
My Conclusions arrived at for Item Date. cont.

Therefore,

Although Taken (Original) maps directly from EXIF DateTimeOriginal, and it is **this** EXIF item that limits the filetypes supported, Item Date does not have this limitation and has many options from which to map, some of those consist of Windows Properties that may – not always, in turn map from EXIF Metadata. The most common of these – and the only ones I have found given the filetypes I have tested, are:

EXIF tags:

EXIF DateTimeOriginal
EXIF DateDigitized

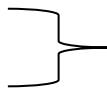


Item Date assumes these Creation Date Values if EXIF Data is available for the supported image filetypes.

Taken (Original) will also hold this value.

Windows Properties:

→ **Date Created**
→ **Date Modified**



Item Date will use the earlier of the two date values for both image filetypes not supported by EXIF DateTimeOriginal and for non-image filetypes that are not Document filetypes.

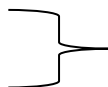
System.Photo.DateTaken



Item Date assumes the value from System.Item.Date. If the filetype is a supported image filetype, Windows will map out to System.Photo.DateTaken which in turn maps from EXIF DateTimeOriginal.

Document Property:

Date Last Saved



Item Date may assume this value if the filetype is a 'recognized' Document filetype and has Metadata available.

As for Microsoft's claim of mapping to other Metadata, e.g., System.Media.DateReleased for an MP3 file, my findings dispute this by and large. I tested many MP3 and video files and found that unless it is a file type supported by Taken (Original), and MP3 and video files are not supported, System.ItemDate assumes the value from the earlier date of the two Windows Properties, period.

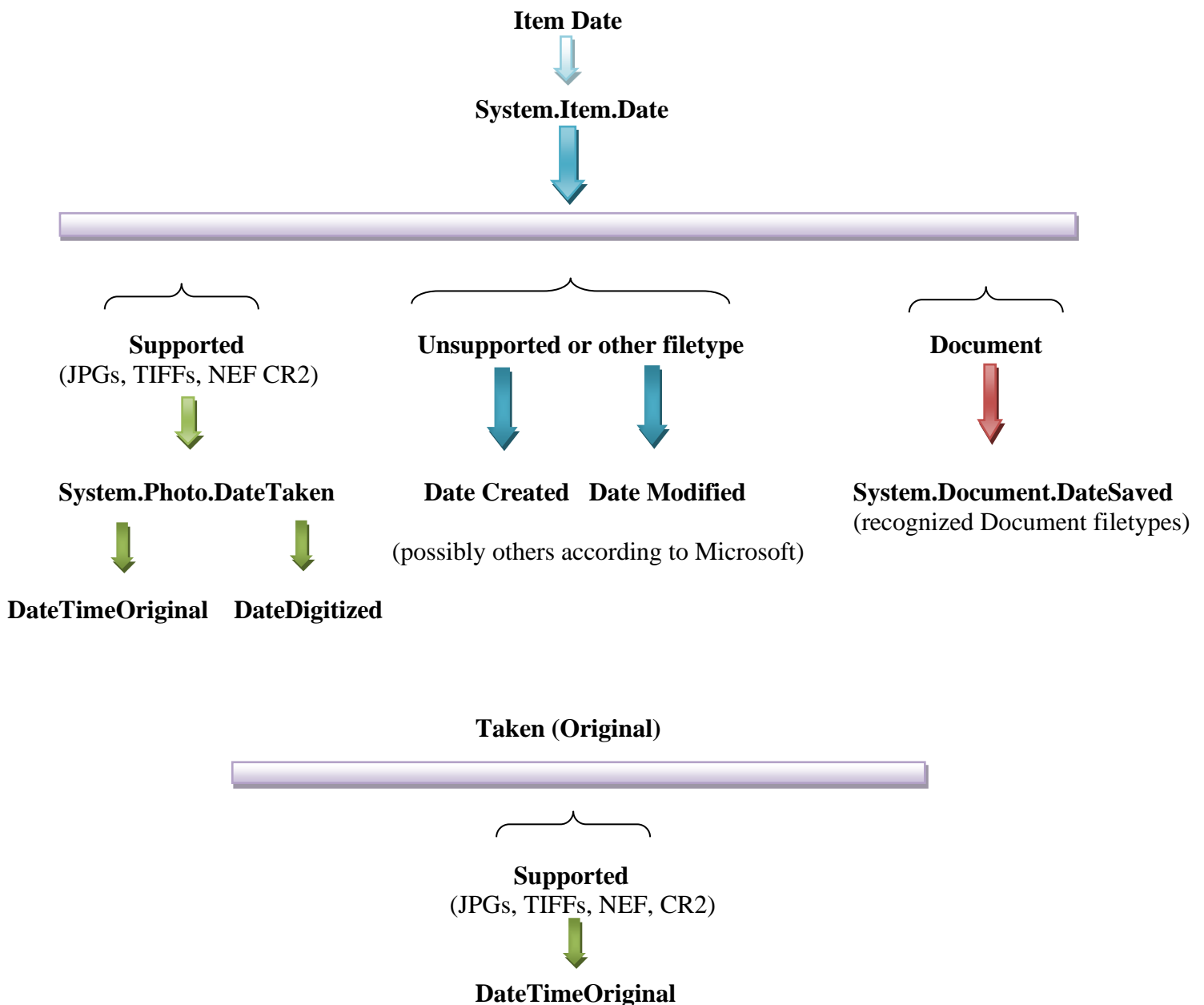
Document files, though, are another matter. I have found that Item Date can assume the Windows Metadata property, System.Document.DateSaved (Date Last Saved) value if available, of recognized Document filetypes. What constitutes a recognized Document filetype will be explained.

I find no other values in consideration, but let's say for the sake of argument, that it is true that other values are assumed. Who am I to question Microsoft?

Section # 8: Auto Datev3.4 New Additions

My Conclusions arrived at for Item Date. cont.

Still confused? Perhaps this visual will clarify (as I see it anyway):



All of the above named properties are Windows Metadata with the exceptions of **DateTimeOriginal** and **DateDigitized** which are EXIF properties. Windows properties that assume values from EXIF are more accurate than those that are taken from other Windows Properties, or assigned by the Windows OS directly, e.g., **Date Created**.


Section # 8: Auto Datev3.4 New Additions

The Tests Used


Conclusion #1 Tests

For other file types, takes its value from the earlier date of Date Created or Date Modified.

This is an Access database I created for this test and assigned the various Timestamps you see here.


Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	8/3/2017 1:56:46 PM	12/8/2020 11:10:11 PM	12/8/2020 11:10:11 PM	8/3/2017 1:56 PM

a. I Change the original Date Created from 8/3/17 to current, 12/21/20. Item Date is currently 8/3/17 before change.

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	12/21/2020 10:24:22 A...	12/8/2020 11:10:11 PM	12/8/2020 11:10:11 PM	12/8/2020 11:10 PM


result: Item Date changes to reflect the original Date Modified, 12/8/20. Date Created and Date Accessed are ignored.

b. I Changed the original Date Modified from 12/8/20 to current, 12/21/20, leaving the Date Created also as current:

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	12/21/2020 10:24:22 A...	12/21/2020 10:26:20 AM	12/8/2020 11:10:11 PM	12/21/2020 10:24 AM

result: Item Date reflects the change and changes to 12/21/20. Date Accessed of 12/8/20 is ignored.


c. I Took the original file (Date Created is 8/3/17) and changed the Date Modified from 12/8/20 to current, 12/21/20:

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	8/3/2017 1:56:46 PM	12/21/2020 10:28:00 AM	12/8/2020 11:10:11 PM	8/3/2017 1:56 PM

result: Item Date doesn't change – reflects original date 8/3/2017, the Date Created value.

Section # 8: Auto Datev3.4 New Additions

d. I Changed the Date Created to an earlier date from 8/3/17 to 2/10/06. Item Date is currently 8/3/17 before change.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Tim Master.accdb	Tim Master.accdb	2/10/2006 7:21:00 PM	12/8/2020 6:13:54 PM	12/8/2020 6:13:54 PM	2/10/2006 7:21 PM

result: Item Date changes to 2/10/06, the Date Created. Date Accessed and Date Modified are ignored.


Further research concluded that Date Accessed was not a factor in determining Item Date's value.

This next set of tests will be using two different dates for Date Modified and Date Created.


e. I Changed Date Modified (Written) to a date (11/24/19) later than 8/3/2017 (refer to 'set to:' in each photo below).

```
<Search: *.*>
STAMP file: Tim Master.accdb          9,306,112 .a.. 12-08-20 11:10:11 pm
  set to: 11-24-19 10:34:39 am      F5 Previous  F6 Now    F7 Timestamp (written ) F9 Mode
Enter date and time                Tab Original F3 Last  ↑ History ← OK  Esc Cancel
```

result: As expected, Item date did not change from original value, 8/3/2017, a reflection of Date Created.

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	8/3/2017 1:56:46 PM	11/24/2019 10:34:39 AM	12/8/2020 11:10:11 PM	8/3/2017 1:56 PM

f. I Changed the original Date Created from 8/3/17 to current, 12/21/20.

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	12/21/2020 10:37:44 A...	11/24/2019 10:34:39 AM	12/8/2020 11:10:11 PM	11/24/2019 10:34 AM


result: Item Date changes to take the value of Date Modified, 11/24/19, the earlier date.

Section # 8: Auto Datev3.4 New Additions

g. I Changed the original timestamps around so that Date Created is earlier (11/24/19) than Date Modified (12/8/20).

```

L<Search: *.*.*>
STAMP file: Tim Master.accdb          9,306,112 .a.. 8-03-17 12:56:46 pm
  set to: 11-24-19 10:34:39 am      F5 Previous F6 Now F7 Timestamp (created) F9 Mode
Enter date and time      Tab Original F3 Last ↑ History ← OK Esc Cancel
  
```

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	11/24/2019 10:34:39 A...	12/8/2020 11:10:11 PM	12/8/2020 11:10:11 PM	11/24/2019 10:34 AM


result: Item Date changed to the earlier date, 11/24/19, without requiring changing Date Modified.

Hmmm... this disputes a claim in my earlier internet research (not shown) that **only** the Date Modified value is assumed for other filetypes. Now I know this not to be the case.

h. I Changed Date Accessed from 12/8/20 to an earlier date, 03/02/14:

```

L<Search: *.*.*>
STAMP file: Tim Master.accdb          9,306,112 .a.. 3-02-18 10:44:59 am
  set to: 03-02-14 10:44:59 am      F5 Previous F6 Now F7 Timestamp (accessed) F9 Mode
Enter date and time      Tab Original F3 Last ↑ History ← OK Esc Cancel
  
```

Name	New Name	Created	Modified	Accessed	Item Date ▲
 Tim Master.accdb	Tim Master.accdb	8/3/2017 1:56:46 PM	12/8/2020 11:10:11 PM	3/2/2014 10:44:59 AM	8/3/2017 1:56 PM

result: Item Date doesn't change, despite the earliest date value; proof that Date Accessed has no influence in the determination.

Final Conclusion from these tests:

1. For a filetype that is **not supported** by Taken (Original) and correspondingly EXIF DateTakenOriginal, Item Date looks at Date Created and Date Modified and takes the earliest date value between the two.
2. When tested against other non-media types I came to the same findings.
3. For a **supported** image filetype, if EXIF Metadata doesn't exist (no EXIF Metadata in the file), then Item Date will take the earliest date between the Windows Properties, Date Created and Date Modified.

Section # 8: Auto Date

v3.4 New Additions

Conclusion #2


For a filetype that **is** supported by Taken (Original) and correspondingly EXIF DateTimeOriginal, Item Date will assume this value over the Windows Properties, Date Created or Date Modified:

This next set of tests will test other factors.

- For supported image filetypes that do have EXIF Metadata, Both Item Date and Taken (Original) will assume this value directly from EXIF DateTimeOriginal.

This example shows that Item Date has the same value as Date Created and Taken (Original).

The fact that Taken (Original) holds a value is proof enough that EXIF data does exist for this file.

Name	New Name	Created	Modified	Accessed	Item Date ▲	Taken (Original)
 DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	3/2/2014 10:44:59 AM	8/7/2017 1:00:00 AM	9/1/2009 1:38 PM	9/1/2009 1:38:35 PM

But if you want further proof, a look at the header reveals the EXIF DateTimeOriginal value:

```

ZTreeWin v2.4.197 - M: DSCN0001.JPG
M:\test BRU 3\DSCN0001.JPG
00000000 .....Exif..II*.....(.....1.....2.....
000000C8 ...COOLPIX L20 V1.0-2009:09:01 13:38:35-.....".....!.....'.....0220.....
00000190 .....|.....0100.....@.....
00000258 .....<.....-2009:09:01 13:38:35-2009:09:01 13:38:35-...
00000320 .....
  
```

You can also look at the Properties for the file:

The Details tab displays the EXIF value, DateTimeOriginal as Date taken.



Section # 8: Auto Datev3.4 New Additions


This next set of tests will verify whether Item Date will take its value from the earlier date of the Windows Property or use the EXIF DateTimeOriginal date regardless.

b. I will change all of the dates at once to earlier than the current Item Date of 2009:

```

<<Search: *.*.*>
STAMP file: DSCN0001.JPG                2,592,944 .a.. 3-02-14 10:44:59 am
  set to: 03-02-01 10:44:59 am      F5 Previous F6 Now F7 Timestamp (ALL  ) F9 Mode
Enter date and time      Tab Original F3 Last ↑ History ← OK Esc Cancel
  
```

Results:

Name	New Name	Created	Modified	Accessed	Item Date ▲	Taken (Original)
 DSCN0001.JPG	DSCN0001.JPG	3/2/2001 10:44:59 AM	3/2/2001 10:44:59 AM	3/2/2001 10:44:59 AM	9/1/2009 1:38 PM	9/1/2009 1:38:35 PM

There's the proof.

Item Date remains at 2009 even if the Windows Property Timestamps are earlier. This successfully validates that when EXIF Metadata exists, Item Date will take that value, regardless of the date values held by Date Created or Date Modified, and when it doesn't, Item Date assumes the value of the earlier between the two Windows Properties aforementioned with the exception of (recognized) Document filetypes. These will be tested next.

Section # 8: Auto Date

v3.4 New Additions

How I Tested Document Metadata

Document Metadata

Document specific Metadata are properties added by the program that created the document. These are stored as part of the document and are not part of the File System. This Metadata is updated as the document is modified, saved, etc.

The two we are concerned with consist of –

Content Created (System.Document.DateCreated)

Content Created is the point at which the document is created by the user but no content has been entered. At this time, Content Created will share the same value as the Windows Property, Date Created. It is only after any content has been entered into the document and saved that Content Created is updated. After this, the timestamp does not change. Date Created is not updated in this manner and therefore no longer holds the same value as Content Created.

Item Date **does not** assume this value (at least in my testing and research).

Date Last Saved (System.Document.DateSaved)

Self explanatory. It is the date the document was last saved from a program capable of loading, modifying and saving the document. It does not necessarily have to be the program that originally created the document. This is updated whenever the document is saved after a modification. It is not the same item as the Windows Property Date Modified or Date Accessed.

Item Date **may** use this value if available (and only if the file is a recognized document filetype)

Notes:

1. These two Metadata items are not only reserved for Document filetypes, but appear in most filetypes as long as Metadata is present (I look at this as an anomaly to its purpose). You can see these Properties by bringing up the ‘Show List of File Properties’ from BRU.
 - a. The Content Created item behaves in the same manner as detailed above. It is assigned at the creation of the file and ‘initially’ shares the same timestamp as Date Created. The date is not supposed to change once established (content has been entered and saved the first time).
 - b. The Date Last Saved item also behaves in the same manner as detailed above. This date is subject to change. The difference between these properties when applied to a non-document filetype and a ‘recognized’ document filetype is that the Metadata item, Date Last Saved, is not a consideration in determining Item Date for a non-document filetype.

Section # 8: Auto Datev3.4 New Additions

Notes:

1. If you want to view the 'Standard Attributes' timestamps of a file, right click on the file and select Properties. Under the General tab, it will list the current Windows timestamp properties, Date Created, Date Modified and Date Accessed, as extracted from the File System.
2. To find the Content Created or Date Last Saved Metadata for Microsoft related documents, go to the Details tab and look under, Origin. To find the same Metadata for other types of documents, use BRU's 'Show List of File Properties' option by right clicking on the file and selecting from the Context Menu.
3. Under the listing of Windows Properties from the 'Show List of File Properties, you can also perform a search for the Label, Date Last Saved, or under the Name, System.Document.DateSaved.
4. How I concluded that Item Date uses the Date Last Saved value.
 - a. I altered the Windows Timestamp Property values of a Microsoft Word document with numerous values – no change in Item Date. Upon reviewing the properties I saw that the only date that corresponded with Item Date's current value was Date Last Saved.
 - b. I Loaded the document into Microsoft Word, made a minor change and Item Date changed to today's date. Reviewing the properties also verified that Date Last Saved reflects this change. The following pages will fully document my findings.

I started with a Microsoft Word document file and from this, I created the rest of the filetypes to be tested.

.docx –	Microsoft Word
.rtf –	Microsoft Word Rich Text Format
.pdf –	Adobe Portable Document Format
.odt –	Open Office Open Document Text file
.tmd –	SoftMaker TextMaker Document
.txt –	Plain Text File

The Microsoft Word document file had a Date Last Saved value initially of 10/9/20.







Date last saved	10/9/2020 8:31 AM	System.Document.DateSaved
-----------------	-------------------	---------------------------

The remainder of the files all shared the same Date Last Saved value with the Date Created value of 1/4/21 because they were all created within the same time period.

Date last saved	1/4/2021 9:58 PM	System.Document.DateSaved
-----------------	------------------	---------------------------







Section # 8: Auto Datev3.4 New Additions

Original timestamps.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Menu (food).docx	Menu (food).docx	10/29/2017 6:45:16 PM	10/9/2020 8:32:04 AM	10/9/2020 8:32:04 AM	10/9/2020 8:31 AM
 Menu (food).odt	Menu (food).odt	1/4/2021 9:58:50 PM	1/4/2021 9:58:50 PM	1/4/2021 9:58:50 PM	1/4/2021 9:58 PM
 Menu (food).pdf	Menu (food).pdf	1/4/2021 10:01:16 PM	1/4/2021 10:01:26 PM	1/4/2021 10:01:26 PM	1/4/2021 10:01 PM
 Menu (food).rtf	Menu (food).rtf	1/4/2021 9:59:06 PM	1/4/2021 9:59:18 PM	1/4/2021 9:59:18 PM	1/4/2021 9:59 PM
 Menu (food).tmd	Menu (food).tmd	1/4/2021 9:58:29 PM	1/4/2021 9:58:30 PM	1/4/2021 9:58:30 PM	1/4/2021 9:58 PM
 Menu (food).txt	Menu (food).txt	1/4/2021 9:59:33 PM	1/4/2021 9:59:40 PM	1/4/2021 9:59:40 PM	1/4/2021 9:59 PM







All files, with the exception of the Word document, are using the current timestamp. This is in all likelihood based on the earliest value between Date Created and Date Modified, all of which, at this stage in the testing, are identical. The Word document's Item Date is either based on Date Last Saved or the value for the Date Modified. Currently unproven.

First I changed all of the Date Created values to an earlier date of 3/2/14.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Menu (food).docx	Menu (food).docx	3/2/2014 7:21:00 PM	10/9/2020 8:32:04 AM	10/9/2020 8:32:04 AM	10/9/2020 8:31 AM
 Menu (food).odt	Menu (food).odt	3/2/2014 7:21:00 PM	1/4/2021 9:58:50 PM	1/4/2021 9:58:50 PM	1/4/2021 9:58 PM
 Menu (food).pdf	Menu (food).pdf	3/2/2014 7:21:00 PM	1/4/2021 10:01:26 PM	1/4/2021 10:01:26 PM	1/4/2021 10:01 PM
 Menu (food).rtf	Menu (food).rtf	3/2/2014 7:21:00 PM	1/4/2021 9:59:18 PM	1/4/2021 9:59:18 PM	1/4/2021 9:59 PM
 Menu (food).tmd	Menu (food).tmd	3/2/2014 7:21:00 PM	1/4/2021 9:58:30 PM	1/4/2021 9:58:30 PM	3/2/2014 7:21 PM
 Menu (food).txt	Menu (food).txt	3/2/2014 7:21:00 PM	1/4/2021 9:59:40 PM	1/4/2021 9:59:40 PM	1/4/2021 9:59 PM

The TextMaker file is the only file that changed to the earlier date. All others were unaffected. The Word document continues to remain at 10/9/20, and all indications are that Item Date is using the Date Last Saved value for this filetype because it did not change to the earlier date of Date Created.







Using the original files, I Changed the Date Modified to an earlier date of 2/10/06.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Menu (food).docx	Menu (food).docx	10/29/2017 6:45:16 PM	2/10/2006 7:21:00 PM	10/9/2020 8:32:04 AM	10/9/2020 8:31 AM
 Menu (food).odt	Menu (food).odt	1/4/2021 9:58:50 PM	2/10/2006 7:21:00 PM	1/4/2021 9:58:50 PM	2/10/2006 7:21 PM
 Menu (food).pdf	Menu (food).pdf	1/4/2021 10:01:16 PM	2/10/2006 7:21:00 PM	1/4/2021 10:01:26 PM	2/10/2006 7:21 PM
 Menu (food).rtf	Menu (food).rtf	1/4/2021 9:59:06 PM	2/10/2006 7:21:00 PM	1/4/2021 9:59:18 PM	2/10/2006 7:21 PM
 Menu (food).tmd	Menu (food).tmd	1/4/2021 9:58:29 PM	2/10/2006 7:21:00 PM	1/4/2021 9:58:30 PM	2/10/2006 7:21 PM
 Menu (food).txt	Menu (food).txt	1/4/2021 9:59:33 PM	2/10/2006 7:21:00 PM	1/4/2021 9:59:40 PM	2/10/2006 7:21 PM

Section # 8: Auto Datev3.4 New Additions







The Word Document remained at 10/9/20, proof that the Word Document is using the Date Last Saved value. All of the other files changed to the earlier date, an indication that these filetypes may be using the earlier date between Date Created and Date Modified and not the Date Last Saved value.

Using the original files, I Changed both the Date Created and Date Modified to 12/13/08.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Menu (food).docx	Menu (food).docx	12/13/2008 12:38:09 P...	12/13/2008 12:38:09 P...	10/9/2020 8:32:04 AM	10/9/2020 8:31 AM
 Menu (food).odt	Menu (food).odt	12/13/2008 12:38:09 P...	12/13/2008 12:38:09 P...	1/4/2021 9:58:50 PM	12/13/2008 12:38 PM
 Menu (food).pdf	Menu (food).pdf	12/13/2008 12:38:09 P...	12/13/2008 12:38:09 P...	1/4/2021 10:01:26 PM	12/13/2008 12:38 PM
 Menu (food).rtf	Menu (food).rtf	12/13/2008 12:38:09 P...	12/13/2008 12:38:09 P...	1/4/2021 9:59:18 PM	12/13/2008 12:38 PM
 Menu (food).tmd	Menu (food).tmd	12/13/2008 12:38:09 P...	12/13/2008 12:38:09 P...	1/4/2021 9:58:30 PM	12/13/2008 12:38 PM
 Menu (food).txt	Menu (food).txt	12/13/2008 12:38:09 P...	12/13/2008 12:38:09 P...	1/4/2021 9:59:40 PM	12/13/2008 12:38 PM

Same results as before. Item Date reflects the change with all filetypes except the Microsoft Word document.

I changed the Date Created, Date Modified and Date Accessed to all different timestamp values.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Menu (food).docx	Menu (food).docx	9/24/2011 10:14:38 AM	4/6/2007 2:43:13 PM	11/17/2005 1:17:44 AM	10/9/2020 8:31 AM
 Menu (food).odt	Menu (food).odt	9/24/2011 10:14:38 AM	4/6/2007 2:43:13 PM	11/17/2005 1:17:44 AM	4/6/2007 2:43 PM
 Menu (food).pdf	Menu (food).pdf	9/24/2011 10:14:38 AM	4/6/2007 2:43:13 PM	11/17/2005 1:17:44 AM	4/6/2007 2:43 PM
 Menu (food).rtf	Menu (food).rtf	9/24/2011 10:14:38 AM	4/6/2007 2:43:13 PM	11/17/2005 1:17:44 AM	4/6/2007 2:43 PM
 Menu (food).tmd	Menu (food).tmd	9/24/2011 10:14:38 AM	4/6/2007 2:43:13 PM	11/17/2005 1:17:44 AM	4/6/2007 2:43 PM
 Menu (food).txt	Menu (food).txt	9/24/2011 10:14:38 AM	4/6/2007 2:43:13 PM	11/17/2005 1:17:44 AM	4/6/2007 2:43 PM

The Microsoft Word document is the only filetype that retained the original, Date Last Saved value. All others used the earliest date between Date Created and Date Modified.

Preliminary conclusion:







Microsoft only refers to a document file as one of its own and does not recognize the other formats as document types for purposes of mapping to the Last Date Saved value if available.

This includes .rtf (Rich Text Format) surprisingly since it belongs to the Microsoft family of formats.

Section # 8: Auto Datev3.4 New Additions

Continuation of the testing waited until a few days later, Jan 6, 2021.

In this second series of tests, I loaded each of the documents from the first test into their respective editor programs, made minor changes, then saved each modified document and checked the results.

Name ▲	New Name	Created	Modified	Accessed	Item Date
 Menu (food).docx	Menu (food).docx	9/24/2011 10:14:38 AM	1/6/2021 2:19:55 PM	1/6/2021 2:19:55 PM	1/6/2021 2:19 PM
 Menu (food).odt	Menu (food).odt	9/24/2011 10:14:38 AM	1/6/2021 2:20:14 PM	1/6/2021 2:20:14 PM	1/6/2021 2:20 PM
 Menu (food).pdf	Menu (food).pdf	9/24/2011 10:14:38 AM	1/6/2021 2:21:37 PM	1/6/2021 2:21:37 PM	1/6/2021 2:21 PM
 Menu (food).rtf	Menu (food).rtf	9/24/2011 10:14:38 AM	1/6/2021 2:21:59 PM	1/6/2021 2:21:59 PM	1/6/2021 2:21 PM
 Menu (food).tmd	Menu (food).tmd	9/24/2011 10:14:38 AM	1/6/2021 2:22:26 PM	1/6/2021 2:22:26 PM	9/24/2011 10:14 AM
 Menu (food).txt	Menu (food).txt	9/24/2011 10:14:38 AM	1/6/2021 2:22:41 PM	1/6/2021 2:22:41 PM	1/6/2021 2:22 PM

For each file tested, the following holds true:

- a. The Date Last Saved Metadata is modified to the current timestamp, the date the file was edited, Jan 6, 2021.

Date last saved	1/6/2021 2:19 PM	System.Document.DateSaved
-----------------	------------------	---------------------------

- b. Every file with the exception of the TextMaker document, changed Item Date to reflect the Date Last Saved value, even though this value is not the earliest date available. If Item Date was using the earliest date between Date Created and Date Modified, then Item Date would have assumed the Date Created value of 9/24/11 and not the value of 1/6/21.

This is conclusive proof that **IF** the filetype is recognized as a document by Windows and **IF** the Date Last Saved Metadata is available, then Item Date assumes this value.

I have stated that, '**IF Windows recognizes the file as a document filetype**', at various places in this section, because the TextMaker document file, with the extension .tmd, did not use the Date Last Saved Metadata for the value of Item Date, but instead fell back to using the earliest date between Date Created and Date Modified.

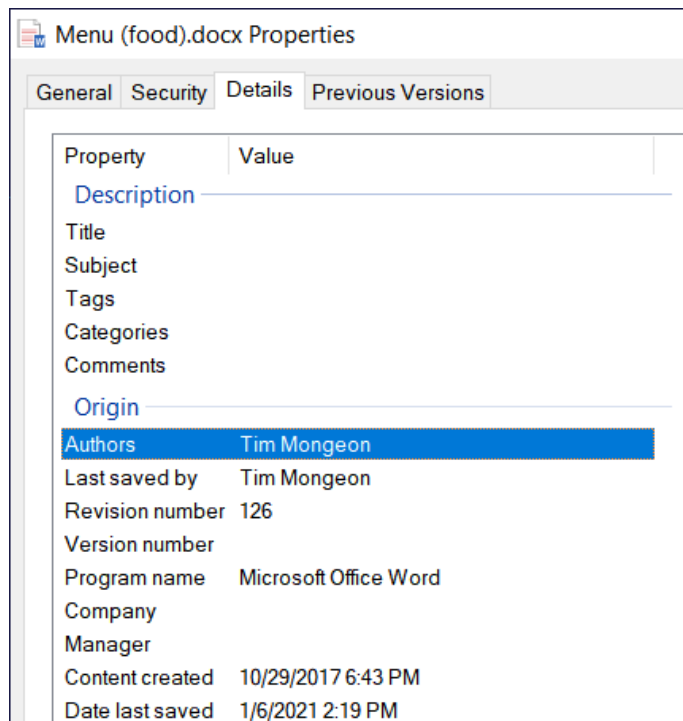
This concludes that **Item Date will only assume the value of Date Last Saved if the file is recognizable as a document filetype**. I should also note that I performed additional testing for verification, the results of which are not included herein.

How Windows determines what is recognizable as a document, I am not certain. Windows obviously looks at the file extension for the filetype, and .tmd is obviously a proprietary format that is not as well known as, for instance, Open Office filetypes, so that may be the sole explanation, but again, not verified. I should also mention that I did verify that I have the .tmd extension associated with the TextMaker program, thus Windows is well aware of the program association.

Section # 8: Auto Date

v3.4 New Additions

Also of note is that Microsoft Office adds its own additional Metadata to its document files. This can be seen in the Details tab of the document's properties sheet under Origin.



Final conclusions:

Preliminary findings were wrong, because the OS does recognize more filetypes as documents than originally thought. This includes, PDF, RTF, ODT and TXT. There are probably more but these were the ones tested. I also found that if a filetype is not recognized by the OS as a document filetype, then Item Date will treat that filetype as a 'non-supported or other filetype' and assume the earlier value between Date Created and Date Modified. Date Last Saved will also assume this value (again, I look at this as an anomaly).

The explanation for the preliminary findings in error is that these files, other than the Microsoft Word file, were never edited in the first phase of testing. They were created and therefore any value for Last Saved in the Metadata was premature. It wasn't until the files were edited and saved, that Item Date assumed the value in the second phase.

Notes:

1. Other Office documents, .PPT (PowerPoint), .xlsx (Excel), were not tested. It is presumed though, that because Office considers these filetypes as documents, Item Date may also assume the Date Last Saved of these filetypes.

Section # 8: Auto Date

v3.4 New Additions

Anomalies: exe

This particular .exe file holds no Metadata. Just as with Taken (Original), if there is no data available, then the Item Date column for that entry will be blank.

Name	New Name	Created	Modified	Accessed	Item Date
apm.exe	apm.exe	12/25/2018 10:47:51 A...	12/25/2018 10:47:53 AM	12/25/2018 10:47:51 AM	

But that's odd because it does have values for the Windows Properties, Date Created and Date Modified. Why then doesn't Item Date assume the earliest of these two values as I concluded earlier?

Upon examining the file, using the Windows File Properties list from within BRU, I verified there was no Metadata that could be extracted.



Properties		
Label	Value	Name

Under Windows Properties accessed through the file's Properties option using the Context Menu, the Date Created, Date Modified, and Date Accessed File Attributes are listed under General, but the Details tab, used for displaying if any Metadata is available, is empty except for data reported from the File System.

The image shows two screenshots of the Windows File Properties dialog for 'apm.exe'. The left screenshot shows the 'General' tab with the following information:

- Type of file: Application (.exe)
- Description: apm.exe
- Location: H:\Documents
- Size: 109 KB (112,209 bytes)
- Size on disk: 112 KB (114,688 bytes)
- Created: Tuesday, December 25, 2018, 10:47:51 AM
- Modified: Tuesday, December 25, 2018, 10:47:53 AM
- Accessed: Tuesday, December 25, 2018, 10:47:51 AM

The right screenshot shows the 'Details' tab, which is mostly empty except for a 'Description' section containing the following information:

Property	Value
Description	
File description	
Type	Application
File version	
Product name	
Product version	
Copyright	
Size	109 KB
Date modified	12/25/2018 10:47 AM
Language	

When validating this finding using the Nirsoft Property System Viewer (freeware), however, the program displayed all of the missing Metadata properties (not shown).

Section # 8: Auto Date

v3.4 New Additions

Here is another example. One has Item Date, the other does not. Verification that not all exe files exhibit this problem.

Name ▲	New Name	Size	Created	Modified	Accessed	Item Date
wildcrypt.exe	wildcrypt.exe	22 KB	12/24/2018 8:57:44 AM	12/24/2018 8:57:47 AM	12/24/2018 8:57:44 AM	
WildGem.exe	WildGem.exe	1 MB	12/3/2018 7:33:58 PM	12/3/2018 7:33:58 PM	12/3/2018 7:33:58 PM	12/3/2018 7:33 PM

I again tested using the Nirsoft Property System View program. Using not only the file above but other test files, I found that all files, those with and those without **an Item Date in BRU, had Metadata including System.ItemDate**. The question becomes, why doesn't BRU see the System.ItemDate **if available** to determine BRU's Item Date value?

PropertySystemView - M:\test BRU 3\wildcrypt.exe

File Edit View Options Help

Load properties of the following path:

M:\test BRU 3\wildcrypt.exe

Go (F8)

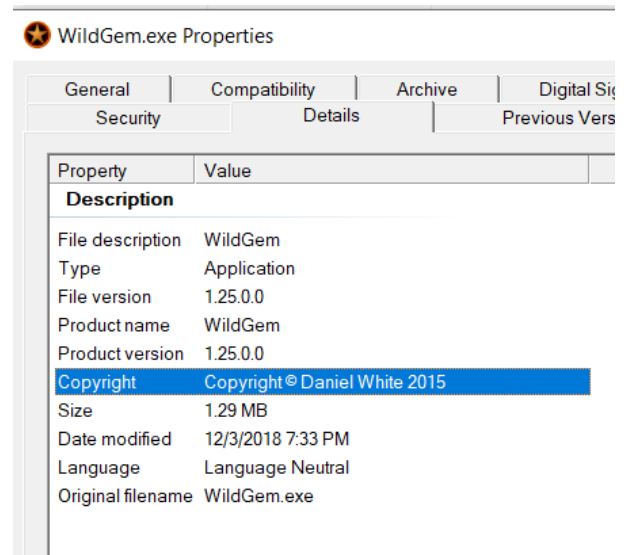
Canonical Name	Display Name	Formatted Value
		wildcrypt.exe
System.AppUserModel.ID	AppUserModelId	
System.AppUserModel.ParentID	Parent id	
System.AppZonIdentifier		
System.ComputerName	Computer	
System.ContentType	Content type	application/x-msdownload
System.DateAccessed	Date accessed	12/24/2018 8:57 AM
System.DateCreated	Date created	12/24/2018 8:57 AM
System.DateImported	Date imported	12/24/2018 8:57 AM
System.DateModified	Date modified	12/24/2018 8:57 AM
System.Document.DateCreated	Content created	12/24/2018 8:57 AM
System.Document.DateSaved	Date last saved	12/24/2018 8:57 AM
System.ExpandoProperties		
System.FileAttributes	Attributes	N
System.FileAttributesDisplay		
System.FileExtension	File extension	.exe
System.FileName	Filename	wildcrypt.exe
System.FileOwner	Owner	
System.FilePlaceholderStatus		6
System.IsFolder	Files	Files
System.IsShared	Shared	No
System.ItemDate	Date	12/24/2018 8:57 AM
System.ItemFolderNameDisplay	Folder name	test BRU 3
System.ItemFolderPathDisplay	Folder path	M:\test BRU 3
System.ItemFolderPathDisplayNarrow	Folder	test BRU 3 (M:)
System.ItemName		wildcrypt.exe

Section # 8: Auto Date

v3.4 New Additions

Although BRU only displays what it extracts, the data **is** there at least according to the Nirsoft program, **but BRU is not extracting it for some reason**. Is it because BRU is set to extract only that Metadata which Windows displays in the Details tab of the file's property sheet?

Using WildGem.exe, a file that does have an Item Date in BRU, the Details tab doesn't display much, but there is evidence of some Metadata:



Therefore, at least for this .exe, Metadata does exist and BRU displays the Item Date for this file – see previous page. But this reveals very little as to an explanation of why it can extract one but not the other.

And because **Date Created and Date Modified exist**, why doesn't System.ItemDate have a value?

Here's why.

The explanation as far as BRU is concerned, is that BRU only 'reads' whatever Metadata values are there for extraction of the Item Date value. The properties of Date Created, Date Modified and Date Accessed, if you remember, belong to the 'Standard Attributes' of the Resident Attributes found in the File System and are not stored within the file as is other Metadata.

The fact that the Metadata contents of the file is void is why BRU cannot read a value for Item Date. It only reports the news, and doesn't make it. The General tab of the Properties sheet echoes back the information reported from the File System. The Details tab displays some of the Metadata if 'any' exists in the file. If it is blank that means the file doesn't contain any. If System.ItemDate has no value, then Item Date has no value because they are the same.

This doesn't explain though, how the Windows File Properties reports no Metadata, as the Properties sheet shows, but the Nirsoft program provides a full listing. When I contacted Nirsoft as to why, they informed me that they obtain this data through the Windows Property System API (<https://docs.microsoft.com/en-us/windows/win32/api/propsys/>). TGRMN also verified that they use the same API for the extraction of the Metadata, so the mystery continues.

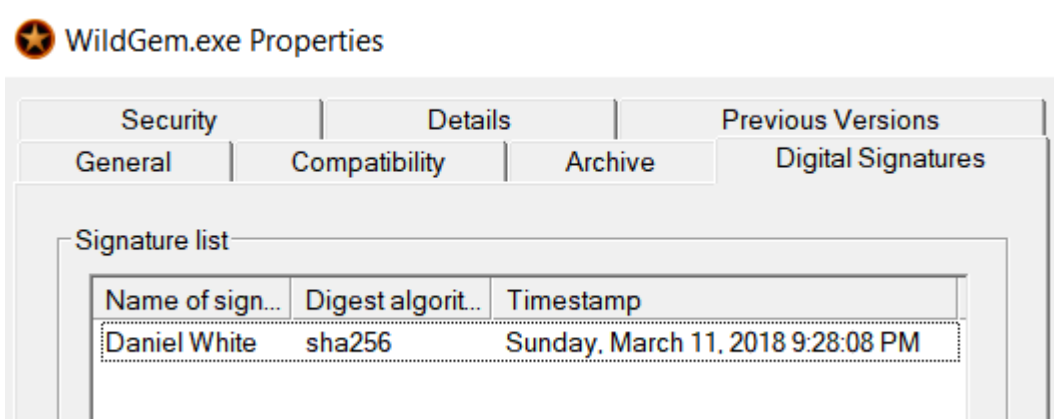
At least so far, the only filetype I have found that exhibits this strange behavior is .exe files, not .msi., archives, document files, etc. I have tried doing a comparison between the two .exe files to see if there was anything different or missing from the property listings as reported by the Nirsoft program. There was none.

Section # 8: Auto Date

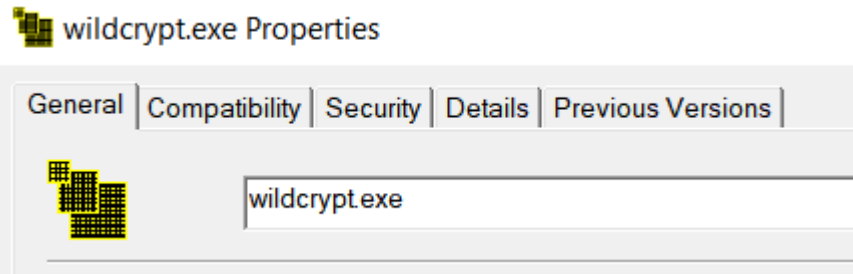
v3.4 New Additions

A further test was done as to a theory proposed that the problem could be related with Digital Signatures.

I looked at .exe's, some of which had Digital Signatures with a Digital Signature Timestamp and some without. The files that have Digital Signatures are identified by a separate tab in the Properties sheet for the file:



And those that do not, display as:



Simple enough. The file with the Digital Signature had a value for Item Date. Could this be the missing factor?

Unfortunately no; and here's why.

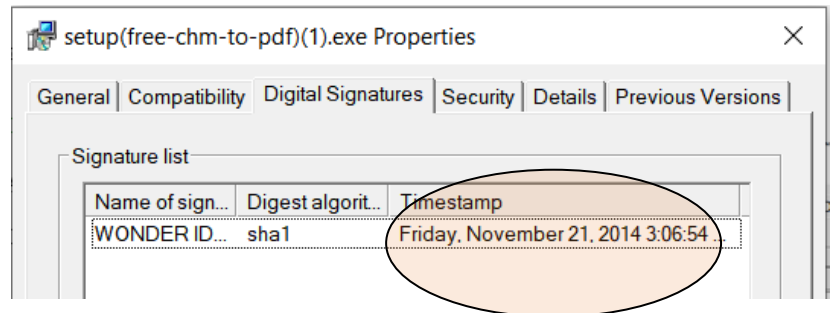
Section # 8: Auto Date

v3.4 New Additions

I took a file with a Digital Signature:

setup(free-chm-to-pdf)(1).exe

I



This .exe file has Metadata and displays Item Date in BRU with a value, 11/15/2020. This value, you will notice, is not that of the Digital Signature Timestamp 11/21/2014, but of the earliest date between Date Created and Date Modified.

Name ▲	New Name	Created	Modified	Accessed	Item Date
setup(free-chm-to-pdf)(1).exe	setup(free-chm-to-...	11/15/2020 10:57:53 A...	1/6/2021 10:04:31 AM	1/6/2021 10:04:31 AM	11/15/2020 10:57 AM

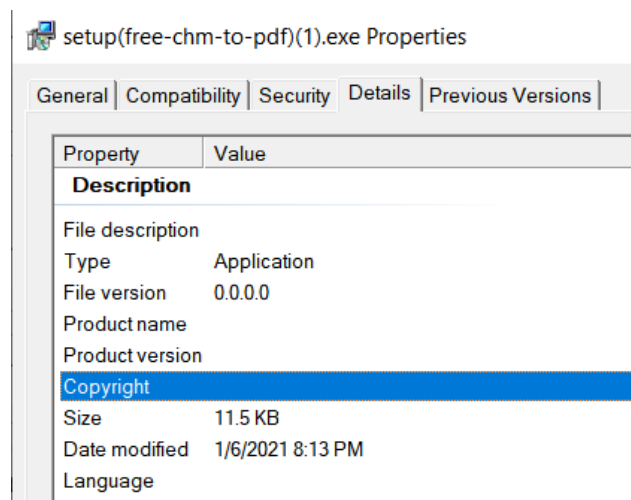
The theory in question is not whether Item Date is using the Digital Signature Timestamp for its value, but that the very presence of the Digital Signature is somehow contributing to Windows using the File System's Date Created or Date Modified rather than simply disregarding the value as it did with wildcrypt.exe.

I next removed the Digital Signature from the file, expecting and hoping that Item Date would be blank as a result.

It was not. No change.

Name ▲	New Name	Created	Modified	Accessed	Item Date
setup(free-chm-to-pdf)(1).exe	setup(free-chm-to-...	11/15/2020 10:57:53 A...	1/6/2021 10:04:31 AM	1/6/2021 10:04:31 AM	11/15/2020 10:57 AM

Finally, I removed all of the Metadata from the file:



Item Date still held a value.


Name ▲	New Name	Created	Modified	Accessed	Item Date
setup(free-chm-to-pdf)(1).exe	setup(free-chm-to-...	1/6/2021 8:13:57 PM	1/6/2021 8:13:57 PM	1/6/2021 8:13:57 PM	1/6/2021 8:13 PM

Theory failed. Digital Signatures are not a contributing factor as to why certain .exe files hold no value for Item Date.

Section # 8: Auto Datev3.4 New Additions**Anomalies: Non-Document or Other Filetype Breaks Microsoft's Rules**

For purposes of clarification, the following illustrates how Windows manages a 'non-recognized document filetype or other filetype', when changing timestamps and in doing so, breaks its own rules of behaviour.

This is an example of an MP3 file with all dates set to 8/31/19

Name ▲	New Name	Size	Created	Modified	Accessed	Item Date
 Portishead-Roads.mp3	Portishead-Roads....	13 MB	8/31/2019 10:32:12 PM	8/31/2019 10:32:27 PM	8/31/2019 10:32:12 PM	8/31/2019 10:32 PM

Here is the pertinent Windows Metadata:

Label	Value	Name
Folder name	test BRU 3	System.ItemFolderNameDisplay
Type	MP3 Audio File (VLC)	System.ItemTypeText
Name	Portishead-Roads.mp3	System.ItemNameDisplay
Size	12.5 MB	System.Size
Attributes	A	System.FileAttributes
Date modified	8/31/2019 10:32 PM	System.DateModified
Date created	8/31/2019 10:32 PM	System.DateCreated
Date accessed	8/31/2019 10:32 PM	System.DateAccessed
	Portishead-Roads	System.ItemNameDisplayWithoutExtension
Content type	audio/mpeg	System.ContentType
Title	Portishead-Roads	System.Title
Authors		System.Author
Comments	Recorded and Tagge	System.Comment
Content created	8/31/2019 10:32 PM	System.Document.DateCreated
Date last saved	8/31/2019 10:32 PM	System.Document.DateSaved
Date	8/31/2019 10:32 PM	System.ItemDate
	Music	System.KindText
Participants		System.ItemParticipants
Activity		System.StorageProviderAggregatedCustomStates
Date imported	8/31/2019 10:32 PM	System.DateImported

As you know, even though this is not a document filetype, the Document Metadata of Content Created and Date Last Saved are available. Regardless, Windows will treat this file as a non-document filetype where the Date Last Saved value will **not** be a consideration in determining Item Date, nor should it be.

Section # 8: Auto Datev3.4 New Additions**The Rules (repeated from previous):****Content Created and Last Date Saved**

These Dates are added to the file via the Application, e.g., Microsoft Word - <https://docs.microsoft.com/en-us/windows/win32/properties/props-system-document-datecreated>

Indicates the date and time that a document was created. This information is stored in the document, not obtained from the file system.

The value is in two phases. The first is when the document is created. Content Created will share the same value as Date Created. The second phase is when the content is entered and saved for the first time. Content Created is updated to reflect the current timestamp and will not change from this point forward.

System.DateImported

The date and time the file was imported into a private application database. For example, this property can be used when a photo is imported into a photo database.

Breaking the Rules:

First off, let's look at the Windows Property Names for Content Created and Date Last Saved:

Label	Name
Content Created	(System.Document.DateCreated)
Date Last Saved	(System.Document.DateSaved)


Notice that these are **Document** Metadata. They are only supposed to be created for Document filetypes and even then, only those filetypes that Windows recognizes as Documents. But I have seen them in all sorts of non-document filetypes. The rules are broken, but that doesn't invalidate my findings. As long as they are present, then I have to test how they apply and I have. My conclusions can be viewed under, 'How I tested Document Metadata'.

Now this may change in future Windows revisions and updates, because everything about Windows is not written in stone anymore. Here today, gone tomorrow – usually with no warning. Microsoft, in addition, has undocumented algorithms that are embedded in the code that do strange things. These particular anomalies could be related.


The program creating the file, e.g., Microsoft Word, adds this Metadata. Does this mean that the MP3 created by a program from Jaksta added this? If so, Windows allowed it. I don't have answers for that. But what I do know is how Windows handles these non-document filetypes in determining the Item Date value. It doesn't use them. Instead, as I have already concluded, non-document filetypes will base the value on the earliest date between Date Created and Date Modified. Only those document filetypes, as recognized by Windows, will use the Date Last Saved value.

Section # 8: Auto Datev3.4 New Additions

Here is that MP3 example of a non-document file again.

Name ▲	New Name	Size	Created	Modified	Accessed	Item Date
 Portishead-Roads.mp3	Portishead-Roads...	13 MB	8/31/2019 10:32:12 PM	8/31/2019 10:32:27 PM	8/31/2019 10:32:12 PM	8/31/2019 10:32 PM

I Changed the Date Modified value to 3/2/2014.

Name ▲	New Name	Size	Created	Modified	Accessed	Item Date
 Portishead-Roads.mp3	Portishead-Roads...	13 MB	8/31/2019 10:32:12 PM	3/2/2014 7:21:00 PM	8/31/2019 10:32:12 PM	3/2/2014 7:21 PM

result: Item Date as expected assumed the value of the earlier date, 3/2/2014, of either Date Created or Date Modified.

But that's not all that changed...

Properties

Label	Value	Name
Folder name	test BRU 3	System.ItemFolderNameDisplay
Type	MP3 Audio File (VLC)	System.ItemTypeText
Name	Portishead-Roads.mp3	System.ItemNameDisplay
Size	12.5 MB	System.Size
Attributes	A	System.FileAttributes
Date modified	3/2/2014 7:21 PM	System.DateModified
Date created	8/31/2019 10:32 PM	System.DateCreated
Date accessed	8/31/2019 10:32 PM	System.DateAccessed
	Portishead-Roads	System.ItemNameDisplayWithoutExtension
Content type	audio/mpeg	System.ContentType
Title	Portishead-Roads	System.Title
Authors		System.Author
Comments	Recorded and Tagged	System.Comment
Content created	3/2/2014 7:21 PM	System.Document.DateCreated
Date last saved	3/2/2014 7:21 PM	System.Document.DateSaved
Date	3/2/2014 7:21 PM	System.ItemDate
Date imported	3/2/2014 7:21 PM	System.DateImported

Changed these Windows Properties:

Date Modified
(System.DateModified)
8/31/2019 to 3/2/2014

Content Created
(System.Document.DateCreated)
8/31/2019 to 3/2/2014

Date Last Saved
(System.Document.DateSaved)
8/31/2019 to 3/2/2014

Date
(System.ItemDate)
8/31/2019 to 3/2/2014

Date Imported
(System.DateImported)
8/31/2019 to 3/2/2014

Section # 8: Auto Datev3.4 New Additions

Results.


Content Created date is not supposed to change once established. And if you remember, it is established once the file has had content entered and saved for the first time. Changing a timestamp should not be relevant to this action.

Date Imported, according to Microsoft's own definition, is updated, or created for that matter, if the file is imported into some application database. Does changing the timestamp from within BRU or using a File Manager program like ZTree to alter the timestamps qualify? I wouldn't believe so.

Date Last Saved was also affected. Changing Date Modified should have no effect on this value as well.

However this is uncharted territory so who knows?

Using the original file again, when I changed the Date Created to 3/2/01

Name ▲	New Name	Size	Created	Modified	Accessed	Item Date
 Portishead-Roads.mp3	Portishead-Roads...	13 MB	3/2/2014 7:21:00 PM	8/31/2019 10:32:27 PM	8/31/2019 10:32:12 PM	3/2/2014 7:21 PM

Results:

Once again, as expected, Item Date assumed the earliest value between Date Created and Date Modified.

However, interestingly enough...

Content created	3/2/2014 7:21 PM	System.Document.DateCreated
Date last saved	8/31/2019 10:32 PM	System.Document.DateSaved


Date Last Saved did NOT change as it did when I changed Date Modified. I am, however, unsure of what significance this plays, if any, because this Document Metadata property shouldn't even be there in the first place.

The other timestamp values followed the same course as before.

Date Created	8/31/2019	to	3/2/2014
Date Modified	no change		
Date Imported	8/31/2019	to	3/2/2014

Section # 8: Auto Datev3.4 New Additions

Now I will change both the Date Created and Date Modified values to 3/2/2014.

Name ▲	New Name	Size	Created	Modified	Accessed	Item Date
 Portishead-Roads.mp3	Portishead-Roads...	13 MB	3/2/2014 7:21:00 PM	3/2/2014 7:21:00 PM	8/31/2019 10:32:12 PM	3/2/2014 7:21 PM

Results:

Item Date – expected behaviour. Takes the earliest value between Date Modified and Date Created. In this case they are the same.

Content created	3/2/2014 7:21 PM	System.Document.DateCreated
Date last saved	3/2/2014 7:21 PM	System.Document.DateSaved

Because changing Date Modified affected Date Last Saved prior, it was therefore affected this time and changed to 3/2/2014. Content Created once again changed as well to 3/2/2014 and all other values listed on previous page in green, changed as expected, at least in accordance with this strange scenario. I should also note that ID3 Metadata for this filetype had no bearing on any of these outcomes and was unaffected by any of these changes.

As to my reasoning of these anomalies, I'm just putting this out there. I have no explanation and no theories.

Notes:

1. These tests were conducted on other non-document filetypes, not just the single sample MP3 file used in these examples, with the same conclusive or should I say, inconclusive findings.

Final Conclusions on the Anomalies section:

1. Regarding the .exe anomaly, TGRMN has no explanation at this time as to the discrepancy between BRU's extraction methods and Nirsoft. In fairness, it should be pointed out that the Windows Property Sheet substantiates BRU's conclusion of no Metadata available for the file in question. Consequently, it is perhaps Nirsoft's findings that are at fault in this matter. In either regard, I consider the matter closed.
2. Document Metadata should not exist for filetypes that are non-documents, but it does. This, however is not taken in consideration when determining Item Date's value for these filetypes, and therefore a moot point.

Section # 8: Auto Datev3.4 New Additions

A final thought on a problem that some users may come up against that is fairly common.

Scenario:

Windows Properties' Date Modified is Earlier than the Date Created value.

- a. This occurs when the file is copied from another medium storage, e.g., USB, FTP, etc. Windows changes the Date Created to the date of when the file was copied but the Date Modified remains as is. This also occurs if transferring files from one PC to another, e.g., setting up a new system.
- b. If you copy files, the OS modifies the original Date Created to the date of when files are copied. This does not occur when moving files unless moving files from one File System to another, e.g., Linux to Windows, at which point, the move operation is really creating a new file; thus you get a new Date Created followed by deletion of the old file on the previous File System.
- c. Restoration of a backup, depending on the program used and its options, may change the Date Created to the date of the restoration.

Prevention – none. This behaviour is part of the Windows OS.

Remedy –

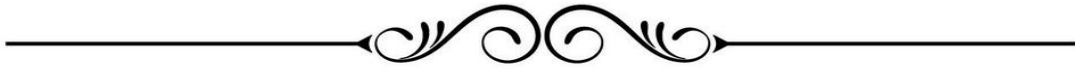
Third party freeware utilities can be used to copy or move files without affecting the dates. This solution is not a prevention, but a work-around, because these utilities will bypass the OS.

After the fact –

Only commercial utilities are available.

Final Notes on Item Date:

1. For more information, refer to, Understanding Metadata, in this section.
2. The Item Date value, as with all of the other Metadata values, are evaluated outside of BRU. BRU only extracts whatever values are available, it cannot create or alter them, with the exception of the Windows properties, Date Created, Date Modified and Date Accessed.

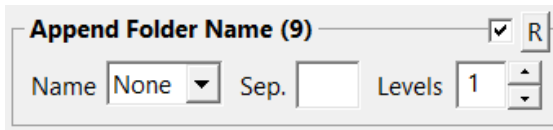


Section # 9 - Append Folder Name

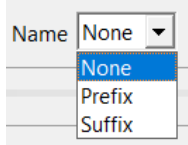
Section # 9: Append Folder Name

Append Folder Name – The names of the directory and sub directory levels can be appended as a **prefix** or **suffix**.

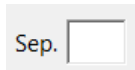
This is useful where you have lots of directories, each containing the same group of files, and you want to merge all the files into a single folder all with unique names. You could also use ‘Prevent Duplicates’ from the Renaming Options Menu.



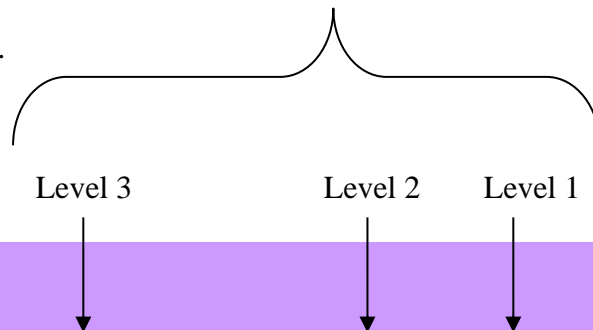
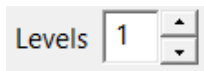
1. Select where to append under Name.



2. Select if you want to use any **separator** characters to distinguish between the appended data and the original filename.

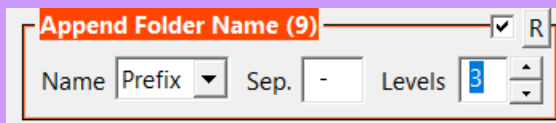


3. Select how many directory **levels** to include.



Example,

File, Cat.jpg located in D:\Documents and Settings\Administrator\Pictures



Level 3 as Prefix using a <hyphen> as a separator results in New Name:

Documents and Settings-Administrator-Pictures-Cat.jpg

Note – any colon or backslash characters are converted to <hyphen> characters automatically by BRU.

For more information, refer to –

Renaming Options Menu

‘Append Folder Name Levels and Rearrange them’ in the JavaScript section of ‘Volume 2

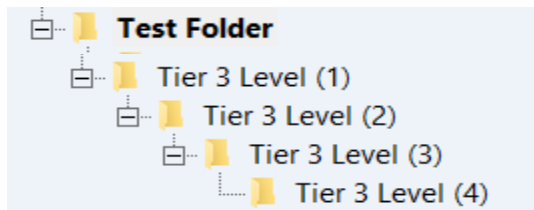
Section # 9: Append Folder Name

v3.42 New Additions

Append Negative Value for Specific Directory Name

Added ability to use negative values for 'Levels' in 'Section #9: Append Folder Name'. A negative level number will only append that specific directory name to the file name instead of the current sub-directory path.

This is a directory hierarchy I created that has four levels of subdirectories coming off of the root, Test Folder.



File #9
File #10
File #11
File #12

In each of these directories I have placed a single file.

In the above directory hierarchy, there are 5 levels including the root directory. The range for Levels is from:

Level 0

to

Level 5

or

Level 0

to

Level -5

Section # 9: Append Folder Namev3.42 New Additions

Example #1:

Name: Prefix
 Separator: <space>

File #11 resides in Tier 3 Level (3). Range of levels is thereby 0 through -4, including the root directory, Test Folder.

Level 0

Name	New Name	Sub Dir. ▼
File #11	File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

At the default, Level 0, no appending, but New name does reflect a change because of a specified Separator at Prefix.

Level -1

New Name	Sub Dir. ▼
Tier 3 Level (3) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

Level -1 specifies the current directory in which the file resides, Tier 3 Level (3).

Level -2

New Name	Sub Dir. ▼
Tier 3 Level (2) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

Level -2 specifies the subdirectory above the current directory one level up.

Level -3

New Name	Sub Dir. ▼
Tier 3 Level (1) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

Level -3 specifies the subdirectory two levels up from current directory, first subdirectory off from root, Test Folder.

Level -4

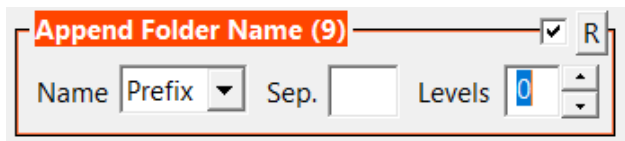
New Name	Sub Dir. ▼
Test Folder File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

Level -4 specifies the subdirectory three levels up from the current directory. For this example, it is the root directory.

Section # 9: Append Folder Name

v3.42 New Additions

Example #2:



Name: Prefix
Separator: <space>

File #11 resides in Tier 3 Level (3). Range of levels is thereby 0 through 4, including the root directory, Test Folder.

Level 0

Name	New Name	Sub Dir.
File #11	File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

At the default, Level 0, no appending, but New name does reflect a change because of a specified Separator at Prefix.

Level +1



New Name	Sub Dir.
Tier 3 Level (3) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

At Level 1, the current subdirectory path is specified. The Current subdirectory contains the example file, File #11.

Level +2



New Name	Sub Dir.
Tier 3 Level (2) Tier 3 Level (3) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

At Level 2, the subdirectory path includes the current subdirectory and one directory level up from there.

Level +3



New Name	Sub Dir.
Tier 3 Level (1) Tier 3 Level (2) Tier 3 Level (3) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

At Level 3, the subdirectory path includes the current subdirectory and two directory levels up from there.

Level +4



New Name	Sub Dir.
Test Folder Tier 3 Level (1) Tier 3 Level (2) Tier 3 Level (3) File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

At Level 4, in this example, the full subdirectory path is specified and includes the root directory, Test Folder.

Section # 9: Append Folder Name

v3.42 New Additions

Notes:

1. Subfolder must be enabled in '[Section #12: Filters](#)'.
2. How many levels are displayed is determined by the Sub-folders Level setting in '[Section #12: Filters](#)'.

For example:

If I limit the levels to three, there is no display of the 4th Tier. Only the third Tier containing File #11 would be the maximum level available:

Filters (12)

Mask: * Match Case RegEx Folders Hidden Files Subfolder Lvl: 3 Path Min

Condition:

Name	New Name	Sub Dir. ▼
File #11	File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)
File #10	File #10	\Tier 3 Level (1)\Tier 3 Level (2)
File #9	File #9	\Tier 3 Level (1)
File #7	File #7	\Tier 2 Level 1\Tier 2 Level 2\Tier 2 Level 3
File #6	File #6	\Tier 2 Level 1\Tier 2 Level 2
File #5	File #5	\Tier 2 Level 1
File #3	File #3	\Tier 1 Level a\Tier 1 Level b\Tier 1 Level c
File #2	File #2	\Tier 1 Level a\Tier 1 Level b
File #1	File #1	\Tier 1 Level a

But if I have the level at zero, which is the default maximum...

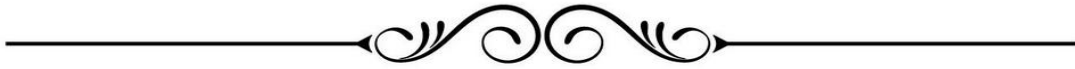
Filters (12)

Mask: * Match Case RegEx Folders Hidden Files Subfolder Lvl: 0 Path Min

Condition:

... now you can see the 4th Tier containing File #12:

Name	New Name	Sub Dir. ▼
File #12	File #12	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)\Tier 3 Level (4)
File #11	File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)
File #10	File #10	\Tier 3 Level (1)\Tier 3 Level (2)
File #9	File #9	\Tier 3 Level (1)
File #8	File #8	\Tier 2 Level 1\Tier 2 Level 2\Tier 2 Level 3\Tier 2 Level 4
File #7	File #7	\Tier 2 Level 1\Tier 2 Level 2\Tier 2 Level 3
File #6	File #6	\Tier 2 Level 1\Tier 2 Level 2
File #5	File #5	\Tier 2 Level 1
File #4	File #4	\Tier 1 Level a\Tier 1 Level b\Tier 1 Level c\Tier 1 Level d



Section # 10 - Numbering

Section # 10: Numbering

Numbering – Adds a sequential sequence, numeric or alpha, to a group of files, also known as ‘Autonumbering’.

Notes:

1. You can sort the file list using the column headers, just as you would do in Windows Explorer.
2. ‘Section #4: Case’ has no effect on the Alpha Base sequence.
3. The files will always be **processed** in the order of the *displayed sequence* –. In other words, the displayed files’ position in the Content Pane will determine the **numeric sequence** of numbers (or alpha) applied to the filenames.

To further demonstrate what I mean by the order of the displayed sequence -

As it stands now,

A.	Name ▲	New Name
A	DSCN0001.JPG	DSCN0001.JPG
A	DSCN0029.JPG	DSCN0029.JPG
A	DSCN0032.JPG	DSCN0032.JPG
A	DSCN0055.JPG	DSCN0055.JPG

The files will be processed in this order:

DSCN0001.JPG
DSCN0029.JPG
DSCN0032.JPG
DSCN0055.JPG

By changing the order of the files, the order in which the files will be processed is also changed.

A.	Name ▲	New Name
A	DSCN0032.JPG	DSCN0032.JPG
A	DSCN0001.JPG	DSCN0001.JPG
A	DSCN0055.JPG	DSCN0055.JPG
A	DSCN0029.JPG	DSCN0029.JPG

The files will now be processed in the following order:

DSCN0032.JPG
DSCN0001.JPG
DSCN0055.JPG
DSCN0029.JPG

For more information refer to –

- ‘Sorting’ under the Display Options Menu
- ‘List - Reposition’, under the Actions Menu
- ‘Apply Random Sort to Current List’ under the Actions Menu
- ‘Rename in Reverse Order’, under the Renaming Options Menu

Section # 10: Numbering

Mode

Mode Select among the following -

Start	<input type="button" value="None"/>	None –	no changes made (default)
Pad	<input type="button" value="Prefix"/>	Prefix –	append at the beginning of filename
Break	<input type="button" value="Suffix"/>	Suffix –	append at the end of filename
	<input type="button" value="Pre.+Suff."/>	Pre. + Suff. –	append at both the beginning and end of the filename
	<input type="button" value="Insert"/>		

Prefix:

Name	New Name
DSCN0032.JPG	1DSCN0032.JPG
DSCN0001.JPG	2DSCN0001.JPG
DSCN0055.JPG	3DSCN0055.JPG
DSCN0029.JPG	4DSCN0029.JPG

Suffix:

Name	New Name
DSCN0032.JPG	DSCN00321.JPG
DSCN0001.JPG	DSCN00012.JPG
DSCN0055.JPG	DSCN00553.JPG
DSCN0029.JPG	DSCN00294.JPG

Prefix + Suffix

Name	New Name
DSCN0032.JPG	1DSCN00321.JPG
DSCN0001.JPG	2DSCN00012.JPG
DSCN0055.JPG	3DSCN00553.JPG
DSCN0029.JPG	4DSCN00294.JPG

Mode at Insert – specify by position where in the filename the numbering sequence should appear

Name	New Name
DSCN0032.JPG	DSCN01032.JPG
DSCN0001.JPG	DSCN02001.JPG
DSCN0055.JPG	DSCN03055.JPG
DSCN0029.JPG	DSCN04029.JPG

Position 1 2 3 4 5

Notes:

1. Under BRU criteria other than RegEx, the position starts with 1. Under Regular Expressions and programs like RegEx Buddy the position starts with 0.

Section # 10: Numbering

Start

Start
 Incr.

Specify a starting number (or in the case of alpha, a letter designation, 1=A, 2 =B, etc.) and increment value.

Start = 2, Increment = +1

Start = 2, Increment = +2

Numbering (10)

Mode at

Start
 Incr.

Numbering (10)

Mode at

Start
 Incr.

Numeric

Name ▲	New Name
DSCN0032.JPG	2DSCN0032.JPG
DSCN0001.JPG	3DSCN0001.JPG
DSCN0055.JPG	4DSCN0055.JPG
DSCN0029.JPG	5DSCN0029.JPG

Name ▲	New Name
DSCN0032.JPG	2DSCN0032.JPG
DSCN0001.JPG	4DSCN0001.JPG
DSCN0055.JPG	6DSCN0055.JPG
DSCN0029.JPG	8DSCN0029.JPG

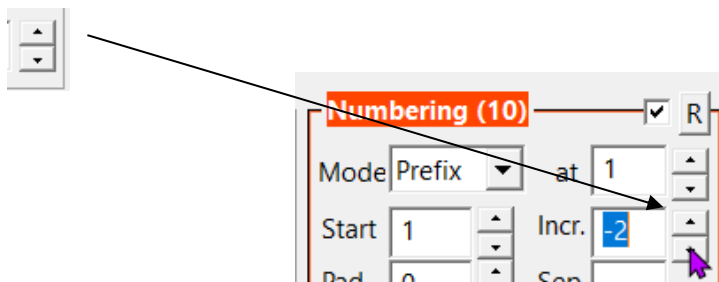
Alpha

Name ▲	New Name
DSCN0032.JPG	B DSCN0032.JPG
DSCN0001.JPG	C DSCN0001.JPG
DSCN0055.JPG	D DSCN0055.JPG
DSCN0029.JPG	E DSCN0029.JPG

Name ▲	New Name
DSCN0032.JPG	B DSCN0032.JPG
DSCN0001.JPG	D DSCN0001.JPG
DSCN0055.JPG	F DSCN0055.JPG
DSCN0029.JPG	H DSCN0029.JPG

Note:

1. A Negative number entered into the Increment field will display from the Start value down to zero, then negative values thereafter. Negative numbers can be entered directly into the Insert 'At' or Start 'Incr.' data entry fields or you can use the Up/Down indicators:



Section # 10: Numbering

Pad

Pad

Specify how many places the numbering sequence should occupy. Any unused places are ‘padded’ from the left with a ‘0’ (zero), ‘a’ (lowercase), or ‘A’ (uppercase) character, as appropriate dependent on ‘Type’ selected (see ‘Type’).

- 0 e.g. 00001 (padded with zeroes)
- a e.g. aaaab (padded with lowercase a’s)
- A e.g. AAAB (padded with uppercase A’s)

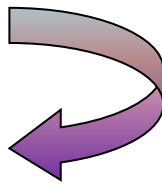
The positions occupied by the sequence, including any padding, are counted from **right to left**:

e.g.,

Padding is set a 3

Sequence: 0 0 1

Position: 3 2 1



Example:

Numbering (10) R

Mode Prefix at 1

Start 1 Incr. 1

Pad 3 Sep.

Numeric

Name	New Name
DSCN0032.JPG	001DSCN0032.JPG
DSCN0001.JPG	002DSCN0001.JPG
DSCN0055.JPG	003DSCN0055.JPG
DSCN0029.JPG	004DSCN0029.JPG

Alpha

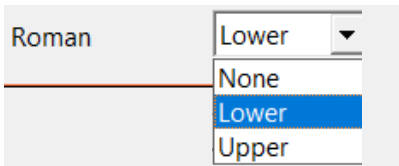
Name	New Name
DSCN0032.JPG	AAADSCN0032.JPG
DSCN0001.JPG	AABDSCN0001.JPG
DSCN0055.JPG	AACDSCN0055.JPG
DSCN0029.JPG	AADDSCN0029.JPG

Section # 10: Numbering

Roman Numerals



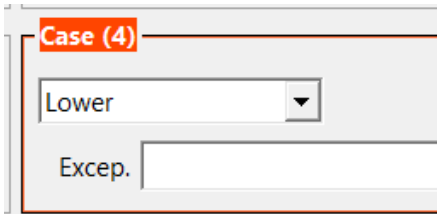
This has nothing to do with a numbering sequence, but since it is related to controlling numbers, it appears here. If Roman Numerals exist in the current filename, the option is given to do nothing or convert them to upper or lower case. This is done to preserve them when a rename action will change the overall case of the filename.



Example:

String = Mark IV 204.56 Alphoze.jpg

‘Section #4: Case’



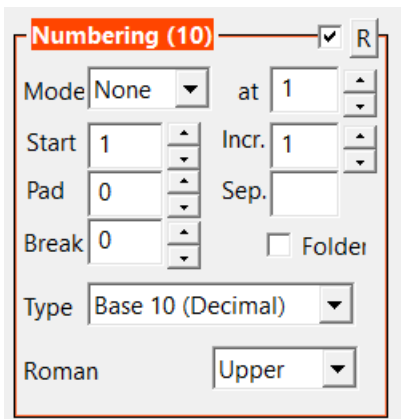
Set to Lower

New Name without ‘Roman’ enabled:

Name ▲	New Name
Mark IV 204.56 Alphoze.jpg	mark iv 204.56 alphoze.jpg

The result is that the Roman Numeral IV has been altered.

Using ‘Section #10: Numbering’ with ‘Roman’ enabled



Roman: Upper

Name ▲	New Name
Mark IV 204.56 Alphoze.jpg	mark IV 204.56 alphoze.jpg

The result is that the Roman Numeral IV is unaltered.

Notes:

1. The other settings of Numbering, Mode, Start, Pad, etc. have nothing to do with the ‘Roman Numeral’ function.
2. **BRU v3.42 or higher, disregard this section and refer to v3.43 New Additions supplement that follows..**

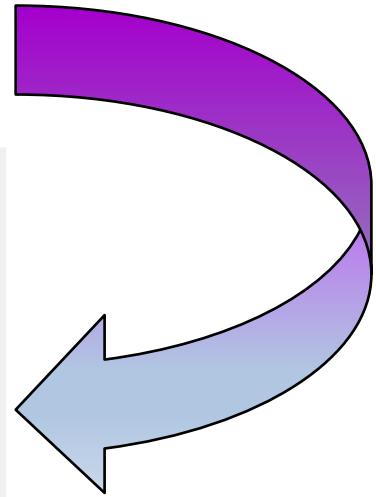
Section # 10: Numbering

v3.43 New Additions

Roman Numerals section removed and placed under 'Type' section.

Before:

Current:



All previous functionality pertaining to Roman Numerals has been modified. Case conversion is no longer performed.

The function of Case Conversion of existing Roman Numerals in the string has been moved to the more appropriate, 'Section #4: Case'. This is how to use Roman Numerals under 'Section #4: Case':

Using the previous example, slightly modified:

Example #1:

String = mark IV 204.56 alphoze.jpg

Name ▲	New Name
mark IV 204.56 alphoze.jpg	Mark IV 204.56 Alphoze.JPG

The Roman Numeral is preserved.

Section # 10: Numberingv3.43 New Additions

Example #2:

String = mark iv 204.56 alphoze.jpg

Name ▲	New Name
mark iv 204.56 alphoze.jpg	Mark IV 204.56 Alphoze.Jpg

Case conversion is also performed when necessary under ‘[Section #4: Case](#)’.

Instead, the Roman Numeral functionality under ‘[Section #10: Numbering](#)’, now performs in the same manner as the other Base types; to append a prefix or suffix **to an existing string** or to insert a sequence **within an existing string**.

Example #3:

String = My Cat has milk to drink

Mode: Suffix
 Start: 4
 Type: Roman Numerals
 Case: Upper

Name ▲	New Name
My Cat has milk to drink	My Cat has milk to drink IV

The Roman Numeral, IV (representing Start = 4), presented in uppercase (Case set to Upper), suffixed to the string.

Section # 10: Numberingv3.43 New Additions

Example #4:

Samples:

My Cat has milk to drink
 mark iv 204.56 alphoze.jpg
 'abc'x
 700 8 is next
 a, and, as, at, but, by, en, for, if, in, of, on

Mode: Suffix
 Start: 4
 Increment: 1
 Type: Roman Numerals
 Case: Upper

Adding more samples will append the incrementing value to each selected string while ignoring any strings that are not selected.

This prevents the string,

e.g., 'abc'x

from having a value of 'VI' instead of 'V' because the unselected string, 'mark iv 204.56 alphoze.jpg', is not included..

Results:

Name ▲	New Name
My Cat has milk to drink	My Cat has milk to drink IV
mark iv 204.56 alphoze.jpg	mark iv 204.56 alphoze.jpg
'abc'x	'abc'x V
700 8 is next	700 8 is next VI
a, and, as, at, but, by, en, for, if, in, of, on	a, and, as, at, but, by, en, for, if, in, of, on VII

Section # 10: Numbering

v3.43 New Additions

Case

The Case item is also a new addition. In previous versions, it was reserved for the Roman Numeral Case conversion. Now it can be used for both the appended Roman Numerals and the A-Z, a-z Alpha Base Types as well, although, it doesn't really do that much when applied to the Alpha Base Types. Not very useful and probably not even intended for that purpose.

Example:

Samples:

My Cat has milk to drink
 mark iv 204.56 alphoze.jpg
 'abc'x
 700 8 is next
 a, and, as, at, but, by, en, for, if, in, of, on

Mode: Suffix
 Start: 1
 Increment: 1
 Type: a-z
 Case: None (same – do not change)

Results:

Name	New Name
My Cat has milk to drink	My Cat has milk to drink a
mark iv 204.56 alphoze.jpg	mark iv 204.56 alphoze.jpg
'abc'x	'abc'x b
700 8 is next	700 8 is next c
a, and, as, at, but, by, en, for, if, in, of, on	a, and, as, at, but, by, en, for, if, in, of, on d

The Alpha Base Type **a-z or A-Z** is what determines initially if the letters that are appended are to be in lowercase or uppercase. I say initially because if you change the CASE to either lowercase or uppercase, this will supplant the Type specification. In the example below, although the Type a-z has been specified, the appended value is in uppercase because of the Case specification of Upper.

e.g.,

Results:

Name	New Name
My Cat has milk to drink	My Cat has milk to drink A

Section # 10: Numbering

A Quick Review before continuing ...

‘Section #10: Numbering’ – is a facility where an Autonumber can be appended to the Prefix, Suffix, both the Prefix and Suffix, or inserted at an **exact** position **within** the string. **It cannot be inserted at a variable position.**

Example:

Numbering not active. Mode is at ‘None’. Filename unaffected.

Name	New Name
Name cat Name abc.jpg	Name cat Name abc.jpg

Numbering Active. Mode is at ‘Prefix’.
Start number at 1 and Auto increment at 1.

Name	New Name
Name cat Name abc.jpg	1Name cat Name abc.jpg

Numbering Active. Mode is at ‘Suffix’.
Start number at 1 and auto increment at 1

Name	New Name
Name cat Name abc.jpg	Name cat Name abc1.jpg

Numbering Active. Mode is at ‘Prefix and Suffix’.
Start number at 1 and auto increment at 1.

Name	New Name
Name cat Name abc.jpg	1Name cat Name abc1.jpg

Numbering Active. Mode is at ‘Insert’. Position is at 1.

Name	New Name
Name cat Name abc.jpg	N1ame cat Name abc.jpg

- Na1me cat Name abc.jpg Position is 2
- Nam1e cat Name abc.jpg Position is 3
- Name1 cat Name abc.jpg Position is 4
- Name1cat Name abc.jpg Position is 5
- Name c1at Name abc.jpg Position is 6

Note:
The ‘N’ in the beginning of the string is considered position ‘0’, or the ‘Prefix’ position. This is why the position of the Autonumber at **position 1** begins with the position of the ‘a’ character in the text and not at the ‘N’ character. Most positions in the BRU sections other than, ‘Section #1: RegEx’, begin at **Position 1**.

Etc.

Section # 10: Numbering

Sep(arator)

Sep.

Specify a separator character(s) that will be used to distinguish the numbering sequence from the filename.

Using the Sep(arator) –

This option provides a character(s) as a delimiter for the Autonumber’s placement in the string. For example if a <space> character is used in Sep. with Mode at ‘Prefix’:

Numbering (10) R

Mode Prefix at 0

Start 1 Incr. 1

Pad 0 Sep.

goes from this...

1Name cat Name abc.jpg

... to this.

1 Name cat Name abc.jpg

More than one character can be entered here as well:

Numbering (10) R

Mode Prefix at 0

Start 1 Incr. 1

Pad 0 Sep. -

Sep: <space> <hyphen> <space>

Resulting in -

1 - Name cat Name abc.jpg

Section # 10: Numbering

Using The Special Character ‘ : ’ Colon

There is also a special function that uses the ‘ : ’ (colon) to act as a ‘placeholder’ for the Autonumber.

If you enter the special character ":" (colon), this will be replaced with the Autonumber at runtime. So a Sep. value of ABC:DEF: would result in ABC1DEF1, ABC2ABC2 etc.

Illustrated here:

Sep: ABC:DEF:
Mode: Prefix
Inc.: 1

clarified to ABC : DEF :

results in –

Name	New Name
Name-some text.jpg	ABC1DEF1 Name-some text.jpg
Name cat Name abc.jpg	ABC2DEF2 Name cat Name abc.jpg

I said before that the Autonumber cannot be placed at a variable position within the string. Some people get confused in the purpose of the colon. They presume, that by placing the colon anywhere in the filename itself, the Autonumber would appear at **that** position in the New Name. It does not.

Here is the confusion –

i.e.,

This is a test : some text here.jpg

This is another test : some more text and a number : here.jpg

some might assume would result in:

This is a test 1 some text here.jpg

This is another test 2 some more text and a number 3 here.jpg

This would be great except **that is not how this works**, besides, Windows does not recognize the use of the colon in filenames; it is considered an illegal character. In addition, the Autonumber increment would only increment **per filename not within the filename** itself. ⁽¹⁾

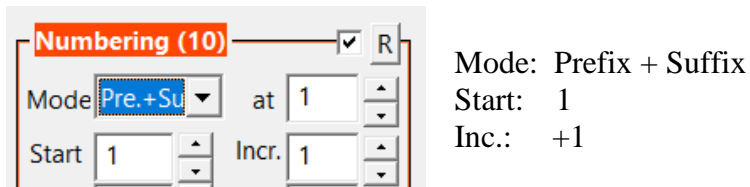
Section # 10: Numbering

This is how it does work –

Using Autonumber in it's correct usage:

The following examples demonstrate how the Autonumber increments on a per filename basis.

Example 1 -



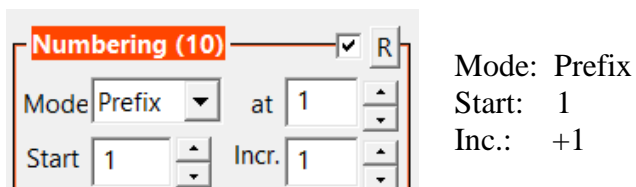
results in:

Name ▲	New Name
test.jpg	1test1.jpg

You can clearly see that the same value '1' is applied to both the prefix and suffix and does not result in, '1 test 2.jpg'.

Example 2 –

Applying to multiple files:



results in:

Name ▲	New Name
test file a.jpg	1test file a.jpg
test file b.jpg	2test file b.jpg
test file c.jpg	3test file C.jpg



results in:

Name ▲	New Name
test file a.jpg	1test file a1.jpg
test file b.jpg	2test file b2.jpg
test file c.jpg	3test file c3.jpg

It is each filename that gets incremented.

Section # 10: Numbering

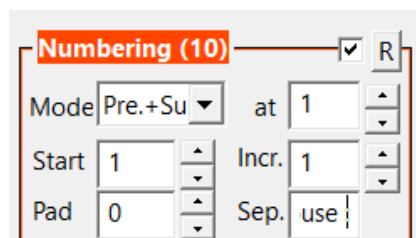
The use of the colon is to be able to specify *WHERE* in the **Sep. delimiter string**, you want the Autonumber to appear *NOT WHERE* in the **filename string**.

Example:

Sample dataset –

Filename a (some text here).jpg

Filename b (some text here).jpg



Mode: Prefix + Suffix

Start: 1

Inc.: +1

Sep.: Cat : Mouse : House :

results in:

Name	New Name
Filename a (some text here).jpg	Cat 1 Mouse 1 House 1 filename a (some text here)Cat 1 Mouse 1 House 1.jpg
Filename b (some text here).jpg	Cat 2 Mouse 2 House 2 filename b (some text here)Cat 2 Mouse 2 House 2.jpg

original filename is not affected by colon

For each colon in each filename, the Autonumber is applied to the position indicated **within** the **Sep. value**

(1) There is, however, a technique to do this that can be found in 'Volume 2 of the Bulk Rename Utility Operations Manual – Expert's Corner', under 'Add' in the 'RegEx' section.

Also refer to –

'Add Autonumber in Middle of String' under the 'Add' section of RegEx, in Volume 2 of Bulk Rename Utility Operations Manual: Expert's Corner', which explains this further.

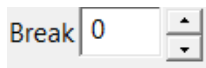
Section # 10: Numbering

Resetting an (Auto) Numbering back to start value

If the start value is 1, the filenames will be appended with 1, 2, 3, 4, etc., incrementing each time. However, if you want the value to start back at '1' ...

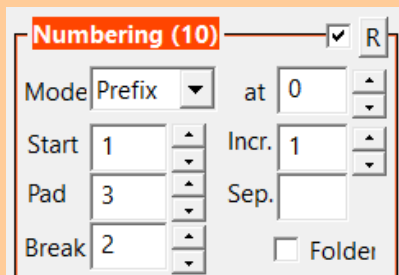
2 methods by which this can happen:

Break



Use Break to reset the 'counter' when the specified character position changes.

Example,



Mode: Prefix at 0 position
Start: 1 at increment +1
Sep: <space>

Pad is set a 3 positions
Break is set at (position) 2

Results in:

Name ▲	New Name
DSCN0001.JPG	001 DSCN0001.JPG
DSCN0002.JPG	002 DSCN0002.JPG
DSCN0014.JPG	003 DSCN0014.JPG
DSCN0015.JPG	004 DSCN0015.JPG
DSCN0016.JPG	005 DSCN0016.JPG
DSCN0017.JPG	006 DSCN0017.JPG
DSCN0018.JPG	007 DSCN0018.JPG
DSCN0019.JPG	008 DSCN0019.JPG
DSCN0025.JPG	009 DSCN0025.JPG
DSCN0026.JPG	001 DSCN0026.JPG

When the second position changes, the counter resets back to 001.



e.g., 009 (DSCN0025.JPG) to 010 (DSCN026.JPG), 'DSCN026' becomes '001 DSCN026'
10 will change the second position occupied by zero to one, so counter resets to '001'

Section # 10: Numbering

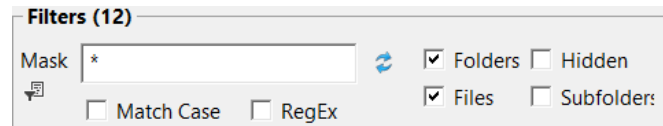
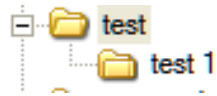
Second method:

Folder



This will reset the counter for each new sub directory. This also requires that 'Section #12: Filters' have 'Subfolders' option enabled (recursion), otherwise you will see no difference.

Here are two directories, Test and a subdirectory, Test 1



Test contains the following files:

Name
Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg
SAM_0030.JPG
SAM_0035.JPG

Illustration #1

Test 1 subdirectory contains the following files:

Name
Becca BW.jpg
Becca Colour QT BW.jpg
Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg
Becca QT Distorted.jpg
Becca QT FI Chalk.jpg

Illustration #2

In the Navigation Pane click on the Test directory, and it will display illustration #1 again. Now click on the 'Subfolders' option in 'Section #12: Filters' this turns on 'Recursion':

Name
Becca BW.jpg
Becca Colour QT BW.jpg
Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg
Becca QT Distorted.jpg
Becca QT FI Chalk.jpg
Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg
SAM_0030.JPG
SAM_0035.JPG



The window displays all of the files in both directories.

← Illustration #3

Section # 10: Numbering

Example:

Using the directories from the previous page,

Numbering (10) R

Mode: Prefix at 0

Start: 1 Incr.: 1

Pad: 3 Sep.:

Break: 0 Folder

Mode is set a Prefix
 Start is set at 1 with an increment of 1
 Pad is set at 3
 Sep.(arator) is <space>
 Folder is enabled

Results in:

Name	New Name
Becca BW.jpg	001 Becca BW.jpg
Becca Colour QT BW.jpg	002 Becca Colour QT BW.jpg
Becca Moving Sun Faded.jpg	003 Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg	004 Becca Purple Vignette.jpg
Becca QT Distorted.jpg	005 Becca QT Distorted.jpg
Becca QT FI Chalk.jpg	006 Becca QT FI Chalk.jpg
Belvedere Plantation 6474.jpg	001 Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg	002 Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg	003 Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg	004 Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg	005 test5.jpg
Belvedere Plantation 6492.jpg	006 Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	007 Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	008 Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	009 Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	010 Belvedere Plantation 6512.jpg
SAM_0030.JPG	011 SAM_0030.JPG
SAM_0035.JPG	012 SAM_0035.JPG

Test 1

Test

Notice that the files in the Test directory are renamed using a sequence of 001 through 012 and the files in the Test 1 subdirectory begin at 001 through 006. This is because the counter resets for each new directory that is processed. If the Folder option was not enabled, the files would be renamed using a sequence of 001 through 018 and the counter would not reset.

Section # 10: Numbering

Using Break and Folder options together

Example:

Using the same example, but this time:

Numbering (10) R

Mode: Prefix at 0

Start: 1 Incr.: 1

Pad: 3 Sep.:

Break: 2 Folder

Break is set at (position) 2 and Folder is enabled

Results in:

Name	New Name
Becca BW.jpg	001 Becca BW.jpg
Becca Colour QT BW.jpg	002 Becca Colour QT BW.jpg
Becca Moving Sun Faded.jpg	003 Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg	004 Becca Purple Vignette.jpg
Becca QT Distorted.jpg	005 Becca QT Distorted.jpg
Becca QT FI Chalk.jpg	006 Becca QT FI Chalk.jpg
Belvedere Plantation 6474.jpg	001 Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg	002 Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg	003 Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg	004 Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg	005 test5.jpg
Belvedere Plantation 6492.jpg	006 Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	007 Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	008 Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	009 Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	001 Belvedere Plantation 6512.jpg
SAM_0030.JPG	002 SAM_0030.JPG
SAM_0035.JPG	003 SAM_0035.JPG

Test 1

Test

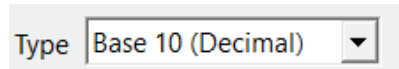
The files are renamed in the Test directory using a sequence of 001 through 009. Sequence number 010 resets the counter back to 001, so the remaining files in Test return back to 001 and continue to 003 (for a total of 13 files). The Test 1 directory only contains 6 files, so the files are renamed using a sequence of 001 through 006.

Notes:

1. For more information, see '[Section #12: Filters](#)'.

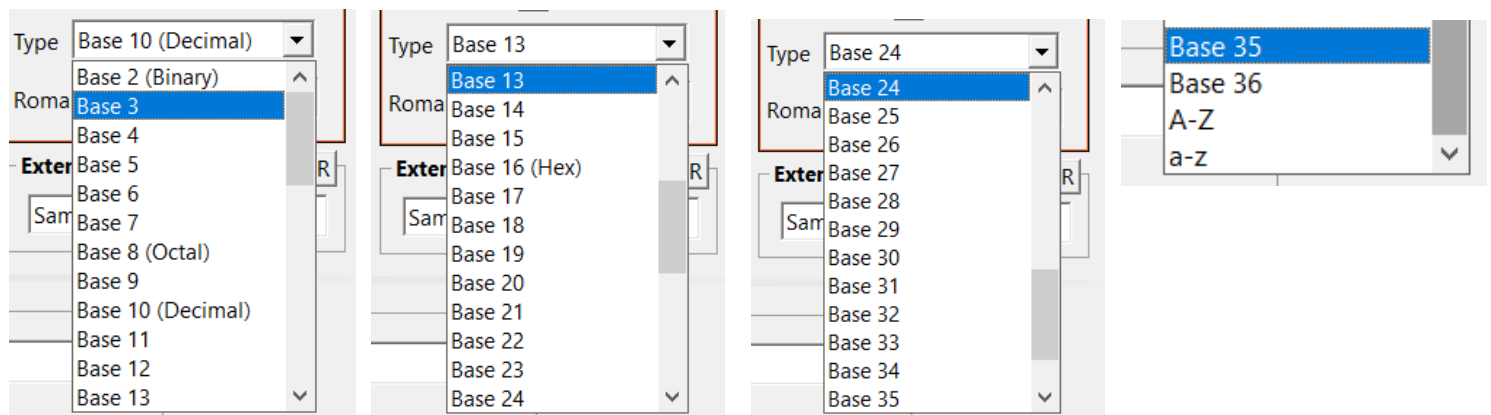
Section # 10: Numbering

Type



Type is used to indicate what type of numbering system or Base to use for the sequence.

These are the available Numeric Bases along with two Alpha Bases, A-Z (uppercase) and a-z (lowercase):



Notes:

(1) The **base** is what you count up to before it starts over. This corresponds to the digit referenced in the base name.

e.g.,

Base 10, Base 8, etc.

(2) Zero is always counted as part of the base. In a Base 6 system for instance, the base is –
0 1 2 3 4 5.

For example, the numeric digits, 0-9 make up the base in the Base 10 (decimal) system –

0 1 2 3 4 5 6 7 8 (9) (10)

The digit '9' marks the end of the base before it changes over to '10'. The '10' is the equivalent of the **zero** digit again – 1-0 and starts the base over until it reaches '19' where '20' – 2-0 'will continue the base, etc.

For this reason, I refer to the numeric digit number '10', as the changeover point. The changeover value in a numeric base system, is always the numeric digit '10' initially.

Section # 10: Numbering

Notes: cont.

- (3) In numeric bases past base 10, **letters** are used to fill up places past the numeric digit 9 to the end of the base before changing over to 10.

Examples,

Base 11 (Undenary)

0 1 2 3 4 5 6 7 8 9 A 10

Base 12 (Duodecimal)

0 1 2 3 4 5 6 7 8 9 A B 10

Base 13 (Tridecimal)

0 1 2 3 4 5 6 7 8 9 A B C 10

you get the idea.

On the following page is a chart that I created that illustrates each available Base:

1. The numeric digits leading up to the first changeover point are in **BOLD black**. These are the numeric digit values that make up the Base of each system.
2. The first changeover value is in **BOLD red**. This value is always '10' in a numeric base system and represents the changeover point where the base starts over the numeric sequence. The next changeover value in the sequence would be '20', then '30' and so forth. In binary, the second changeover value would be 100, then 110 etc.
3. The digits after the initial changeover value are in **yellow**.
4. I have also included the name of each Base System.

Section # 10: Numbering

Base	Name	Example
2	Binary	0 1 10 11 100 101 110 111 1000 1001 1010 1011 1100 ..
3	Ternary	0 1 2 10 11 12 20 21 22 100 101 102 110 111 112 120..
4	Quaternary	0 1 2 3 10 11 12 13 20 21 22 23 30 31 32 33 100 101 102 103 110 ..
5	Quinary	0 1 2 3 4 10 11 12 13 14 20 21 22 23 24 30 31 32 33 34 40 41 42 43 44 100 ..
6	Senary	0 1 2 3 4 5 10 11 12 13 14 15 20 21 22 23 24 25 30 31 32 33 34 35 40 41 42 43 44 45 50 ..
7	Septenary	0 1 2 3 4 5 6 10 11 12 13 14 15 16 20 21 22 23 24 25 26 30 31 32 33 34 35 36 40 41 42 43 44 45 46 50 ..
8	Octal	0 1 2 3 4 5 6 7 10 11 12 13 14 15 16 17 20 21 22 23 24 25 26 27 30 31 32 33 34 35 36 37 40 ..50 ..
9	Nonary	0 1 2 3 4 5 6 7 8 10 11 12 13 14 15 16 17 18 20 21 22 23 24 25 26 27 28 30 31 32 33 34 35 36 .. 40 .. 50
10	Decimal	0 1 2 3 4 5 6 7 8 9 10 .. 19 20 21 22 23 24 25 26 ..
11	Undenary	0 1 2 3 4 5 6 7 8 9 A 10 .. 19 1A 20 .. 29 2A ..
12	Duodecimal	0 1 2 3 4 5 6 7 8 9 A B 10 .. 19 1A 1B 20 .. 29 2A ..
13	Tridecimal	0 1 2 3 4 5 6 7 8 9 A B C 10 .. 19 1A 1B 1C 20 .. 29 2A ..
14	Tetradecimal	0 1 2 3 4 5 6 7 8 9 A B C D 10 .. 19 1A 1B 1C 1D 20 .. 29 2A ..
15	Pentadecimal	0 1 2 3 4 5 6 7 8 9 A B C D E 10 .. 19 1A 1B 1C 1D 1E 20 .. 29 2A ..
16	Hexadecimal	0 1 2 3 4 5 6 7 8 9 A B C D E F 10 .. 19 1A 1B 1C 1D 1E 1F 20 .. 29 2A ..
17	Heptadecimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G 10 .. 19 1A 1B 1C 1D 1E 1F 1G 20 .. 29 2A ..
18	Octodecimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H 10 .. 19 1A 1B 1C 1D 1E 1F 1G 1H 20 .. 29 2A ..
19	Enneadecimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I 10 .. 19 1A 1B 1C 1D 1E 1F 1G 1H 1I 20 .. 29 2A ..
20	Vigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J 10 .. 19 1A 1B 1C 1D 1E 1F 1G 1H 1I 1J 20 .. 29 2A ..
21	Unvigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K 10 .. 19 1A 1B 1C 1D 1E 1F 1G 1H 1I 1J 1K 20 .. 29 2A ..
22	Duovigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L 10 .. 19 1A 1B 1C 1D 1E 1F 1G 1H 1I 1J 1K .. 20 .. 29 2A
23	Trivigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M 10 .. 19 1A 1B 1C 1D 1E 1F 1G 1H 1I 1J 1K 1L 1M 20..
24	Tetravigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N 10 .. 19 1A 1B .. 1N 20..
25	Pentavigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O 10 .. 19 1A 1B .. 1O 20..
26	Hexavigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P 10 .. 19 1A 1B .. 1P 20..
27	Septemvigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q 10 .. 19 1A 1B.. 1Q 20..
28	Octovigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R 10 .. 19 1A 1B.. 1R 20..
29	Enneavigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S 10 .. 19 1A 1B .. 1S 20..
30	Trigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T 10 .. 19 1A 1B .. 1T 20..
31	Untrigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U 10 .. 19 1A 1B .. 1U 20..
32	Duotrigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V 10 .. 19 1A 1B .. 1V 20..
33	Tritrigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W 10 .. 19 1A 1B .. 1W 20..
34	Tetratrigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X 10 .. 19 1A 1B .. 1X 20..
35	Pentatrigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y 10 .. 19 1A 1B .. 1Y 20..
36	Hexatrigesimal	0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 10 .. 19 1A 1B .. 1Z 20..

1. The base of each numeric system is displayed in **BOLD black**. The **yellow** are samples of following sequences.
2. The **changeover point** is where each base begins the sequence over.
3. The first changeover value in a numeric based system is always '10'. In the chart it can be observed in **BOLD red**.
4. There are two Alpha options that are representative of uppercase and lowercase letters and not part of any base:

A-Z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z **AA** AB AC..BA BB BC .. CA CB ..

a-z a b c d e f g h i j k l m n o p q r s t u v w x y z **aa** ab ac.. ba bb bc .. ca cb cc .. da db dc .. ea eb ec ..

Example of using Alpha option, A-Z, (with Pad set to 3 places):

```
AAA - 2956214110111744859-mp3.mp3
AAB - 3045862950031069794-mp3.mp3
AAC Aidan Oliver; Emma Kirkby; The King's Consort; Mich...
AAD Alvaro Cassuto- Sinfonia in B flat I- Allegro.mp3
```

Section # 10: Numbering**Known Issues:****Deselecting a File with a Non-Consecutive Selection Also Deselects the Previous File that held Focus**

1. Remember that selections can either be Consecutive or Non-Consecutive.

a. Consecutive selections refer to selections of files that are done in sequence – top to bottom, bottom to top and selecting all files in between. This is done using the Shift key + Mouse or the Shift + Down/Up arrow keys.

b. In addition, all files can be selected using Ctrl + A.

i.e.,

Name	New Name
<input type="checkbox"/> 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	0001 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
<input type="checkbox"/> 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6	0002 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6
<input type="checkbox"/> UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	0003 UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6
<input type="checkbox"/> FILENAME [1998] (text)	0004 FILENAME [1998] (text)
<input type="checkbox"/> FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]	0005 FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]
<input type="checkbox"/> FILENAME (1994)[text]	0006 FILENAME (1994)[text]

c. Non-Consecutive selections refer to selections that do not follow in sequence. This is done using the Ctrl key + Mouse.

i.e.,

Name	New Name
<input type="checkbox"/> 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
<input type="checkbox"/> 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6	0001 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6
<input type="checkbox"/> UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	0002 UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6
<input type="checkbox"/> FILENAME [1998] (text)	FILENAME [1998] (text)
<input type="checkbox"/> FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]	0003 FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]
<input type="checkbox"/> FILENAME (1994)[text]	0004 FILENAME (1994)[text]
<input type="checkbox"/> CAN-FirstLast.docx	CAN-FirstLast.docx
<input type="checkbox"/> CAN-First Last.docx	0005 CAN-First Last.docx

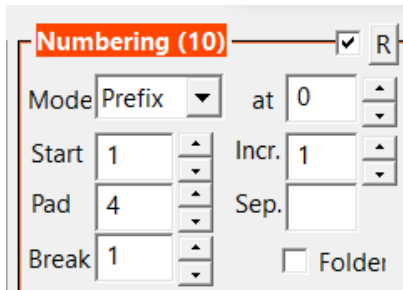
2. The issue is that the last file in focus, meaning the last file that was selected individually, and this does not matter if the selection was done consecutively or non-consecutively, will be deselected when another file is deselected non-consecutively using Ctrl key + Mouse.

Section # 10: Numbering

Known Issues:

To Reproduce the problem:

1. Activate Numbering and give it some settings.



2. Select a file – it does not have to be the first file in the Content Pane, but let's use that. The first file now has focus.

Name	New Name
14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	0001 14692-2A-010-A-24-0-T2-62-14692-2A-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-T6-109-14692-2A-1100-A-32-5 T6
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6

3. Select additional files consecutively or non-consecutively – it doesn't matter. For clarity, this example will use consecutively:

Name	New Name
14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	0001 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6	0002 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	0003 UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6
FILENAME [1998] (text)	0004 FILENAME [1998] (text)
FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]	0005 FILENAME (text - a to z) (1974)(text - a to z)[text -a to z]
FILENAME (1994)[text]	0006 FILENAME (1994)[text]
CAN-FirstLast.docx	0007 CAN-FirstLast.docx

- a. The last file to hold focus is the last selected file – 'CAN-FirstLast.docx'

Section # 10: Numbering

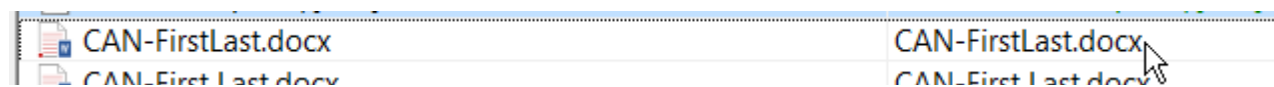
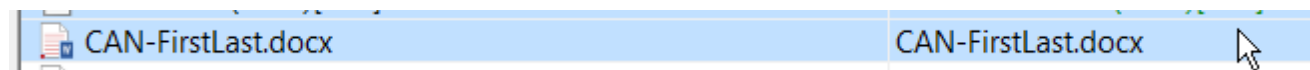
Known Issues:

4. Using Ctrl + Mouse, deselect a file, - again, it doesn't matter which. I will deselect, 'FILENAME [1998] (text)'

Name	New Name
14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	0001 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6	0002 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	0003 UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6
FILENAME [1998] (text)	FILENAME [1998] (text)
FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]	0005 FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]
FILENAME (1994)[text]	0006 FILENAME (1994)[text]
CAN-FirstLast.docx	CAN-FirstLast.docx

5. Now you can see that not only did the file that I deselected become deselected, but the last file that held focus was also affected. It is selected but no longer under the criteria's evaluation.

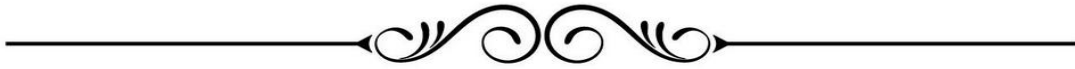
6. Work-around, **re-select the file** using Ctrl + Mouse (twice) – once to remove the selection and then again to select.



Name	New Name
14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2	0001 14692-2A-010-A-24-0-T2-62-14692-2A-010-A-24-0 T2
14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6	0002 14692-2A-1100-A-32-5-T6-109-14692-2A-1100-A-32-5 T6
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	0003 UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6
FILENAME [1998] (text)	FILENAME [1998] (text)
FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]	0004 FILENAME (text - a to z) (1974)(text - a to z)[text - a to z]
FILENAME (1994)[text]	0005 FILENAME (1994)[text]
CAN-FirstLast.docx	0006 CAN-FirstLast.docx

Update:

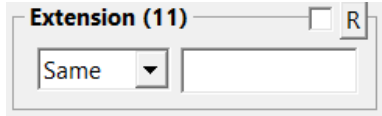
TGRMN has addressed this issue to a satisfactory conclusion. I leave the original text of the problem to help those who for whatever reason have not or will not update to the newest version.



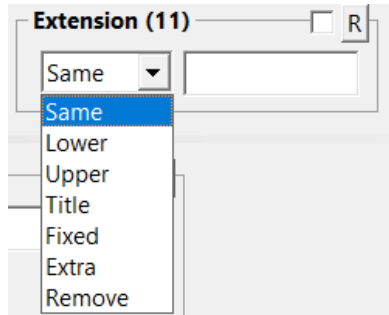
Section # 11 - Extension

Section # 11: Extension

Extension – you can elect to change the filename extension during the rename action.



Your options are:



Same – no changes are made.

Lower/ Upper/Title – change the Case

Lower –

Name ▲	New Name
test file a.JPG	test file a.jpg

Upper –

Name ▲	New Name
test file a.jpg	test file a.JPG

Title –

Name ▲	New Name
test file a.jpg	test file a.Jpg

Fixed – specify an extension to use for all selected files. This is entered in the data entry field to the right. This replaces any previous extension.



Name ▲	New Name
test file a.JPG	test file a.Tim

Remove – remove the current file extension and replace it with nothing

Name ▲	New Name
test file a.JPG	test file a

Section # 11: Extension

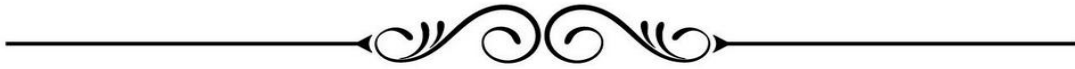
Extra – add a secondary extension. Windows supports use of the dot character in more than one place within the filename, so files could be renamed with a secondary dot extension,

If you had a bunch of files with no windows identifiable extension, for instance, a previous rename operation may have gone awry, you could add the proper extension.



Name ▲	New Name
<input type="checkbox"/> Becca BW.music	becca bw.music.jpg
<input type="checkbox"/> Becca Colour QT BW.music	becca colour qt bw.music.jpg
<input type="checkbox"/> Becca Moving Sun Faded.mu...	becca moving sun faded.music.jpg
<input type="checkbox"/> Becca Purple Vignette.music	becca purple vignette.music.jpg
<input type="checkbox"/> Becca QT Distorted.music	becca qt distorted.music.jpg
<input type="checkbox"/> Becca QT FI Chalk.music	becca qt fl chalk.music.jpg

Using the 'Fixed' option would not work because it would remove '.music' when all that is needed is to add an extension that could identify the file type.



Section # 12 - Filters

Section # 12: Filters

Filters – Specify a filespec (filename, with or without extension, aka file mask) that determines what is displayed in the Content Pane. This has no direct effect on which files or folders are actually selected for processing other than to make the selection easier by displaying only those files that match against the filter.

Filters (12)

Mask Folders Hidden Name Len Max
 Match Case RegEx Files Subfolders Path Len Min Max
 Condition

Navigation Pane ↓

Content Pane ↓

Bulk Rename Utility		Attributes	Name ▲	New Name	Type
		A	Belvedere Plantation 6474.jpg	Belvedere Plantation 6474.jpg	JPG File
		A	Belvedere Plantation 6476.jpg	Belvedere Plantation 6476.jpg	JPG File
		A	Belvedere Plantation 6479.jpg	Belvedere Plantation 6479.jpg	JPG File
		A	Belvedere Plantation 6488.jpg	Belvedere Plantation 6488.jpg	JPG File
		A	Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg	JPG File
		A	Belvedere Plantation 6492.jpg	Belvedere Plantation 6492.jpg	JPG File
		A	Belvedere Plantation 6493.jpg	Belvedere Plantation 6493.jpg	JPG File
		A	Belvedere Plantation 6497.jpg	Belvedere Plantation 6497.jpg	JPG File
		A	Belvedere Plantation 6508.jpg	Belvedere Plantation 6508.jpg	JPG File
		A	Belvedere Plantation 6512.jpg	Belvedere Plantation 6512.jpg	JPG File
		A	SAM_0030.JPG	SAM_0030.JPG	JPG File
		A	SAM_0035.JPG	SAM_0035.JPG	JPG File

Mask – used to enter a filespec that is used to filter the display (limit what is seen) in the Content Pane. The default is: * - show all (or the more traditional filespec of *.*). This is also called a 'Wildcard' character, because the asterisk is used to represent something else. e.g., *.mp3, displays only Mp3 (audio) file types identified by the .mp3 extension.

Mask

Example,

Mask

displays:

Name ▲	New Name
Belvedere Plantation 6474.jpg	Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg	Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg	Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg	Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg	Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	Belvedere Plantation 6512.jpg

Section # 12: Filters

Notes:

1. '*' is the same as '*.*' but not '*.' (neither '*.' or '*.*' is valid in the Filters field)
2. Multiple filespecs can be entered using a <space> delimiter.
3. When entering or changing a filespec, the change won't take effect (doesn't refresh) unless you click on a different object or move to another field, Press F5 to refresh all or use the 'Filter Refresh' button.





4. A 'not' Boolean operator can be entered using an exclamation point. The operator is used as 'do Not include'.

e.g., *!*Sam* Display **only** files beginning with SAM (don't include All but do include *Sam)

Results in:











Test directory

Name ▲
 SAM_0030.JPG
 SAM_0035.JPG

e.g., * !Sam* Display all files **except** those beginning with SAM (include all but not Sam)

Results in:




Test directory

Name ▲
 Belvedere Plantation 6474.jpg
 Belvedere Plantation 6476.jpg
 Belvedere Plantation 6479.jpg
 Belvedere Plantation 6488.jpg
 Belvedere Plantation 6490.jpg
 Belvedere Plantation 6492.jpg
 Belvedere Plantation 6493.jpg
 Belvedere Plantation 6497.jpg
 Belvedere Plantation 6508.jpg
 Belvedere Plantation 6512.jpg

e.g., * !*QT* Display all files **except** those containing the string QT (include all but not *QT)

Results in:

Test 1 directory

Name ▲
 Becca BW.music
 Becca Moving Sun Faded.music
 Becca Purple Vignette.music

Specifying multiple **not** selections (using <space> delimiter):

e.g., "* !*.doc !*.mp3" This will select everything **except** (older type) Word documents and MP3 files (include all but not .doc or .mp3).

Section # 12: Filters

Include sub-section

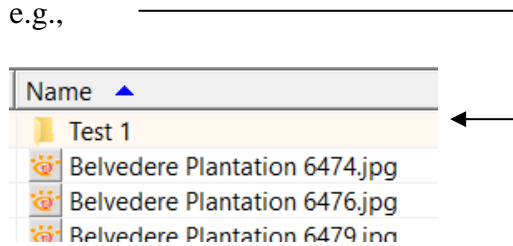
Folders Hidden
 Files Subfolder:

This indicates what items can be included in the display:

Files – all files in the selected directories (default)

Folders – include folders

e.g.,



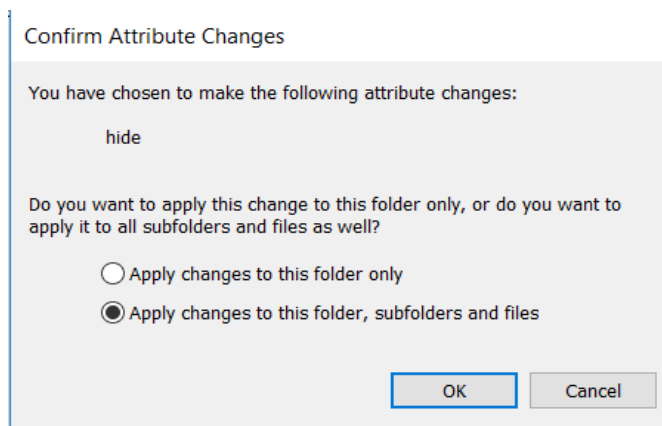
Hidden – include hidden files

Subfolders (Recursion)

Recursion. Sounds like stomach-upset and it very well could be if handled in the wrong manner. Recursion directs Windows to perform an action not only on the current directory and its files, but the inclusion of any subsequent sub-directories and their file contents.

Certain Windows functions can perform recursion.

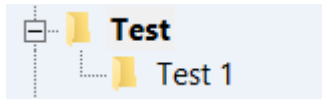
For example:



BRU can use recursion as well. When selecting the Subfolders option, BRU is directed to include the contents of the subdirectories. Any files in any subdirectories below the current directory will have all of its files displayed without having to click on each separate directory. This is called a '**Recursive Scan**'. It allows for the processing of any or all of these files in one operation.

Section # 12: Filters

Here are two directories, Test, and a subdirectory, Test 1



Test contains:

Name ▲
Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg
SAM_0030.JPG
SAM_0035.JPG

Test 1 subdirectory contains:

Name ▲
Becca BW.jpg
Becca Colour QT BW.jpg
Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg
Becca QT Distorted.jpg
Becca QT FI Chalk.jpg

Sub Folders option enabled:

Name ▲
Becca BW.jpg
Becca Colour QT BW.jpg
Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg
Becca QT Distorted.jpg
Becca QT FI Chalk.jpg
Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg
SAM_0030.JPG
SAM_0035.JPG

The Content Pane changes to display **all** the files in both directories.

Note:

1. A scan refers to *reading* the files in the directories that were selected using the Navigation Pane (the Tree Hierarchy). The files are then displayed in the Content Pane. BRU refers to the resulting display as the File List. Because, however, the File List can contain both files and folders, I use the more generic 'Content Pane' in this document to avoid confusion.
2. With recursion set, if you were to scan a high-level folder such as C:\ or C:\Program Files, it could take a long time processing potentially tens of thousands of files, so be aware of this. Slow network issues have also been reported.
 - a. If, after selecting a directory in the Navigation Pane, the program seems to freeze, look at the status bar:

Press ESC to cancel. Scanning folder I:\Go Thru 3...

This indicates the program is busy scanning. If you press 'Esc', the program will stop the scan (after a moment or two).

Scan aborted - list may be incomplete

3. Be careful using recursion and look for unintended consequences; e.g., You forgot and left 'Subfolders' enabled and now you have just mislabeled many files through a Rename action. Time to reach for that 'Undo' function.
4. This also works in conjunction with the 'Sub. Dir' column (see Display Options Menu for more information).



Section # 12: Filters**File Name Minimum Length/ Maximum Length**

Values entered here limit the files/folders displayed based on the (greater or equal to) **length of the filename (including dot character and extension)**.

Example, this filter limits the maximum file length to 13 characters.

when applied against the Test directory, results in:

e.g., SAM_0030.JPG = 12 characters including extension

Name ▲	
	SAM_0030.JPG
	SAM_0035.JPG

Path Minimum Length/ Maximum Length



Values entered here limit the files/folders displayed based on the (greater or equal to) **length of the pathname**. Unfortunately, BRU doesn't display the **full** path. You also can't rename by path length. I have discussed adding these to a future modification of BRU with TGRMN.

Example, this limits the path that includes the file length to a maximum of 154 characters.

when applied against this directory:

F:\Music\10,000 Maniacs with Natalie Merchant (up until 1994)\John and Mary (Ramsey - took over after Merchant)\Pinwheel Galaxy\

Results in only two selections that met the criteria:

Name ▲	
	John & Mary- Gaze.mp3
	John & Mary- The Drone.mp3

Notes:

1. File length includes ALL characters that make up the filename - the name, the dot character and the extension.

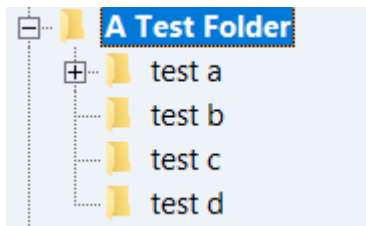
Section # 12: Filters

BRU's Sub Dir Column vs Full path

I don't like getting ahead of myself, but to avoid confusion, I feel it is important to bring up some topics now that will be discussed in a latter part of the book.

If you select Subdirs to turn on Recursion, as already mention in notation #4, there is a display column setting you can enable that will display both the current directory and the preceding directory levels above for each file displayed.

For example, If I have this directory structure:



If I enable the Sub Dir. column,

Name	New Name	Sub Dir.
Word1-Word2.com - 2009 r.ext	Word1-Word2.com - 2009 r.ext	\test d
Word1-Word2.com - 2009 r.ext	Word1-Word2.com - 2009 r.ext	\test c
Windows 7 Quick Reference Card.pdf	Windows 7 Quick Reference Card.pdf	\test a\adjusted
Welcome.doc	Welcome.doc	\test a\adjusted
UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	UK-JoLii1-JoLii2-JoLii3-JoLii4-JoLii5-JoLii6	
Receipt_php.txt	Receipt_php.txt	\test a\adjusted
Receipt_php.mht	Receipt_php.mht	\test a\adjusted
Receipt_php.htm	Receipt_php.htm	\test a\adjusted
nrefs ic	nrefs ic	\test a\adjusted

You can see that for each file, it provides the current directory and any directories preceding, but not the full path.

A true path would be, i.e.,

H:\A Test Folder\test a\adjusted\Windows 7 Quick Reference Card.pdf

In Sub Dir. column it displays as:

\test a\adjusted

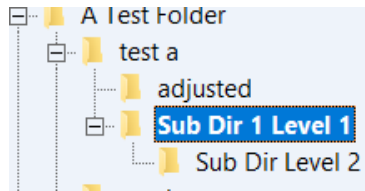
This indicates that the file, 'Windows 7 Quick Reference Card.pdf', is displayed as:

\test a
\adjusted
Windows 7 Quick Reference Card.pdf

Unlike a full path, it does not display the parent root, '**A Test Folder**', or the root directory of the drive, '**H:**'

Section # 12: Filters

It only works on files – for example, using an empty directory -



I have a directory, 'Sub Dir 1 Level 1' that is currently empty. It displays as:

Name	New Name ▼	Sub Dir.
Sub Dir Level 2	Sub Dir Level 2	

This is because the directory **is** empty (void of any files) and Sub Dir. is meant to display file paths not folder paths.

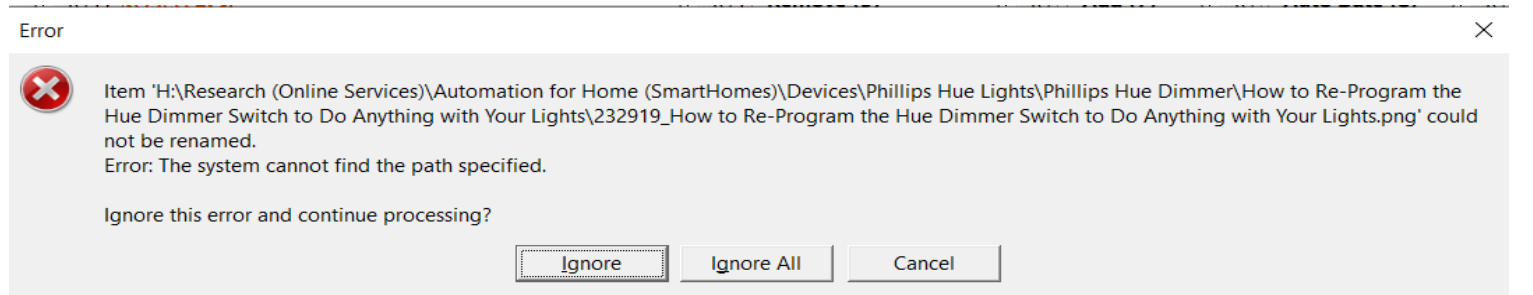
Long Path Support

e.g., This path exceeds the Windows limitation of 260 characters (The Max_Path limit of the WinAPI) -

H:\RESEARCH\Browsers\Browser (general or applies to all)\Printing\Prevent Firefox or Internet Explorer from Printing the URL on Every Page\Prevent Firefox or Internet Explorer from Printing the URL on Every Page -- the How-To Geek.png

Name	New Name ▼	Sub Dir.
Prevent Firefox or Internet Explorer from Printing t...	Browser (general or applies to all)PrintingPrevent Firefox or Internet Explorer from Printing the URL...	\Prevent Firefox or Internet Explor

BRU **Doesn't** Support long File Paths – e.g., attempting to rename a file with an existing path that exceeds **260** characters:

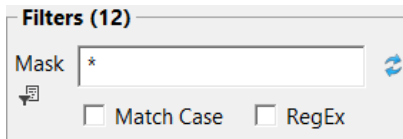


You can, however, filter the display in displaying only those files and file paths that meet a certain length, but when it comes to renaming using paths, without the ability to see the full path, you are at a slight disadvantage. I have reached out to TGRMN Software for adding a 'Full Path' column and supporting long file paths.

Notes:

1. For more information, refer to the discussion of the Sub Dir. Column setting.
2. 'Section #9: Append Folder Name' can append the names of subdirectory levels above and including the current directory. Although you can see these changes – the subdirectory names appear in New Name, this is still not a true full path display. It is also doubtful that the Rename operation will be permitted if the renaming length is exceeded.
3. The Sub Dir. column is useful for displaying paths associated with recursion. Must enable 'Subfolders' in 'Section #12: Filters' for data to be displayed in the Sub Dir. display column.

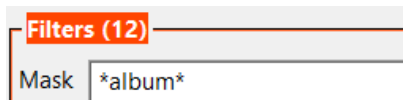
Section # 12: Filters



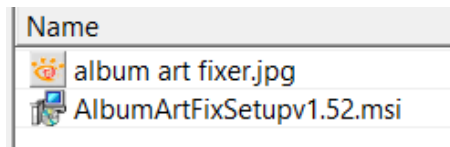
Match Case

Directs BRU to use **Case Sensitivity** when filtering out the file list. Any filtered data is normally Case Insensitive, meaning, any mix of upper and lowercase letters as part of folder or filenames are ignored. If this is set, then the filter will include both lower and uppercase letters in evaluating the filtering of the data.

Example:



Filter Case Insensitive



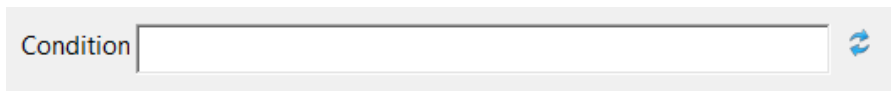
Filter Case Sensitive



RegEx

Directs BRU to interpret the entered 'Mask' as a **Regular Expression**. BRU allows RegEx to be used in evaluating a filter. For more information, see '[Section #1: RegEx](#)' and the 'Regular Expressions Manual' in both the Appendix in this volume and in Volume II. There is also a small section in Volume II dedicated to RegEx as it is used in Filters.

Condition



In this data entry field, you can enter a **JavaScript Condition** to filter files/folders. If the Condition evaluates to **true** for an object (filename or folder), that object is included in the file list, otherwise it is not. Using a JavaScript Condition you can **include/filter** files based on name, date, EXIF, size, attributes, length, etc.

For more information, refer to both the JavaScript section in the Appendix in this volume and you will find a dedicated section to JavaScript in Volume II.

Section # 12: Filters

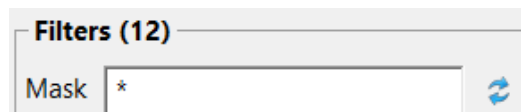
Notes:

1. The pathname includes all of the directories in the path as well as the length of the filename with dot and extension.

F:\Music\10,000 Maniacs with Natalie Merchant (up until 1994)\John and Mary (Ramsey - took over after Merchant)\Pinwheel Galaxy\John & Mary- The Drone.mp3 = **154 characters**

2. The files still have to meet the path and or filename limits to display, but in a Recursive Scan, (Subfolders option is enabled) subdirectories and their files will still be scanned (reads all files) regardless of length. This could slow things down if there are a lot of files. See 'Subfolders' option in this section for more information.

3. The button next to the **Mask data entry field** will rescan the directory and apply the filter.



4. The button next to the **Condition data entry field** will rescan the directory using the JavaScript condition. It will also display any errors in the JavaScript (similar to the 'Test' button on the JavaScript Code Entry form).



5. See 'JavaScript' under 'Special Menu' for more information.

6. JavaScript requires Commercial version of BRU (inexpensive license).

7. Refer to 'JavaScript' section in '[Volume 2 Bulk Rename Utility Operations Manual – Expert's Corner](#)' for more information on JavaScript and some examples of using the Condition data entry field.

8. F7 will clear the Content Pane of all files. It does not delete them. It just removes them from the File List. This is useful if you want to Drag and Drop files and want a 'clear' window to work with. To use F7, make sure that the Content Pane has focus (click on the Content Pane Window to change focus). Now press F7.

As an alternative, you can also use Ctrl + Drag and Drop to accomplish the same thing.

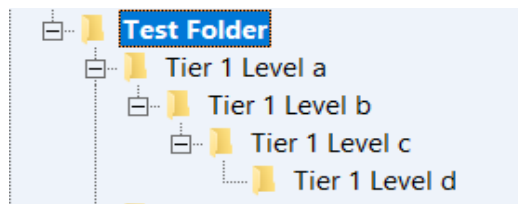
For more information, see 'Drag and Drop from Explorer' in an earlier section of this volume.

Section # 12: Filtersv3.42 New Additions**Set Subfolder Level to Control Recursive Scanning**

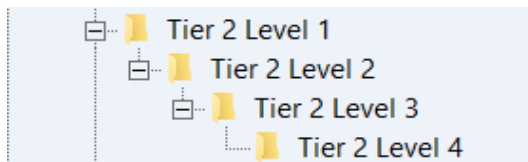
Added specification of subfolder level to set the maximum level of recursive scanning under ‘[Section #12: Filters](#)’.

This will help speed up recursive scanning, especially useful on Networked drives when a lot of files are involved. This way you only need to include those levels of folders that you require.

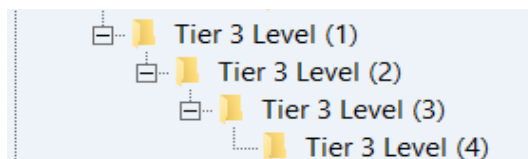
In each of these directories I have placed a single file ...



File #0
File #1
File #2
File #3
File #4



File #5
File #6
File #7
File #8



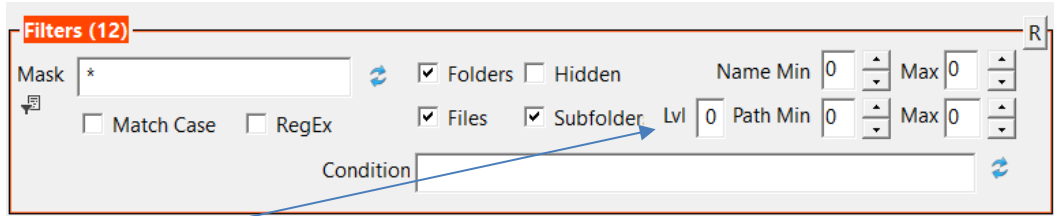
File #9
File #10
File #11
File #12

... It displays as ...

New Name	Sub Dir. ▼
File #12	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)\Tier 3 Level (4)
File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)
File #10	\Tier 3 Level (1)\Tier 3 Level (2)
File #9	\Tier 3 Level (1)
File #8	\Tier 2 Level 1\Tier 2 Level 2\Tier 2 Level 3\Tier 2 Level 4
File #7	\Tier 2 Level 1\Tier 2 Level 2\Tier 2 Level 3
File #6	\Tier 2 Level 1\Tier 2 Level 2
File #5	\Tier 2 Level 1
File #4	\Tier 1 Level a\Tier 1 Level b\Tier 1 Level c\Tier 1 Level d
File #3	\Tier 1 Level a\Tier 1 Level b\Tier 1 Level c
File #2	\Tier 1 Level a\Tier 1 Level b
File #1	\Tier 1 Level a
File #0	
Tier 3 Level (4)	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)
Tier 3 Level (3)	\Tier 3 Level (1)\Tier 3 Level (2)
Tier 3 Level (2)	\Tier 3 Level (1)
Tier 2 Level 4	\Tier 2 Level 1\Tier 2 Level 2\Tier 2 Level 3
Tier 2 Level 3	\Tier 2 Level 1\Tier 2 Level 2
Tier 2 Level 2	\Tier 2 Level 1
Tier 1 Level d	\Tier 1 Level a\Tier 1 Level b\Tier 1 Level c
Tier 1 Level c	\Tier 1 Level a\Tier 1 Level b
Tier 1 Level b	\Tier 1 Level a
Tier 1 Level a	
Tier 2 Level 1	
Tier 3 Level (1)	

Section # 12: Filters

v3.42 New Additions



Level 0 is the default. It will recurse all folder levels. In the example, the maximum level is 4. From there you can begin specifying levels.

Level 4

to

Level 1

<input checked="" type="checkbox"/> Folders	<input type="checkbox"/> Hidden	Name Min
<input checked="" type="checkbox"/> Files	<input checked="" type="checkbox"/> Subfolder	Lvl <input type="text" value="4"/> Path Min

<input checked="" type="checkbox"/> Folders	<input type="checkbox"/> Hidden	Name Min
<input checked="" type="checkbox"/> Files	<input checked="" type="checkbox"/> Subfolder	Lvl <input type="text" value="1"/> Path Min

I'll demonstrate one file to help you to understand. I will use File #12 that resides 4 levels down from its root, Tier 3.

Level 4. All preceding sub-directories from Level 4 through Level 1 are included. Since there are 4 maximum levels, setting Level as 4 is the same as specifying 0 levels. File #12 resides in the Tier 3 Level (4) directory.

New Name	Sub Dir. ▼
File #12	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)\Tier 3 Level (4)

Level 3. All preceding sub-directories from Level 3 to Level 1 are included. Because File #12 was on Level 4, the next file, File #11 in Tier 3 Level (3) directory, is displayed.

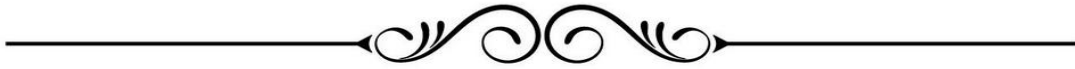
New Name	Sub Dir. ▼
File #11	\Tier 3 Level (1)\Tier 3 Level (2)\Tier 3 Level (3)

Level 2. All preceding sub-directories from Level 2 to Level 1 are included. Because File #11 was on Level 3, the next file, File #10 in Tier 3 Level (2) directory, is displayed

New Name	Sub Dir. ▼
File #10	\Tier 3 Level (1)\Tier 3 Level (2)

Level 1. All preceding sub-directories from Level 1 are included. Level 1 is the only preceding directory. The root directory, Tier 3, is never shown in the Sub-Dir. column because it is not a sub-directory. Because File #10 was on Level 2, the next file, File #9 in Tier 3 Level (1) directory, is displayed.

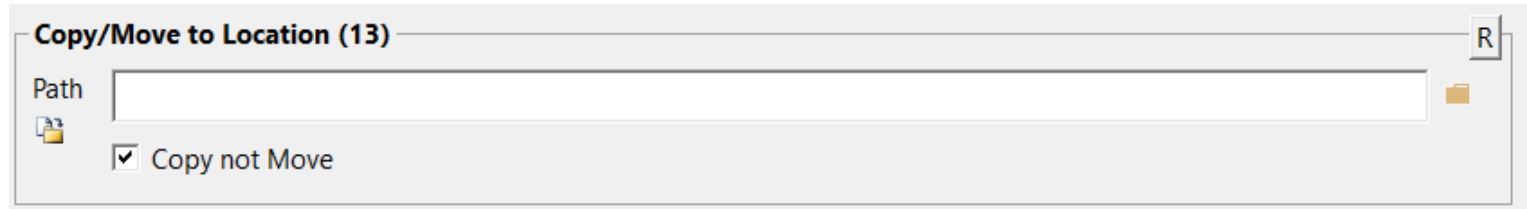
New Name	Sub Dir. ▼
File #9	\Tier 3 Level (1)



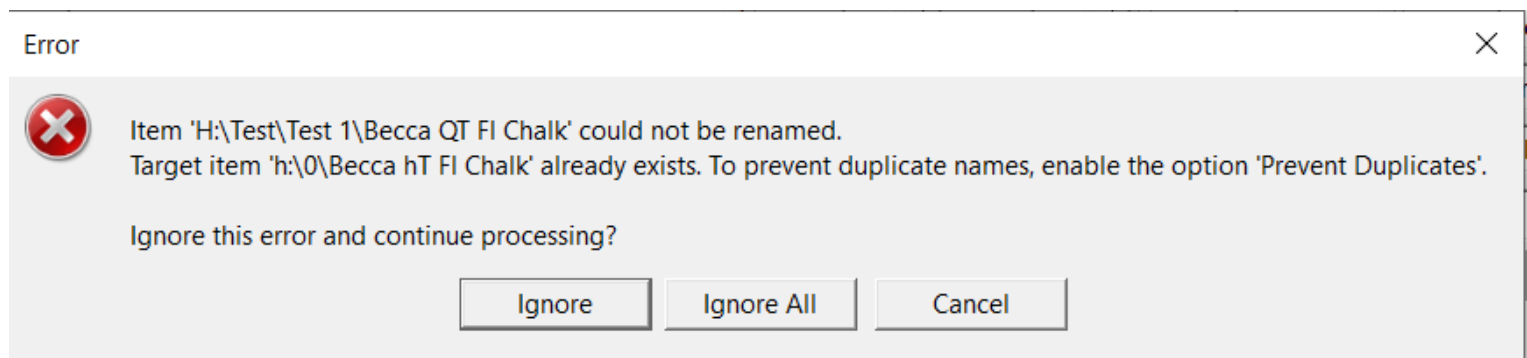
Section # 13 – Copy/Move to Location

Section # 13: Copy/Move to Location

Copy/Move to Location – option to copy or move [processed](#) files to a different specified directory. Copied files will leave the original files intact. Processed files are those files that have already been renamed.



If **'Allow Overwrite / Delete Existing Files During Renaming If Needed'** is set under the Advanced Options of the Renaming Options Menu, files with same name will be overwritten. If not, this message will be generated:



If you choose to ignore the error, processing will continue on with the next file.

Notes:

1. This restriction is overridden if **'Prevent Duplicates'** is enabled from the Renaming Options Menu.

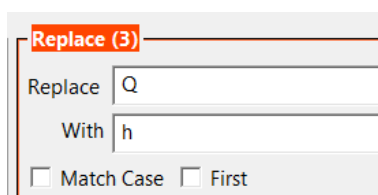
What this does is to append `'_#'` as a suffix to the file name,

i.e., Becca hT FI Chalk becomes Becca hT FI Chalk_1

where:

`_#` represents a count of each duplicate named file.

2. FYI, the above error example is the output result of **'Section #3: Replace'** set to .. Replace: Q With: h



Section # 13: Copy/Move to Location

This is how it works.

After renaming some files, I want to move them to the Test 2 directory. In this example I will remove the extension of some files in the Test 1 directory.

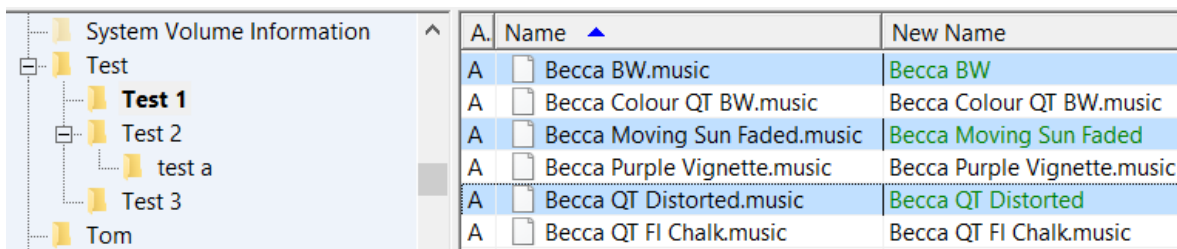
Current contents of Test 2 directory:

```
H:\Test\Test 2
<directory: *.*>
Andrew Gregory Macintyre.jpg
Doctor My Eyes - Jackson Browne.mp3
Donna Summer- Love to Love You Baby (extended original version).mp3
Donna Summer- Mac Arthur Park Suite (original extended version).mp3
Filename a (some text here).jpg
Filename b (some text here).jpg
Graham James Edward Miller.jpg
Graham the aB ab abc abC ABC Miller.jpg
HeLLO WoRlD.j
Here is another string 7 0 no date this time
Star Wars Episode USD100 123x12 aNbc.jpg
test file a.jpg
```

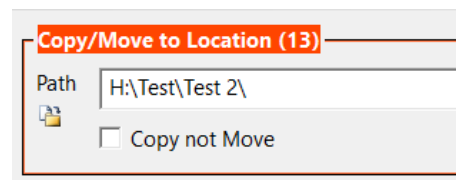
1. Specify my renaming criteria.



2. Select my files to be renamed in the Test 1 directory.



3. Specify the directory to which these files are to be moved.



Result:

```
H:\Test\Test 2
<directory: *.*>
Andrew Gregory Macintyre.jpg
Becca BW
Becca Moving Sun Faded
Becca QT Distorted
Doctor My Eyes - Jackson Browne.mp3
Donna Summer- Love to Love You Baby (extended original version).mp3
Donna Summer- Mac Arthur Park Suite (original extended version).mp3
Filename a (some text here).jpg
Filename b (some text here).jpg
Graham James Edward Miller.jpg
Graham the aB ab abc abC ABC Miller.jpg
HeLLO WoRlD.j
Here is another string 7 0 no date this time
Star Wars Episode USD100 123x12 aNbc.jpg
test file a.jpg
```

The three renamed files, Becca*, have been moved to the Test 2 directory.

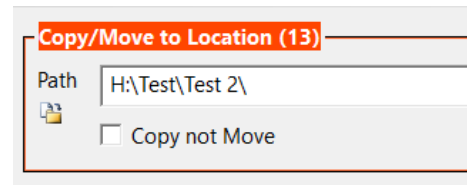
Section # 13: Copy/Move to Location

Notes:

1. Only files can be copied or moved, not the directories with their file contents.
2. 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).
3. If the target directory does not exist, it will be created.

Using the same example as before,

... Removing the extension from three files and moving them ...



The directory, Test 2 under the Test directory off of the H: drive, did not currently exist but was created 'on the fly' (American expression meaning, on demand) as the Copy/Move operation was being performed.

4. If the 'Copy' option is used, the original files are not renamed, only the copied files.

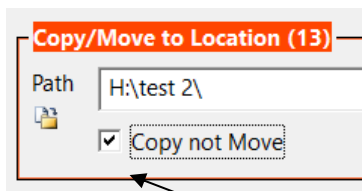
Using the same example as before,

... Removing the extension from three files and moving them ...

Files selected from Test 1 directory:

Name ▲	New Name
Becca BW.music	Becca BW
Becca Colour QT BW.music	Becca Colour QT BW
Becca Moving Sun Faded.music	Becca Moving Sun Faded

Target location specified as:

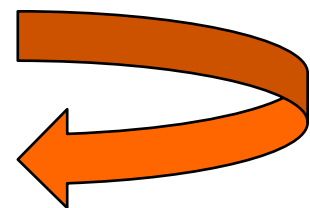
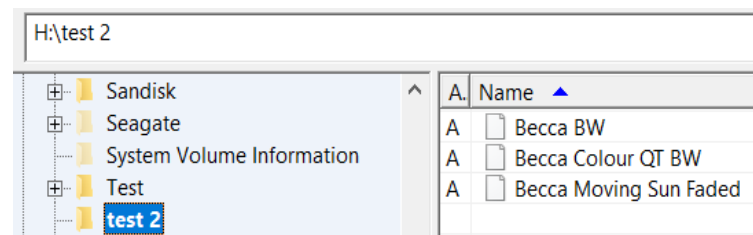
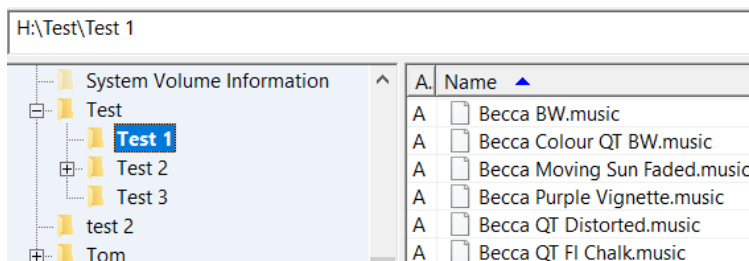


Files will be Copied not Moved.

Result:

Files remain unaltered in original directory, Test

The Renamed files are copied over to Test 2 directory



Section # 13: Copy/Move to Location

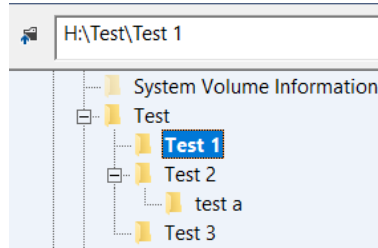
Because of the confusion among Forum participants over this subject, I will elaborate on my notations further.

1. Only files can be copied or moved, not the directories themselves with their file contents. 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). { Taken from notations #1 and #2 from the previous page }.

This means, using this function, I cannot simply move one directory including its contents to another location.

Example:

The current directory structure for the Test directory.



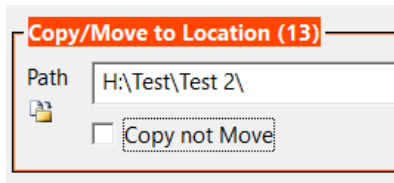
If Test 1 contains the following files:

```
ZTreeWin v2.4.197 - H: test file b.jpg
H:\Test\Test 1
<directory: *.*>
Becca BW.music
Becca Colour QT BW.music
Becca Moving Sun Faded.music
Becca Purple Vignette.music
Becca QT Distorted.music
Becca QT Fl Chalk.music
```

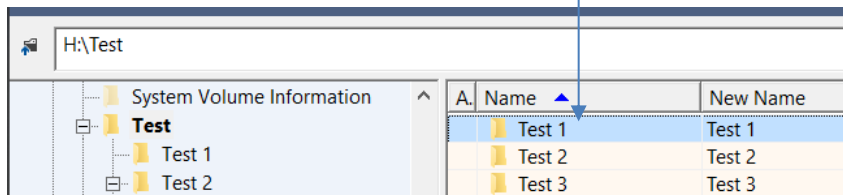
I want to move the Test 1 directory along with its contents to become a subdirectory under the Test 2 directory so I end up with –

H:\Test\Test 2\Test 1\

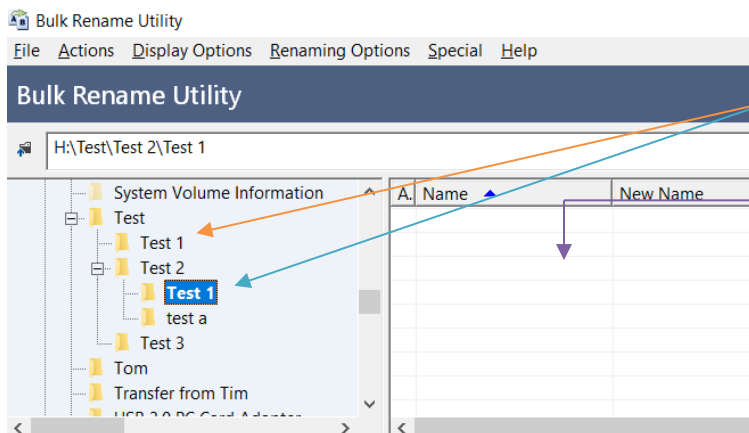
1. I specify Test 2 as the target directory:



2. The directory, Test 1, is selected...



3. Click on Rename:



Directory Test 1 was NOT moved but COPIED as a subdirectory under Test 2, and the contents of that subdirectory is EMPTY.

You will find, though, that the original contents of the Test 2 directory under Test are still intact, unaltered.

Conclusion:

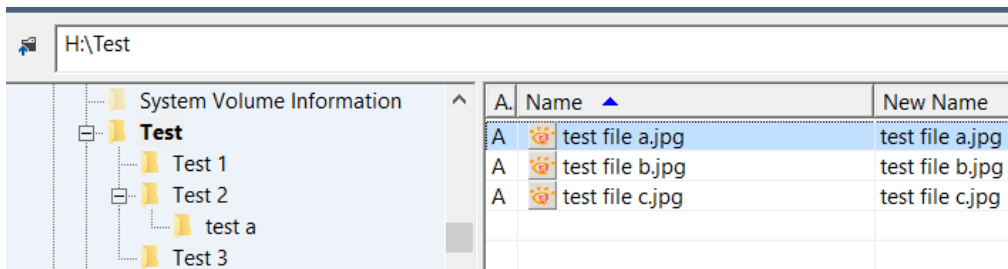
BRU will only copy the directory structure, not the contents, if you attempt to Copy/move a directory.

Section # 13: Copy/Move to Location

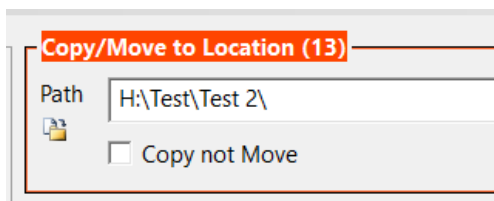
Let's examine this behaviour in more depth.

This is what happens when I select a file that does not meet the criteria for renaming using Copy/Move to Location.

1. I have selected a file, test file a.jpg, located in the main Test directory.



2. I Specify my target directory location.

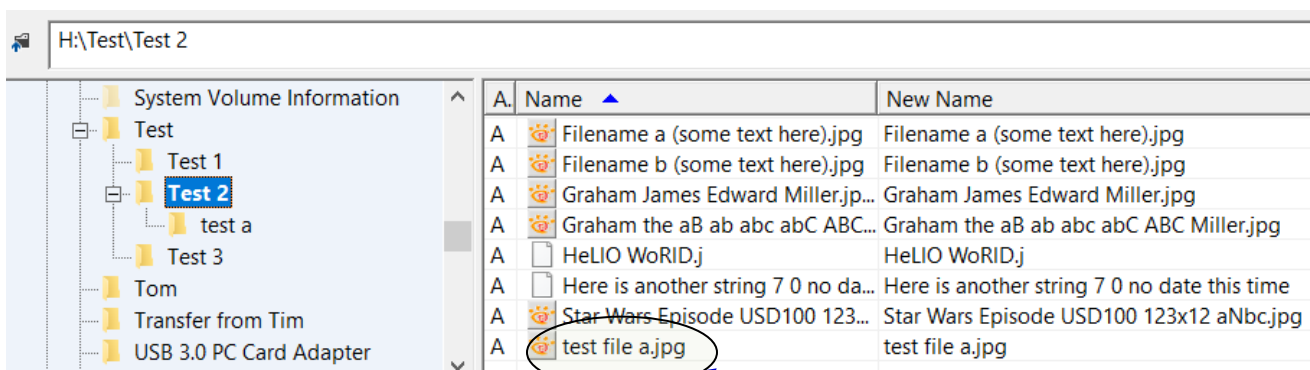


I have no other criteria other than the Move/To Location specification. Therefore, New Name shows no value other than the original Name unchanged.

Name ▲	New Name
test file a.jpg	test file a.jpg
test file b.jpg	test file b.jpg
test file c.jpg	test file c.jpg

3. Click on Rename.

Result:



The file is successfully moved to the Test 2 directory **unaltered**.

What does this mean? A Great Tip coming up, that's what.

Section # 13: Copy/Move to Location

Moving Files from one Directory to Another Location with Files content!

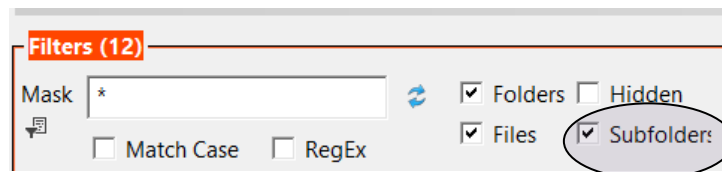
This is done in two parts:

1. Create the directory to which the files will be moved.

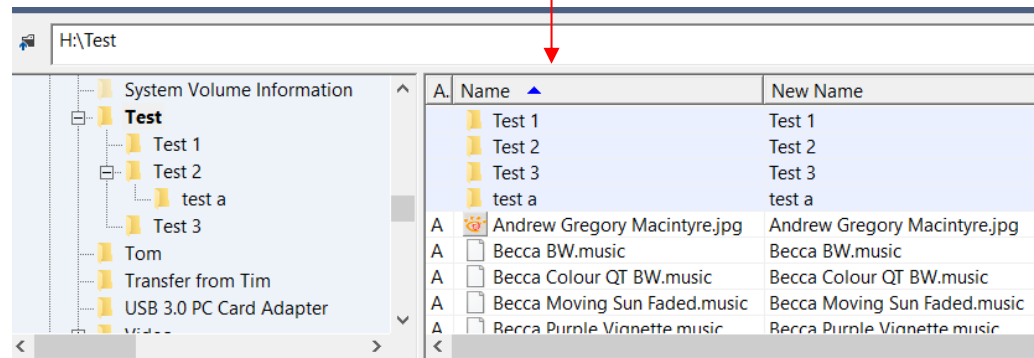
You can use any File Manager including Windows Explorer. I have already shown you how to do it in BRU, by specifying the directory and then issuing the Move/Copy.

For this example I will use BRU to create an empty directory structure.

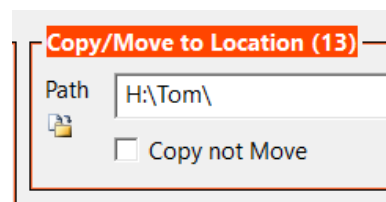
- a. Turn on Recursion.



- b. Select the directories.

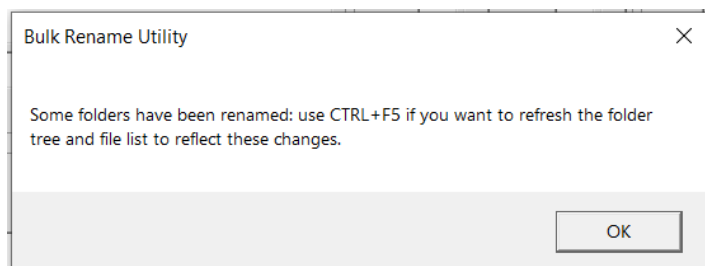


- c. Specify the target location.

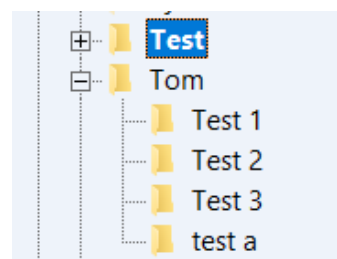


- d. Click Rename.

- e. Click Ctrl + F5 to refresh.



- f. Directory structure has been recreated under Tom, albeit empty of any file content.



Section # 13: Copy/Move to Location

2. Now we move the files. Leave recursion on to move all of the files into a single directory or turn recursion off and perform the operation on each directory individually. To save time for this experiment, I will opt to move them all at once into a single directory, H:\Tom\Test 1.

1. Select my files.

A	Name	New Name
	Test 1	Test 1
	Test 2	Test 2
	Test 3	Test 3
	test a	test a
A	Andrew Gregory Macintyre.jpg	Andrew Gregory Macintyre.jpg
A	Becca BW.music	Becca BW.music
A	Becca Colour QT BW.music	Becca Colour QT BW.music
A	Becca Moving Sun Faded.music	Becca Moving Sun Faded.music
A	Becca Purple Vignette.music	Becca Purple Vignette.music
A	Becca QT Distorted.music	Becca QT Distorted.music
A	Becca QT Fl Chalk.music	Becca QT Fl Chalk.music
A	Doctor My Eyes - Jackson Brow...	Doctor My Eyes - Jackson Browne.mp3
A	Donna Summer- Love to Love Y...	Donna Summer- Love to Love You Baby (extended original version).mp3
A	Donna Summer- Mac Arthur Par...	Donna Summer- Mac Arthur Park Suite (original extended version).mp3
A	Filename a (some text here).jpg	Filename a (some text here).jpg
A	Filename b (some text here).jpg	Filename b (some text here).jpg
A	Graham James Edward Miller.jp...	Graham James Edward Miller.jpg
A	Graham the aB ab abc abC ABC...	Graham the aB ab abc abC ABC Miller.jpg
A	HeLIO WoRID.j	HeLIO WoRID.j
A	Here is another string 7 0 no da...	Here is another string 7 0 no date this time
A	Star Wars Episode USD100 123...	Star Wars Episode USD100 123x12 aNbc.jpg
A	test file a.jpg	test file a.jpg
A	test file b.jpg	test file b.jpg
A	test file c.jpg	test file c.jpg

2. Select my target directory.

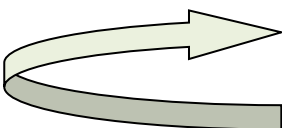
Copy/Move to Location (13)

Path

Copy not Move

3. Click Rename.

```
H:\Tom\Test 1
<directory: *.*>
Andrew Gregory Macintyre.jpg
Becca BW.music
Becca Colour QT BW.music
Becca Moving Sun Faded.music
Becca Purple Vignette.music
Becca QT Distorted.music
Becca QT Fl Chalk.music
Doctor My Eyes - Jackson Browne.mp3
Donna Summer- Love to Love You Baby (extended original version).mp3
Donna Summer- Mac Arthur Park Suite (original extended version).mp3
Filename a (some text here).jpg
Filename b (some text here).jpg
Graham James Edward Miller.jpg
Graham the aB ab abc abC ABC Miller.jpg
HeLIO WoRID.j
Here is another string 7 0 no date this time
Star Wars Episode USD100 123x12 aNbc.jpg
test file a.jpg
test file b.jpg
test file c.jpg
```

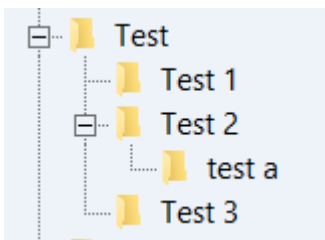


All files have been moved intact with no renaming required! Neat, huh?

Section # 13: Copy/Move to Location

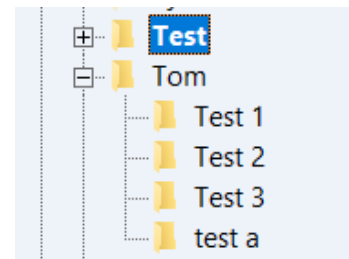
Notes:

1. When I refer to 'Main' it is because it is the **root** directory of this directory structure and **not** a subdirectory branching off from it.
2. The subdirectory, test a, was originally a subdirectory under Test2, but because I included it in the transfer operation, it became a subdirectory directly under the Test directory of Tom. There is no current method of recreating more than 1 level of directories using this method. In other words, I can't recreate the original structure of:



to the new location.

Instead, I have to settle for this:



test a directory becomes a subdirectory off of Tom instead of off of Test 2

I can always move the test a directory manually using a File Manager.

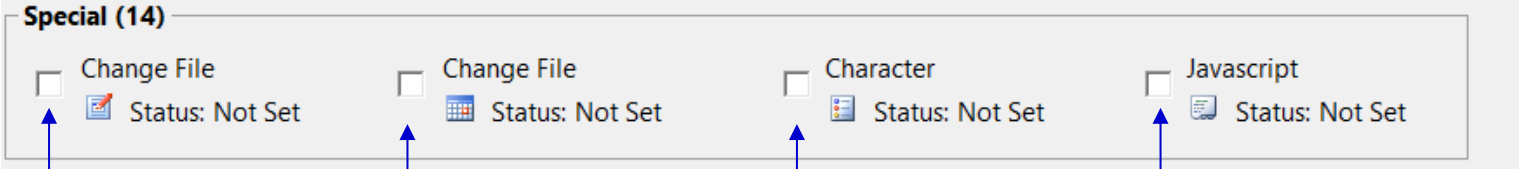
3. You can also use my tip for moving the directories to a location that doesn't exist because BRU will create it on the fly just as it does normally when specifying non-existent target directories for renaming operations.







Section # 14 – Special

Section # 14: Special

The **Special** Section is made up of the following:



Special (14)

- Change File  Status: Not Set
- Change File  Status: Not Set
- Character  Status: Not Set
- Javascript  Status: Not Set

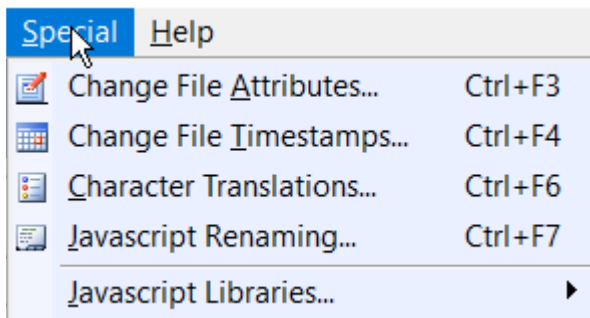
Change File – Change File Attributes
File Attributes consist of ‘Read-Only’, ‘Archive’, ‘Hidden’, and System.

Change File – Change File Timestamps
Time stamps consist of ‘Date Created’, ‘Date Modified’, and ‘Date Accessed’.

Character – Character Translations
Defines a set of rules that will translate one specific character or sequence of characters into a different character or sequence of characters.

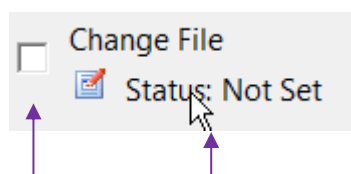
JavaScript - Use JavaScript in Renaming
(ONLY available with purchase of an inexpensive Commercial license)

This section has an entire menu equivalent – ‘Special Menu’



Section 14, therefore, is located on the User Interface as a convenience.

For each of these criteria there is a button, **Status Not Set**. When you click on either it or the little blank checkbox, it will bring up the dialog box with available settings and options that pertains to that criteria.



Section # 14: Special

File Change (Attributes):

File Change (Timestamp):

Character (Translations):

For further information on using these criteria, please refer to the appropriate section dedicated to each of these functions earlier in the book. More information can be found under the 'Special Menu' of the Menu section as well.

JavaScript:

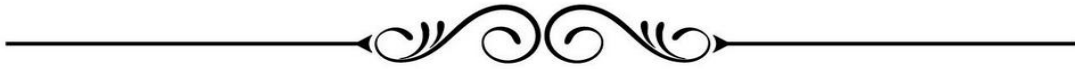
This code entry form affords you a working space to both write and optionally test the code's syntax before committing to it.

Notes:

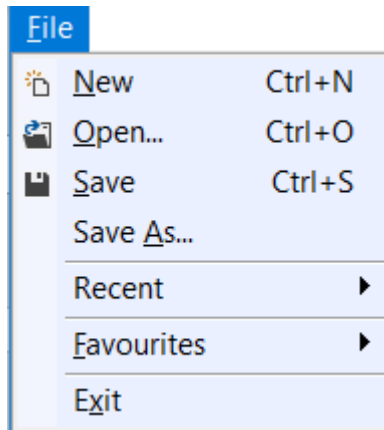
'JavaScript' form requires a knowledge of JavaScript as well as the Commercial version of BRU to perform the rename function. Testing and code syntax checking are included in freeware edition.

Also refer to 'JavaScript' section in '[Volume II Bulk Rename Utility Operations Manual – Expert's Corner](#)'.

Menus



File Menu



File Menu

New (Ctrl + N)

This will unload any previous [Favourite](#) file and ‘Reset’ all the criteria but it has no effect on any loaded ‘Imported Rename Pairs’. In addition, if ‘Save on Exit’ is set and ‘New’ is still in effect (there is currently no Favourites file loaded), then you will be reminded to create a file upon exit. For more information, refer to ‘Understanding Favourites’ earlier in this volume.

The screenshot displays the Bulk Rename Utility interface with the following sections:

- RegEx (1)**: Match and Replace fields, with an 'Include Ext.' checkbox.
- Replace (3)**: Replace and With fields, with a 'Match Case' checkbox.
- Remove (5)**: First n, Last n, From, to, Chars, Words, Crop, and Trim options.
- Add (7)**: Prefix, Insert, at pos., Suffix, and Word Space options.
- Auto Date (8)**: Mode, Type, Fmt, Sep., Seg., Custom, and Cent. Off. options.
- Numbering (10)**: Mode, at, Start, Incr., Pad, Sep., Break, Folder, Type, and Roman options.
- Name (2)**: Name dropdown menu.
- Case (4)**: Same and Excep. dropdown menus.
- Move/Copy Parts (6)**: None, 1, None, 1, and Sep. options.
- Append Folder Name (9)**: Name, Sep., and Levels options.
- Extension (11)**: Same and dropdown menu.
- Filters (12)**: Mask, Match Case, RegEx, Folders, Hidden, Files, Subfolders, Name Len, Path Len Min, and Max options.
- Copy/Move to Location (13)**: Path and Copy not Move checkbox.
- Special (14)**: Change File, Character, and Javascript options.

At the bottom, the status bar shows '924 Objects (1 Selected)'. The software title is 'Bulk Rename Utility 64-bit (Unicode)', version 3.3.1.0, with a 'Check for Updates' link. Buttons for 'Preview', 'Reset', 'Revert', and 'Rename' are visible.

Open (Ctrl + O)

Open a previously saved [Favourites](#) file (saved with a ‘.bru’ extension). A Favourites file (or for you Americans – Favorites), is a file that contains all of the set criteria so you don’t have to enter it from scratch each time for a particular renaming task. For more information, see ‘Favourites’ in this section and ‘Understanding Favourites’ earlier in this volume.

Save (Ctrl + S)

Saves the current set criteria as a [Favourites](#) File. If the file doesn’t already exist, a new name will be requested. If the file exists, it will be overwritten. For more information, refer to ‘Understanding Favourites’ earlier in this volume.

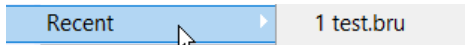
Save As

Request to Save a [Favourites](#) file with a new name. For more information, refer to ‘Understanding Favourites’ earlier in this volume.

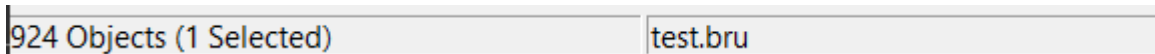
File Menu

Recent

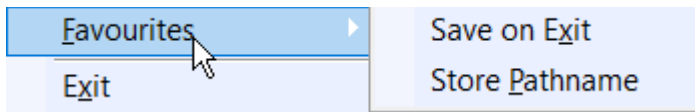
For your convenience, BRU remembers the last few [Favourite](#) files for easy selection without having to use the ‘Open’ dialog.



The most recent file, will also be displayed in the status bar unless ‘New’ was previously selected.



Favourites



A Favourites file saves all of the settings, including criteria, selection and even the current directory. You can have multiple Favourites files each consigned to a particular task.

– *Save on Exit*

If set, this will automatically save any changes to the current Favourites file on exiting the program. If there is no current Favourites file, a new name will be requested upon exit by presenting the Save dialog box.

– *Store Pathname*



If set, the next time this Favourites file is loaded, you will be placed into the same directory at the time this file was last saved. This allows you to continue previous work. If not, set, you will remain in the current directory when the Favourites file is loaded.

1. For more information, refer to ‘Understanding Favourites’ earlier in this volume.

[This Page Intentionally Blank]



Actions Menu

Actions	
<u>S</u> elect All	Ctrl+A
<u>D</u> eselect All	Ctrl+D
<u>I</u> nvert Selection	Ctrl+I
Select from Clipboard	
<u>J</u> ump to Path...	Ctrl+J
Rename Object <u>M</u> anually	F2
 <u>R</u> efresh Files	F5
Refresh <u>T</u> ree	Ctrl+F5
 Show/Hide Tree	F11
<u>Z</u> oom...	F8
<u>L</u> ist	▶
<u>I</u> mport Rename-Pairs	▶
Debug New <u>N</u> ame	
<u>R</u> eset All Renaming Criteria	Ctrl+T
Re <u>v</u> ert All Criteria to Last Saved	Ctrl+E
<u>P</u> review	Ctrl+P
<u>R</u> ename	Ctrl+R
<u>U</u> ndo Rename	Ctrl+Z
Create Undo <u>B</u> atchfile	Ctrl+B

Actions Menu

Selection

- Individually Select with Keyboard and Mouse

Select which files are to be processed in the Content Pane using ‘**Ctrl + Left Mouse Click**’ to individually select files or use ‘**Shift Key + Down Arrow**’ to select files consecutively.

Note:

1. The files will always be processed in the order of the *displayed sequence*.

For more information see -

- ‘Sorting’ under the Display Options Menu
- ‘List - Reposition’, under the Actions Menu
- ‘Apply Random Sort to Current List’, under the Actions Menu
- ‘Rename in Reverse Order’, under the Renaming Options Menu
- notation #6 under ‘Program Notes’ section earlier in this volume

- Select All (Ctrl + A)

Selects all files and directories displayed in the Content Pane.

Name
Bing Crosby - Silent Night.mp3
Carol of the Bells (for 12 cellos) - The Piano Guys.mp3
Carol of the Bells - Lindsey Stirling.mp3
Dance of the Sugar Plum Fairy - Lindsey Stirling.mp3
Do You Hear What I Hear - Bing Crosby.mp3
Elements - Lindsey Stirling (Dubstep Violin Original Song).mp3
God Rest Ye Merry Gentlemen We Three Kings - Bare Naked Ladies, Sarah McLachlan.mp3
Have Yourself a Merry Little Christmas - Frank Sinatra.mp3
It's Beginning to Look a Lot Like Christmas - Bing Crosby.mp3

- Deselect All (Ctrl + D)

Unselects any files and directories previously selected in the Content Pane.

- Invert Selection (Ctrl + I)

Reverses the selections in the Content Pane. If a file or directory was selected previously, it becomes unselected. If a file or directory was previously unselected, it becomes selected.

Before -

Name
Bing Crosby - Silent Night.mp3
Carol of the Bells (for 12 cellos) - The Piano Guys.mp3
Carol of the Bells - Lindsey Stirling.mp3
Dance of the Sugar Plum Fairy - Lindsey Stirling.mp3
Do You Hear What I Hear - Bing Crosby.mp3
Elements - Lindsey Stirling (Dubstep Violin Original Song).mp3
God Rest Ye Merry Gentlemen We Three Kings - Bare Naked Ladies, Sarah McLachlan.mp3
Have Yourself a Merry Little Christmas - Frank Sinatra.mp3
It's Beginning to Look a Lot Like Christmas - Bing Crosby.mp3

After -

Name
Bing Crosby - Silent Night.mp3
Carol of the Bells (for 12 cellos) - The Piano Guys.mp3
Carol of the Bells - Lindsey Stirling.mp3
Dance of the Sugar Plum Fairy - Lindsey Stirling.mp3
Do You Hear What I Hear - Bing Crosby.mp3
Elements - Lindsey Stirling (Dubstep Violin Original Song).mp3
God Rest Ye Merry Gentlemen We Three Kings - Bare Naked Ladies, Sarah McLachlan.mp3
Have Yourself a Merry Little Christmas - Frank Sinatra.mp3
It's Beginning to Look a Lot Like Christmas - Bing Crosby.mp3

Actions Menu

- Select From Clipboard

If a previous list of files or directories was saved via the clipboard, e.g. the output from a 'Dir \B | clip' command, the program will select which files or directories from the clipboard match any files or directories currently displayed in the Content Pane.

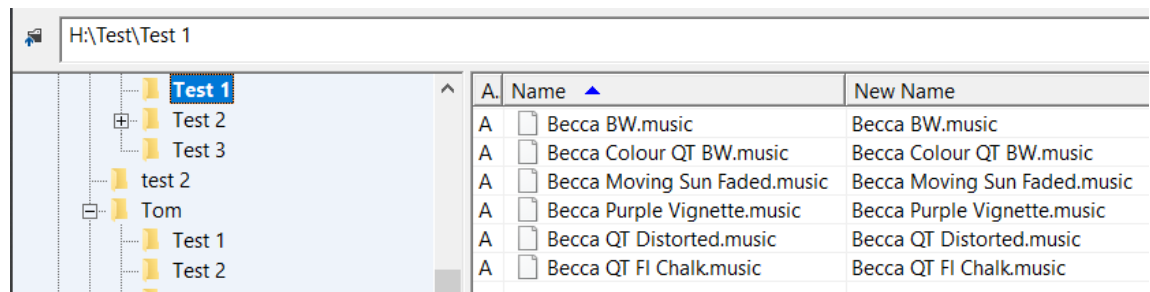
Notes:

1. It does not place the contents of the clipboard into the Content Pane, it only matches against it.
2. Matches will be highlighted by row based on the Name column (Clipboard contents must 'exactly' match).
3. In the example Dir command above, 'clip', an executable, is only available in Vista or higher. Clip redirects command output from the command line to the Windows clipboard.

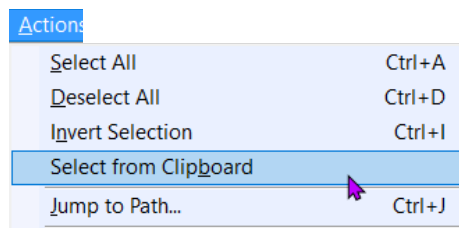
Example of clipboard contents containing list of files:

```
Becca BW.music
Becca Colour QT BW.music
Becca Moving Sun Faded.music
```

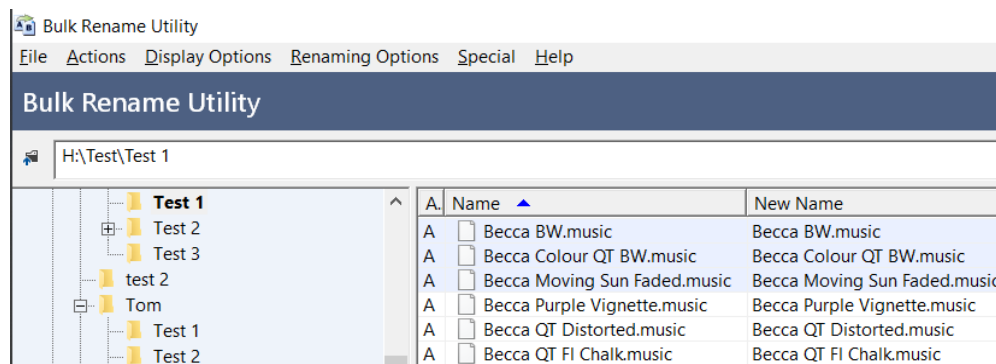
Before:



Issue command:



After:



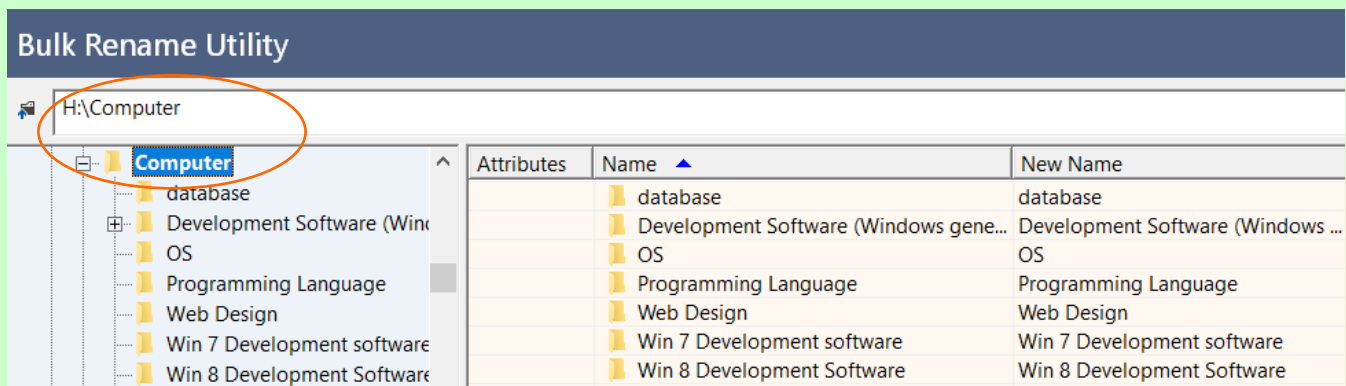
Actions Menu

Another Example. This time I am going to match against directories.

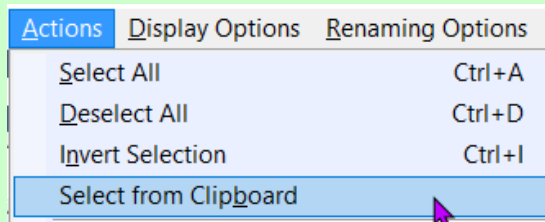
clipboard contains the following directories:

database
OS
Win 7 Development software

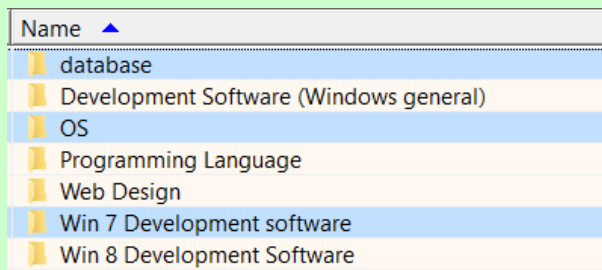
In the Navigation Pane the ‘\Computer’ directory is currently selected.



From Actions Menu, click on ‘Select from Clipboard’.



Results in:



The directories above were matched with the contents of the clipboard providing the selections highlighted.

Actions Menu

Jump to Path (Ctrl + J)

Initiates the 'Jump Directly to Path' dialog box. This allows you to navigate directly to the user specified path. If you don't know the path, you can click on '...' and browse for it. BRU will switch to the path in the Navigation Pane, update the Address line and display any file contents in the Content Pane.



Jump to Path (Ctrl + J): [v3.4 New Additions](#)

Network/UNC Paths are Supported.

In a network, the Universal Naming Convention (UNC) is a way to access a shared file in a computer without having to specify (or know) the storage device it is on. In Windows, the UNC can be used instead of the local naming system.

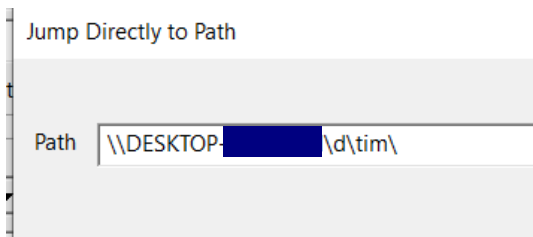
The UNC name format is:

```
\\servername\sharename\path\filename
```

instead of:

```
d:\path\directory name
```

Example:



Where:

[\\DESKTOP-](#) [redacted]

server name (name of computer being accessed)

\d
\tim\

share name. In this case the share is a drive's root directory
pathname

filename

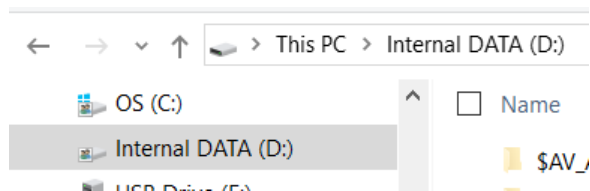
not used because Jump to Path only supports paths not filenames.

Actions Menu

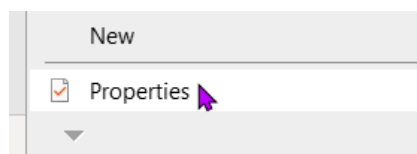
Jump to Path (Ctrl + J): v3.4 New Additions

If you want to find the UNC Network path name for a share -

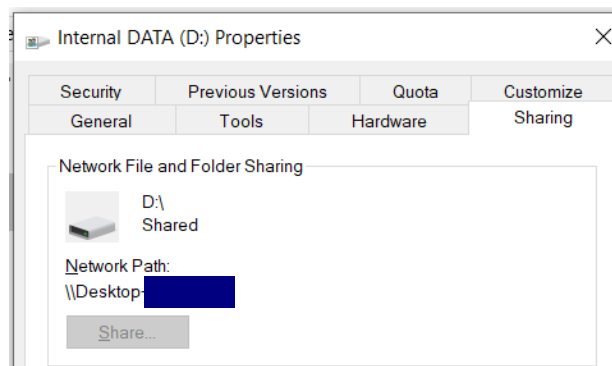
1. In Explorer, navigate to a share. I temporarily created the D share for this example.



2. Right click and select 'Properties'



Brings up the Properties dialog box:



Under Sharing tab, you can see the UNC Network path. Here you can use Copy and Paste to copy the path into BRU.

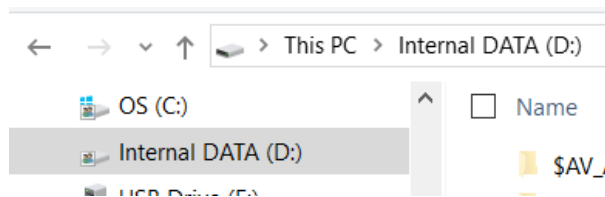
Actions Menu

[Jump to Path \(Ctrl + J\): v3.4 New Additions](#)

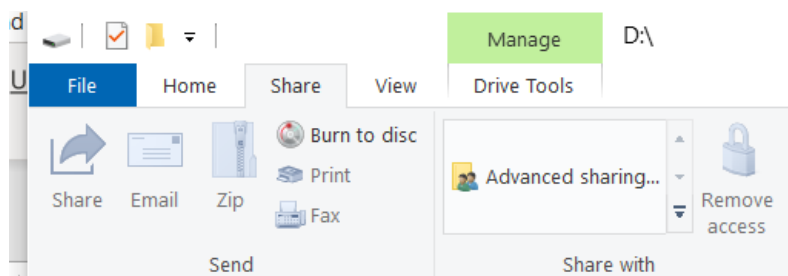
Here are some additional methods to view and copy the Network Path:

Using the same D share,

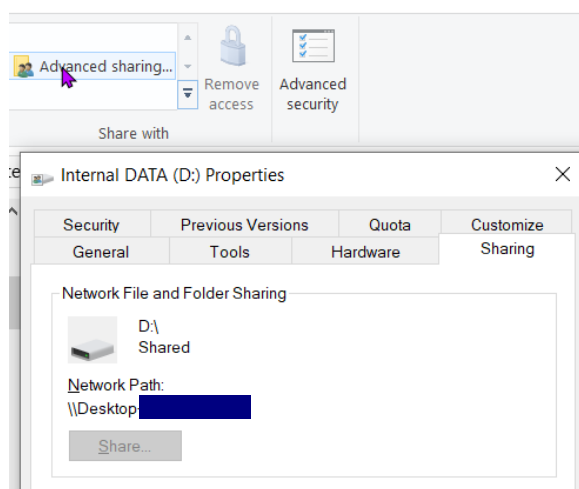
1. In Explorer, navigate to a share.



2. In the Share tab in the top menu, select Advanced Sharing.



3. This brings up the Share Dialog box directly:



You can use Copy and Paste to copy the path into BRU.

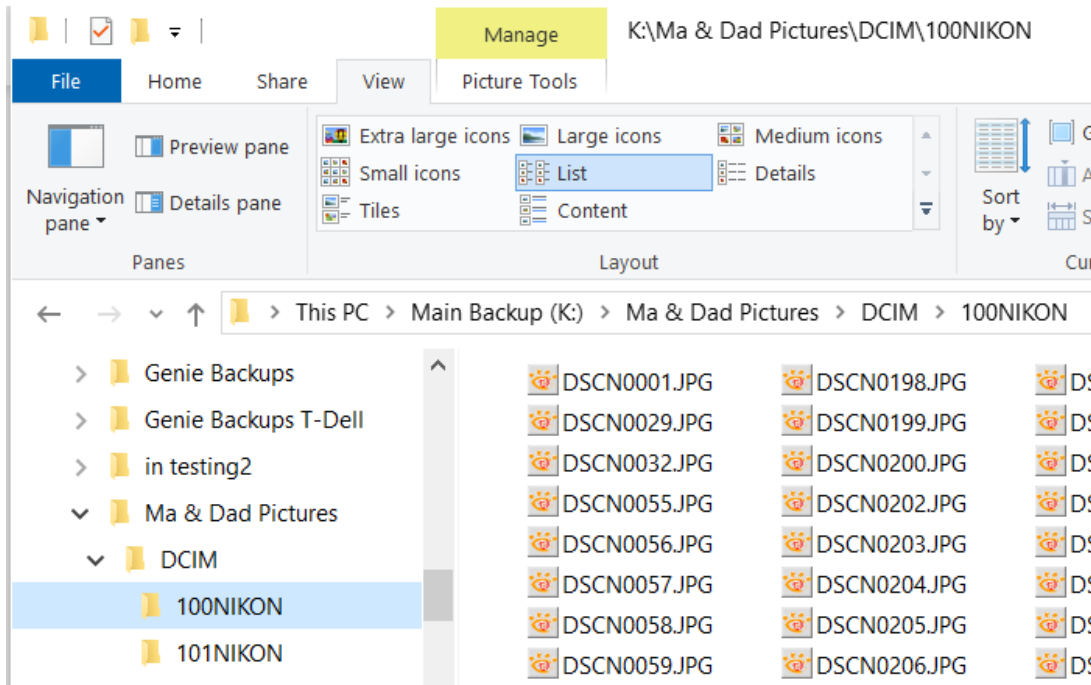
Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

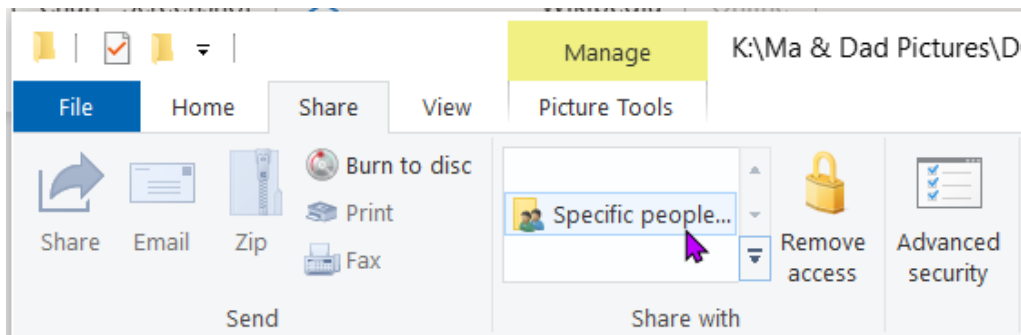
For this next method, I created another temporary share, this time a directory on an external drive.

Create a share:

1. In Windows Explorer, navigate to the folder you want to share. Since this is a test, I decided on one of my photo directories:



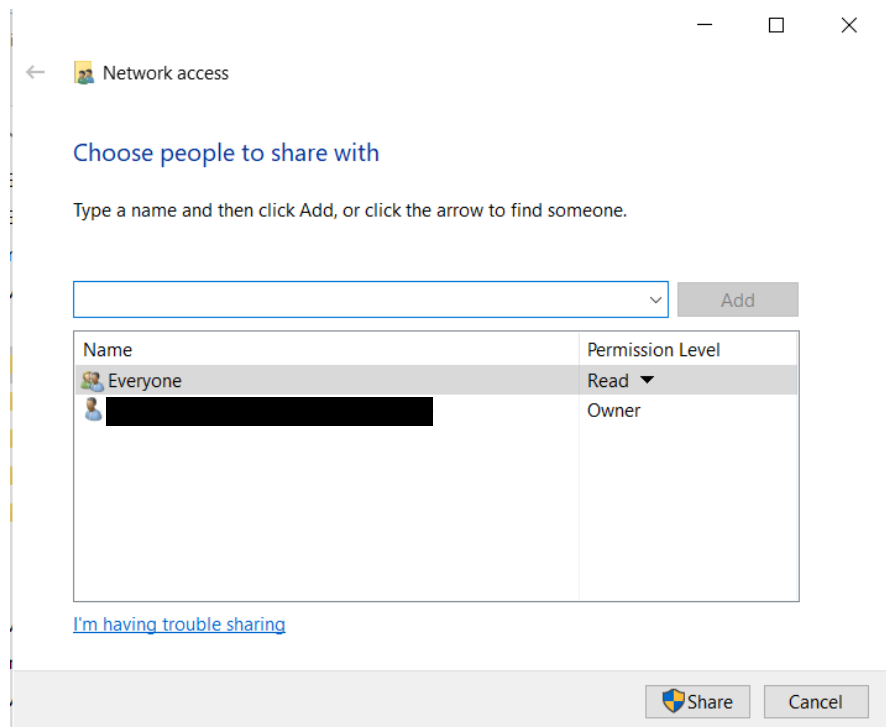
2. Under the Share tab, click on 'Specific People'



Actions Menu

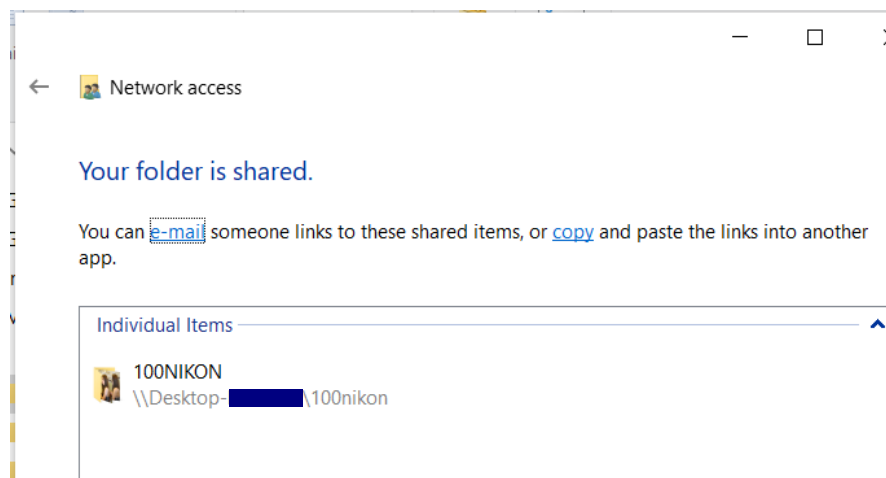
[Jump to Path \(Ctrl + J\): v3.4 New Additions](#)

3. This opens the Share dialog box:



I added the group, 'Everyone' providing Read access only. You can specify this or any other defined group.

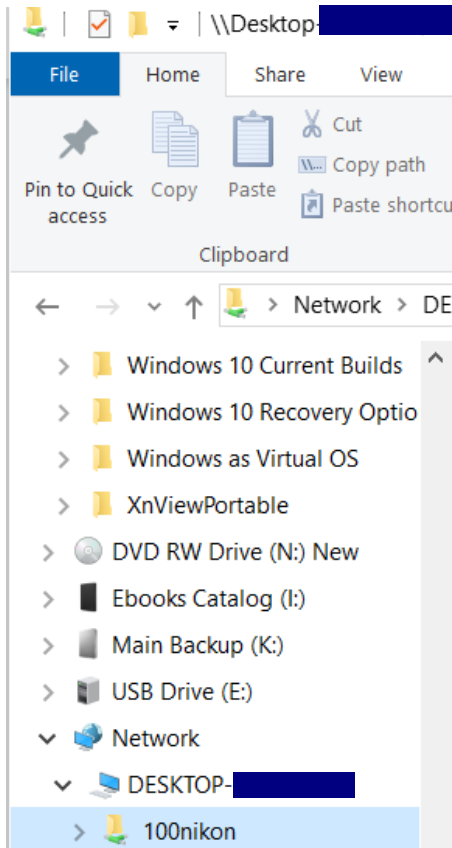
4. Click on the 'Share' button:



Actions Menu

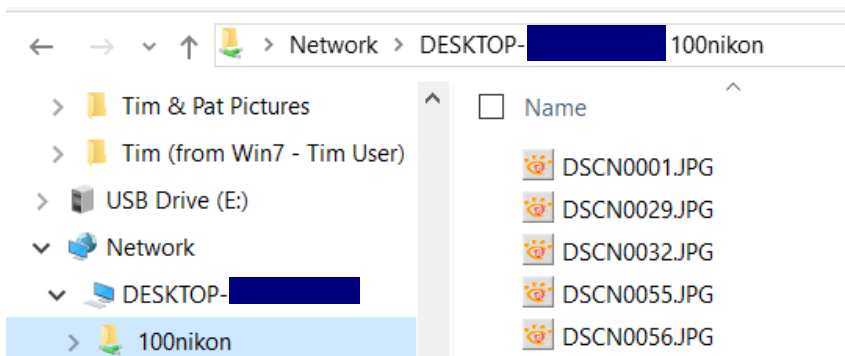
[Jump to Path \(Ctrl + J\): v3.4 New Additions](#)

Open Windows Explorer and navigate to Networks. Under Networks, you should see your new share:



To view and copy the Network path, here are a couple more methods:

1. Navigate to the Network in the Tree Hierarchy of the Navigation Pane and select the device that holds the share. In this particular example it was on my local network under DESKTOP-[redacted]



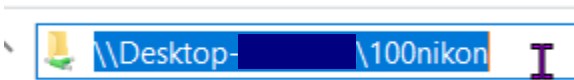
Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

2. Click inside the file share to display the pseudo path in the Explorer address bar.



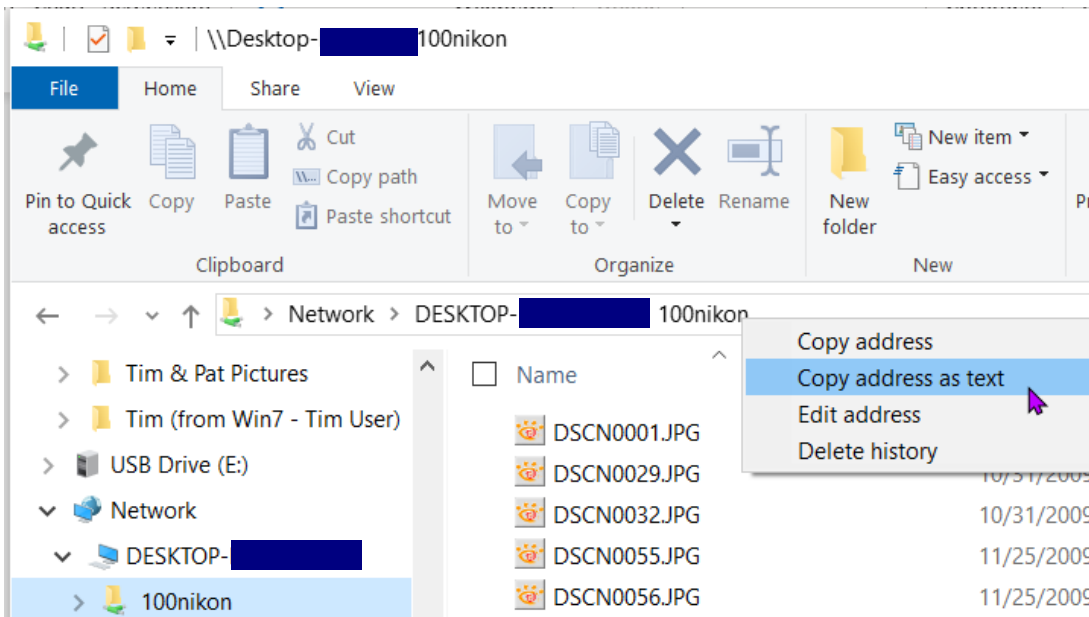
3. Click on the address bar and it will change to the Network path:



4. Copy the highlighted address to the clipboard buffer.

or... replace step 2 above with ...

2. right click on the pseudo path and select 'Copy address as Text':



Actions Menu

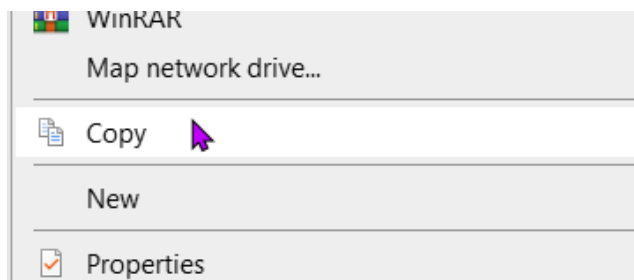
Jump to Path (Ctrl + J): v3.4 New Additions

or

2. Right click on the shared directory in the Navigation Pane,



3. .. and select 'Copy'.



The UNC Path is copied to the Clipboard buffer:

[\\Desktop-\[redacted\]100nikon](#)

Any of these methods will work.

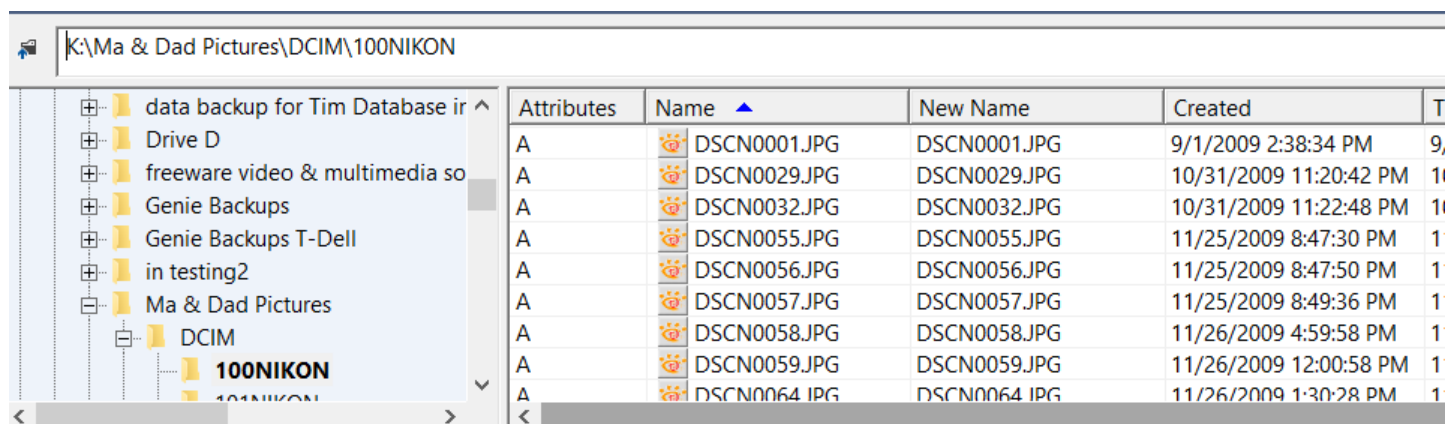
Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

The reason I created a temporary share on a specific directory and not on a root directory of a drive was to illustrate that the UNC is **not** simply applying the filepath with an added syntax of `\\<path>` but **is** using the assigned network **Share name**.

Path is:

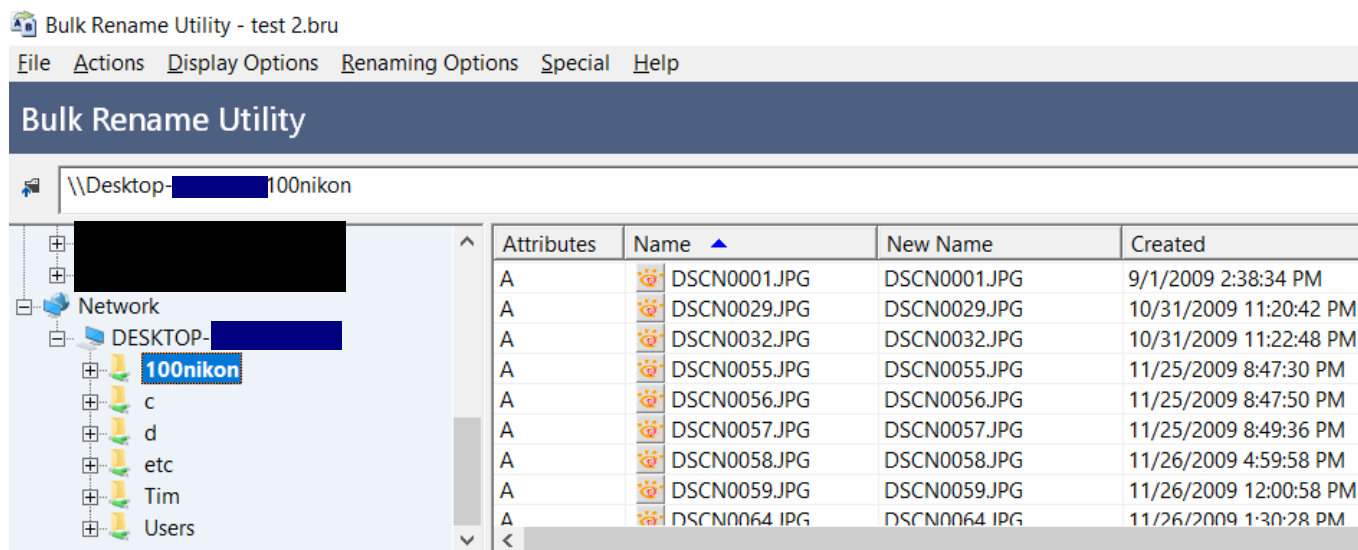
K:\Ma & Dad Pictures\DCIM\100NIKON



Network Share is:

//Desktop-██████████100nikon

and this would be the UNC path as well:

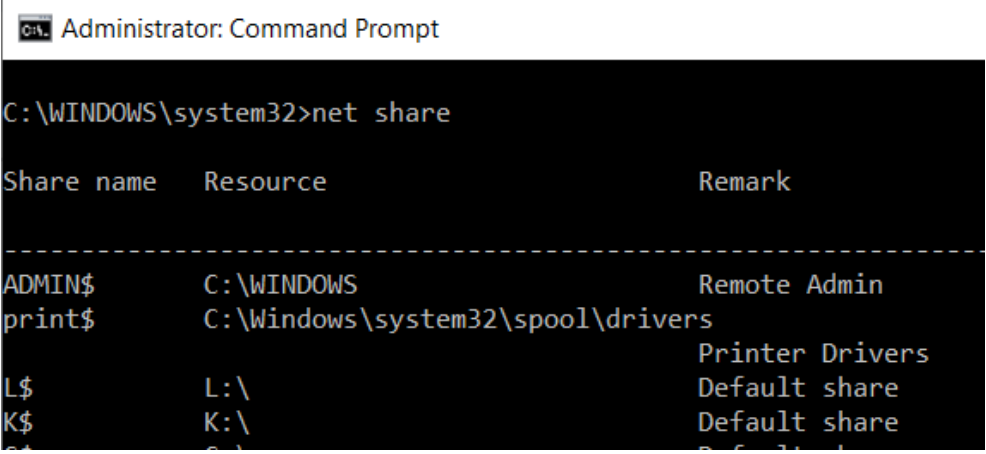


Actions Menu

[Jump to Path \(Ctrl + J\): v3.4 New Additions](#)

If you want to view all the available shares on the current computer, use the net command.

1. From a command prompt, type 'net share'.



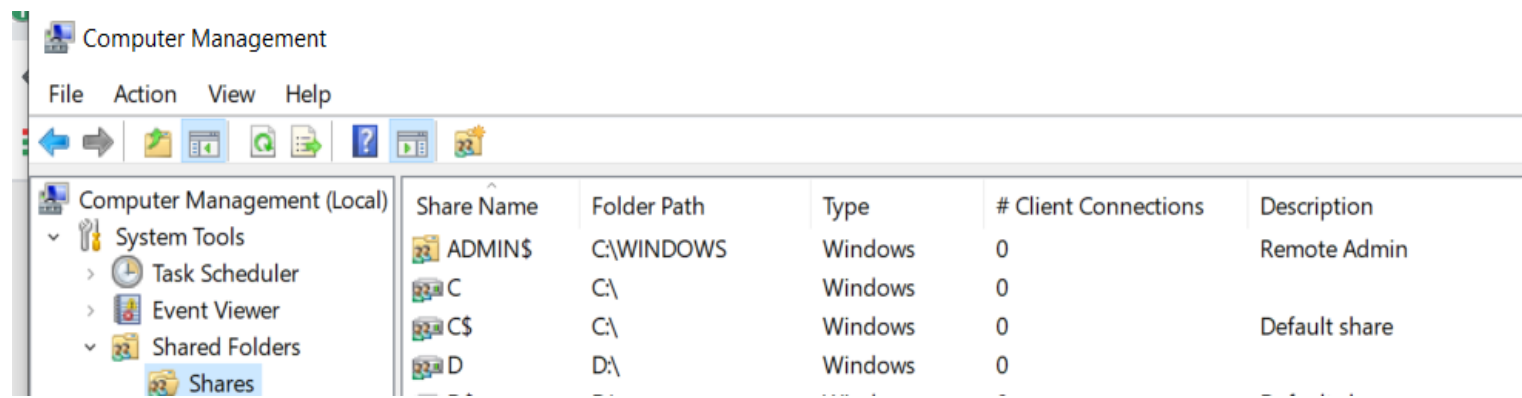
```
Administrator: Command Prompt
C:\WINDOWS\system32>net share

Share name      Resource                Remark
-----
ADMIN$          C:\WINDOWS              Remote Admin
print$          C:\Windows\system32\spool\drivers Printer Drivers
L$              L:\                     Default share
K$              K:\                     Default share
```

The advantage of this over simply viewing the Network in the Explorer Navigation Pane is that this will display all shares including hidden shares.

This can also be done from Computer Management:

1. Right click on Start Menu and select Computer Management.
2. Navigate to System Tools\Shared Folders\Shares



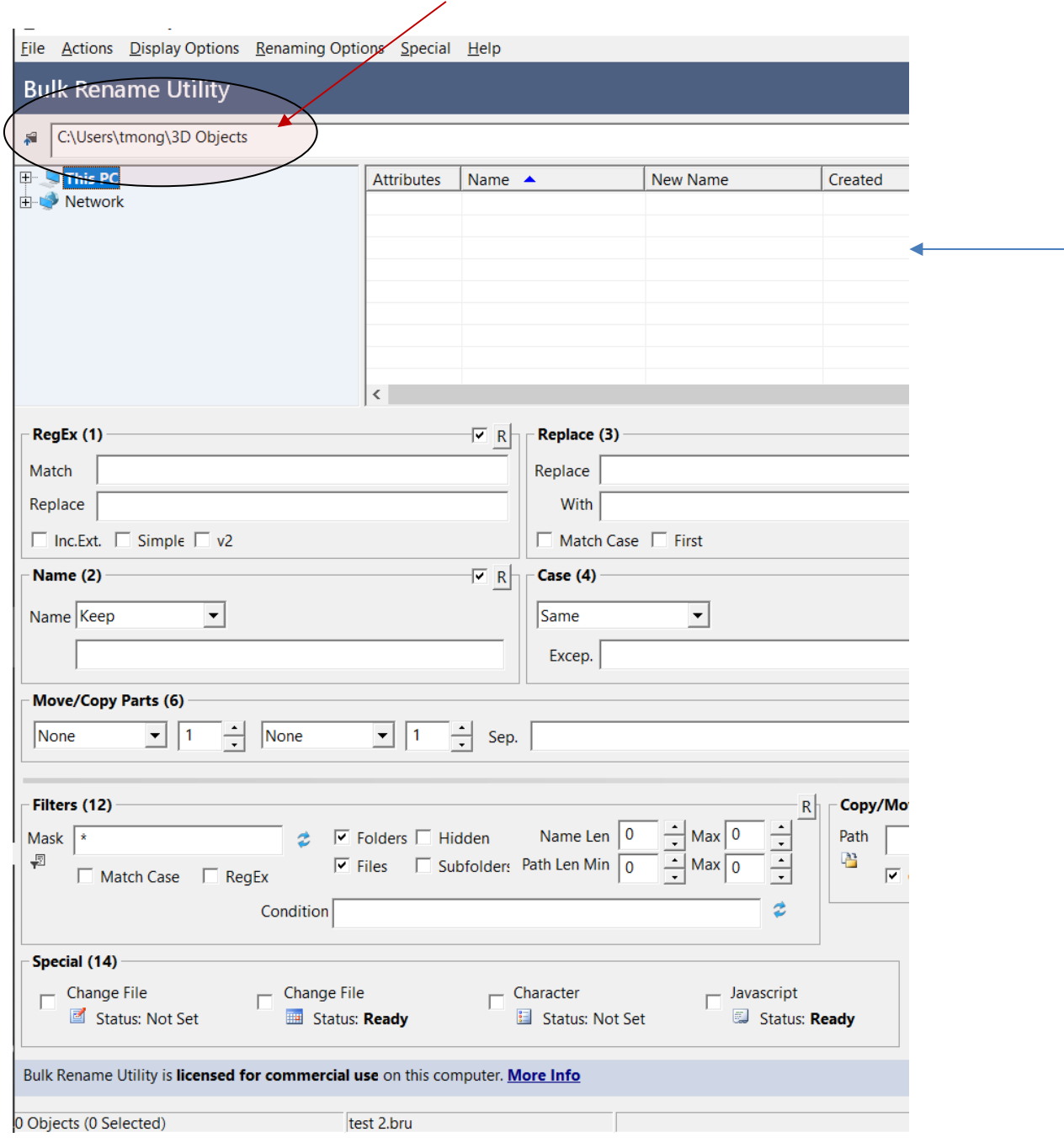
Now you can use Jump to Path to quickly navigate to a particular shared folder in BRU.

Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

Using the new UNC support for Jump to Path.

BRU's current path is set to C:\Users\tmong\3D Objects.

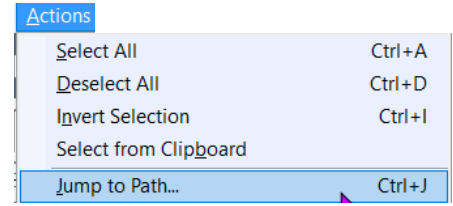


You will notice that the Content Pane is empty. I deliberately picked an empty directory to provide a visual contrast.

Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

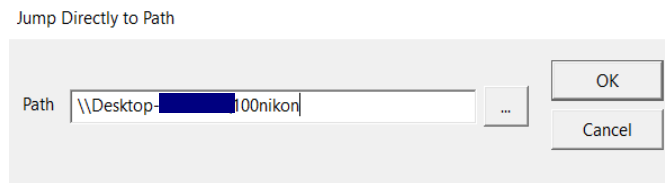
1. Anywhere in BRU, issue the command, **Ctrl + J** or use the menu item, 'Jump to Path' from the Actions menu.



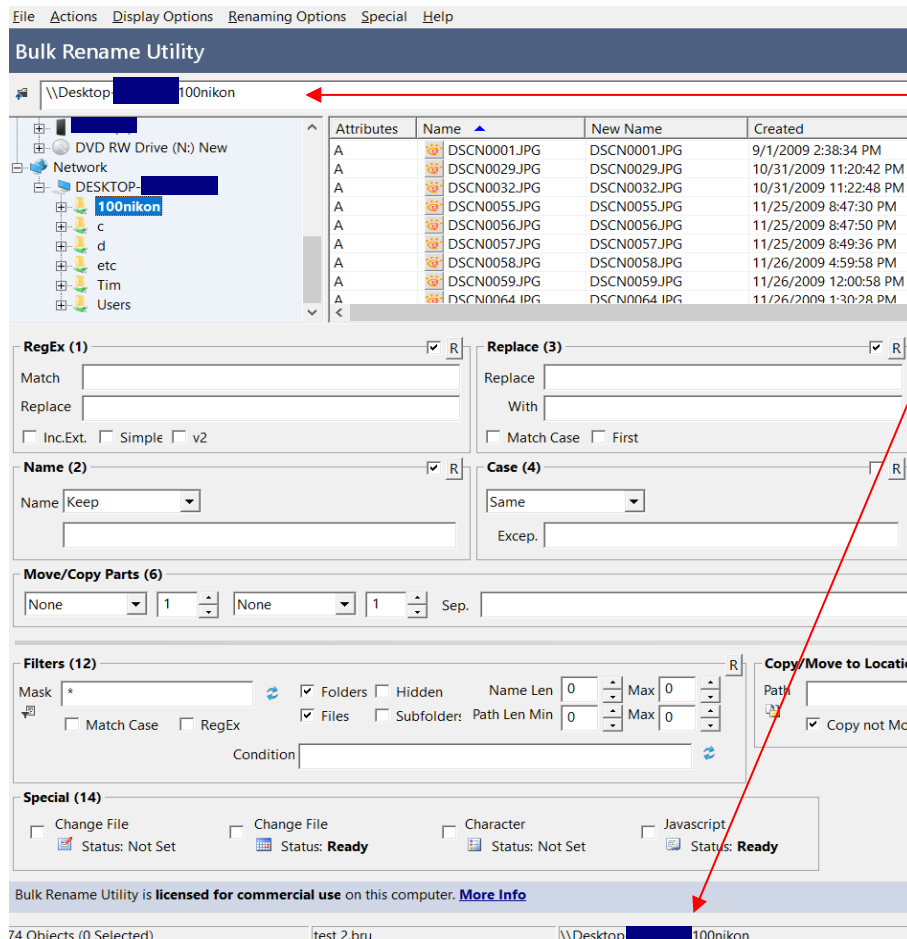
2. This initiates the Jump to Path dialog box: It will always display the current path initially.



3. Enter in your UNC path and click OK.



4. The path is changing, the Status Bar at the bottom may indicate 'Jump To Path'. Depending on how many files will need to be read, there may be a delay before the Content Pane displays the files.



a. Path bar at top will change to reflect new path: [\\Desktop-100nikon](#)

b. Status bar at bottom will also reflect this change.

c. Navigation Pane changes to new directory path and highlights current directory:

`\\100nikon`

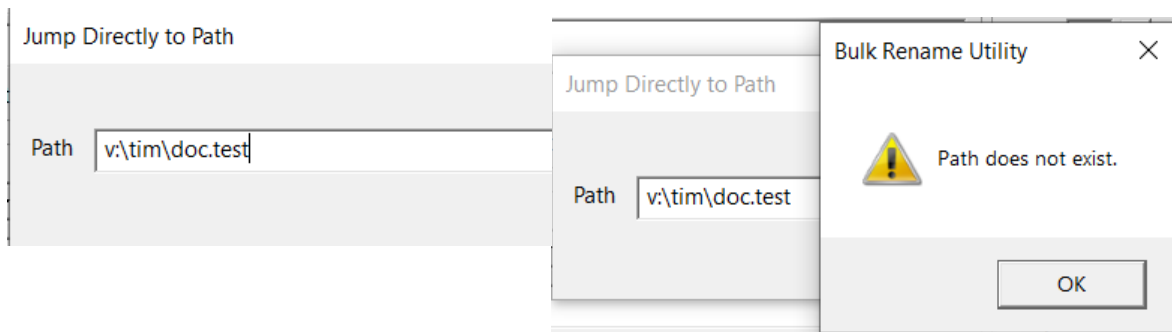
d. Content Pane will display the files, if any, contained within that directory.

Actions Menu

[Jump to Path \(Ctrl + J\): v3.4 New Additions](#)

Advantages over other methods:

1. Completely bypasses the Win32 Namespace Windows API and instead uses the File System directly. This eliminates a lot of problems with accessing Network Drives including some reported speed issues.
2. The Windows API was also limited to Max_Path and responsible for the parsed path and filename length restriction of 260 characters. This addresses one issue of BRU's non-support of Long File Paths, although still not supported in renaming operations.
3. If a Mapped drive is changed, the file is inaccessible and pointing to it via a path would produce an error:



4. The same would happen if the mapped drive was assigned a different drive designation or a user on the network changed the mapping. UNC paths usually do not change. BRU, however, still validates the existence of the UNC path.
5. A mapped drive can only be accessed by a logged on user. A UNC pathname does not require this.
6. The directory structure under a mapped drive may also be changed due to updates or repairs so as to make mapped drives impossible to maintain. UNC only requires to maintain a list of computers on the network so the user always has access to the correct resources.
7. Printers and other devices can also be addressed using UNC (not in BRU certainly).
8. UNC access can be faster than accessing through Mapped drives especially if user authentication is involved. But generally speaking, this is also due to the fact that UNC bypasses the OS.
9. Mapped drives are more convenient for the end user but a nightmare for the system administrator who has to maintain a list of all of the mapped drives on the network as well as the UNC lists.

Notes:

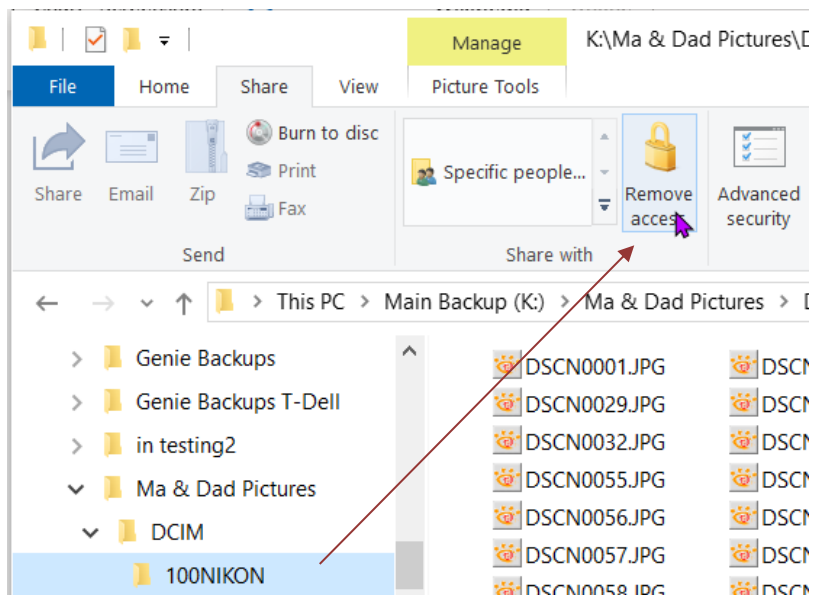
1. Please refer to Long File Paths in the Appendix of Volume II for a more detailed explanation of this.

Actions Menu

[Jump to Path \(Ctrl + J\): v3.4 New Additions](#)

Oh, by the way:

To remove the temporary share, use the 'Remove Access' button from the Share tab of Windows Explorer on the highlighted shared directory in the Navigation Pane:



Actions Menu

Jump to Path (Ctrl + J): [v3.4 New Additions](#)

This begs the question, will the UNC support work under other BRU functions? Let's find out. For this next example I will test it using 'Section #13: Copy/Move to Location':

I set up a simple rename test using 'Section #3: Replace' and specify the share for Section #13: Copy/Move to Location'.

The screenshot shows the Bulk Rename Utility interface. At the top, a file list is displayed for the path L:\0 - BRU Test\0. The list includes columns for Attributes, Name, New Name, Created, and Taken (Original). The file 'HeLIO WoRID' is highlighted, and its new name is 'HeLIO GoRID'. Below the list, several configuration panels are visible: 'RegEx (1)' with 'Replace (3)' set to 'W' and 'With G'; 'Name (2)' set to 'Keep'; 'Case (4)' set to 'Same'; 'Move/Copy Parts (6)' set to 'None'; and 'Filters (12)' with 'Copy/Move to Location (13)' selected and the path '\\Desktop-[redacted]100nikon\' specified. The 'Copy not Move' checkbox is checked.

Use the Preview button from the Actions Menu (or Press Ctrl + P)

Current Name	New Name	Action	Current Folder	New Folder
HeLIO WoRID	HeLIO GoRID	Name Change - Location Change	L:\0 - BRU Test\0\	\\Desktop-[redacted]100nikon\

Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

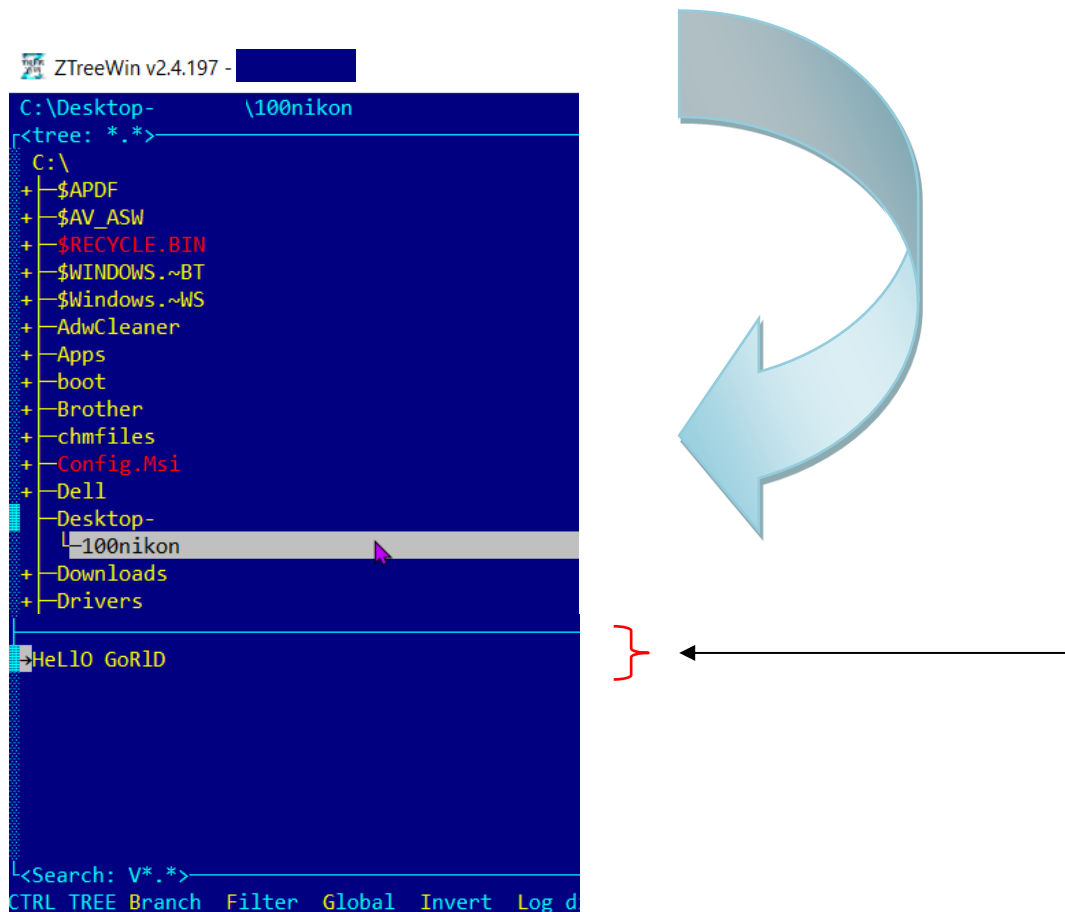
Did it work?

No.

If you look at the preview again, the UNC syntax of '\\\' was changed to: '\'.


Current Name	New Name	Action	Current Folder	New Folder
HeLIO WoRID	HeLIO GoRID	Name Change - Location Change	L:\0 - BRU Test\A\	\Desktop-[REDACTED]100nikon\

So instead of referring to a Network path, the single '\' refers to a directory off of the root. And since the directory structure, 'Desktop-[REDACTED]100nikon\' did not already exist, BRU created it and copied the renamed file over just as I had directed.



So, at least for now UNC support is limited to Jump to Path.

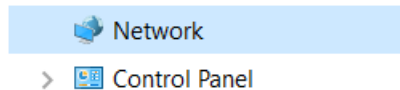
Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

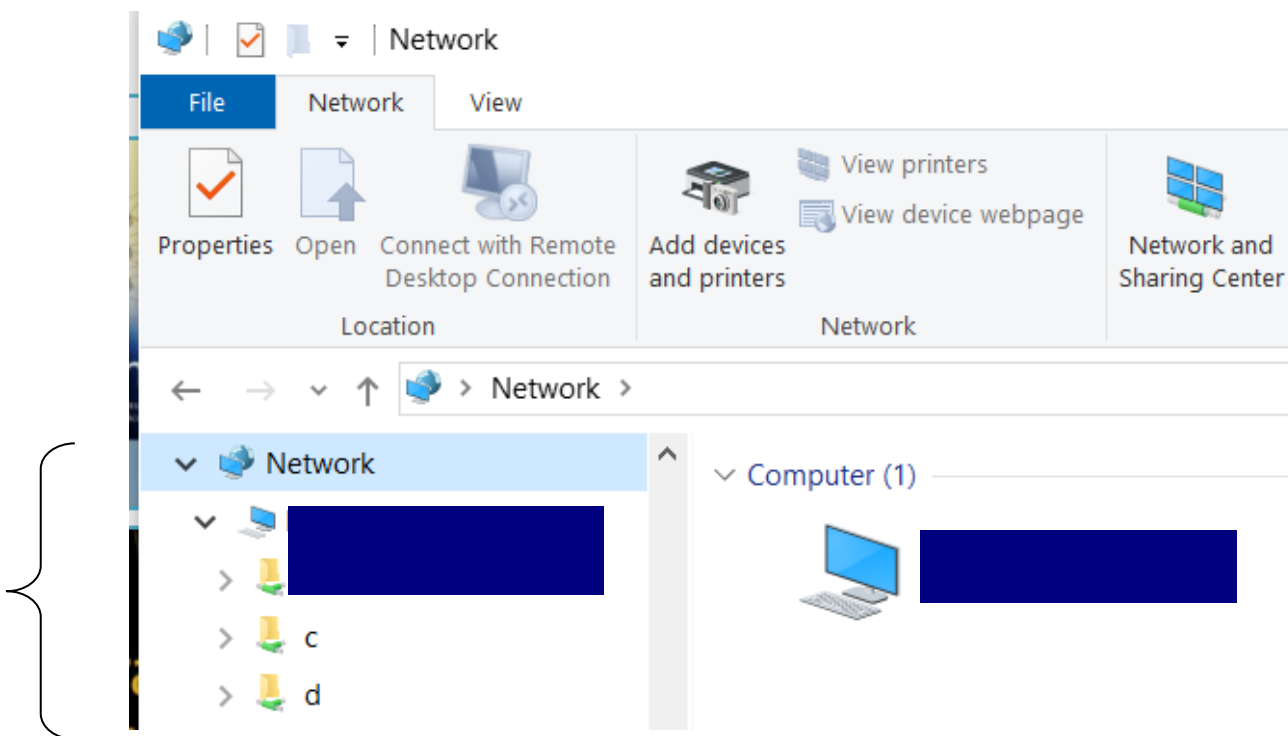
Troubleshooting UNC under Jump to Path

It should be noted that in order for the UNC path option to function properly under Windows 10, the Network Discovery under Windows Explorer must be in working order.

What I mean by this is, if you go to Network in the Tree Hierarchy of the Navigation Pane under Windows Explorer and it displays **empty**



instead of i.e., -



.. then you have a problem and UNC will not work. If you try, the directory will not change and the Content Pane will go blank. If this happens, click on the address bar and press Enter to refresh the files from the original directory to restore the file listing.

Why would this occur and why have I mentioned it at all? Because a lot of people are afflicted with this, including myself, because of some fouled up Windows Update in the past – which is my excuse, or it could be some other problem. With Windows, you just never know. Search Google to see how many mentions there are on this subject.

In a working system, Network should display any Shares along with all computers currently online and connected to your system. It may also display any networked printers or other peripherals available to the Network.

Actions Menu

Jump to Path (Ctrl + J): v3.4 New Additions

☺ Network 101 – of course make sure you have Network Discovery turned on in the first place:

In the Advanced options of the Network and Sharing Center -

Change sharing options for different network profiles

Windows creates a separate network profile for each network you use. You can choose specific options for each profile.

Private (current profile) _____ (⬆)
 Network discovery _____

When network discovery is on, this computer can see other network computers and devices and is visible to other network computers.

- Turn on network discovery ←
 Turn on automatic setup of network connected devices.
 Turn off network discovery

and turn on Sharing:

All Networks _____ (⬆)
 Public folder sharing _____

When Public folder sharing is on, people on the network, including homegroup members, can access files in the Public folders.

- Turn on sharing so anyone with network access can read and write files in the Public folders
 Turn off Public folder sharing (people logged on to this computer can still access these folders)

One sure way to know if you have the problem is to issue the ‘net view’ from an elevated command prompt.

Administrator: Command Prompt

```
Microsoft Windows [Version 10.0.18362.1016]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>net view
System error 1231 has occurred.

The network location cannot be reached. For information about network troubleshooting, see Windows Help.
```

If you get this error or something similar, welcome to the club.

Actions Menu

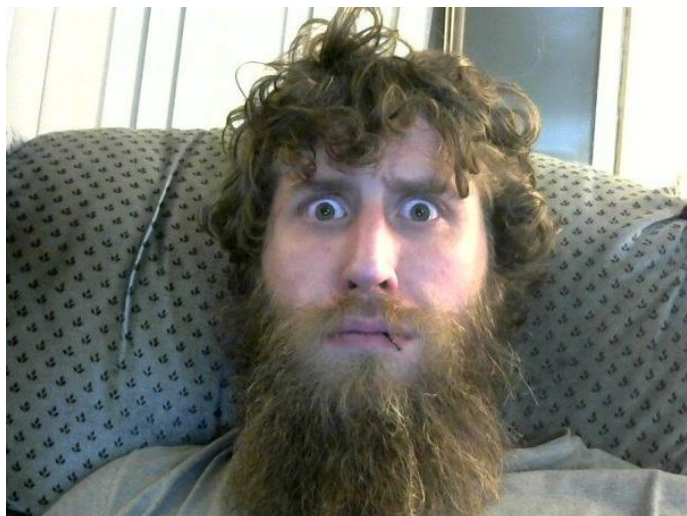
Jump to Path (Ctrl + J): v3.4 New Additions

If indeed you do have this problem then you may want to try some of the remedies out there. For myself, they didn't work. What did work is that I have both an Ethernet and Wireless connection set up. I switch the Ethernet to Wireless (pull the Ethernet cable from the router and it automatically switches over because I have both connections set up), then when the Wireless is recognized, I switch back (plug the cable back in), and it fixes it – at least temporarily until the next Windows Update usually.

If you don't have this setup, then you will have to try other alternatives.

Update January 13, 2021

Windows did another update. Guess what? Lost the Network again (along with the usual loss of my printer and sound). Tried my trick of switching from Ethernet to Wireless but the system 'learned' and this time it didn't work.



Kindly send your donations to Tim Mongeon, Home for Overworked and Unpaid Writers, Level 3 Locked Ward

FYI. This is not my photograph.

So more trial and error produced this final working solution:

Under Services:

function discovery resource publication (fdPHost)

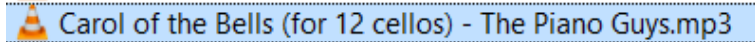
If it is not running, start it. If it is already running, restart it. Make certain you have it is set to AUTOMATIC. On my system, it was already running so I restarted it and once more problem fixed (temporarily). Can't wait until the next Windows Update so it gets screwed with some more, and perhaps that insidious Windows AI, getting far more intelligent than I can keep up with in fixes, will shut my Network folder down again.

This does not fix the above System Error, e.g., 1231, but you can ignore it if your Network comes back in Windows Explorer because now the UNC path will work in Jump to Path.

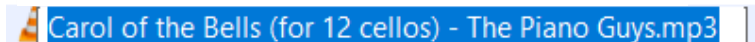
Actions Menu

Rename Object Manually (F2)

For this to work, you have to have only one file/folder selected in the Content Pane. This is the standard F2 Windows Explorer function. Press Esc to cancel the rename.

A screenshot of a file name 'Carol of the Bells (for 12 cellos) - The Piano Guys.mp3' inside a text input field. The text is highlighted in blue, and a small orange bell icon is visible on the left side of the input field.

Now you can rename the file by typing. Click ENTER to save, or just move to a different filename.

A screenshot of a file name 'Carol of the Bells (for 12 cellos) - The Piano Guys.mp3' inside a text input field. The text is highlighted in blue, and a small orange bell icon is visible on the left side of the input field.

Notes:

1. If you are in 'Recursive mode', ('Sub Folders' enabled in '[Section #12: Filters](#)') the File List will be refreshed.

Refresh Files (F5)

This will refresh the Content Pane.

Refresh Tree (Ctrl + F5)

This will refresh the Navigation Pane. Note that you can also refresh the contents of the selected branch by collapsing and re-expanding the branch.

Show/Hide Tree (F11)

This will hide or show the Tree Hierarchy (the Navigation Pane). This is useful if you are performing a great amount of renaming in a single folder. The "tree" can be removed to give you more space in which to work.

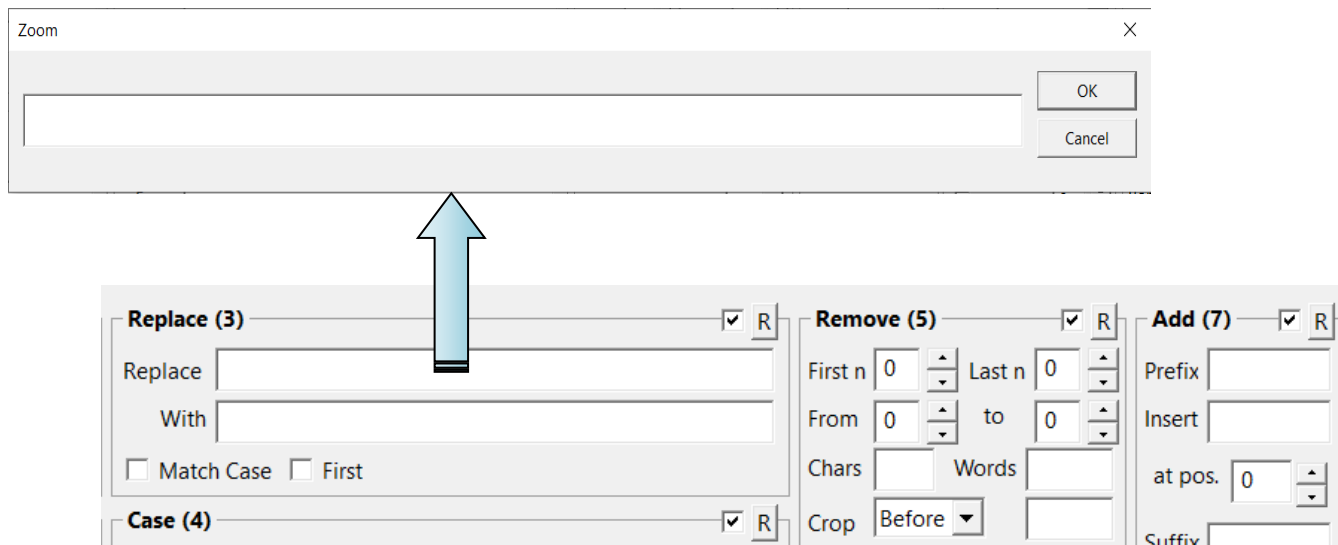
Notes:

1. The tree will always be visible at program startup, even if it was hidden when you closed the program.
2. I refer to the tree as the 'Tree Hierarchy' because it is an Object Tree. An Object Tree displays hierarchical objects in a tree view.

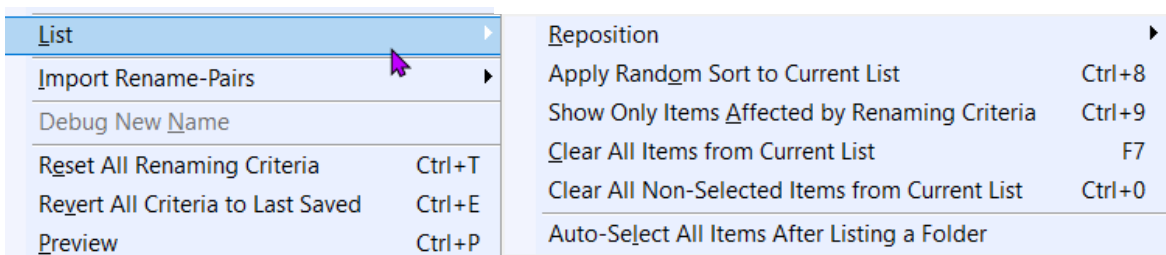
Actions Menu

Zoom (F8)

This will display a larger data entry field for easier editing, if you press **F8** while in most of the data entry fields of the criteria sections. #1 - #13.



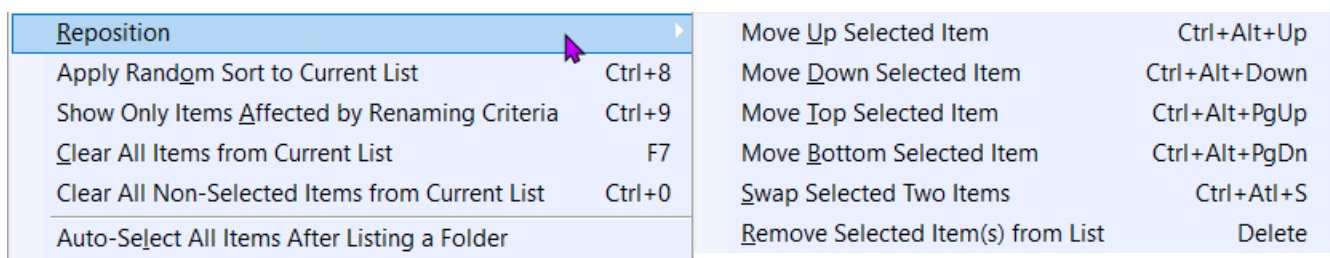
List



The List command acts on the files and folders listed in the Content Pane. The content presented in the pane is often referred to as the 'File List'.

- Reposition

Allows you to reposition/reorganize items in the File List, useful for renumbering. Reposition offers several sub-commands.

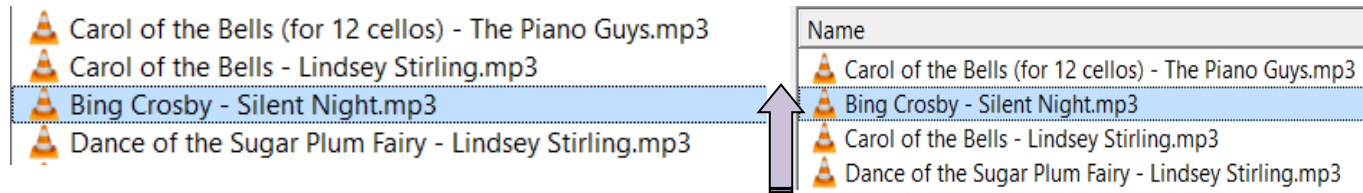


Actions Menu

- Reposition

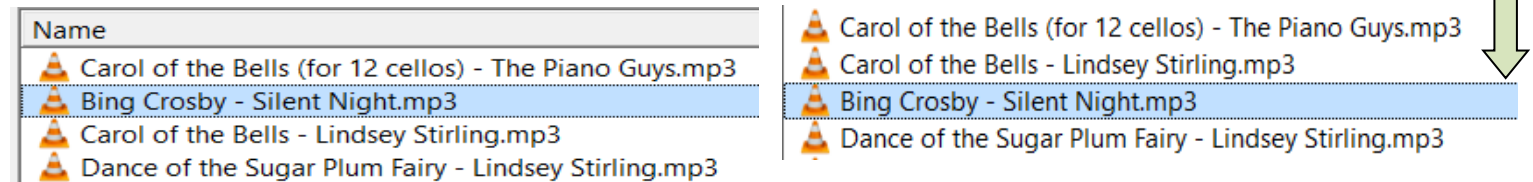
Move Up Selected Item (Ctrl + Alt + Up Arrow)

Moves the selected item *up* one line position in the list. For smooth, rapid movement, use the Keyboard Shortcuts.



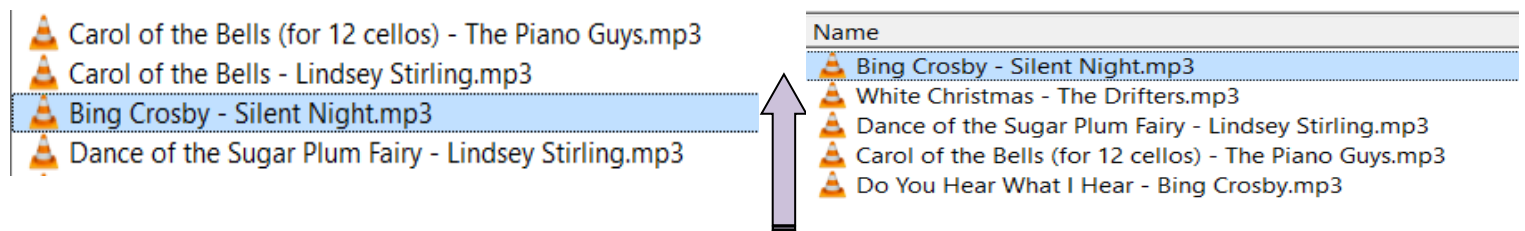
Move Down Selected Item (Ctrl + Alt + Down Arrow)

Moves the selected item *down* one line position in the list. For smooth, rapid movement, use the Keyboard Shortcuts.



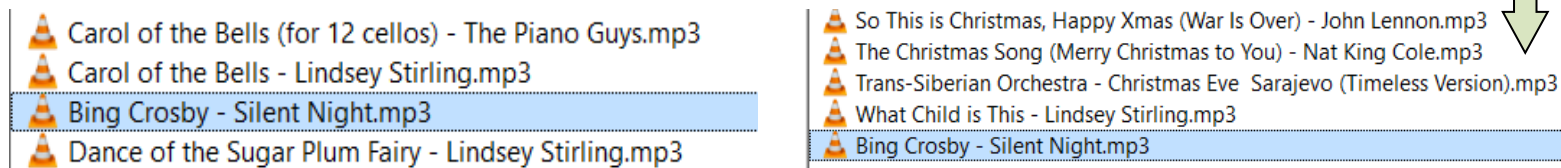
Move Top Selected Item (Ctrl + Alt + PgUp)

Moves the selected item to the *top* of the list. For smooth, rapid movement, use the Keyboard Shortcuts



Move Bottom Selected Item (Ctrl + Alt + PgDn)

Moves the selected item to the *bottom* of the list. For smooth, rapid movement, use the Keyboard Shortcuts

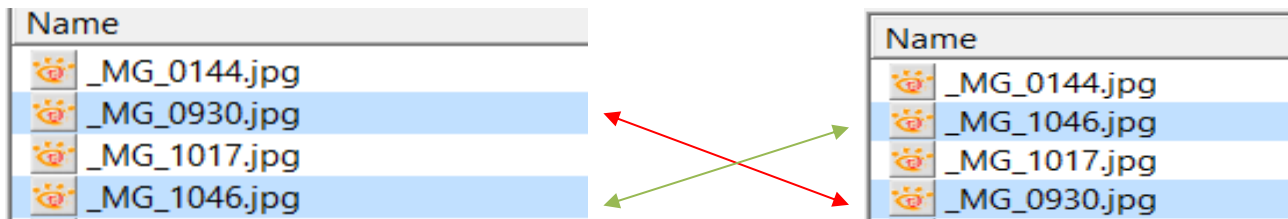


Actions Menu

- *Reposition*

Swap Two Selected Items (Ctrl + Alt + S)

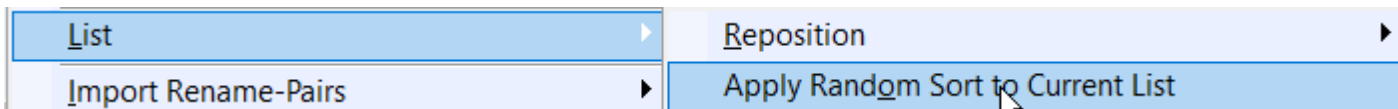
Repositions two selected items each replacing the other's *position* in the list.



Remove From List (Delete key)

This removes the file from the list, BUT does not delete the actual file. If you wish to delete the physical file, right click on the file to bring up the context menu and select 'Delete'.

- *Apply Random Sort to Current List (Ctrl + 8)*



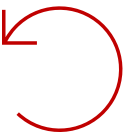
This allows you to sort the file list in a random sequence. Useful if you want to create a slideshow in a random display-sequence.

Before –

Name ▲
Becca Moving Sun Faded.jpg
Becca Purple Vignette.jpg
Becca QT Distorted.jpg
Becca QT FI Chalk.jpg
Becca Water Color.jpg
Becca Water Reflection.jpg
Belvedere Plantation 6404.jpg
Belvedere Plantation 6472.jpg
Belvedere Plantation 6474 inn

After -

Name ▲
Pinewood Derby_2010-13.jpg
Smithsonian Zoo 279.jpg
Belvedere Plantation 6404.jpg
Copy of North Adams October 2008 019 copy.jpg
Zachary 5693.jpg
Bubbles04.jpg
Copy of North Adams October 2008 021.jpg
Tomb of Unknown Soldiers Guard 4.jpg

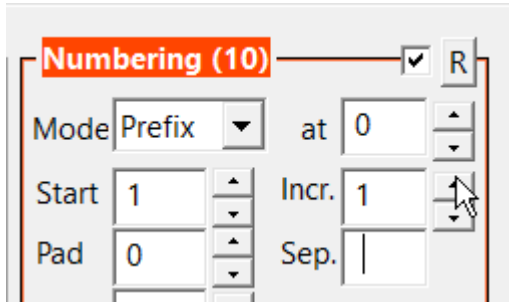


Actions Menu

- Apply Random Sort to Current List

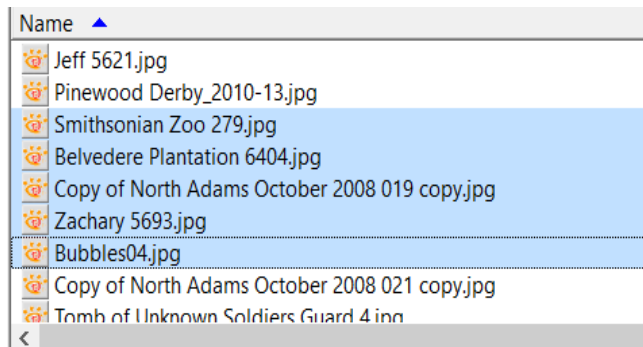
Once you've sorted in random order, apply a numeric auto-number prefix to keep the list in that order.

example -



‘[Section #10: Numbering](#)’, showing Prefix selected with Increment set at one and an added <space> separator.

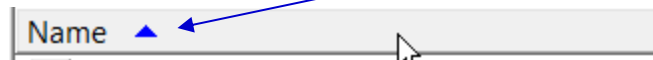
Before:



After:



Order can be restored if desired by re-sorting the list by ‘Name’. Click on the ‘Name’ Title bar.

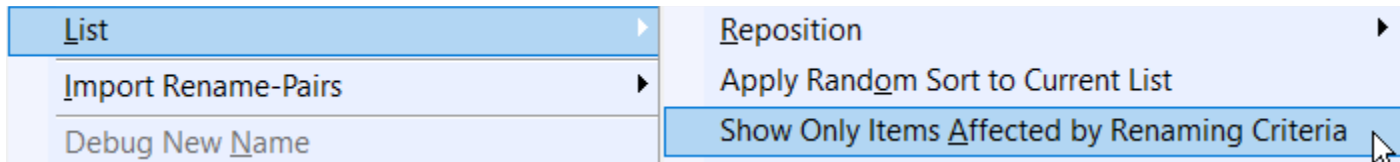


Notes:

1. If you use the keyboard shortcuts, you can move the file quickly and easily rather than using the menu requiring clicking on the item for each instance.
2. For more information, refer to ‘[Section #10: Numbering](#)’.

Actions Menu

- Show Only Items Affected by Renaming Criteria (Ctrl + 9)



This allows you to only display those files that will be renamed using the criteria. Especially useful when you have a large list. Select files first before applying.

Example:

Here is a list of various files. I have selected (highlighted) only a few files for this illustration.

Before:

Name ▲	New Name
2012-01-20_193417.png	2012-01-20_193417.png
2012-01-20_193825.png	2012-01-20_193825.png
2012-01-20_194059.png	2012-01-20_194059.png
_MG_0144.jpg	_TM_0144.jpg
_MG_0930.jpg	_TM_0930.jpg
_MG_1017.jpg	_TM_1017.jpg
_MG_1046.jpg	_TM_1046.jpg
_MG_1051.jpg	_TM_1051.jpg
MG_1059.ina	MG_1059.ina

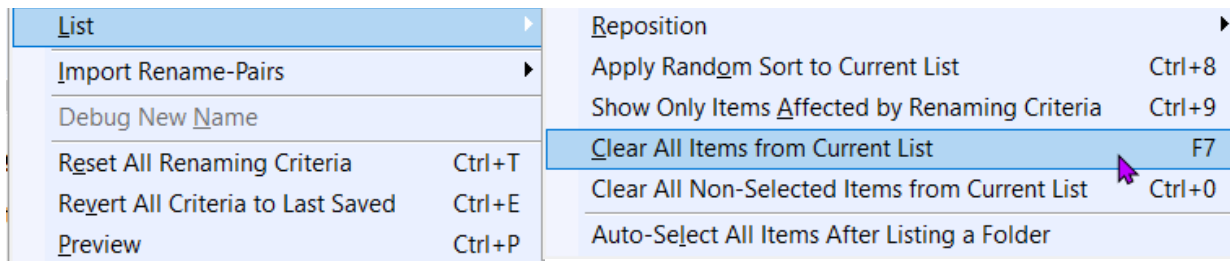
To limit the scope of the list and only view those files affected by the current criteria...

After:

Name ▲	New Name
_MG_0144.jpg	_TM_0144.jpg
_MG_0930.jpg	_TM_0930.jpg
_MG_1017.jpg	_TM_1017.jpg
_MG_1046.jpg	_TM_1046.jpg
_MG_1051.jpg	_TM_1051.jpg

Actions Menu

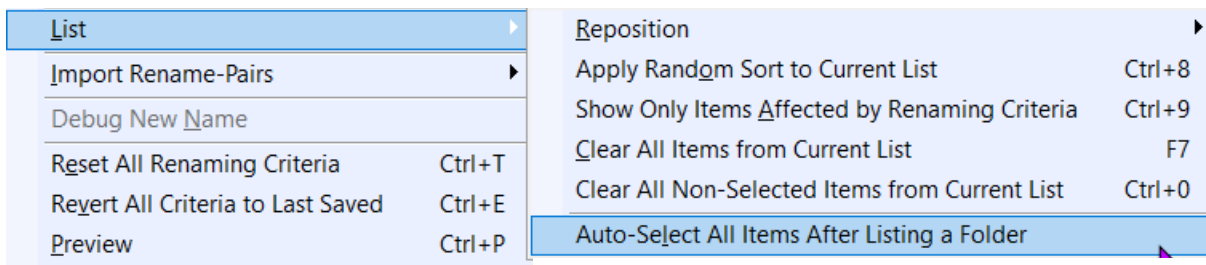
- Clear All Items from Current List



Clears 'all' items from list (Content Pane), not just selected. Files are removed from view and not actually deleted. To restore view, use F5 or 'Refresh Files' from 'Actions Menu'.

- Auto-Select All Items After Listing a Folder

All files and folders in the Content Pane are selected automatically whenever the list of files/folders is built.



Name	New Name
flac.txt	tack.txt
flac113b(tools).exe	tack113b(tools).exe
FLAC_Comment_Editor v1.10.zip	tack_Comment_Editor v1.10.zip
http_mcplugins.sourceforge.net_enc_flac.html.JPG	http_mcplugins.sourceforge.net_enc_tack.html.JPG
http_mcplugins.sourceforge.net_in_flac.html.JPG	http_mcplugins.sourceforge.net_in_tack.html.JPG
http_omniencoder.autobotcity.net.JPG	http_omniencoder.autobotcity.net.JPG
http_wiki.hydrogenaudio.org_index.php_title_EAC_and_Flac.JPG	http_wiki.hydrogenaudio.org_index.php_title_EAC_and_tack.JPG
http_wiki.hydrogenaudio.org_index.php_title_WMPTSE__How_to_change...	http_wiki.hydrogenaudio.org_index.php_title_WMPTSE__How_to_change_t...
http_wiki.slimdevices.com_index.cgi_EACBeginners.JPG	http_wiki.slimdevices.com_index.cgi_EACBeginners.JPG
http_www.losslessaudioblog.com_wmmnce_lossless_guide.JPG	http_www.losslessaudioblog.com_wmmnce_lossless_guide.JPG

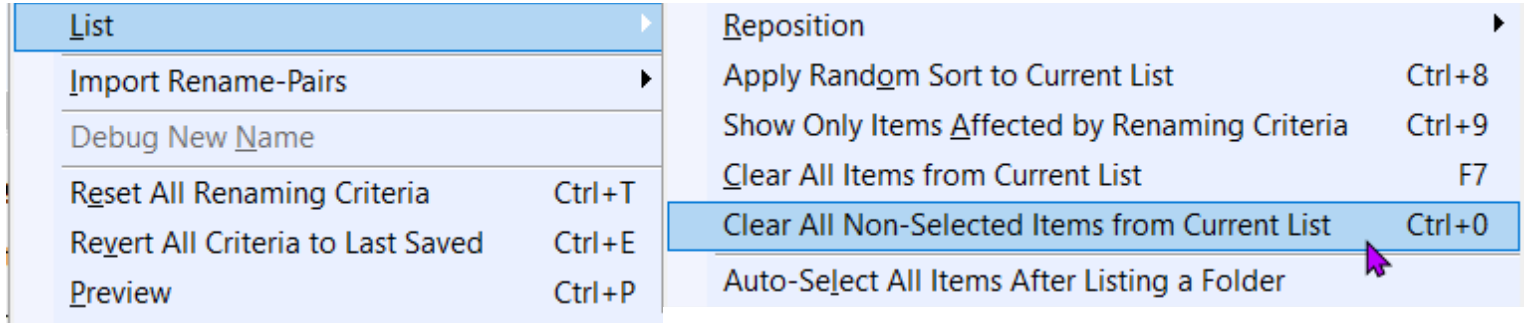
Notes:

1. Applies to:
 - a. When the BRU program starts.
 - b. When the Files are refreshed using F5 or 'Refresh Files' from Actions Menu.
 - c. When changing to another folder in the Navigation Pane. The files in the Content Pane will be selected for the changed folder.
2. Any criteria set will be applied to the selected files. New Name will reflect any changed (qualified) filenames. This function is a convenience that can save a little time.

Actions Menu

v3.4 New Additions

- Clear All Non-Selected Items from Current List (Ctrl + 0)



Clears 'all' non-selected items (without highlight) from list (Content Pane). Files are removed from view and not actually deleted. To restore view, use F5 or 'Refresh Files' from 'Actions Menu'.

Before:

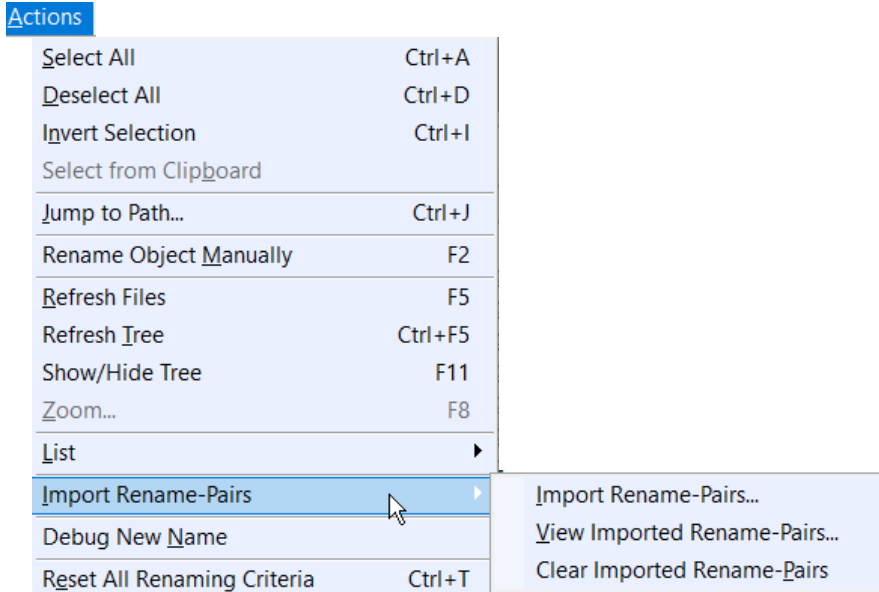
Attributes	Name ▲	New Name	Created	Taken (Original)
A	DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM
A	DSCN0032.JPG	DSCN0032.JPG	10/31/2009 11:22:48 PM	10/31/2009 5:22:48 PM
A	DSCN0055.JPG	DSCN0055.JPG	11/25/2009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

After:

Attributes	Name ▲	New Name	Created	Taken (Original)
A	DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM
A	DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	10/31/2009 5:20:43 PM
A	DSCN0032.JPG	DSCN0032.JPG	10/31/2009 11:22:48 PM	10/31/2009 5:22:48 PM
A	DSCN0055.JPG	DSCN0055.JPG	11/25/2009 8:47:30 PM	11/25/2009 3:47:32 PM
A	DSCN0056.JPG	DSCN0056.JPG	11/25/2009 8:47:50 PM	11/25/2009 3:47:50 PM

Actions Menu

Import Rename-Pairs (Rename from a Text File)



You can perform a rename action using a text file containing the files to be renamed in this format:

<original name>|<new name>

One entry per line.

where:

the **<original name>** is separated from the **<new name>** using a pipe character ' | '.

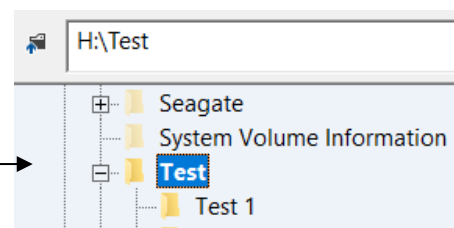
Pipe character is found on most keyboards as the key above the backslash character.

e.g.

```
Track001.mp3|Headlong.mp3
Track002.mp3|Rushes.mp3
TRACK003.mp3|AnywhereIs.mp3
```

- Import Rename-Pairs

1. Navigate to the directory containing the files to be renamed, e.g., →
2. From the Actions Menu select, 'Import Rename-Pairs'.
3. Load in the text file from the, 'Open File', dialog box.
 - a. Select your files in the Name column.
 - b. If successful, the New Name column will contain a preview of the renamed files.



Actions Menu

- Import Rename-Pairs cont.

Example,

I have a text file called do.txt containing the following filenames taken from my Test directory:

```
Belvedere Plantation 6474.jpg|test1.jpg
Belvedere Plantation 6476.jpg|test2.jpg
Belvedere Plantation 6479.jpg|test3.jpg
Belvedere Plantation 6488.jpg|test4.jpg
Belvedere Plantation 6490.jpg|test5.jpg
```

Once loaded, and the files selected, results in:

Name	New Name
Belvedere Plantation 6474.jpg	test1.jpg
Belvedere Plantation 6476.jpg	test2.jpg
Belvedere Plantation 6479.jpg	test3.jpg
Belvedere Plantation 6488.jpg	test4.jpg
Belvedere Plantation 6490.jpg	test5.jpg
Belvedere Plantation 6492.jpg	Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	Belvedere Plantation 6508.jpg

Notes:

1. The text file cannot contain the file paths. It won't work. Therefore you are limited to having all of the files referenced in the Rename-Pairs listing reside in the same directory (another suggestion I have made to TGRMN).
2. The text file can reside anywhere and does not have to be in the same directory as the intended files.
3. The file only tells the program what will be renamed. You still have to use the Navigation Pane to select the directory containing the files.
4. Case is ignored when BRU is comparing filenames in the text file to the filenames in the folder.
5. Use the 'Clear Imported-Pairs' command from this sub-menu to remove the text file entries from the Content Pane.
6. BRU will read ANSI and Unicode (UTF-16) text files.
7. The Imported rename-pairs can be viewed using 'View Imported Rename Pairs' from this sub-menu.
8. When the file is loaded, BRU will reflect this by placing a RED notice next to the program info:

Rename Pairs Imported	Bulk Rename Utility 64-bit (Unicode) TGRMN Software 2001-2019 Version: 3.3.1.0 Check for Updates
--------------------------------------	---

9. Double-Clicking on the RED notice above will bring up the 'View Imported Rename Pairs' window.

Actions Menu

- Import Rename-Pairs cont.

Directory List Method

To make the listing itself, you need a program that can create a text file from a directory. You may need to look around to find one. Even so, once you have the listing, you will need to add the pipe character. This can be done by loading the list into a word processor or text editor and use that program's find and replace, specifying the '^p' character (paragraph) as the Find and '|^p' as the Replace.

Spreadsheet Program Method

Here's a method for using a Spreadsheet program to create a Renamed-Pairs list. It was originally meant for Volume II, but I decided it would be a better fit in Volume I. The idea was taken from the BRU forums by the Contributor, KenP with added material by myself.

Contents of old.txt (old filenames)

```
3052503155104.Bourjois_FDTAirMat_01_3
AWOLNATION - Run (B52 Remix)_217279053 - B52
b&B_296170807 - psupac
[Word1-Word2] 2010
[Word] 2006
Word1-Word2.com - 2009 r
```

Contents of new.txt (new filenames)

```
3052503155104.Bourjois
AWOLNATION - Run (B52 Remix)
B_296170807
[Word1-Word2] 2019
[Word] 2008
Word1-Word2.com - 2010
```

In Excel or other Spreadsheet program enter the following into row 1:

	A	B	C	D	E	F	G	H
1	"	<paste old filenames>	"		"	<paste new filenames>	"	=A1&B1&C1&D1&E1&F1&G1
2		3052503155104.Bourjois	"		"	3052503155104.Bourjois	"	=A1&B1&C1&D1&E1&F1&G1
3		AWOLNATION - Run (B52 Remix)_217279053 - B52				AWOLNATION - Run (B52 Remix)		
4		b&B_296170807 - psupac				B_296170807		
5		[Word1-Word2] 2010				[Word1-Word2] 2019		
6		[Word] 2006				[Word] 2008		
7		Word1-Word2.com - 2009 r				Word1-Word2.com - 2010		

Column H contains a formula. When you press ENTER after entering the formula or you move to another cell it will evaluate the formula giving you this:

old name		new name
H		
"3052503155104.Bourjois_FDTAirMat_01_3" "3052503155104.Bourjois"		

This is the result you want for each of the words in your word list. To get this formatting, copy the cell data for each cell that does not contain a word list, beginning with cell A1, down to the end of the word list A6.

Actions Menu

- Import Rename-Pairs cont.

e.g.,

Copy each value in the columns, **A – H** down to the end of the word list, but do not include cell **B** or cell **F** because these contain the word lists.

1. Place cursor on cell **A1** that contains the “ ” and press Ctrl + C to copy the cell into the clipboard.

	A
1	"
2	
3	
4	
5	
6	

2. Move down to cell **A2** (the next cell down) and select the cells **A2** through **A6** using the Down arrow.

	A
1	"
2	"
3	"
4	"
5	"
6	"

3. Press Ctrl + V to paste
4. Repeat this for cells **C1**, **D1**, **E1** and **G1**.

5. Move to cell **H1** that contains the formula.
 - a. If the formula is displayed, click on cell **H1** and press <Enter> to get the text back.

H
"3052503155104.Bourjois_FDTAirMat_01_3" "3052503155104.Bourjois"

6. Copy cell H1 using Ctrl + C. Move down to cell **H2** and select through cell **H6**. Paste as you did with other cells.

	A	B	C	D	E	F	G	H	I
1	"	3052503155104.Bourjois	"		"	3052503155104.Bourjois	"	"3052503155104.Bourjois_FDTAirMat_01_3" "3052503155104.Bourjois"	
2	"	AWOLNAT	"		"	AWOLNAT	"	"AWOLNATION - Run (B52 Remix)_217279053 - B52" "AWOLNATION - Run (B52 Remix)"	(B52 Remix)"
3	"	b&B_2961	"		"	B_296170807	"	"b&B_296170807 - psupac" "B_296170807"	
4	"	[Word1-W	"		"	[Word1-W	"	"[Word1-Word2] 2010" " [Word1-Word2] 2019"	
5	"	[Word] 20	"		"	[Word] 20	"	"[Word] 2006" " [Word] 2008"	
6	"	Word1-W	"		"	Word1-W	"	"Word1-Word2.com - 2009 r" "Word1-Word2.com - 2010"	

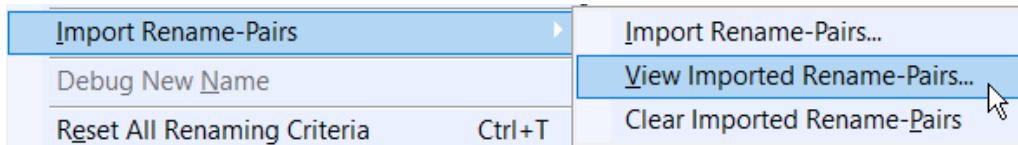
7. Select cell **H1** and including cell **H1**, select down to cell **H6**.

H
"3052503155104.Bourjois_FDTAirMat_01_3" "3052503155104.Bourjois"
"AWOLNATION - Run (B52 Remix)_217279053 - B52" "AWOLNATION - Run (B52 Remix)"
"b&B_296170807 - psupac" "B_296170807"
"[Word1-Word2] 2010" " [Word1-Word2] 2019"
"[Word] 2006" " [Word] 2008"
"Word1-Word2.com - 2009 r" "Word1-Word2.com - 2010"

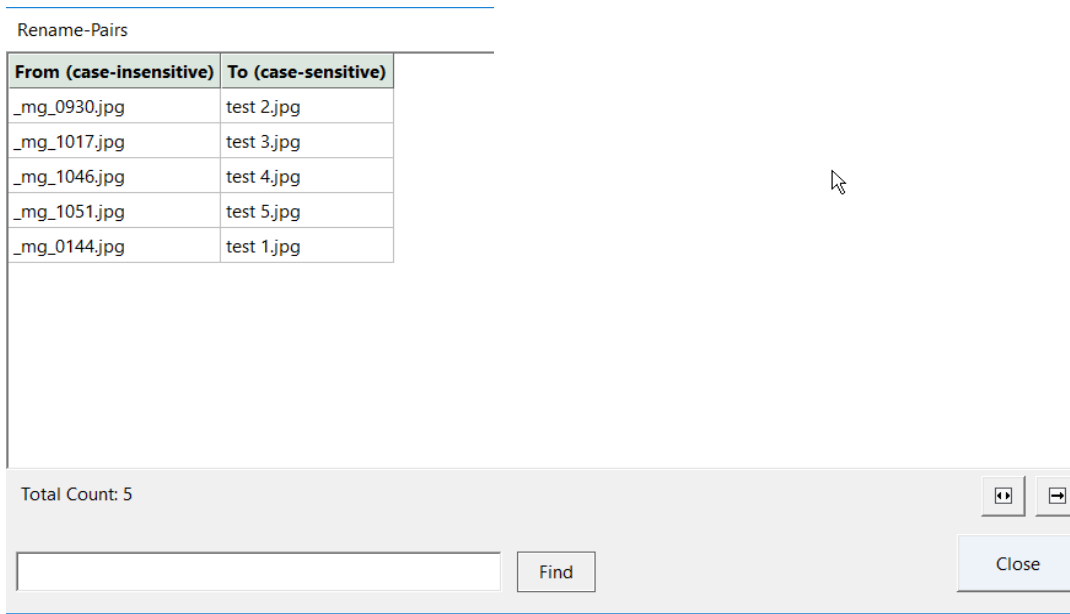
8. Copy and paste this data into a text file and you now have your list.

Actions Menu

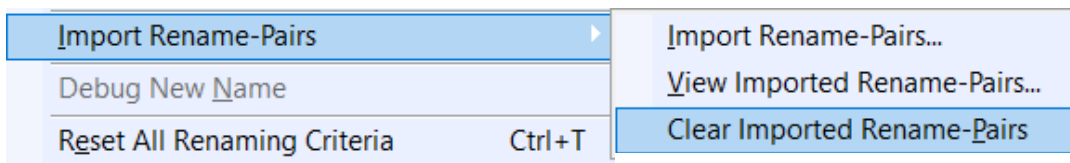
- View Imported Rename Pairs



If you have already imported the Import-Rename Pairs text file, you can see the contents by selecting this option. I reduced the file down to 5 items for this example.



- Clear Imported-Pairs



Unloads the 'Imported Rename-Pairs' file from BRU, and clears the data used for 'New Name'.

Loaded

Name	New Name	Name	New Name
_MG_0144.jpg	test 1.jpg	_MG_0144.jpg	_MG_0144.jpg
_MG_0930.jpg	test 2.jpg	_MG_0930.jpg	_MG_0930.jpg
_MG_1017.jpg	test 3.jpg	_MG_1017.jpg	_MG_1017.jpg
_MG_1046.jpg	test 4.jpg	_MG_1046.jpg	_MG_1046.jpg
_MG_1051.jpg	test 5.jpg	_MG_1051.jpg	_MG_1051.jpg
_MG_1059.jpg	_MG_1059.jpg	_MG_1059.jpg	_MG_1059.jpg
_MG_1068.jpg	_MG_1068.jpg	_MG_1068.jpg	_MG_1068.jpg

Unloaded

Actions Menu

[Import Rename-Pairs \(Rename from a Text File\): v3.4 New Additions](#)

Full Path Support

BRU supports full paths in the Rename Pairs text file. This means that you are no longer restricted to having your source files in one directory **BUT** you *are* restricted to having the files **within a single directory structure**.

You can't have your source files reside anywhere and expect BRU will include them. This feature will hopefully be made available in a future update – I keep pushing for this.

Note that the files cannot contain UNC paths. These only work under the Jump to Path feature.

So what do I mean by a single directory structure? Remember the discussion about Directory Levels? You have a main directory with sub-directories beneath. BRU now supports having the source files, not just in a single directory, but in any directory that branches off from it.

Using my example from previous, I have moved the files to be renamed all within a single directory structure:



Test is the main or Root directory of this structure with the sub-directories, \Test 1, \Test 2, and \Test 3.

here are the contents of each directory -

..\Test

[Belvedere Plantation 6479.jpg](#)

..\Test\Test 1

[Belvedere Plantation 6474.jpg](#)

\Test\Test 2

[Belvedere Plantation 6488.jpg](#)
[Belvedere Plantation 6490.jpg](#)

..\Test\Test 3

[Belvedere Plantation 6476.jpg](#)

Actions Menu

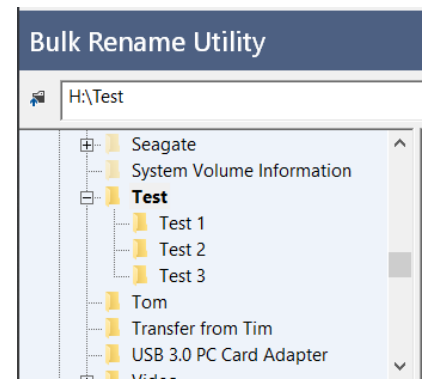
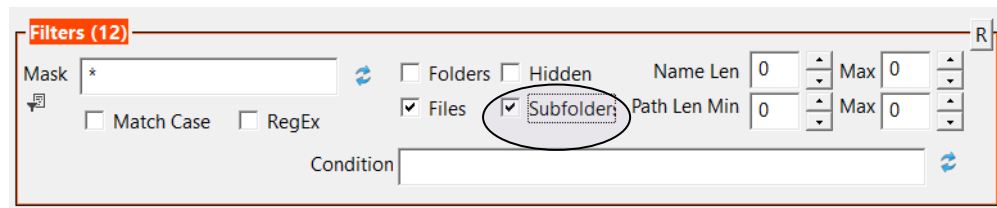
Import Rename-Pairs (Rename from a Text File): v3.4 New Additions

This is the contents of the Import Rename-Pairs text file, do.txt -

```
H:\Test\Test 1\Belvedere Plantation 6474.jpg|test2.jpg
H:\Test\Test 2\Belvedere Plantation 6488.jpg|test3.jpg
H:\Test\Test 2\Belvedere Plantation 6490.jpg|test4.jpg
H:\Test\Test 3\Belvedere Plantation 6476.jpg|test5.jpg
Belvedere Plantation 6479.jpg|test1.jpg
```

The last entry represents the current directory. The current directory is H:\Test and it contains two files, the do.txt file that contains the above file list and the last file entry, Belvedere Plantation 6479.jpg. I could just as easily have used the full path, H:\Test\ Belvedere Plantation 6479.jpg|test1.jpg, but I wanted to demonstrate that you can have both representations in the same Import Rename-Pairs file.

1. Navigate to the Main or root directory, e.g., Test, of your directory structure. →
2. Turn on Recursion so that all of the files display in the directory structure. ↻



Displays as:

Name	New Name	Sub Dir.
Belvedere Plantation 6474.jpg	Belvedere Plantation 6474.jpg	\Test 1
Belvedere Plantation 6476.jpg	Belvedere Plantation 6476.jpg	\Test 3
Belvedere Plantation 6479.jpg	Belvedere Plantation 6479.jpg	
Belvedere Plantation 6488.jpg	Belvedere Plantation 6488.jpg	\Test 2
Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg	\Test 2
do.txt	do.txt	







3. View the Imported List:

Rename-Pairs	
From (case-insensitive)	To (case-sensitive)
belvedere plantation 6479.jpg	test1.jpg
h:\test\test 3\belvedere plantation 6476.jpg	test5.jpg
h:\test\test 2\belvedere plantation 6490.jpg	test4.jpg
h:\test\test 2\belvedere plantation 6488.jpg	test3.jpg
h:\test\test 1\belvedere plantation 6474.jpg	test2.jpg

Actions Menu







Import Rename-Pairs (Rename from a Text File): v3.4 New Additions

4. Select all of the files that will be renamed.

Name ▲	New Name	Sub Dir.
 Belvedere Plantation 6474.jpg	test2.jpg	\Test 1
 Belvedere Plantation 6476.jpg	test5.jpg	\Test 3
 Belvedere Plantation 6479.jpg	test1.jpg	
 Belvedere Plantation 6488.jpg	test3.jpg	\Test 2
 Belvedere Plantation 6490.jpg	test4.jpg	\Test 2
 do.txt	do.txt	

5. Click on the Rename button.

Files have been renamed according to the do.txt Imported Rename-Pairs list.

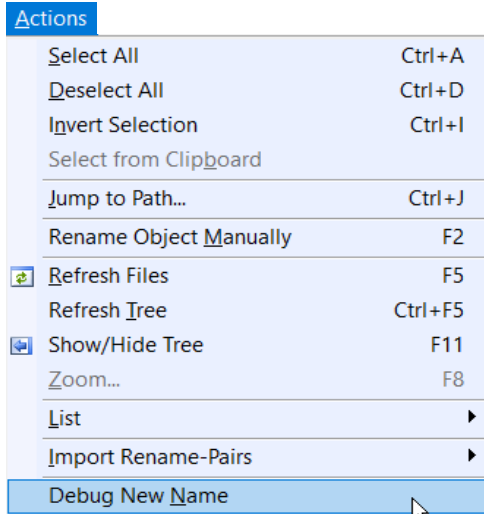
Name ▲	New Name	Sub Dir.
 test2.jpg	test2.jpg	\Test 1
 test5.jpg	test5.jpg	\Test 3
 test1.jpg	test1.jpg	
 test3.jpg	test3.jpg	\Test 2
 test4.jpg	test4.jpg	\Test 2
 do.txt	do.txt	

Notes:

1. Although do.txt is part of the selected files list in step 4 above, it will not be renamed. If it was to be affected, New Name would reflect any changes.
2. Remember that you can select what you want, and only those files that are subject to the criteria entered will be altered.
3. It is good, though, to be careful in your selections to avoid including unintended files that may result in unintended consequences. This has been a Tim Confucius moment.

Actions Menu


Debug New Name



This menu item is available only if you have one single file or folder selected, the purpose of which is to demonstrate how the final New Name was derived. Upon selecting, you are presented with a message box that lists each of the 11 criteria as they were evaluated and their effect, if any, on the new file name. JavaScript, if part of this evaluation, would have its own 12th entry at the end of the listing. This is created ‘on-the-fly’ and cannot be saved as a file.

This is useful when the result is unexpected and helps assist with the debugging process, or in other words, ‘How the hell did I end up with this mess?’

Example,

 Belvedere Plantation 6488.jpg

test 20-01-2012 cat B E D L E M plnion 6488 001.jpg

Filename Generation Debug Results



Original name: Belvedere Plantation 6488.jpg
Components: Name=Belvedere Plantation 6488, Extension=.jpg

Finished Group 1. Name=Belvedere Plantation 6488, Extension=.jpg
 Finished Group 2. Name=Belvedere Plantation 6488, Extension=.jpg
 Finished Group 3. Name=BELVEDLEM Plantation 6488, Extension=.jpg
 Finished Group 4. Name=BELVEDLEM Plantation 6488, Extension=.jpg
 Finished Group 5. Name=BEDLEM Plnion 6488, Extension=.jpg
 Finished Group 6. Name=BEDLEM Plnion 6488, Extension=.jpg
 Finished Group 7. Name=cat B E D L E M Plnion 6488, Extension=.jpg
 Finished Group 8. Name=20-01-2012 cat B E D L E M Plnion 6488, Extension=.jpg
 Finished Group 9. Name=Test 20-01-2012 cat B E D L E M Plnion 6488, Extension=.jpg
 Finished Group 10. Name=Test 20-01-2012 cat B E D L E M Plnion 6488 001, Extension=.jpg
 Finished Group 11. Name=Test 20-01-2012 cat B E D L E M Plnion 6488 001, Extension=.jpg

Final Name: Test 20-01-2012 cat B E D L E M Plnion 6488 001.jpg

Actions Menu

So how *did* I end up with this mess? ☹


Because this book is an entirely new edition of the original, I ended up updating all of the photos, and in doing so, I actually did have to figure out this mess. Oh, this particular recreation may not be 100% accurate to what I had originally. For instance, I believe I actually used tags in the Custom Date format instead of lazily spelling out the date as I have done here, but it is [true](#) enough.

So back to the recreation -

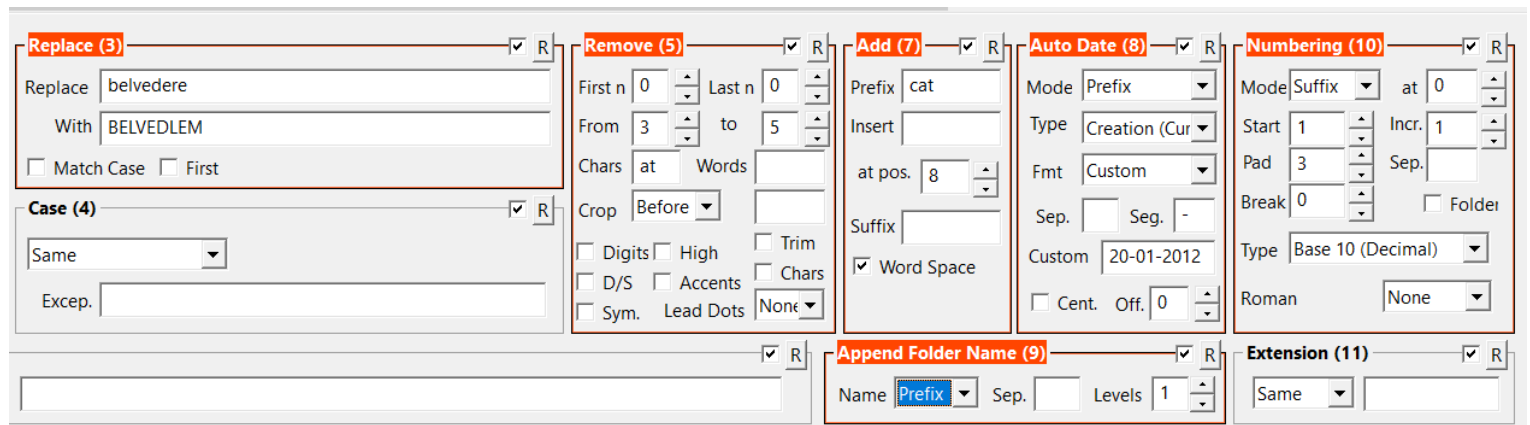
All I had to go on was my original Debug output photo in the original volume to try and figure things out, but isn't that the purpose of this function, after all?

So, for each of the listed criteria in the debug output, I tested and retested each section until the final outcome was close enough to the original.

This is what I came up with:

Name ▲	New Name
 Belvedere Plantation 6488.jpg	Test 220-01-2012 cat B E D L E M Plinion 6488 001.jpg

created from:



The screenshot shows the Bulk Rename Utility interface with several settings panels highlighted in red:

- Replace (3):** Replace `belvedere` with `BELVEDLEM`. Match Case, First.
- Case (4):** Same.
- Remove (5):** First n: 0, Last n: 0, From: 3, to: 5, Chars: at, Words: [empty], Crop: Before. Digits, High, Trim, D/S, Accents, Chars, Sym., Lead Dots: None.
- Add (7):** Prefix: cat, Insert: [empty], at pos.: 8, Suffix: [empty], Word Space.
- Auto Date (8):** Mode: Prefix, Type: Creation (Cur), Fmt: Custom, Sep.: [empty], Seg.: -, Custom: 20-01-2012, Cent., Off.: 0.
- Numbering (10):** Mode: Suffix, at: 0, Start: 1, Incr.: 1, Pad: 3, Sep.: [empty], Break: 0, Folder, Type: Base 10 (Decimal), Roman: None.
- Append Folder Name (9):** Name: Prefix, Sep.: [empty], Levels: 1.
- Extension (11):** Same.

Actions Menu

Here is another *extreme* example broken down into each single segment. This should also provide an insight into how BRU's Order of Evaluation operates— just to illustrate, using test file, DSCN0032.JPG:

First I will apply each criteria individually so you can see how the Debug displays the results for each.

Section #1: RegEx

Name =  DSCN0032.JPG

New Name = REGX2 = 2 REGX1 = DSCN003.JPG

RegEx (1)

Match: (.*)(.)

Replace: REGX2 = \2 REGX1 = \1

Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1, Name=REGX2 = 2 REGX1 = DSCN003, Extension=.JPG

Section #2: Name

Name =  DSCN0032.JPG

New Name = 2300NCSD.JPG

Name (2)

Name: Reverse

Filename Generation Debug Results




Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1, Name=DSCN0032, Extension=.JPG

Finished Group 2, Name=2300NCSD, Extension=.JPG

Section #3: Replace

Name =  DSCN0032.JPG

New Name = REPLACE = catSCN0032.JPG

Replace (3)

Replace: D

With: REPLACE = cat

Match Case

Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1, Name=DSCN0032, Extension=.JPG

Finished Group 2, Name=DSCN0032, Extension=.JPG

Finished Group 3, Name=REPLACE = catSCN0032, Extension=.JPG

Actions Menu

Section #14: Special – Character Translation

Name = DSCN0032.JPG

New Name = DSCP0032.JPG

Special (14)

Change File Status: Not Set

Change File Status: Not Set

Character Status: **Ready**

Javascript Status: Not Set

Character Translations

Character Translations

Enter one entry per line, separated by a comma

N=P

Filename Gene bug Results

Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1. Name=DSCN0032, Extension=.JPG
Finished Group 2. Name=DSCN0032, Extension=.JPG
Finished Group 3. Name=DSCN0032, Extension=.JPG
Finished Group 4. Name=DSCP0032, Extension=.JPG

Selection #4: Case

Name = DSCN0032.JPG

New Name = Dscn0032.JPG

Case (4)

Title

Excep.

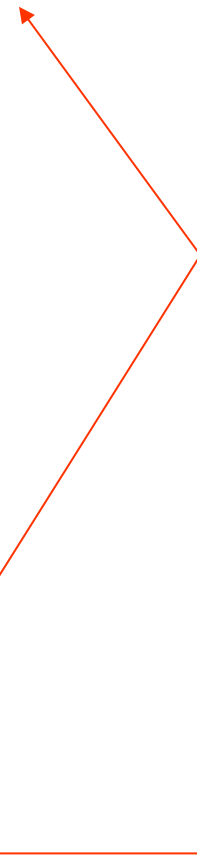
Filename Generation Debu

Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1. Name=DSCN0032, Extension=.JPG
Finished Group 2. Name=DSCN0032, Extension=.JPG
Finished Group 3. Name=DSCN0032, Extension=.JPG
Finished Group 4. Name=Dscn0032, Extension=.JPG

Notes:


1. Notice how Character Translation and Case both appear in Group 4?
2. Character Translation is always performed prior to Case.



Actions Menu

Selection #5: Remove

Name =  DSCN0032.JPG

New Name =  SCN0032.JPG

Remove (5) **R**

First n Last n

From to


Filename Generation D sults




Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

- Finished Group 1. Name=DSCN0032, Extension=.JPG
- Finished Group 2. Name=DSCN0032, Extension=.JPG
- Finished Group 3. Name=DSCN0032, Extension=.JPG
- Finished Group 4. Name=DSCN0032, Extension=.JPG
- Finished Group 5. Name=SCN0032, Extension=.JPG

Selection #6: Move/Copy Parts

Name =  DSCN0032.JPG

New Name =  D Move = 2 Move = SCN003.JPG

Move/Copy Parts (6)

Move last n To pos. Sep.

Filename Generation Debug Results




Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

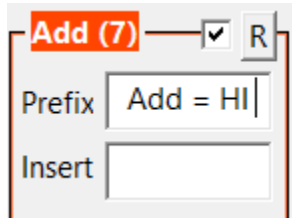
- Finished Group 1. Name=DSCN0032, Extension=.JPG
- Finished Group 2. Name=DSCN0032, Extension=.JPG
- Finished Group 3. Name=DSCN0032, Extension=.JPG
- Finished Group 4. Name=DSCN0032, Extension=.JPG
- Finished Group 5. Name=DSCN0032, Extension=.JPG
- Finished Group 6. Name=D Move = 2 Move = SCN003, Extension=.JPG

Actions Menu

Selection #7: Add

Name =  DSCN0032.JPG

New Name = Add = HI DSCN0032.JPG




Filename Generation Debug Results



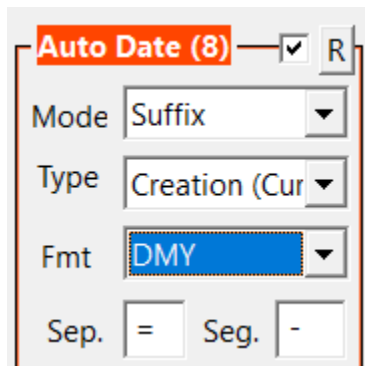
Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1. Name=DSCN0032, Extension=.JPG
 Finished Group 2. Name=DSCN0032, Extension=.JPG
 Finished Group 3. Name=DSCN0032, Extension=.JPG
 Finished Group 4. Name=DSCN0032, Extension=.JPG
 Finished Group 5. Name=DSCN0032, Extension=.JPG
 Finished Group 6. Name=DSCN0032, Extension=.JPG
 Finished Group 7. Name= Add = HI DSCN0032, Extension=.JPG

Selection #8: Auto Date

Name =  DSCN0032.JPG

New Name = DSCN0032 Date = 01-11-09.JPG



Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1. Name=DSCN0032, Extension=.JPG
 Finished Group 2. Name=DSCN0032, Extension=.JPG
 Finished Group 3. Name=DSCN0032, Extension=.JPG
 Finished Group 4. Name=DSCN0032, Extension=.JPG
 Finished Group 5. Name=DSCN0032, Extension=.JPG
 Finished Group 6. Name=DSCN0032, Extension=.JPG
 Finished Group 7. Name=DSCN0032, Extension=.JPG
 Finished Group 8. Name=DSCN0032 Date = 01-11-09, Extension=.JPG

Where:

Sep. equals the string, 'Date = '

Actions Menu

Selection #9: Append Folder Name

Name = DSCN0032.JPG

New Name = 100nikon= Folder DSCN0032.JPG

Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

- Finished Group 1. Name=DSCN0032, Extension=.JPG
- Finished Group 2. Name=DSCN0032, Extension=.JPG
- Finished Group 3. Name=DSCN0032, Extension=.JPG
- Finished Group 4. Name=DSCN0032, Extension=.JPG
- Finished Group 5. Name=DSCN0032, Extension=.JPG
- Finished Group 6. Name=DSCN0032, Extension=.JPG
- Finished Group 7. Name=DSCN0032, Extension=.JPG
- Finished Group 8. Name=DSCN0032, Extension=.JPG
- Finished Group 9. Name=100nikon= Folder DSCN0032, Extension=.JPG

Where:

Sep. = '= Folder '

Selection #10: Numbering

Name = DSCN0032.JPG

New Name = DSCN0032 NUM = 1.JPG

Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG


- Finished Group 1. Name=DSCN0032, Extension=.JPG
- Finished Group 2. Name=DSCN0032, Extension=.JPG
- Finished Group 3. Name=DSCN0032, Extension=.JPG
- Finished Group 4. Name=DSCN0032, Extension=.JPG
- Finished Group 5. Name=DSCN0032, Extension=.JPG
- Finished Group 6. Name=DSCN0032, Extension=.JPG
- Finished Group 7. Name=DSCN0032, Extension=.JPG
- Finished Group 8. Name=DSCN0032, Extension=.JPG
- Finished Group 9. Name=DSCN0032, Extension=.JPG
- Finished Group 10. Name=DSCN0032 NUM = 1, Extension=.JPG

Where:

Sep. = ' NUM = '

Actions Menu

Selection #11: Extension

Name =  DSCN0032.JPG

New Name = DSCN0032.EXT

Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG

Finished Group 1. Name=DSCN0032, Extension=.JPG
 Finished Group 2. Name=DSCN0032, Extension=.JPG
 Finished Group 3. Name=DSCN0032, Extension=.JPG
 Finished Group 4. Name=DSCN0032, Extension=.JPG
 Finished Group 5. Name=DSCN0032, Extension=.JPG
 Finished Group 6. Name=DSCN0032, Extension=.JPG
 Finished Group 7. Name=DSCN0032, Extension=.JPG
 Finished Group 8. Name=DSCN0032, Extension=.JPG
 Finished Group 9. Name=DSCN0032, Extension=.JPG
 Finished Group 10. Name=DSCN0032, Extension=.JPG
 Finished Group 11. Name=DSCN0032, Extension=.EXT

Notes:

1. Only lists 11 criteria because section 12-13 and parts of section 14 have no effect on a file's name change.
 - a. 'Section #12: Filter' – affects only what you see in the File Listing of the Content Pane.
 - b. 'Section #13: Copy/Move Location' only affects the physical file. Has nothing to do with the Renaming process.
 - c. 'Section #14: Special' is made up of items that appear in the Special Menu and are placed here for convenience. Two of the items, 'Change File (Attributes)' and 'Change File (Timestamp)' do not affect the Renaming process and therefore are not listed in the Debug results.

JavaScript if used, appears as the last line and is identified not under a group but as its own entry, 'JavaScript'. JavaScript requires an inexpensive commercial license.

2. Character Translation of Section #14: Special is performed prior to Section #4: Case. Be aware the Debug Log does not show a distinction between these two operations, even though there are.
3. The Bulk Rename Utility log located in `..\Documents\Bulk Rename Utility\`, only gives an account of what files were renamed and does not account for how. The 'Debug New Name' is the only facility that provides the stages.
4. The resulting Log file produced by 'Debug New Name' is done 'on the fly'. Once you close out the message box, there is no electronic transcript kept, unfortunately.
5. The Debug results are divided into 11 groups, each representative of one evaluation. Each corresponding to a numbered section with the exception of 'Character Translation' falling before Section #4: Case, and if active, would be grouped together with Case under Group 4 in the Debug Log.

Actions Menu

And just for fun, Here is what you have been waiting for – all of them active at once, with the exception of JavaScript:

Name = `DSCN0032.JPG`

New Name =

`100nikon= Folder Add = HI 0 Move = r Move = Oncs Replace = Cat = 1xger 2 = 2xge Date = 01-11-09 NUM = 1.EXT`

Filename Generation Debug Results



Original name: DSCN0032.JPG
Components: Name=DSCN0032, Extension=.JPG



Finished Group 1. Name=REGX2 = 2 REGX1 = DSCN003,
Extension=.JPG
Finished Group 2. Name=300NCSD = 1XGER 2 = 2XGER,
Extension=.JPG
Finished Group 3. Name=300NCS REPLACE = cat = 1XGER 2 =
2XGER, Extension=.JPG
Finished Group 4. Name=300ncs Replace = Cat = 1xger 2 = 2xger,
Extension=.JPG
Finished Group 5. Name=00ncs Replace = Cat = 1xger 2 = 2xger,
Extension=.JPG
Finished Group 6. Name=0 Move = r Move = Oncs Replace = Cat =
1xger 2 = 2xge, Extension=.JPG
Finished Group 7. Name= Add = HI 0 Move = r Move = Oncs Replace
= Cat = 1xger 2 = 2xge, Extension=.JPG
Finished Group 8. Name= Add = HI 0 Move = r Move = Oncs Replace
= Cat = 1xger 2 = 2xge Date = 01-11-09, Extension=.JPG
Finished Group 9. Name=100nikon= Folder Add = HI 0 Move = r
Move = Oncs Replace = Cat = 1xger 2 = 2xge Date = 01-11-09,
Extension=.JPG
Finished Group 10. Name=100nikon= Folder Add = HI 0 Move = r
Move = Oncs Replace = Cat = 1xger 2 = 2xge Date = 01-11-09 NUM
= 1, Extension=.JPG
Finished Group 11. Name=100nikon= Folder Add = HI 0 Move = r
Move = Oncs Replace = Cat = 1xger 2 = 2xge Date = 01-11-09 NUM
= 1, Extension=.EXT

Final Name: 100nikon= Folder Add = HI 0 Move = r Move = Oncs
Replace = Cat = 1xger 2 = 2xge Date = 01-11-09 NUM = 1.EXT

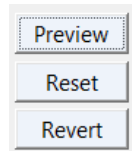
Now you can clearly see how each section evaluated in each corresponding group (any Character Translation and Case appear as one in Group #4 as previously discussed). By the way, it is your turn to figure out what created the mess above by studying just the Debug output from the photo above. Don't cheat and refer back to the previous pages. It will be a good training exercise. Good luck with that. I have given you all the help you need to figure it out. 😊

Actions Menu

Reset All Renaming Criteria (Ctrl + T)

Actions	
Select All	Ctrl+A
Deselect All	Ctrl+D
Invert Selection	Ctrl+I
Select from Clipboard	
Jump to Path...	Ctrl+J
Rename Object Manually	F2
 Refresh Files	F5
Refresh Tree	Ctrl+F5
 Show/Hide Tree	F11
Zoom...	F8
List	▶
Import Rename-Pairs	▶
Debug New Name	
Reset All Renaming Criteria	Ctrl+T

This is the same as the 'Reset' button on the interface:





Master reset to reset all values and settings for 'all' 14 sections back to their default state

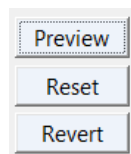
Notes:

1. Has no effect on a loaded Favourites file.
2. Has no effect on a loaded Imported Rename-Pairs file.
3. Current directory position in the Navigation Pane and those files that display in the Content Pane remain unchanged.
4. Any selected files in the Content Pane remain selected.
5. Difference between this and the File Menu option, 'New' is that New will unload the Favourites File, reset all criteria, navigate back to the default location, 'This PC', and remove any JavaScript code from the Code Entry Form.
6. JavaScript selection will be disabled but any script in the Code Entry Form remains.

Revert All Criteria to Last Saved (Ctrl + E)

Actions	
Select All	Ctrl+A
Deselect All	Ctrl+D
Invert Selection	Ctrl+I
Select from Clipboard	
Jump to Path...	Ctrl+J
Rename Object Manually	F2
 Refresh Files	F5
Refresh Tree	Ctrl+F5
 Show/Hide Tree	F11
Zoom...	F8
List	▶
Import Rename-Pairs	▶
Debug New Name	
Reset All Renaming Criteria	Ctrl+T
Revert All Criteria to Last Saved	Ctrl+E

This is the same as the 'Revert' button on the interface:



Displays in menu item only if a Favourites file was previously loaded. Revert restores a Favourite file back to its original version after you have made changes to it.

Note:

1. This will not work if the changes have already been saved. For example, if you have used 'Save', Ctrl + S, or exited the program with 'Save on Exit' set, then the changes are made permanent because the Favourites file has been overwritten.

Actions Menu

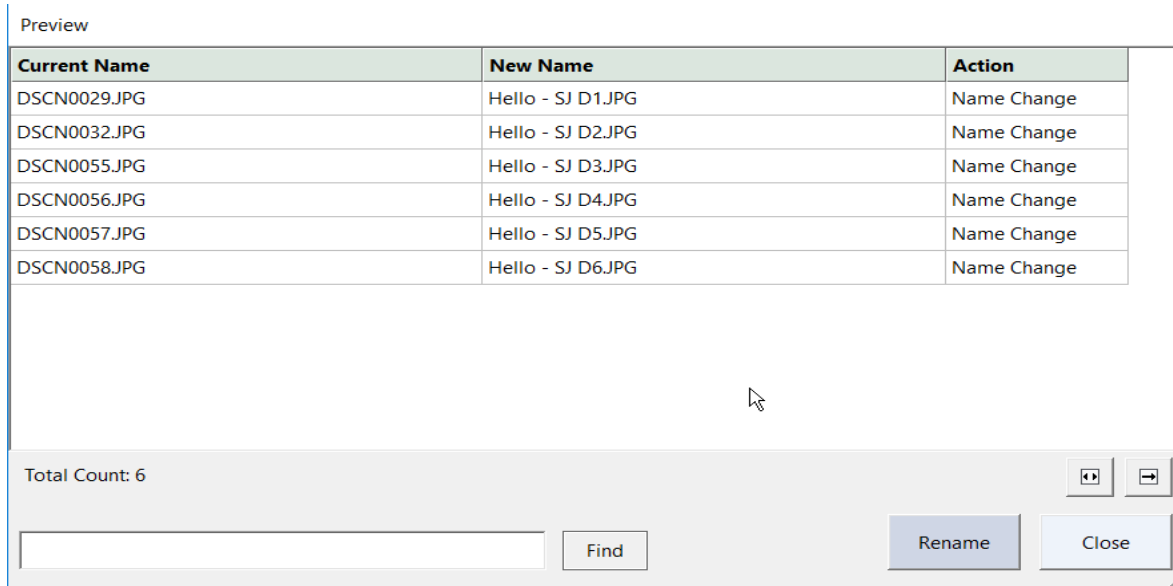
Preview (Ctrl + P)



This is the same as the 'Preview' button on the interface:



Preview all the renaming actions, before actually renaming (name change, timestamp change, attribute change, etc.).

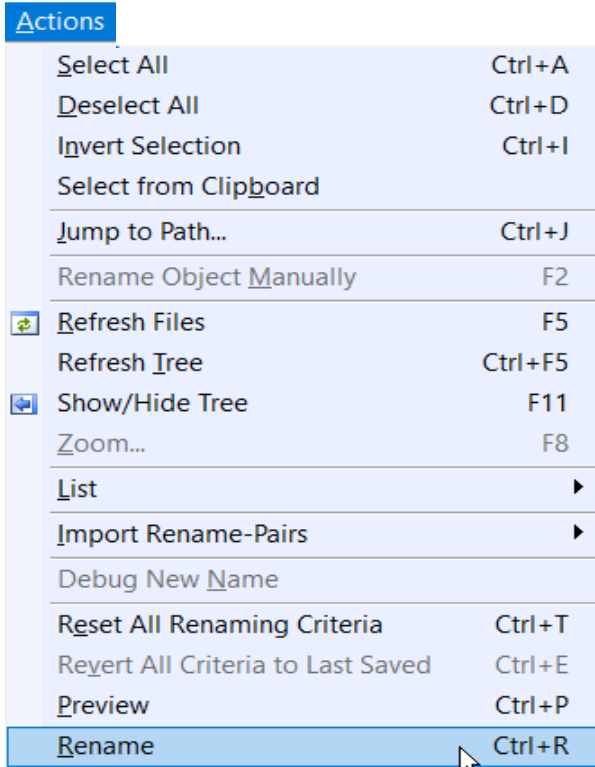


'Preview' is a bit different than just using the New Name column to see a preview of the applied changes.

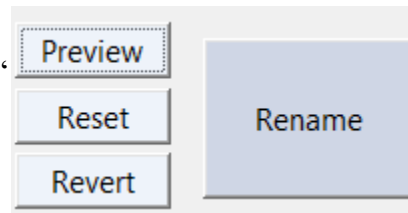
1. Displays only those files affected.
 - a. 'Show Only Items Affected by Renaming ...' from List submenu of Actions Menu will perform similar function.
2. Provides a count of files affected.
3. Has a search facility.
4. Is not a DEBUG – will only show the name change and the action taken to obtain the change and nothing more.

Actions Menu

Rename (Ctrl + R)



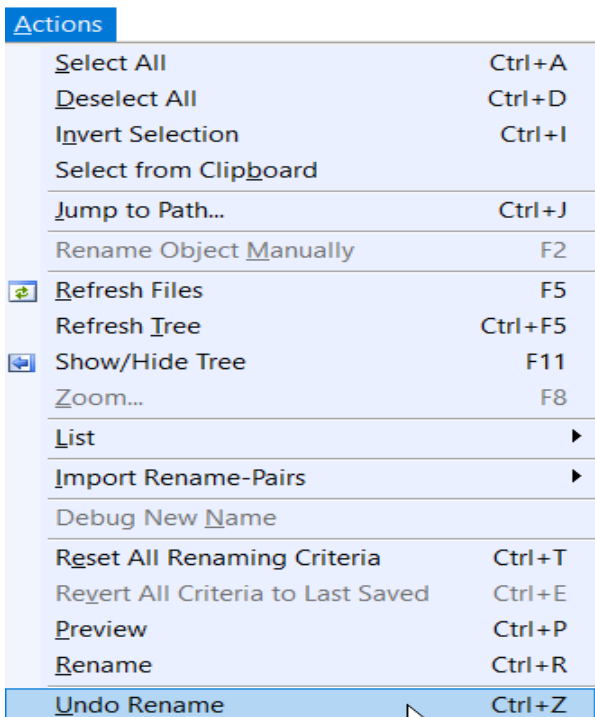
This is the same as the 'Rename' button on the User Interface:



'Rename' will instruct BRU to begin processing the evaluation of the expression specified by the set criteria and any options that may have been selected, and apply the finished equation to all selected files that qualify.

Or, in other words, Rename the stupid files already! 😊

Undo Rename (Ctrl + Z)



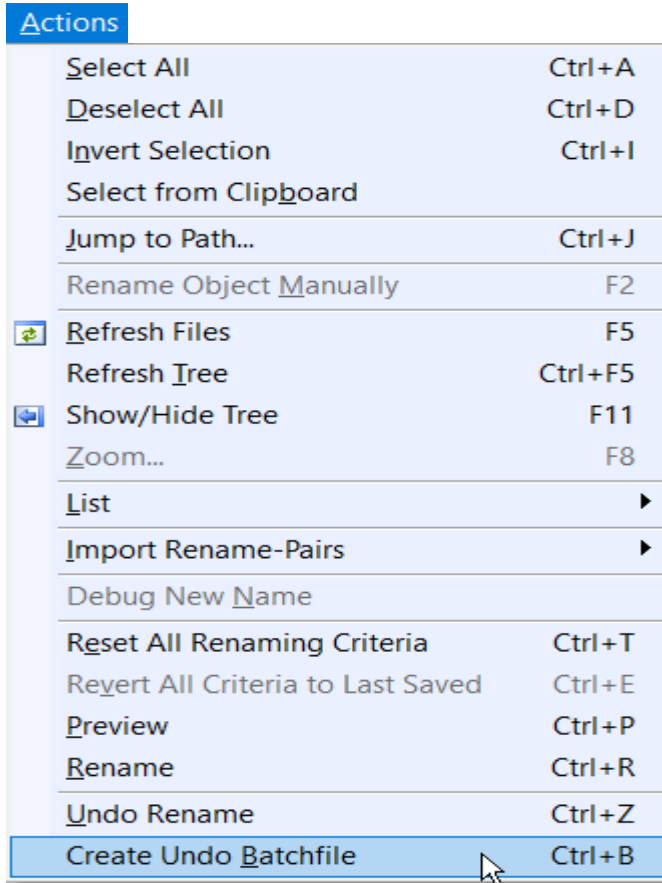
Unintended Consequences. Oh No! You realized too late that you just renamed 177 files to *&%% 1, 2 etc. and now you are screaming a similar sequence of expletives '\$&*%\$!' at yourself. Despite all of the safeties built into the program – 'New Name' column, 'Preview' facility and even a 'Debug Result' function, your attention was distracted for the moment. Fret not. BRU has you covered.

This will undo the last Rename Operation and remove any and all changes made to the files and folders including timestamps, attribute changes, etc. **This is a single use command.** If you were to perform further renaming operations on these files, then you really are '\$&*%\$!' out of luck.

Note: Does not undo 'physical location changes': **Selection #13: Copy/Move to Location.** But does restore 'File Attributes' and 'Timestamp' changes.

Actions Menu

Create Undo Batch File (Ctrl + B)



Although the program employs an excellent Undo feature already, this allows for creating an Undo Batch File that essentially can be run outside of the program anytime as long as those filenames retain their current name.

This is especially useful..

- (1). If the program is closed and restarted later, you will not have the ability to use the normal Undo feature because it clears this status, however the batch file can be run and restores the files back.
- (2). If you need to perform multiple actions, you can use this feature as a timeline of events. Each batch file created after a rename action would preserve the previous names allowing you to trace back to a place before things started to go wrong. This, in conjunction with the Debug New Name feature, could be utilized for further analysis.

Notes:

1. The rename action must be performed first in order to successfully create an Undo Batch File for that action.

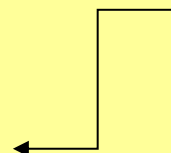
Example:

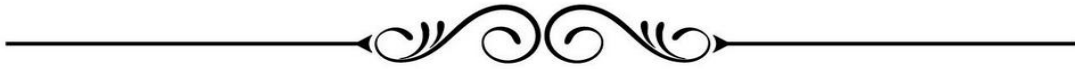
1. Perform the rename action on these files by clicking on the 'Rename' Button

Name	New Name
Belvedere Plantation 6474.jpg	test1.jpg
Belvedere Plantation 6476.jpg	test2.jpg
Belvedere Plantation 6479.jpg	test3.jpg
Belvedere Plantation 6488.jpg	test4.jpg
Belvedere Plantation 6490.jpg	test5.jpg
Belvedere Plantation 6492.jpg	Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	Belvedere Plantation 6508.jpg

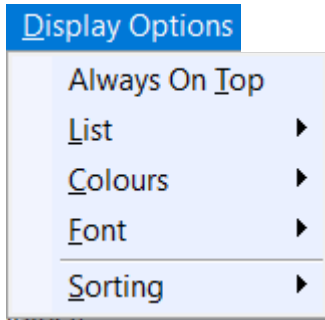
2. Select from the 'Actions Menu', 'Create Undo Batchfile'
3. Give it a name and save the file, e.g., do.bat
4. Just run the do.bat file when you want to restore the files back.

```
F:\test\test 1\do.bat
REM Bulk Rename Utility- UNDO Batch file
REM
REN "F:\test\test5.jpg" "Belvedere Plantation 6490.jpg"
REN "F:\test\test4.jpg" "Belvedere Plantation 6488.jpg"
REN "F:\test\test3.jpg" "Belvedere Plantation 6479.jpg"
REN "F:\test\test2.jpg" "Belvedere Plantation 6476.jpg"
REN "F:\test\test1.jpg" "Belvedere Plantation 6474.jpg"
ECHO Operation complete!
```





Display Options Menu



Display Options Menu

Always on Top

Display Options

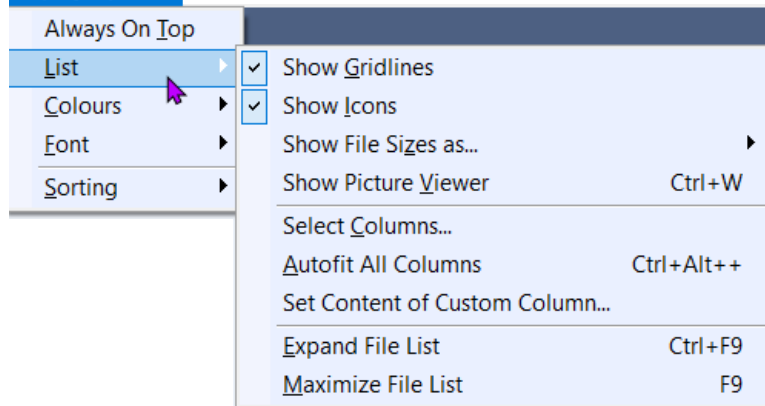
Always On Top

The BRU program will remain on 'top' of other windows regardless of 'focus'. Windows will normally move a window (program, dialog box, message box, etc.) into the background (place it behind other windows) when the user clicks on or uses the Windows task switching (task view) to select a different window.

This new window now has 'focus', meaning, that the user can interact with the window (any Foreground/Background processing Priorities of Windows are in effect). This window also has priority as far as placement and is on 'top' of other windows. By specifying 'Always On Top' you are instructing Windows to display the BRU program 'on screen' even if focus is lost by switching to another window.

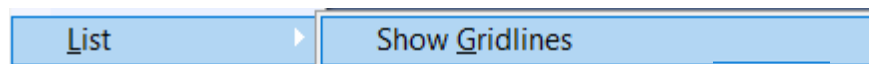
List

Display Options

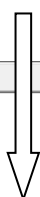


The List item displays selections that affect the visual interface of the Content Pane.

- Show Gridlines

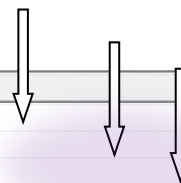


No Gridlines:



Name	New Name
DSCN0001.JPG	DSCN0001.JPG
DSCN0029.JPG	DSCN0029.JPG
DSCN0032.JPG	DSCN0032.JPG
DSCN0055.JPG	DSCN0055.JPG
DSCN0056.JPG	DSCN0056.JPG
DSCN0057.JPG	DSCN0057.JPG
DSCN0058.JPG	DSCN0058.JPG
DSCN0059.JPG	DSCN0059.JPG
DSCN0064.JPG	DSCN0064.JPG

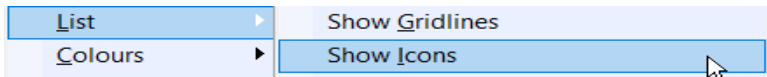
With Gridlines:



Name	New Name
DSCN0001.JPG	DSCN0001.JPG
DSCN0029.JPG	DSCN0029.JPG
DSCN0032.JPG	DSCN0032.JPG
DSCN0055.JPG	DSCN0055.JPG
DSCN0056.JPG	DSCN0056.JPG
DSCN0057.JPG	DSCN0057.JPG
DSCN0058.JPG	DSCN0058.JPG
DSCN0059.JPG	DSCN0059.JPG
DSCN0064.JPG	DSCN0064.JPG

Display Options Menu

- Show Icons

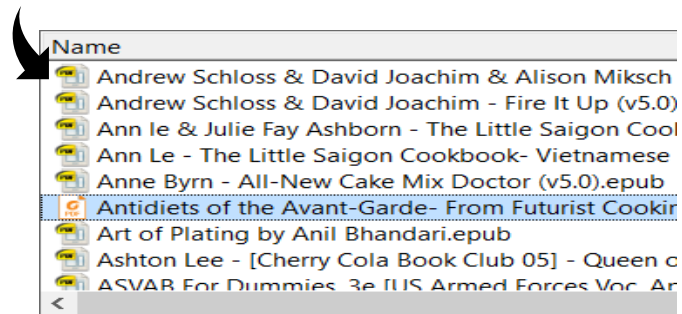


Must refresh files to view changes (F5 or through 'Refresh Files' of the Actions Menu), or momentarily switch to another folder then back again from the Navigation Pane. It has been observed that displaying icons can have an effect on BRU's speed when scanning files and folders, especially with mapped drives.

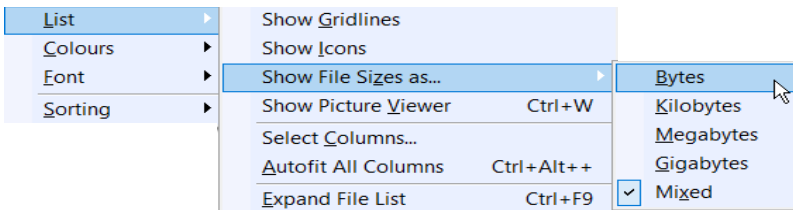
Without Icons:



With Icons:



- Show File Sizes as..



Mixed -

Size
12 MB
9 MB
3 MB
4 MB
6 MB
5 MB
3 MB
650 KB
12 MR

Bytes -

Size
683,377 B
2,897,856 ...
1,561,176 ...
2,332,408 ...
2,037,366 ...
4,957,823 ...
2,923,266 ...
11,569,50...
628,424 B

Kilobytes -

Size
3,390 KB
30,513 KB
667 KB
2,829 KB
1,524 KB
2,277 KB
1,989 KB
4,841 KB

Megabytes -

Size
3 MB
29 MB
0 MB
2 MB
1 MB
2 MB
1 MB
4 MR

Gigabytes -

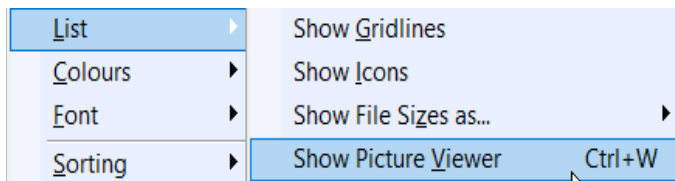
Size
0 GB
0 GB
0 GB
0 GB
0 GB
0 GB
0 GB
0 GB

Notes

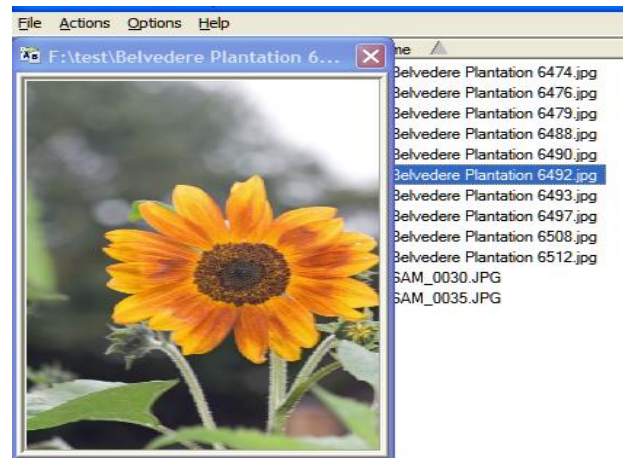
- "Mixed" will display files in the unit most suited to their size (e.g., a file size 1224555 Bytes will be displayed in Megabytes). This is the default view.

Display Options Menu

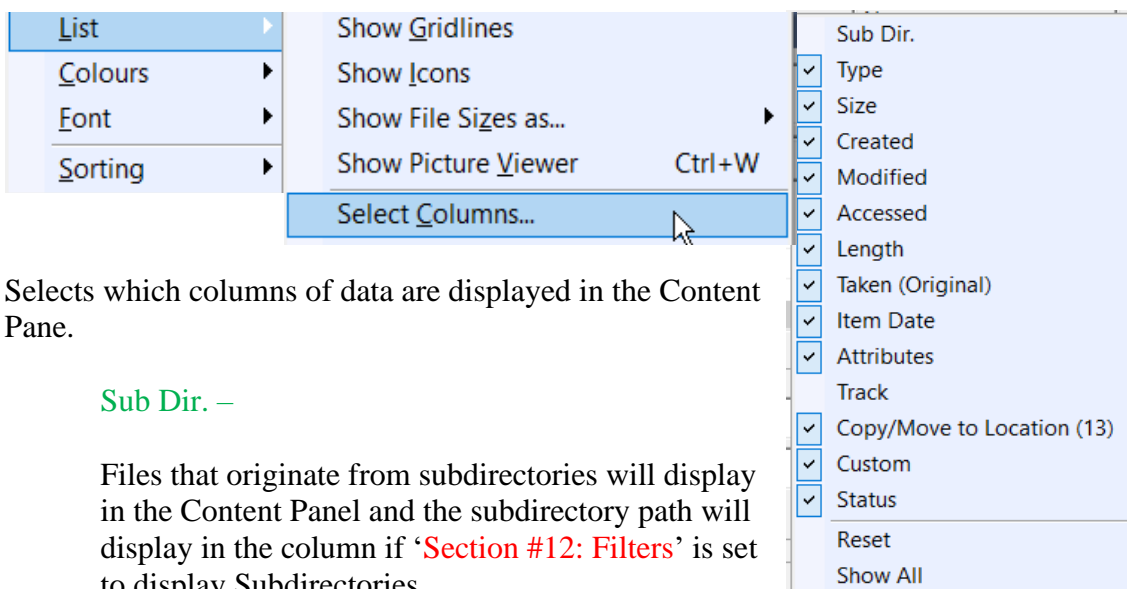
- Show Picture Viewer (Ctrl + W)



Simple resizable window quick image viewer for BMP, GIF, JPG, WMF and EMF file types. Select single image in Content Pane beforehand. Will **not** work if multiple images are selected.



- Select Columns



Selects which columns of data are displayed in the Content Pane.

Sub Dir. –

Files that originate from subdirectories will display in the Content Panel and the subdirectory path will display in the column if 'Section #12: Filters' is set to display Subdirectories.



	Name	New Name	Sub Dir.
	logo_t.png	logo_t.png	\res
	splash.bmp	splash.bmp	\res
	splash.png	splash.png	\res
	commons.txt	commons.txt	\licences
	gnu-crypto.txt	gnu-crypto.txt	\licences
	jgoodies.txt	jgoodies.txt	\licences
	log4j.txt	log4j.txt	\licences
	pdfbox.txt	pdfbox.txt	\licences
	binding-1.4.0.jar	binding-1.4.0.jar	\lib
	commons-in-2.4.jar	commons-in-2.4.jar	\lib

Notes:

1. IF 'Section 12: Filters' is set to display Subdirectories, recursive processing will take place. This means that not only are the files in the current main directory processed but any files in 'all' the sub directories as well (including those that fall below other sub-directories), so be careful with this option.

Display Options Menu

- *Select Columns*

Type –

Refers to file type e.g., .rar, .bat, .jpg and .emf ...



Name ▲	New Name	Type
933e1474c25535ab44f008207a31e9b8_full.emf	933e1474c25535ab44f008207a31e9b8_full.emf	EMF File
extract zip-rar nest.bat	extract zip-rar nest.bat	Windows Batch File
extract zip-rar nest.bat,working for rar	extract zip-rar nest.bat,working for rar	File
test.jpg	test.jpg	JPG File

File types are identified by Windows using the file's extension. In the example above, an .EMF file is a type of image file, a .bat file is a batch file and a .JPG is another type of image file, and BRU identifies a compressed file, e.g., .rar archive, using a generic 'FILE'. But there are many others. BRU gets this information from Windows using the **defined file types** and this includes associated program filetypes.

For example, I have WinRar. BRU identifies the .zip extension as 'WinRar ZIP archive'. A .pdf file is identified on my system as a Foxit Reader PDF Document because the program, Foxit Reader, is associated with that extension. Either of these extensions on another person's system will produce a different identification if there is a different program associated with the filetype, or a generic result if they don't.

Some software installations may also add their own unique filetypes that are proprietary to one company or product. Take as an example, Omnipage. Omnipage is an OCR program and .opd files are uniquely identified as an Omnipage Document.

Here are just a few others I have seen in BRU.

File Folder – Directory	Text Document - .txt	Windows Batch File - .bat
EMF File - .emf	PNG File - .png	DJVU File - .djvu
HTML Document - .htm or .html	Epub File - .epub	Application - .exe
Windows Installer Package - .msi	BMP File - .bmp	Configuration Settings - .ini

MP3 Audio File (VLC) - .mp3

Etc.

You get the idea.

Notes:









1. VLC refers to VLC Media Player, an associated program.

Display Options Menu

- *Select Columns*

Created, Modified, Accessed –

Will display the Windows File Properties timestamp values for each selected column. e.g.,









Name	New Name	Created	Modified	Accessed
 background_col.png	background_col.png	10/15/2018 8:31:23 PM	10/15/2018 8:31:24 PM	10/15/2018 8:31:23 PM
 Booksorber.exe	Booksorber.exe	10/15/2018 1:06:29 PM	8/15/2013 6:55:52 PM	10/15/2018 1:06:29 PM
 Booksorber.jar	Booksorber.jar	10/15/2018 1:06:29 PM	8/15/2013 6:55:52 PM	10/15/2018 1:06:29 PM
 launch-linux.sh	launch-linux.sh	10/15/2018 1:06:29 PM	8/15/2013 5:22:42 PM	10/15/2018 1:06:29 PM
 launch-macos.command	launch-macos.command	10/15/2018 1:06:29 PM	9/26/2012 12:38:48 AM	10/15/2018 1:06:29 PM
 launch-win-alternative.bat	launch-win-alternative.bat	10/15/2018 1:06:29 PM	1/12/2013 3:10:28 PM	10/15/2018 1:06:29 PM
 Logfile.txt	Logfile.txt	10/15/2018 1:06:37 PM	10/15/2018 8:34:43 PM	10/15/2018 1:06:37 PM
 tmp.tmp	tmp.tmp	10/15/2018 8:34:41 PM	10/15/2018 8:34:42 PM	10/15/2018 8:34:41 PM

Notes:

1. These timestamps are assigned by the Windows File System at the time of creation within the Windows OS or at the time the file is imported into Windows from another source. The data is stored in the MFT (Master File Table) of the File System on the volume on which the file resides.
2. BRU extracts this information and it appears in the selected columns.
3. The Windows Properties, Date Created, Date Modified and Date Accessed, can be modified through the Change File Timestamp facility of '[Section #14: Special](#)'.

Size –

Will display the current File Size of each file – but not a Folder's file size. The format – Mixed, Bytes, Kilobytes, Megabytes, and Gigabytes, as determined by the, 'Show File Sizes as...', option (see List selection in this section for further information).

Name	New Name	Size
 background_col.png	background_col.png	158 KB
 Booksorber.exe	Booksorber.exe	981 KB
 Booksorber.jar	Booksorber.jar	229 KB
 launch-linux.sh	launch-linux.sh	121 B
 launch-macos.command	launch-macos.command	143 B
 launch-win-alternative.bat	launch-win-alternative.bat	123 B
 Logfile.txt	Logfile.txt	25 KB
 tmp.tmp	tmp.tmp	6 MB

Display Options Menu

- Select Columns

Length –

Will display the current File length – this includes the dot notation and the extension but does **not** include the file path. Currently, there is no function or option to display the **full** pathname or length. I have submitted a request to TGRM Software to add a **‘Full Path’** column.

i.e.

Name ▲	New Name	Length
933e1474c25535ab44f008207a31e9b8_full.emf	933e1474c25535ab44f008207a31e9b8_full.emf	41
extract zip-rar nest.bat	extract zip-rar nest.bat	24
extract zip-rar nest.bat,working for rar	extract zip-rar nest.bat,working for rar	40
test.jpg	test.jpg	8

Taken (Original) –

This is an EXIF Metadata that refers to the original date an image file was created by the digital device. If the message, ‘Set Menu Option to Extract EXIF data’ is displayed, then BRU is indicating that data exists but the option to view it has not been activated {set option ‘Extract EXIF data (Photos)’ under the ‘ID3 /EXIF Data/ File Properties’ selection of the Renaming Options Menu to activate}.

Name ▲	New Name	Taken (Original)
933e1474c25535ab44f008207a31e9b8_full.emf	933e1474c25535ab44f008207a31e9b8_full.emf	
extract zip-rar nest.bat	extract zip-rar nest.bat	
extract zip-rar nest.bat,working for rar	extract zip-rar nest.bat,working for rar	
test.jpg	test.jpg	

Set menu option to extract EXIF data

Notes:

1. For more information, refer to, ‘Using EXIF Tags’ under **‘Section #7: Add’**

Display Options Menu

- Select Columns

Attributes –

Refers to the File Attributes, [Archive](#), [System](#), [Hidden](#), and [Read-Only](#). This ‘bit’ can be set by Windows, a program, or the user. These will be displayed for both files and folders:

Example of Folder with Attributes set –

Name ▲	New Name	Sub Dir.	Attrib...
MBR Boot Disk ISO file	MBR Boot Disk ISO file		R
MBRSetup.msi	MBRSetup.msi		
MBRWizard Boot Disk.ISO	MBRWizard Boot Disk.ISO	\MBR Boot Disk ISO file	

Example of Files with Attributes set –

Name ▲	New Name	Attrib...
SAM_0030.JPG	SAM_0030.JPG	A
SAM_0035.JPG	SAM_0035.JPG	A
SAM_0061.JPG	SAM_0061.JPG	A
SAM_0072.JPG	SAM_0072.JPG	A

Read-only - Allows a file to be read, but cannot be written or modified. The file is ‘write-protected’ but can be deleted.

Archive – Indicates that this file has been modified since a previous backup. This attribute is commonly used by backup software for determining ‘Incremental’ Backup status. Files can be written, modified and deleted.

Hidden - File is not shown. It will display in Windows Explorer **if** you have the system set to show Hidden Files, but will not include System files. They require a separate option for authorization. BRU allows the display of Hidden Files using an override available in ‘[Section #12: Filters](#)’.

System – Indicates to Windows that this is a system-critical file that should not be tampered with (modified, renamed, moved, etc.) or deleted. Will be hidden under normal circumstances.

Note:

1. For more information, refer to ‘Change File Attributes’ under the ‘Special Menu’

Display Options Menu

- Select Columns

Track –

Track is part of the ID3 MP3 Metadata. To display the data, the ‘Extract ID3 Data (MP3)’ must be set from The ‘ID3 /EXIF Data / File Properties’ selection of the Renaming Options Menu. If it isn’t, even if data is present, all you will see is ‘Track 0’.

Not set:

Name ▲	New Name	Track
🔔 Truth Hurts (From Ratpocalypse).mp3	Truth Hurts (From Ratpocalypse).mp3	0
🔔 Try.mp3	Try.mp3	0
🔔 Try.mp4	Try.mp4	0

Set:

Name ▲	New Name	Track
🔔 Truth Hurts (From Ratpocalypse).mp3	Truth Hurts (From Ratpocalypse).mp3	17
🔔 Try.mp3	Try.mp3	0
🔔 Try.mp4	Try.mp4	0

Copy/ Move to Location (13) –

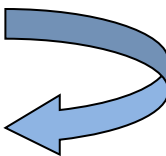
This simply indicates the selected path the renamed files will be copied or moved to based on the options set In ‘[Selection #13: Copy/Move to Location](#)’.

Copy/Move to Location (13)

Path

Copy not Move

Name ▲	New Name	Copy/Move to Location (13)
🔔 Truth Hurts (From Ratpocalypse).mp3	Truth Hurts (From Ratpocalypse).mp3	h:\0\
🔔 Try.mp3	Try.mp3	
🔔 Try.mp4	Try.mp4	



Notes:







- Using ‘Undo Rename’ from the ‘Actions Menu’ or pressing Ctrl + Z does not undo ‘physical location changes’.

Display Options Menu








- *Select Columns*

Status –








Provides two flags. An **OK** flag is indicated after a successful Renumber operation, and an **Error** flag is Indicated along with a generated error message, if the Renumber operation was unsuccessful.

Name ▲	New Name	Status
 933e1474c25535ab44f008207a31e9b8_full.e...	933e1474c25535ab44f008207a31e9b8_full.emf	
 A Course in Combinatorics 2nd Edition.pdf	A Course in Combinatorics 2nd Edition.pdf	
 extract zip-rar nest.bat	extract zip-rar nest.bat	
 extract zip-rar nest.bat,working for rar	extract zip-rar nest.bat,working for rar	
 Introducing Regular Expressions - Unraveling ...	Introducing Regular Expressions - Unraveling Reg...	
 test.jpg	test.jpg	

Example of Error Status (to produce the error, I deleted the file prior to the Renumber action):

Name ▲	New Name	Size	Length	Status
 15.jpg	15.jpg	216 KB	6	
 16.jpg	16.jpg	156 KB	6	
 17.jpg	17.jpg	119 KB	6	
 18.jpg	18.jpg	101 KB	6	
 19.jpg	cat.jpg	131 KB	7	Error. The system cannot find the file specified.
 2.jpg	2.jpg	154 KB	5	
 2.jpg	2.jpg	267 KB	5	

Example of OK Status:

Name ▲	New Name	Size	Length	Status
 15.jpg	15.jpg	216 KB	6	
 16.jpg	16.jpg	156 KB	6	
 17.jpg	17.jpg	119 KB	6	
 18.jpg	18.jpg	101 KB	6	
 19.jpg	19.jpg	131 KB	6	Error. The system cannot find the file specified.
 cat.jpg	cat.jpg	154 KB	7	OK
 3.jpg	3.jpg	267 KB	5	

Notice that previous status flags are retained. To clear the Status, Refresh the File List (F5 or use 'Refresh Files' from the Actions Menu or momentarily change to a different folder and back again from the Navigation Pane (this re-establishes the focus).



Display Options Menu

- Select Columns

Reset –

Resets Content Pane display back to the default columns. Does not affect any other options except columns.

The columns are: ‘Name’, ‘New Name’, ‘Modified’, ‘Size’, and ‘Status’. All other columns are removed.

Name ▲	New Name	Modified	Size	Status	
 DSCN0001.JPG	DSCN0001.JPG	9/1/2009 2:38:34 PM	2 MB		
 DSCN0029.JPG	DSCN0029.JPG	10/31/2009 11:20:42 PM	5 MB		

Show All –

All of the available columns will display:

Name	New Na... ▲	Sub Dir.	Type	Created	Modified	Size	Accessed	Length	Taken (Original)	Item Date	Attribut...	Track	Copy/Mov...	Stati
------	-------------	----------	------	---------	----------	------	----------	--------	------------------	-----------	-------------	-------	-------------	-------

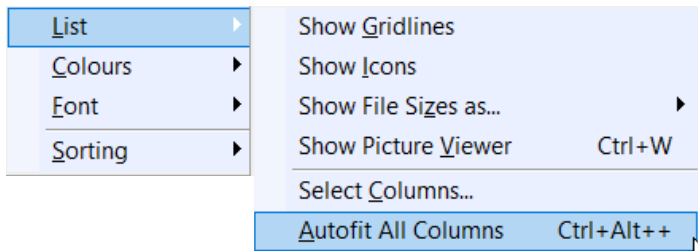
Name New Name Type Created Modified
 Attributes Track Size Accessed Length Status
 Item Date* Custom* Sub Dir* Copy/Move to Location Taken (Original)

* refer to the sub-sections under New Additions below and ‘Section #8: Auto Date’.

* enable recursion under ‘Section #12: Filters’

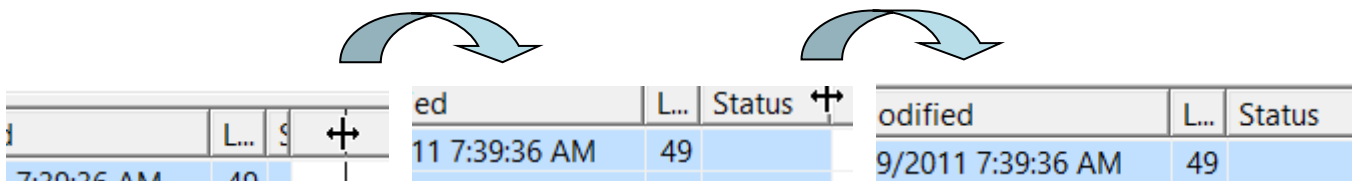
Display Options Menu

- Autofit All Columns (Ctrl + Alt + +)



Autofit adjusts each column's width equal to the longest length of data held in each. This will display all data without being cut off (truncated) and save you from having to make the manual adjustments yourself (expanding each header of the columns individually through mouse select and drag).

For example - using mouse to select and drag the Status header to expand the Status column -



Instead, using Autofit –

Before:

Name	New Name	Sub Dir.	Type	Taken (Original) ▲	Item Date	Custom	
Baseball_RiverDogs_Ga...	Baseball_RiverDog...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	2
Becca BW.jpg	Becca BW.jpg		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	1
Becca Colour QT BW.jpg	Becca Colour QT B...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	3
Becca Moving Sun Fade...	Becca Moving Sun ...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	1
Becca Purple Vignette.j...	Becca Purple Vign...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	1
Becca QT Distorted.jpg	Becca QT Distorte...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	3
Becca QT FI Chalk.jpg	Becca QT FI Chalkj...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	5
Becca Water Color.jpg	Becca Water Color...		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	2
Becca Water Reflection	Becca Water Refle		JPG File	Set menu option to extract...	Set menu opti...	File Size is - Mo...	2

After:

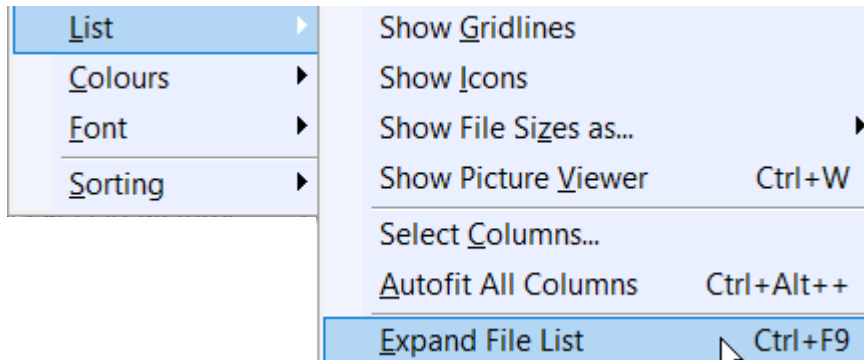
Name	New Name	Sub Dir.	Type	Taken (Original) ▲
Baseball_RiverDogs_Game1_006.jpg	Baseball_RiverDogs_Game1_006.jpg		JPG File	Set menu option to extract EXIF data
Becca BW.jpg	Becca BW.jpg		JPG File	Set menu option to extract EXIF data
Becca Colour QT BW.jpg	Becca Colour QT BW.jpg		JPG File	Set menu option to extract EXIF data
Becca Moving Sun Faded.jpg	Becca Moving Sun Faded.jpg		JPG File	Set menu option to extract EXIF data
Becca Purple Vignette.jpg	Becca Purple Vignette.jpg		JPG File	Set menu option to extract EXIF data
Becca QT Distorted.jpg	Becca QT Distorted.jpg		JPG File	Set menu option to extract EXIF data
Becca QT FI Chalk.jpg	Becca QT FI Chalk.jpg		JPG File	Set menu option to extract EXIF data
Becca Water Color.jpg	Becca Water Color.jpg		JPG File	Set menu option to extract EXIF data
Becca Water Reflection.jpg	Becca Water Reflection.jpg		JPG File	Set menu option to extract EXIF data

Notes:

1. Status has no current value, therefore width = 1 character. This requires manual expanding because Status changes.
2. Columns are equal to the width of the longest data held which may not be visibly apparent in the provided photos.

Display Options Menu

- *Expand File List (Ctrl + F9)*



This enlarges both the Navigation and Content Panes to allow more visibility and still be able to work with most of the criteria sections. It does this by removing Sections 13 and 14 from the user interface.

Notes:

1. This is not the same as the Maximize File List (F9).

Display Options Menu

- Expand File List (Ctrl + F9)

Before:

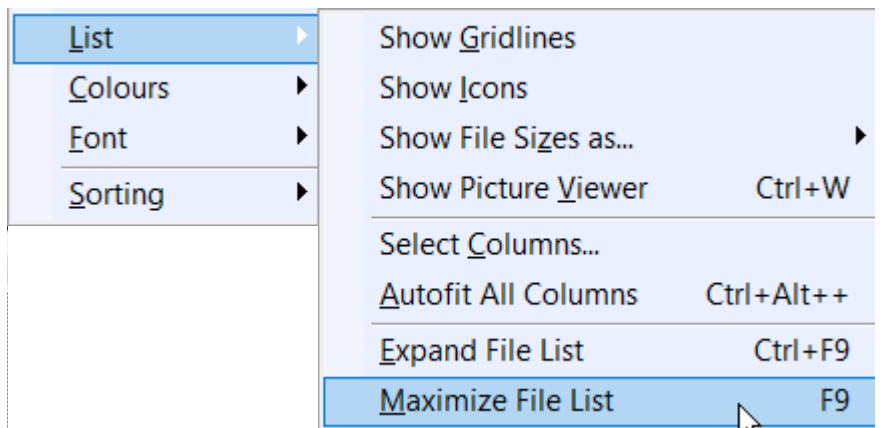
The screenshot shows the Bulk Rename Utility interface. The file list is partially visible, showing files like DSCN0001.JPG through DSCN0064.JPG. The interface includes various configuration panels such as RegEx, Replace, Remove, Add, Auto Date, and Numbering. A status bar at the bottom indicates "74 Objects (0 Selected)" and the current path "K:\Ma & Dad Pictures\DCIM\100NIKON".

After:

The screenshot shows the Bulk Rename Utility interface after the file list has been expanded. The file list now displays all 198 files in the '100NIKON' folder, including DSCN0001.JPG through DSCN198.JPG. The interface elements are identical to the 'Before' screenshot, but the file list is fully populated. The status bar still shows "74 Objects (0 Selected)" and the current path "K:\Ma & Dad Pictures\DCIM\100NIKON".

Display Options Menu

- Maximize File List (F9)



This is the equivalent of using the 'Expand' button on the User Interface.



Maximizes both the Navigation and Content Pane. This allows you to fully concentrate on the Hierarchy Tree and File List. It does this by removing most of the User Interface. The control is a toggle, meaning that if you click it a second time, it will restore back.

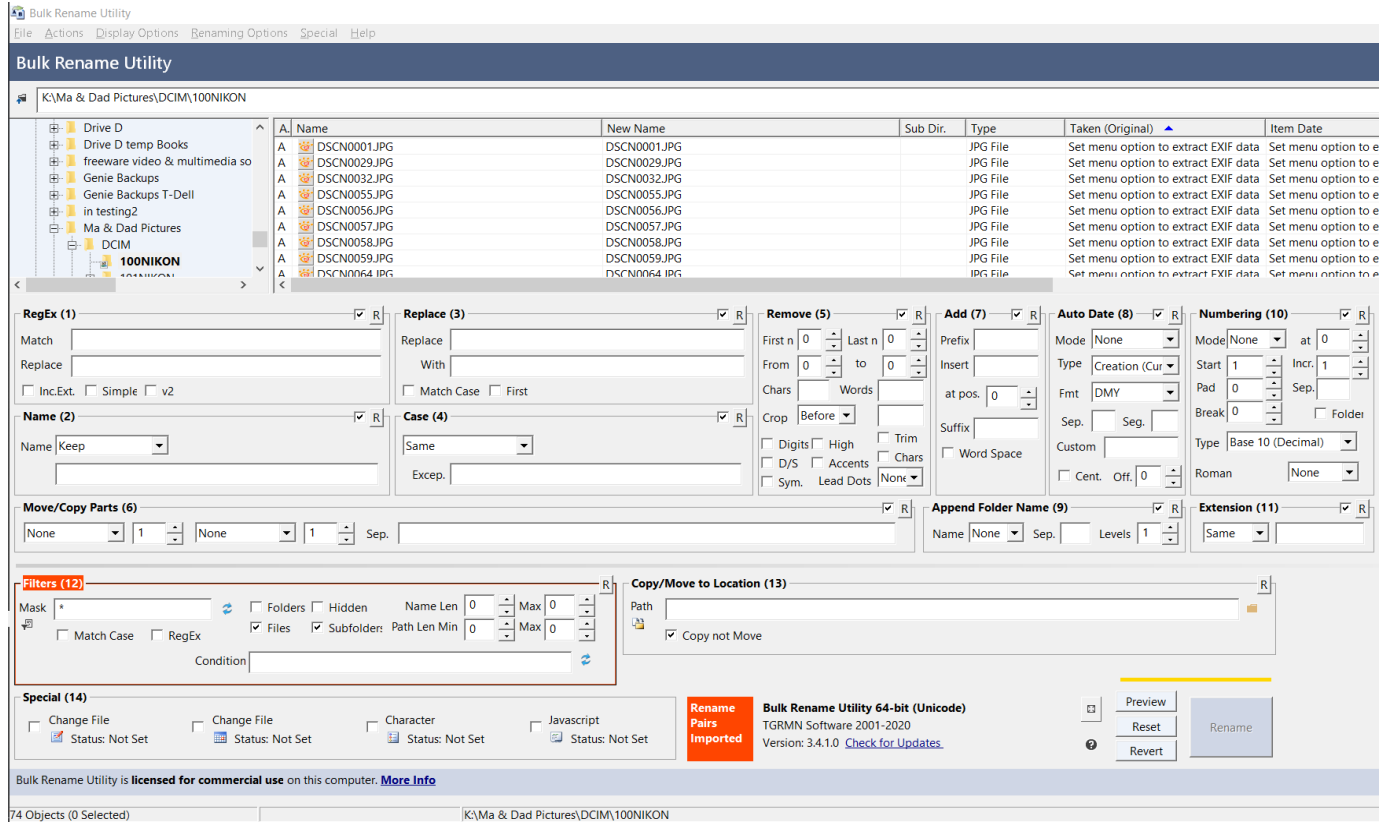
Notes:

1. Refer to 'User Interface' for more information.

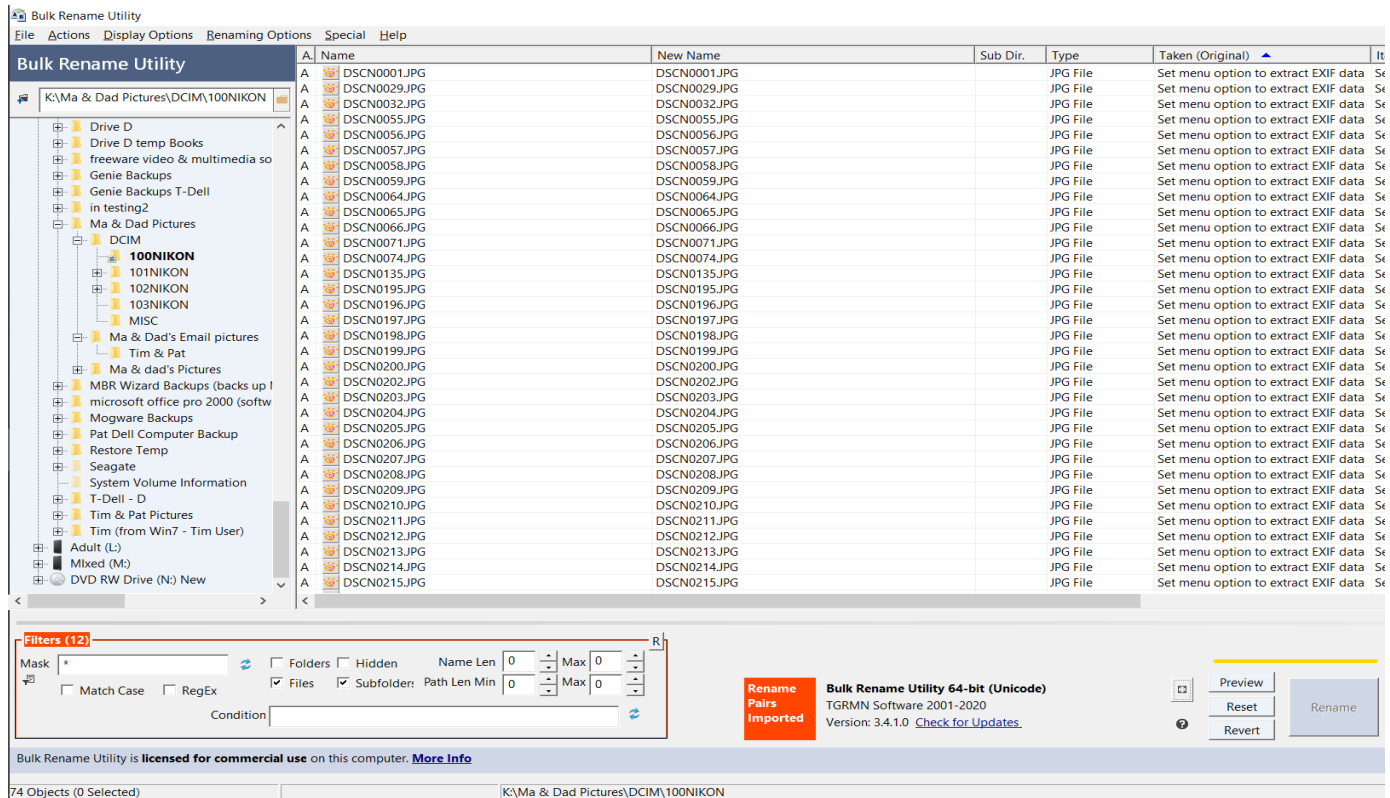
Display Options Menu

- Maximize File List (F9)

Before:

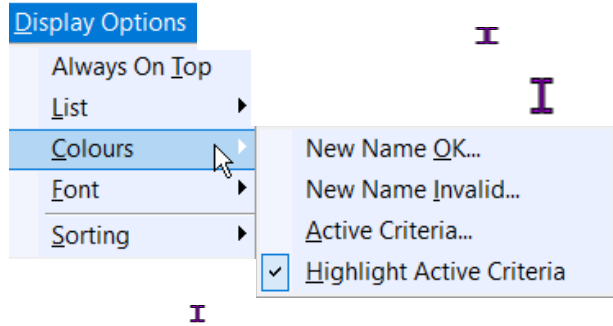


After:



Display Options Menu

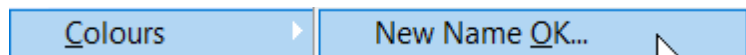
Colours



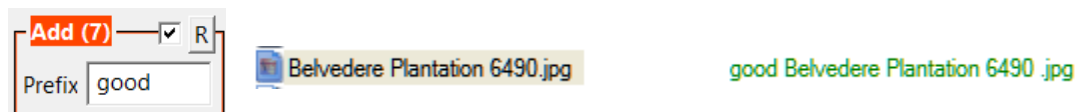
You can select the colour for the following three display characteristics.

- (1) *New Name – Ok* (default is light green in bold)

Hue: 79 Sat: 240 Lum: 60 Red: 4 Green: 128 Blue: 0

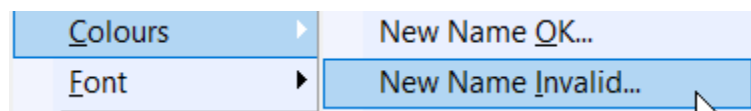


The colour to use if the new filename is acceptable to Windows and does not contain illegal characters or syntax errors.



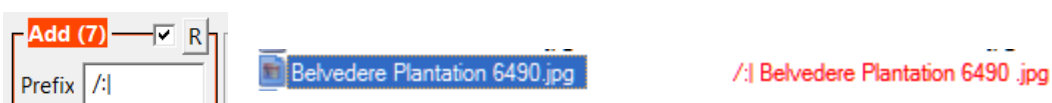
- (2) *New Name – Invalid* (default is red in bold)

Hue: 0 Sat: 240 Lum: 120 Red: 255 Green: 0 Blue: 0



The colour to use if the new filename is not acceptable to Windows and contains illegal characters. This could also be indicative of a possible syntax error.

Illegal characters



Display Options Menu

- (3) *Active Criteria* (default is orange in bold)

Hue: 11 Sat: 240 Lum: 120 Red: 255 Green: 69 Blue: 0



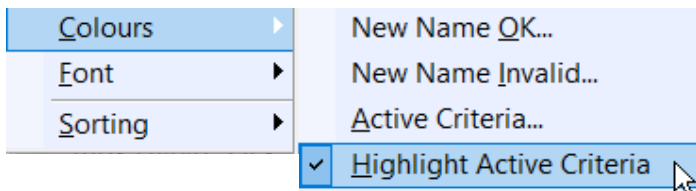
The colour to use if a criteria section is active – meaning that this section is selected and criteria has been entered. Note that ‘Highlight Active Criteria’ must be set for the colour to display.

Inactive

Active

Must restart application for any colour changes to ‘Active Criteria’ to occur.

- *Highlight Active Criteria*



Not Set:

Set:

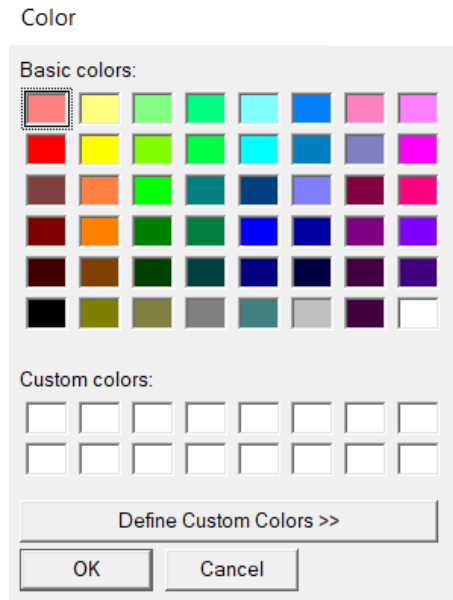
Notes:

1. The colour default is orange and is determined by ‘Active Criteria’ colour selection.

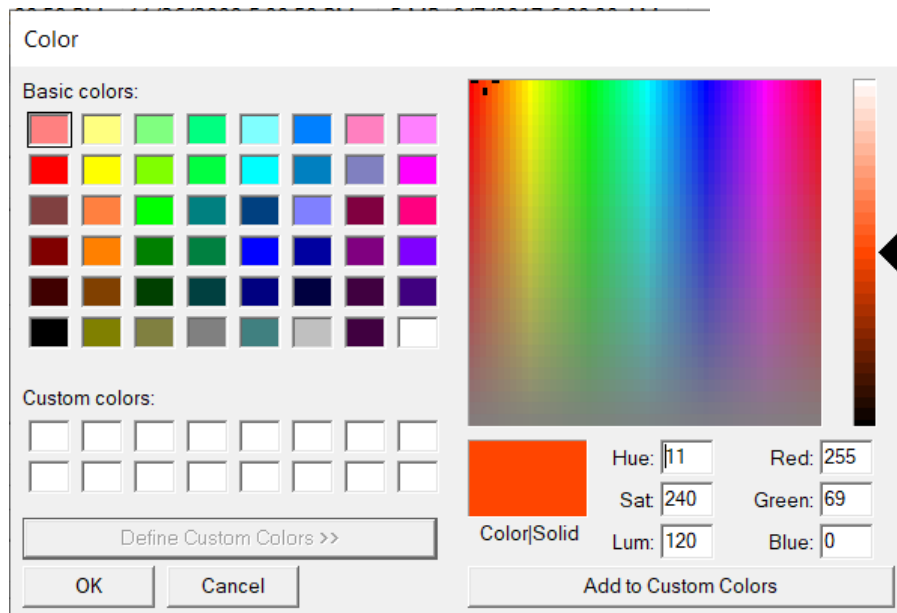
Display Options Menu

Changing the Colours

When you select any of the colour options, it presents you with the Colour selection dialog box.



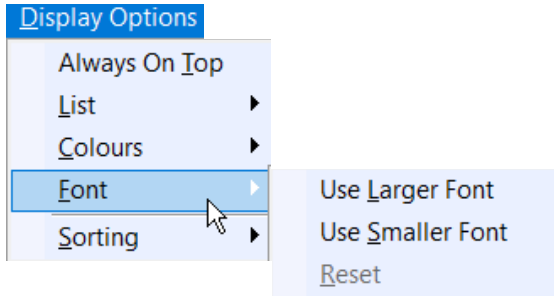
This offers a choice between Basic Colours that you can quickly choose from, along with a possible user defined 16 colours. If you click on the 'Define Custom Colours', the more advanced Colour Selection dialog box appears.



The colour selection here is more refined. Characteristics of Hue, Saturation, Luminance, along with the RGB values for Red, Green Blue can be specified. Once the colour is established, you can save the specifications as a Custom Colour by clicking on the 'Add to Custom Colours'.

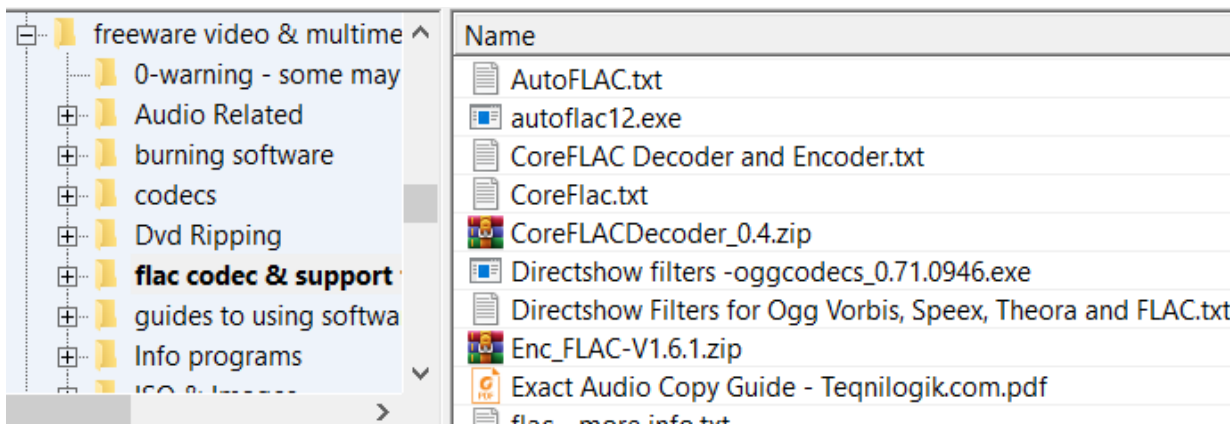
Display Options Menu

Font

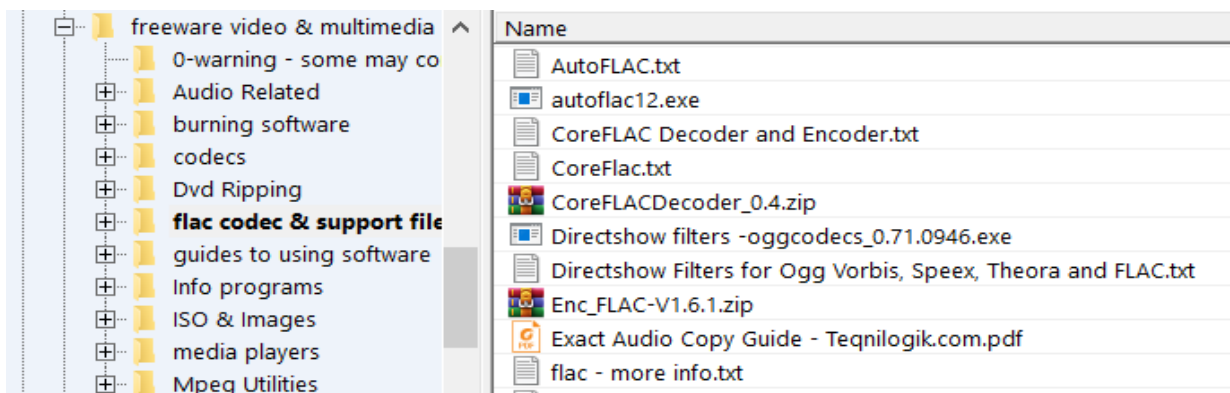


You have two font choices.

- Use Larger Font (default)



- Use Smaller Font



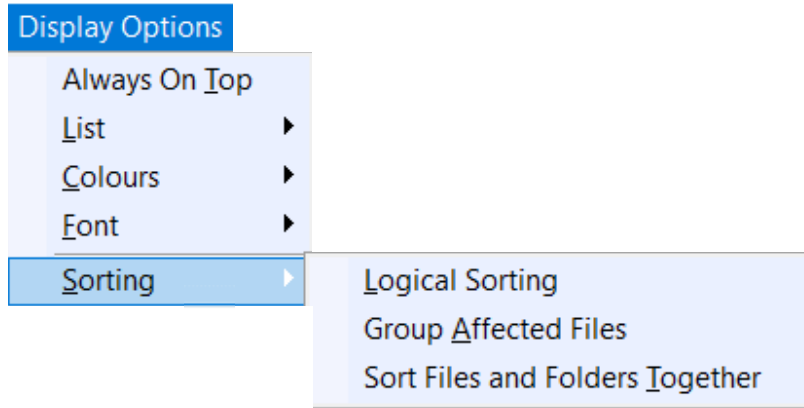
The smaller font also allows for more files to display in both the Navigation and the Content Pane.

Notes:

1. Must restart application for any font change to take effect.
2. Reset (Font) restores back to larger default Font.

Display Options Menu

Sorting



Unlike the other Display Options, Sorting affects the renaming process and is *not* just for Display purposes. In that regard, I think the Sorting function is mislabeled as a Display Option and really belongs in the Actions Menu.

Because BRU processes the files in the order in which they appear in the Content Pane, if you change the sort, you change the files' **order of appearance**, and thereby change the **order of processing**. This becomes especially apparent when '**Section #10: Numbering**' is applied. This is referenced in other sections of the book as the '**order of the displayed sequence**'.

For other BRU functions that affect the order of processing, please refer to:

- 'List - Reposition', under the Actions Menu
- 'Apply Random Sort to Current List', under the Actions Menu
- 'Rename in Reverse Order', under the Renaming Options Menu
- notation #6 under 'Program Notes' section earlier in this volume

- Logical Sorting

BRU uses Absolute Sorting as the default instead of Logical Sorting.

The Logical Sorting sequence (aka numeric sort or numeric order), was adopted as the default sort for all operating systems XP and higher. Logical Sorting sorts numbers from lowest value to highest value; thereby, '2' follows after '1'. Any filenames that contain numerals will be sorted by name first and by their numeric value second; thereby, 'text 2' follows after 'text 1'.

Absolute Sorting

In Absolute Sorting (aka string sort or string order), the sort sequence is based on the first numeric character regardless of the value. If that character is the same, e.g., a '1', it falls to the second character, e.g., '0'. In other words, numeric values are treated as text. The operating system does not differentiate between filenames that contain numeric values and those that do not; thereby, '10' follows after '1' and 'text 10' follows after 'text 1'.







































Display Options Menu

If that seems hard to follow, then this diagram and photos should help to clarify.

LOGICAL SORT







































Low to High

Name First
Numeric Value Second

Name	Name
 1.jpg	 text 1.jpg
 2.jpg	 text 2.jpg
 3.jpg	 text 3.jpg
 4.jpg	 text 4.jpg
 5.jpg	 text 5.jpg
 6.jpg	 text 6.jpg
 7.jpg	 text 7.jpg
 8.jpg	 text 8.jpg
 9.jpg	 text 9.jpg
 10.jpg	 text 10.jpg
 11.jpg	 text 11.jpg
 12.jpg	 text 12.jpg
 13.jpg	 text 13.jpg
 14.jpg	 text 14.jpg
 15.jpg	 text 15.jpg
 16.jpg	 text 16.jpg
 17.jpg	 text 17.jpg
 18.jpg	 text 18.jpg
 19.jpg	 text 19.jpg

ABSOLUTE SORT

First Character Regardless of Value. If Same Character,
Falls to the Second character. Values Treated as Text.

Name	Name
 1.jpg	 text 1.jpg
 10.jpg	 text 10.jpg
 11.jpg	 text 11.jpg
 12.jpg	 text 12.jpg
 13.jpg	 text 13.jpg
 14.jpg	 text 14.jpg
 15.jpg	 text 15.jpg
 16.jpg	 text 16.jpg
 17.jpg	 text 17.jpg
 18.jpg	 text 18.jpg
 19.jpg	 text 19.jpg
 2.jpg	 text 2.jpg
 3.jpg	 text 3.jpg
 4.jpg	 text 4.jpg
 5.jpg	 text 5.jpg
 6.jpg	 text 6.jpg
 7.jpg	 text 7.jpg
 8.jpg	 text 8.jpg
 9.jpg	 text 9.jpg

There may be situations when the files in BRU don't match up with the sort order in a directory, or perhaps you may want the sort ordered differently. This option when enabled directs BRU to use Logical Sorting. Logical Sorting will also require to be re-enabled after a system reboot because BRU defaults back to Absolute Sorting.

Display Options Menu

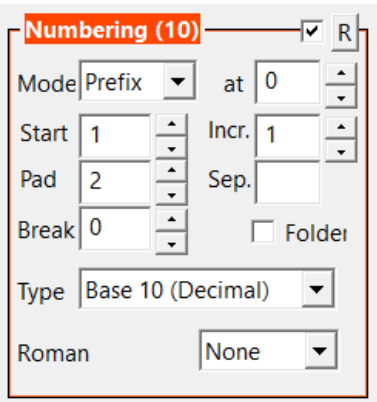
To make a group of files sort by a numeric value that is not logical or absolute-

Suppose you have this group of files that you just sorted by name descending (clicking on the heading in 'Name' column is a toggle – once for Ascending sort a-z, and again for descending z-a):

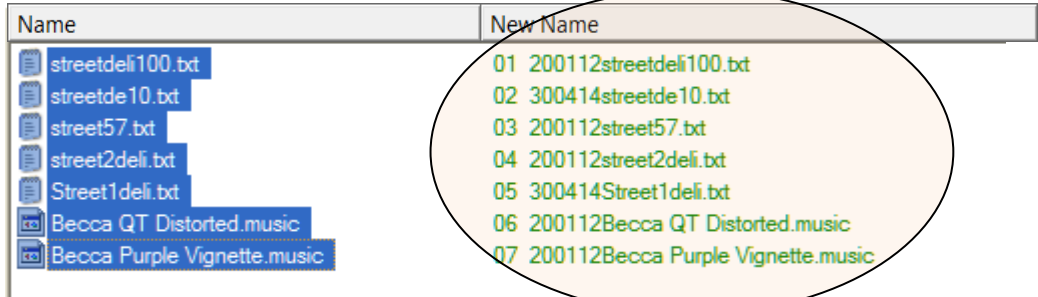
Name	New Name
streetdeli100.txt	200112streetdeli100.txt
streetde10.txt	300414streetde10.txt
street57.txt	200112street57.txt
street2deli.txt	200112street2deli.txt
Street1deli.txt	300414Street1deli.txt
Becca QT Distorted.music	200112Becca QT Distorted.music
Becca Purple Vignette.music	200112Becca Purple Vignette.music

You want to preserve this sort order. The easiest way is to auto append a number using the 'Select #12: Numbering':

This ...



.. yields this:

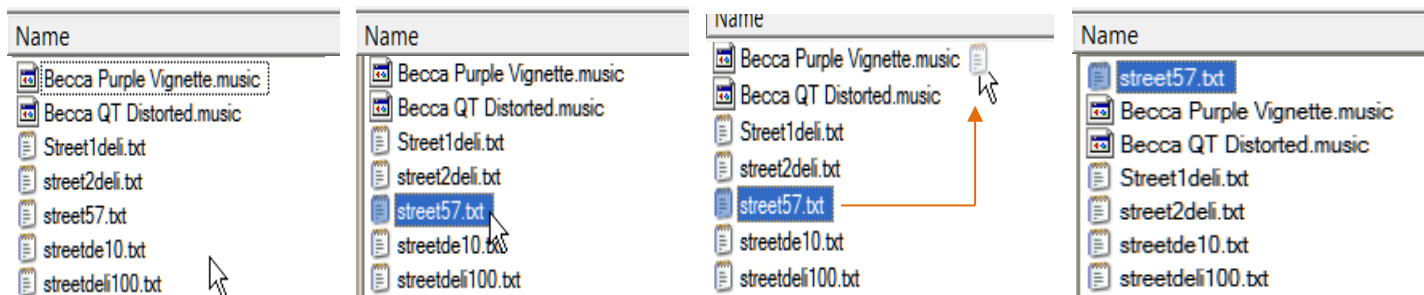


Name	New Name
streetdeli100.txt	01 200112streetdeli100.txt
streetde10.txt	02 300414streetde10.txt
street57.txt	03 200112street57.txt
street2deli.txt	04 200112street2deli.txt
Street1deli.txt	05 300414Street1deli.txt
Becca QT Distorted.music	06 200112Becca QT Distorted.music
Becca Purple Vignette.music	07 200112Becca Purple Vignette.music

This technique also works if you want to preserve the order after randomizing the sort (use the 'Random Sort' command from the Actions menu).

Notes:

1. File sorting can also get messed up if the files were entered into BRU using Drag and Drop.
2. You can manipulate the order by manually dragging each file using the mouse (or use the 'Reposition' option from the List sub-menu of the Actions Menu), and preserve that order using this numbering technique.



Name
Becca Purple Vignette.music
Becca QT Distorted.music
Street1deli.txt
street2deli.txt
street57.txt
streetde10.txt
streetdeli100.txt

Name
Becca Purple Vignette.music
Becca QT Distorted.music
Street1deli.txt
street2deli.txt
street57.txt
streetde10.txt
streetdeli100.txt

Name
Becca Purple Vignette.music
Becca QT Distorted.music
Street1deli.txt
street2deli.txt
street57.txt
streetde10.txt
streetdeli100.txt

Name
street57.txt
Becca Purple Vignette.music
Becca QT Distorted.music
Street1deli.txt
street2deli.txt
streetde10.txt
streetdeli100.txt

Display Options Menu

- Group Affected

You have a group of files with 4 filenames selected individually using non-consecutive selection (Ctrl + Left Mouse Click). They display as:

Name	New Name
streetdeli100.txt	04 200112streetdeli100.txt
streetde10.txt	streetde10.txt
street57.txt	street57.txt
street2deli.txt	03 200112street2deli.txt
Street1deli.txt	02 300414Street1deli.txt
Becca QT Distorted.music	Becca QT Distorted.music
Becca Purple Vignette.music	01 200112Becca Purple Vignette.music

After enabling Group Affected...

Name	New Name
Becca Purple Vignette.music	04 200112Becca Purple Vignette.music
Street1deli.txt	03 300414Street1deli.txt
street2deli.txt	02 200112street2deli.txt
streetdeli100.txt	01 200112streetdeli100.txt
Becca QT Distorted.music	Becca QT Distorted.music
street57.txt	street57.txt
streetde10.txt	streetde10.txt

... files with new filenames, the 'Affected Files', will be grouped together, segregated from the 'Unaffected Files'.

Notes:

1. The display of the newly grouped files [will only take place after you re-sort the column using a column heading](#).

e.g., click on 'Name' column heading to see the files grouped.

- This will not work using the column headings of Attribute, Size or Date to re-sort the files.
- You can sort the file list using the column headers as you would normally in Windows Explorer.

2. The files will always be processed in the order of the *displayed sequence* – especially useful when you're applying '[Section #12: Numbering](#)' criteria.

For more information see -

- 'List - Reposition', under the Actions Menu
- 'Apply Random Sort to Current List', under the Actions Menu
- 'Rename in Reverse Order', under the Renaming Options Menu
- notation #6 under 'Program Notes' section earlier in this volume

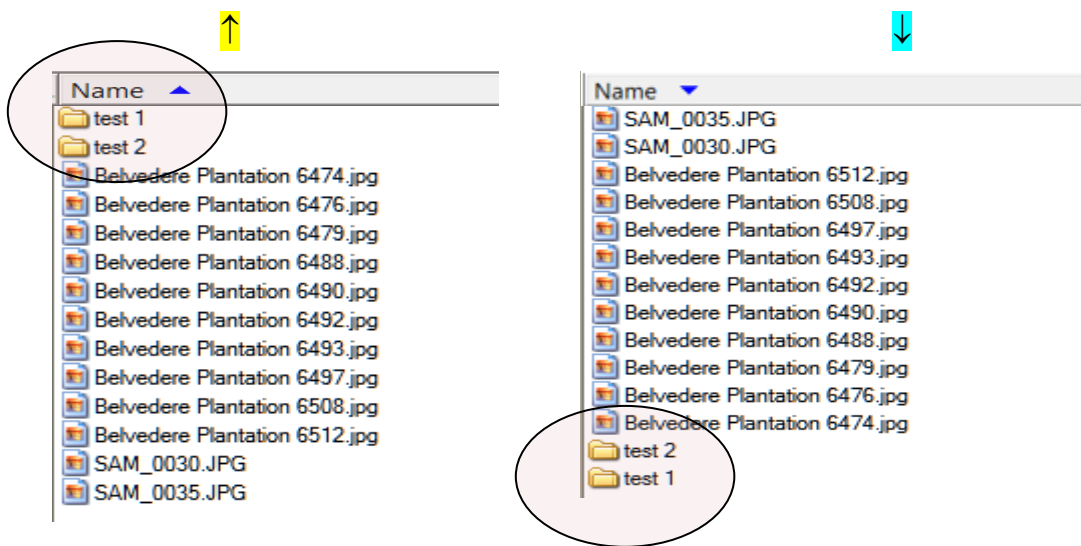
Display Options Menu

- Sort Files and Folders Together

Windows will typically sort the folder names first and then the files.

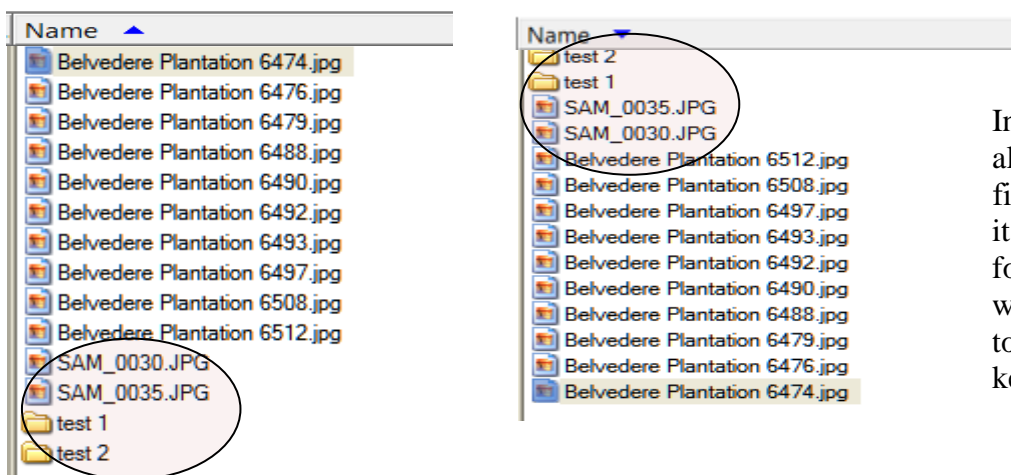
Example,

In an ascending sort the folders appear at the top and are sorted accordingly. The files that appear below these folders are also sorted accordingly and are kept segregated. This behaviour is reversed in a descending sort, while still maintaining the separation.



With, 'Sort Files and Folders Together' enabled, the folders are treated no differently than any other filename and are sorted 'mixed together' accordingly.

In an ascending sort the folders appear at the bottom in this example because, 'T' comes last after 'S'. In a descending sort, 'Test 2' comes before 'Test 1' and 'S' follows. In both of these cases, if there were filenames beginning with 'T', these would be mixed in as well with no separation between files and folders.



In these examples to the left, although there are no other filenames beginning with 'T', it can still be observed that the folder names are sorted together with filenames and appear together rather than sorted and kept segregated.

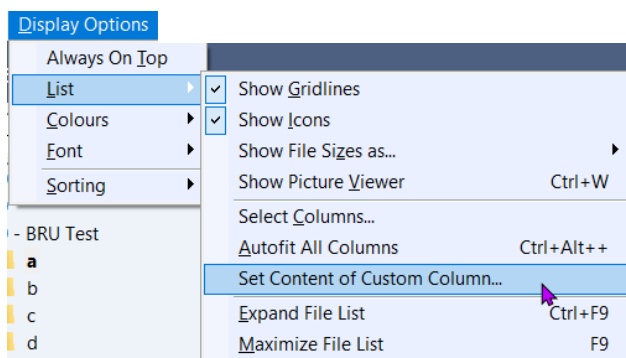
Display Options Menu

v3.4 New Additions

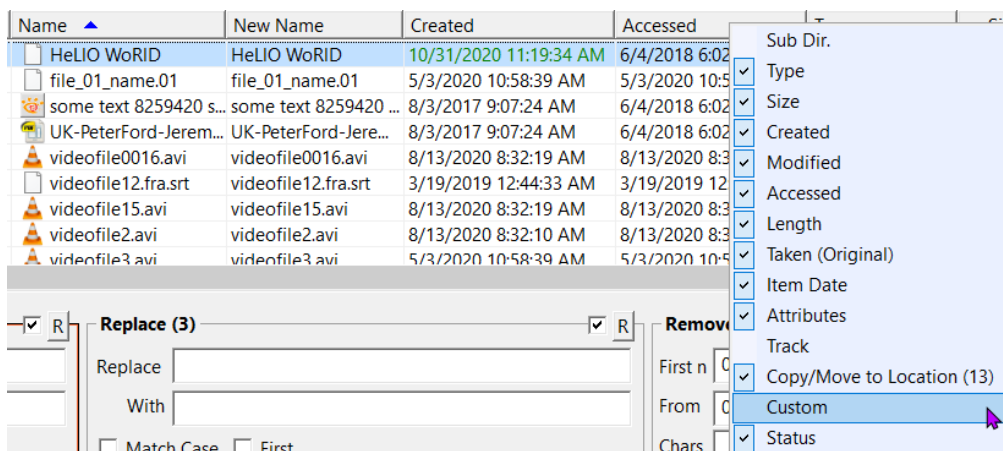
List

Custom Column –

The Custom Column displays information that the user added through the ‘Set Content of Custom Column’ from the List submenu under the Display Options Menu. This added information is defined by the File Metadata, including Windows File Properties tags, version 2 EXIF tags, and Hash tags.



Right click anywhere in the Headings of the Content Pane presents the list of available columns.



From here you can select the ‘Custom Column’ which will add the column to the Content Pane. It will be empty except for a help message, unless you have previously defined the custom data, as will be discussed.

Attributes	Name	New Name	Custom
A	HeLIO WoRID	HeLIO WoRID	Set content of custom column in Display Options -> List
	file_01_name.01	file_01_name.01	Set content of custom column in Display Options -> List

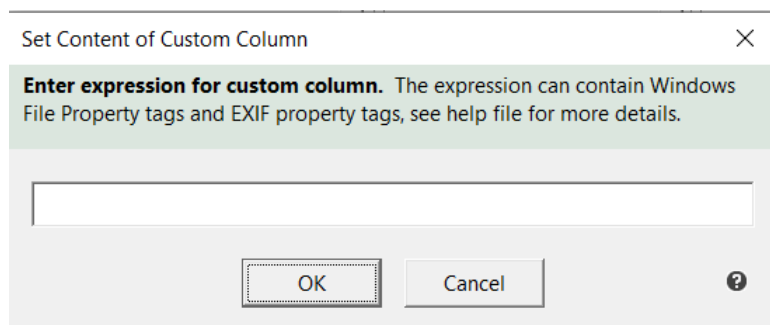
Display Options Menu

v3.4 New Additions

- Set Content of Custom Column

This is where the Custom information that will appear in the column is defined. The definition is set by using File Metadata including Windows File Properties tags, version 2 EXIF tags and Hash tags.

For example, you can set the Custom Column to <(Title)> which would display the Windows File Property, 'Title', for each file. For more information on the tags that can be used for the Custom Column, see 'Custom Column' under the List Menu. Selecting this option brings up:



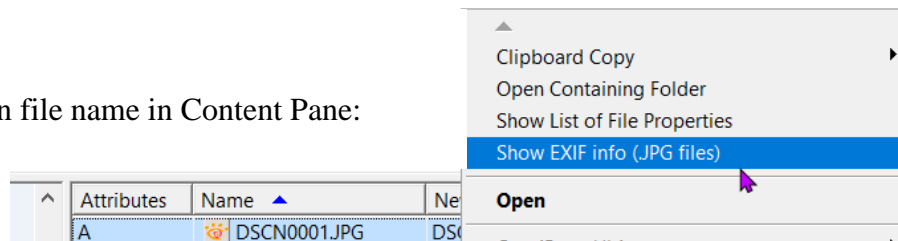
Display Options Menu

v3.4 New Additions

To see what properties are available for a file that can be used to define your custom content –

EXIF Tags –

Right Click on file name in Content Pane:



e.g.,

Version 2 EXIF tags for file, DSCN0001.JPG:

EXIF Property	Value
exif:ImageResolution	3648x2736
exif:ImageWidth	3648
exif:ImageHeight	2736
exif:Make	NIKON
exif:Model	COOLPIX L20
exif:Orientation	1
exif:XResolution	300.000000
exif:YResolution	300.000000
exif:ResolutionUnit	2
exif:Software	COOLPIX L20 V1.0
exif:DateTime	2009:09:01 13:38:35
exif:DateTimeOriginal	2009:09:01 13:38:35
exif:DateTimeDigitized	2009:09:01 13:38:35
exif:ExposureTime	0.016667
exif:FNumber	3.100000
exif:ExposureProgram	2
exif:ISO Speed Ratings	168
exif:FocalLength	6.720000
exif:Flash	24
exif:MeteringMode	5
exif:LensInfo.FocalLengthIn35mm	38.000000

This example lists properties that are only for media and image files that contain Metadata. Other filetypes may have different properties.

Display Options Menu

v3.4 New Additions

Windows Properties –

Right click on file name in Content Pane:



e.g.,

Windows Properties for File, 'HeLIO WoRID':

Label	Value	Name
	0	System.ZoneIdentifier
	{9BF73ECC-0000-0000-0000-100000000000}	System.VolumeId
	6684827675453505162	System.ThumbnailCacheId
Activity		System.StorageProviderAggregatedCustomStates
Size	50.0 KB	System.Size
	fileys; stream	System.Shell.SFGAOfFlagsStrings
Sharing status	Not shared	System.SharingStatus
Shared with		System.SharedWith
File ownership		System.Security.EncryptionOwnersDisplay
	1077936503	System.SFGAOfFlags
Perceived type	Unspecified	System.PerceivedType
	L:\0 - BRU Test\A\HeLIO WoRID	System.ParsingPath
	HeLIO WoRID	System.ParsingName
Offline status		System.OfflineStatus
Availability		System.OfflineAvailability
	No	System.NotUserContent
Network location		System.NetworkLocation
Link target		System.Link.TargetParsingPath
Kind		System.Kind
Type	File	System.ItemTypeText
Item type	File	System.ItemType
Path	HeLIO WoRID (L:\0 - BRU Test\A)	System.ItemPathDisplayNarrow
Path	L:\0 - BRU Test\A\HeLIO WoRID	System.ItemPathDisplay
	HeLIO WoRID	System.ItemNameDisplayWithoutExtension
Name	HeLIO WoRID	System.ItemNameDisplay
	HeLIO WoRID	System.ItemName

Display Options Menuv3.4 New Additions**Windows Properties cont –**

Windows Properties for File, 'HeLlO WoRlD' cont.:

Label	Value	Name
Folder	a (L:\0 - BRU Test)	System.ItemFolderPathDisplayNarrow
Folder path	L:\0 - BRU Test\a	System.ItemFolderPathDisplay
Folder name	a	System.ItemFolderNameDisplay
Date	8/3/2017 9:07 AM	System.ItemDate
Shared	No	System.IsShared
	Files	System.IsFolder
	6	System.FilePlaceholderStatus
Owner	DESKTOP-XXXXXXXXX\Tim	System.FileOwner
Filename	HeLlO WoRlD	System.FileName
File extension		System.FileExtension
Attributes	A	System.FileAttributes
Date last saved	6/4/2018 6:02 PM	System.Document.DateSaved
Content created	8/3/2017 9:07 AM	System.Document.DateCreated
Date modified	6/4/2018 6:02 PM	System.DateModified
Date imported	8/3/2017 9:07 AM	System.DateImported
Date created	8/3/2017 9:07 AM	System.DateCreated
Date accessed	6/4/2018 6:02 PM	System.DateAccessed
Computer	DESKTOP-XXXXXXXXX (this PC)	System.ComputerName
Parent id		System.AppUserModel.ParentID
AppUserModelId		System.AppUserModel.ID

Hash Tags that are available for any file type –

hash:crc32
 hash:keccak
 hash:sha1
 hash:sha256
 hash:sha3
 hash:md5

Notes:

1. For Windows properties, the expression for the Custom Column can use either the **Name** or **Label** of the property.

Display Options Menu

v3.4 New Additions

To use the Custom Column definition, place the property in the proper syntax and enter this in the data entry field provided. The syntax consist of the property placed in parentheses enclosed between angular brackets.

Examples:


Use EXIF – <(exif:Make)>

Set Content of Custom Column

Enter expression for custom column.
File Property tags and EXIF property tag:

<(exif:make)>

OK

Name ▲	New Name	Custom
 DSCN0001.JPG	DSCN0001.JPG	NIKON

Use Windows Property Label – <(Date Last Saved)>

Set Content of Custom Column

Enter expression for custom column.
File Property tags and EXIF property tag:

<(Date last saved)>

Name ▲	New Name	Custom
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID	6/4/2018 6:02 PM

or, use Windows Property Name – <(System.Document.DateSaved)>

Set Content of Custom Column

Enter expression for custom column.
File Property tags and EXIF property tag:

<(System.Document.DateSaved)>

Name ▲	New Name	Custom
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID	6/4/2018 6:02 PM

Use Hash – <(hash:md5)>

Set Content of Custom Column

Enter expression for custom column.
File Property tags and EXIF property tag:

<(hash:md5)>

Name ▲	New Name	Custom
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID	1aa5cd7a0eb73f4718d0ba764e0e9b0c

Display Options Menu

v3.4 New Additions

You can also define more than one property and include literals (any character that can be reproduced on the keyboard) such as <hyphens>, etc.

If you wish to include angle brackets or parentheses then place them **around** the angle brackets. Anything outside of the angle brackets are considered to be literals.

Angle Brackets as literals:

The dialog box shows the text: "Set Content of Custom Column", "Enter expression for custom column.", and "File Property tags and EXIF property tag". The input field contains the expression: "<<(Date modified)>>".

Name ▲	New Name	Custom
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID	<6/4/2018 6:02 PM>

Parentheses as literals:

The dialog box shows the text: "Set Content of Custom Column", "Enter expression for custom column.", and "File Property tags and EXIF property tag". The input field contains the expression: "(<(Date modified)>)".

Name ▲	New Name	Custom
<input type="checkbox"/> file_01_name.01	file_01_name.01	(5/3/2020 10:58 AM)

Multiple Properties defined:

The dialog box shows the text: "Set Content of Custom Column", "Enter expression for custom column. The expression can contain Win File Property tags and EXIF property tags, see help file for more details.", and "File Property tags and EXIF property tag". The input field contains the expression: "File Size is <(System.Size)> - Modified Date is <(Date modified)>".

Name ▲	New Name	Custom
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID	File Size is 50.0 KB - Modified Date is 6/4/2018 6:02 PM)

Notes:

1. The placing of <space> and <hyphen> in angular brackets is something I tend to do personally, and has nothing to do with the syntax described for the Custom Column Definition.

Display Options Menu

v3.4 New Additions

Notes:

1. The items in the file list can be sorted under this column by clicking on the column header.
2. For more information on Windows Properties, EXIF Tags and Hash Tags, refer to headings under the volume section of, Section # 7 – Add, earlier in the book.

Item Date –

Item Date has already been extensively discussed and analyzed under the section, ‘[Section #8:Auto Date](#)’. Item Date is taken from the Windows Property, (Name) System.ItemDate, (Label) Date.

To summarize Item Date:

1. For filetypes that are **not supported** by Taken (Original) and correspondingly, EXIF DateTakenOriginal, or EXIF DateDigitized, Item Date assumes the earliest date of either the Windows Properties of Date Created , or Date Modified with the exception of certain Document filetypes.
2. For filetypes that **are supported** by Taken (Original) and correspondingly, EXIF DateTakenOriginal, or EXIF DateDigitized, Item Date will assume this value. Taken (Original) will also hold this same value.
3. For filetypes that Windows **recognizes as Document filetypes**, Item Date will assume the value of the Windows Property, Last Date Saved.
4. For filetypes that Windows **does not recognize as Document filetypes** will be handled in the same manner as provided for those filetypes that are not supported by Taken (Original). Item Date will assume the earliest date value between the two Windows Properties, Date Created and Date Modified.
5. According to Microsoft, Item Date may be based on properties other than those I have listed, but to date, I have found none.
6. Item Date does not support ‘Seconds’, only ‘Minutes’. Microsoft dropped the support of seconds.

Created	Taken (Original)	Item Date
9/1/2009 2:38:34 PM	9/1/2009 1:38:35 PM	9/1/2009 1:38 PM

Therefore, Taken (Original), if available, is more accurate to the seconds.

7. TGRMN main reasoning for adding support for Item Date is to provide an alternative for media and image files that do not have EXIF Metadata. I applaud this though because of its versatility in use with other data types as well.

Display Options Menu

v3.4 New Additions

The advantages of Item Date over Taken (Original) are:

1. Taken (Original) only works with limited files – as noted previously, JPEG images (.JPG or .JPEG extension), TIFFs (.TIF, .TIFF), Nikon (.NEF) and Canon (.CR2) files under the following prerequisites:
 - a. The Windows Property, System.Photo.DateTaken must have a value.
 - b. System.Photo.DateTaken will not have a value unless EXIF DateTimeOriginal has a value.

whereas:

Item Date is applied to any image filetype under the following prerequisites:

- a. Item Date will assume the value of EXIF DateTimeOriginal if available.
- b. If EXIF DateTimeOriginal is not available, Item Date can assume other EXIF Values if available.

These include:

CreateDate
Photo.DateTimeDigitized

- c. If EXIF Metadata is not available, or for those image filetypes that do not map to EXIF metadata, other Metadata values will be assumed, meaning, that even if Taken (Original) has no value, Item Date most likely will. This includes Windows File Metadata, e.g.,

Date Created
Date Modified

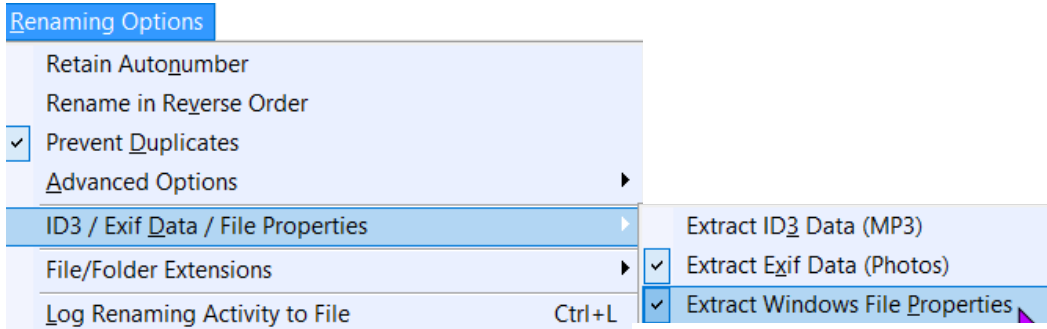
2. Item Date applies to most media, pictures, and video filetypes including HEIC, RAW camera files, etc.,
3. Item Date applies to most (if not all) filetypes, as long as Metadata exists. If there is none, the column value will be blank.
4. Taken (Original) is restricted to the epoch date that limits the earliest recognized date to 01/01/1970 (see ‘Renaming with Dates Prior to January 1, 1971’ in the Appendix of Volume II for further information on the epoch dates). Item Date does not have this restriction (*possibly* no longer restricted in current version).

For further information, refer to the ‘[Section #8:Auto Date](#)’ section.

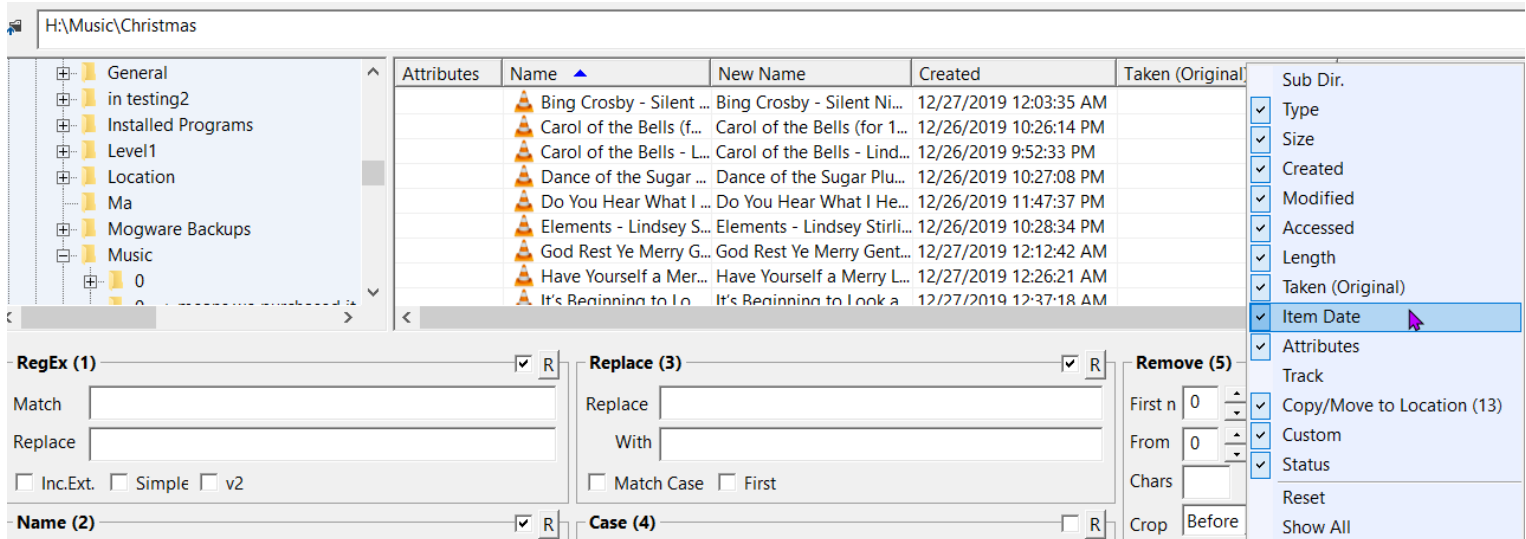
Display Options Menu

v3.4 New Additions

To use, you must first enable the option "Extract Windows File Properties" from the Renaming Options menu.



Right click anywhere in the Headings of the Content Pane presents the list of available columns.



From here you can select the 'Item Date' column which will add the column to the Content Pane.



Display Options Menu

v3.4 New Additions

A Brief primer on EXIF Date Taken accuracy- GIGO

In the Details tab of the Windows Property sheet, you may notice the item, Date Taken. This corresponds to the EXIF Metadata Photo.DateTimeOriginal. Date Taken is just another name for it. If Date Taken is available, both Taken (Original) and Item Date will assume this value. Supported and unsupported filetypes for DateTimeOriginal have previously been discussed.

Although the Item Date for these supported image filetypes is mapped from EXIF Metadata and generally accurate, this accuracy, however, is only as proven as the EXIF data whence it was mapped. This is all dependent on the device that created the file. But there is an old Computer Proverb, GIGO – Garbage In, Garbage Out. You can't expect accuracy if your analysis and conclusion is based on faulty initial data.

Here are the problems associated with the accuracy of the EXIF Date Taken (**DateTimeOriginal**) -

- (1) The Device's date and time setting is inaccurate; therefore the file timestamp will also be inaccurate. This will also occur when the date and time settings of the camera are not updated during travels between different time zones.
- (2) The camera device settings reset when the battery is replaced causing a discrepancy between the photos' timestamps.
- (3) With multiple camera use, there can be sync problems between timestamps of photos taken with different devices.
- (4) A scanned printed photo's digital copy timestamp references the date of the scan, not the date the photo was taken.
- (5) The original timestamp is lost for photos downloaded or exported from an outside source.
- (6) The timestamp has been edited using a third party utility, e.g. EXIFTool

The GIGO caveat must be adhered to by basing the accuracy of an EXIF item over a, e.g., Windows Property.

Notes:

1. In all of these scenarios, the date has to be adjusted back. As already mentioned, EXIFTool can modify this and other Metadata. BRU cannot. BRU can only read and extract the Metadata information for purposes of appending onto a filename in a Rename operation.
2. Professional photographers will always check their battery and camera settings before a shoot, and if multiple cameras are in use, then all of the devices are synchronized. Reserve cameras are also set for any anticipated time zone changes to avoid the problems related to the first scenario.



Renaming Options Menu

Renaming Options

- Retain Autonumber
- Rename in Reverse Order
- Prevent Duplicates
- Advanced Options ▶
- ID3 / Exif Data / File Properties ▶
- File/Folder Extensions ▶
- Log Renaming Activity to File Ctrl+L
- Show Warning Message Before Renaming
- Show Confirmation Message After Renaming

Renaming Options Menu

Retain Autonumber

Renaming Options

Retain Autonumber

Retains the last Autonumber used so the counter doesn't reset back to zero.

If you have criteria set in 'Section #10: Numbering', it will Autonumber selected files as shown below.

Criteria entered

Numbering (10) R

Mode **Prefix** at **0**

Start **1** Incr. **1**

Pad **2** Sep.

Break **0** Folder

Type **Base 10 (Decimal)**

Roman **None**

Results in:

Name ▲	New Name
Belvedere Plantation 6474.jpg	01 Belvedere Plantation 6474.jpg
Belvedere Plantation 6476.jpg	02 Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg	03 Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg	04 Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg	05 Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg	06 Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	07 Belvedere Plantation 6493.jpg

Last value for counter = 07. The counter value is based on the current increment with the Pad value, '2' or two numeric digit places. Current value is now '08'.

Note: Sep. character used in the above example is a <space>

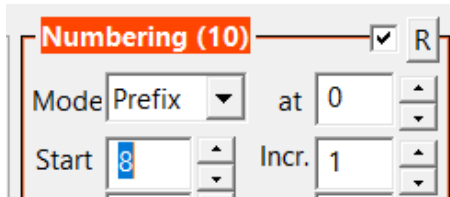
Once you perform the Rename action, the 'counter' resets from the current value, '08' back to '01'. The example below shows that the files prefixed 01 through 07 have been renamed, and when further files are selected afterwards, the file count begins at 01 again.

Name ▲	New Name
01 Belvedere Plantation 6474.jpg	01 Belvedere Plantation 6474.jpg
02 Belvedere Plantation 6476.jpg	02 Belvedere Plantation 6476.jpg
03 Belvedere Plantation 6479.jpg	03 Belvedere Plantation 6479.jpg
04 Belvedere Plantation 6488.jpg	04 Belvedere Plantation 6488.jpg
05 Belvedere Plantation 6490.jpg	05 Belvedere Plantation 6490.jpg
06 Belvedere Plantation 6492.jpg	06 Belvedere Plantation 6492.jpg
07 Belvedere Plantation 6493.jpg	07 Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	01 Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	02 Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	03 Belvedere Plantation 6512.jpg

To clarify, the first renaming operation renamed the files using the Numbering settings from 01 through 07. This operation has been completed. The Auto Number is currently at the value '08', thus in the next renaming operation, the count begins at 08. This is not related to the Break option. Using the Break, the Auto Number value would be reset within the same single renaming operation. The example above is performed in two separate renaming operations.

Renaming Options Menu

You can set the criteria to begin at whatever number you want, so changing the Start to 08 will solve this little dilemma.



	Belvedere Plantation 6497.jpg	08 Belvedere Plantation 6497.jpg
	Belvedere Plantation 6508.jpg	09 Belvedere Plantation 6508.jpg
	Belvedere Plantation 6512.jpg	10 Belvedere Plantation 6512.jpg

If, however, you wish to save yourself the trouble, enable 'Retain Autonumber' and it will start the following group of files at the next incremented number.

Using the same settings for 'Section #10: Numbering' as before:

First Renaming Operation:

Name	New Name
Belvedere Plantation 6476.jpg	02 Belvedere Plantation 6476.jpg
Belvedere Plantation 6479.jpg	03 Belvedere Plantation 6479.jpg
Belvedere Plantation 6488.jpg	04 Belvedere Plantation 6488.jpg
Belvedere Plantation 6490.jpg	05 Belvedere Plantation 6490.jpg
Belvedere Plantation 6492.jpg	06 Belvedere Plantation 6492.jpg
Belvedere Plantation 6493.jpg	07 Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	Belvedere Plantation 6512.jpg

Second Renaming Operation –

Without enabling 'Retain Autonumber':

Name	New Name
02 Belvedere Plantation 6476.jpg	02 Belvedere Plantation 6476.jpg
03 Belvedere Plantation 6479.jpg	03 Belvedere Plantation 6479.jpg
04 Belvedere Plantation 6488.jpg	04 Belvedere Plantation 6488.jpg
05 Belvedere Plantation 6490.jpg	05 Belvedere Plantation 6490.jpg
06 Belvedere Plantation 6492.jpg	06 Belvedere Plantation 6492.jpg
07 Belvedere Plantation 6493.jpg	07 Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	01 Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	02 Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	03 Belvedere Plantation 6512.jpg

With 'Retain Autonumber' enabled:

Name	New Name
01 Belvedere Plantation 6474.jpg	01 Belvedere Plantation 6474.jpg
02 Belvedere Plantation 6476.jpg	02 Belvedere Plantation 6476.jpg
03 Belvedere Plantation 6479.jpg	03 Belvedere Plantation 6479.jpg
04 Belvedere Plantation 6488.jpg	04 Belvedere Plantation 6488.jpg
05 Belvedere Plantation 6490.jpg	05 Belvedere Plantation 6490.jpg
06 Belvedere Plantation 6492.jpg	06 Belvedere Plantation 6492.jpg
07 Belvedere Plantation 6493.jpg	07 Belvedere Plantation 6493.jpg
Belvedere Plantation 6497.jpg	08 Belvedere Plantation 6497.jpg
Belvedere Plantation 6508.jpg	09 Belvedere Plantation 6508.jpg
Belvedere Plantation 6512.jpg	10 Belvedere Plantation 6512.jpg

Notes:

- 'Retain Autonumber' must be enabled prior to the first renaming operation to retain the Autonumber for subsequent renaming operations.
- The Autonumber can be retained even after you shut down the program by using the 'Favourites'. For further information, refer to 'Open\ Save' from the File Menu.

Renaming Options Menu




Rename in Reverse Order

Renaming Options







Retain Auto <u>n</u> umber
Rename in Reverse Order

Files are generally renamed from top to bottom, e.g., the first item in the list followed by the second item ... (also refer to references in this book regarding the **order of the displayed sequence**). But this can cause you problems if files with those filenames already exist.

For example, Let's say you have...

Name ▲
 1.jpg
 2.jpg
 3.jpg

.. and you want to rename...

Name ▲	New Name
 1.jpg	 2.jpg
 2.jpg	 3.jpg
 3.jpg	 4.jpg

The first rename (1.jpg to 2.jpg) would fail because 2.jpg already exists.

Error



Item 'H:\Test\Test 1\1.jpg' could not be renamed.
Target item 'H:\Test\Test 1\2.jpg' already exists. To prevent duplicate names, enable the option 'Prevent Duplicates'.

Ignore this error and continue processing?

Ignore

Ignore All

Cancel

This option removes the problem, processing the files in reverse order –




Rename Operation Complete!



3 item(s) successfully processed.

3.jpg renamed to 4.jpg
2.jpg renamed to 3.jpg
1.jpg renamed to 2.jpg

Results in:

Name ▲
 2.jpg
 3.jpg
 4.jpg

Renaming Options Menu

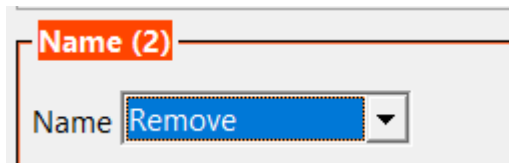
Notes:

1. Example adapted from BRU Manual.
2. BRU manual considers this an ‘Advanced’ option. Not to be confused with the ‘Advanced Options’ menu selection.
3. Don’t forget to turn off this option when through.

In case you were curious..

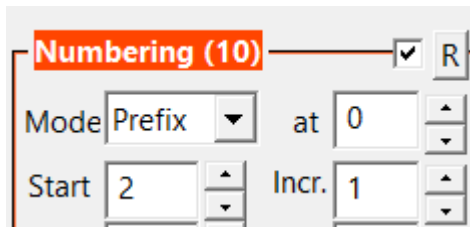
Example was created using -

‘Section # 2: Remove’ –



Name ▲	New Name
1.jpg	.jpg
2.jpg	.jpg
3.jpg	.jpg

‘Section # 10: Numbering’ –



Name ▲	New Name
1.jpg	21.jpg
2.jpg	32.jpg
3.jpg	43.jpg

Remove and Numbering Together results in:

Name ▲	New Name
1.jpg	2.jpg
2.jpg	3.jpg
3.jpg	4.jpg

Order of Evaluation – Remove (2) is performed first, removing the filename, then Numbering (10) is applied, starting at the ‘2’ value and incrementing + 1 for each filename thereafter.

Renaming Options Menu

Prevent Duplicates

Renaming Options

Retain Autonumber
Rename in Reverse Order

Prevent Duplicates

This will prevent situations where a rename operation would fail because the resulting filename would create ‘an Existing File’ error. If this option is set, the program will automatically append a suffix with a numeric using the format:

<numeric> (underscore followed by a numeric character 1 through 99)

The program would first use ‘_1’ as the suffix, and failing that would continue with the next number ‘_2’ all the way up to 99 before generating an error.

Example (with option set):

Replace (3)

Replace

With

	Name ▲	New Name
	text 1.jpg	text 2.jpg
	text 2.jpg	text 2.jpg
	text 3.jpg	text 3.jpg
	text 4.jpg	text 4.jpg

Results in:

	Name ▲	New Name
	text 2_1.jpg	text 2_1.jpg
	text 2.jpg	text 2.jpg
	text 3.jpg	text 3.jpg
	text 4.jpg	text 4.jpg

Notes:

1. This option is set by default.

Renaming Options Menu

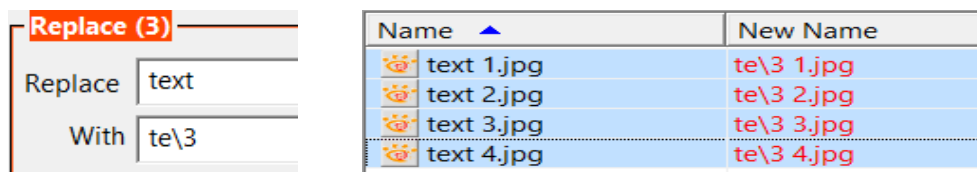
Advanced Options



- Allow Using \' in Renaming Criteria for Creation of New Folders

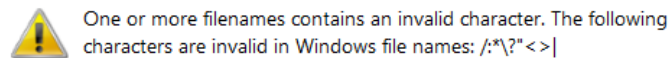
This option can create new folders during the renaming process. The creation of the New folders cannot be undone by using the 'Undo Rename' option from the 'Actions Menu' or by pressing Ctrl + Z. You can, however, still undo the renaming process itself.

The following simple example will create a new sub directory, 'te' off of the current directory path, '..\Keep\a', and move the renamed files into it.



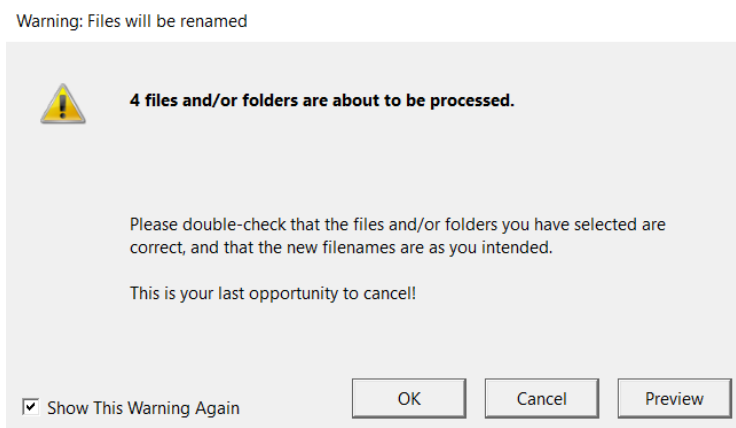
1. Typically, when you try to rename these files,

it will generate an error:



Renaming Options Menu

Click Yes again and... well, this program really gives you a chance to change your mind doesn't it?



Note:

The Preview will not indicate that directories are about to be created so best to heed the previous '3' warnings if you want to back out.



Current Name	New Name	Action
text 1.jpg	te\3 1.jpg	Name Change
text 2.jpg	te\3 2.jpg	Name Change
text 3.jpg	te\3 3.jpg	Name Change
text 4.jpg	te\3 4.jpg	Name Change

Click 'OK'. The backslash will create a directory, 'te'.

4. The following message indicates that the Tree in the Navigation Pane has been affected by the renaming operation.

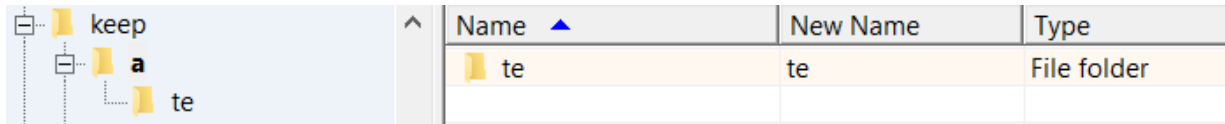
Bulk Rename Utility

Some folders have been renamed: use CTRL+F5 if you want to refresh the folder tree and file list to reflect these changes.

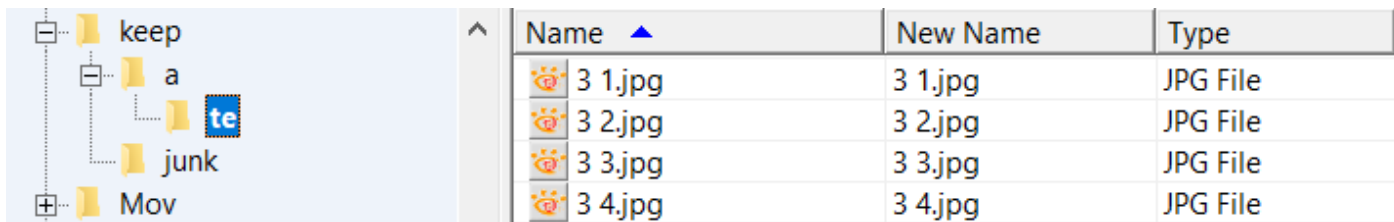
Renaming Options Menu

Look at both the Navigation Pane (after Ctrl + F5) and the Content Pane to see the results.

In this photo you can see the new file path of `'..\Keep\te'`...



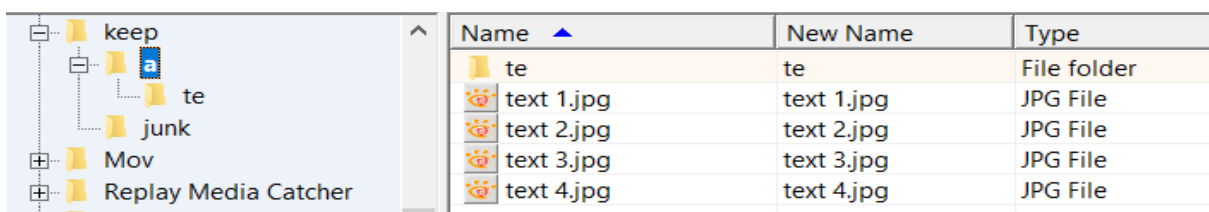
In this photo you can see that the renamed files have been moved into the new sub directory, 'te'...



This doesn't change the file content. The files are unaltered. They have just been renamed and moved.

Although you cannot remove the 'te' directory through the 'Undo', it will restore the file names and even move them back into their original directory.

After 'Undo':



Any unwanted sub directories can be deleted manually using a File Manager or as an alternative, use the 'Delete' option from the right-click Context Menu.

Notes:

1. Use this option with caution.
2. Before using, it is recommended to backup your files.
3. The process itself is known as 'Folderize'. User-defined hierarchies of directory/subdirectory structures are created for the purpose of organizing files through sorting them into these folders.
 - a. 'Section #13: Copy/Move To Location', 'Section #9: Append Folder Name', 'Section #3: Replace' and 'Section #7: Add', could all be used to create a Folderize system.
 - b. This could be used to reorganize files into folders based on their timestamps, for instance.
 - c. The process existed **long** before the term, 'folderize'. I was doing this when folders were paper files in a cabinet.

Renaming Options Menu

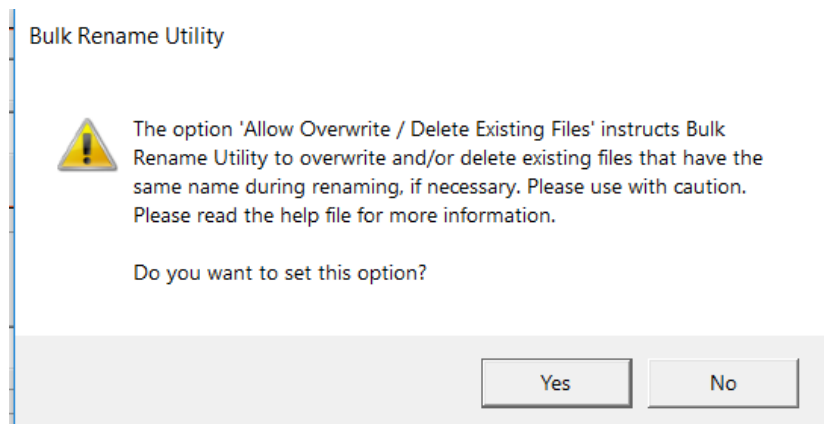
- Allow Overwrite / Delete Existing Files During Renaming If Needed

BRU will generate an error message if the renaming operation will result in a file with the same name, or if the 'Prevent Duplicates' option is set, will append a numeric to make the renamed filename(s) unique. If, however, the 'Allow Overwrite ...' option is set, those other options will be superseded and the resulting files will be overwritten. This also applies to '[Section #13: Copy/Move to Location](#)'.

Example:

Replace (3)		Name ▲	New Name
Replace	2	text 1.jpg	text 1.jpg
With	3	text 2.jpg	text 3.jpg
		text 3.jpg	text 3.jpg
		text 4.jpg	text 4.jpg

Click on Rename...






Name ▲	New Name	Type	Size	Created	Modified	Length	Status
text 1.jpg	text 1.jpg	JPG File	516 KB	9/1/2016 12:04:19 AM	9/1/2016 12:04:19 AM	10	
text 3.jpg	text 3.jpg	JPG File	453 KB	8/31/2016 7:04:20 PM	8/31/2016 7:04:20 PM	10	OK
text 3.jpg	text 3.jpg	JPG File	411 KB	8/31/2016 7:04:20 PM	8/31/2016 7:04:20 PM	10	
text 4.jpg	text 4.jpg	JPG File	426 KB	8/31/2016 7:04:20 PM	8/31/2016 7:04:20 PM	10	

Result – text 2.jpg has been renamed to text 3.jpg overwriting the original text 3.jpg file.

Name ▲	New Name
text 1.jpg	text 1.jpg
text 3.jpg	text 3.jpg
text 4.jpg	text 4.jpg

Renaming Options Menu

The Undo process **will** rename the changed file text 3.jpg back to text 2.jpg, **BUT** the original text 3.jpg has been overwritten and is permanently lost.

Name ▲	New Name
 text 1.jpg	text 1.jpg
 text 2.jpg	text 2.jpg
 text 4.jpg	text 4.jpg

Notes:

1. This option supplants the ‘Prevent Duplicates’ option.

ID3 / EXIF Data / Files Properties

ID3 / Exif Data / File Properties	▶	Extract ID3 Data (MP3)
File/Folder Extensions	▶	Extract Exif Data (Photos)
Log Renaming Activity to File	Ctrl+L	Extract Windows File Properties

These options must be set before BRU is allowed to use Metadata in the renaming process. This will extract the Metadata from the files as they are scanned. Functions that require this include ‘[Section #7: Add](#)’, ‘[Section #8: Auto Date](#)’ and [JavaScript](#).

- Extract ID3 Data (MP3)

This must be enabled before any criteria can be set using ID3 MP3 tags extracted from MP3 files. This includes ‘[Section #7: Add](#)’, which can use specific ‘[Substitute Tags](#)’ as appended data:

%r - Artist
 %t - Title
 %k - Track Number

Notes:

1. This option may slow down the processing *a lot*. If you don't need these fields then leave this option disabled to speed up the processing.
2. Only MP3 files with ID3 tags v1.0 and v1.1 are supported. v2.x ID3 tags are not supported at this time.
3. Refer to, ‘Using Substitution Tags’, under ‘[Section #7: Add](#)’ for more information.

Renaming Options Menu

- Extract EXIF Data (Photos)

This must be enabled before any criteria can be set using EXIF data extracted from supported image filetypes. This includes **'Section #7: Add'**, which can use specific **'Substitute Tags'** as appended data:

%a – Aperture
%c - Comments
%e – Exposure
%f - Focal Length
%xb - Exposure Bias

...and **'Section #8: Auto Date'** can extract 'Date Taken (Original)'.

Notes:

1. This option may slow down the processing *a lot*. If you don't need these fields then leave this option disabled to speed up the processing.
2. Supported image formats include only the following: .JPG, .JPEG, .TIF, .TIFF, .CRW, .CR2, .NEF.
3. May require a 'Files Refresh' from the Actions Menu before the Date Taken (Original)' column data can be viewed in the Content Pane. This option must be enabled in order to extract any information that appears in this column.
4. Refer to **'Section #7: Add'** and **'Section #8: Auto Date'** for more information.
5. This setting must be enabled before the extended EXIF tags can be utilized. Refer to 'Using EXIF Tags' under **'Section #7: Add'**.

- Extract Windows File Properties

This option must be set enabled before using Windows File Properties data extracted from files. This includes **'Section #7: Add'**.

Every file has Metadata values that are assigned through Windows. These are called 'Windows Properties', not to be confused with other Metadata e.g. ID3 and EXIF. BRU supports using Windows Property values in **'Section #7: Add'** in 'Prefix', 'Suffix' and 'Insert' data entry fields (also supported under **'Section #14: JavaScript'**).

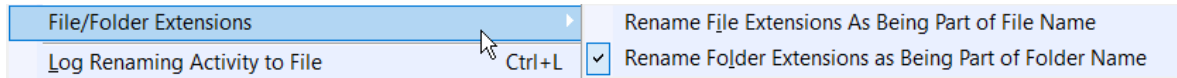
To see the available properties for a file, right click and select, 'Show List of File Properties' from the context menu.

Notes:

1. This option may slow down the processing *a lot*. If you don't need these fields then leave this option disabled to speed up the processing.
2. For more information, refer to 'Using Windows File Properties' under **'Section #7: Add'**.

Renaming Options Menu

File / Folder Extensions

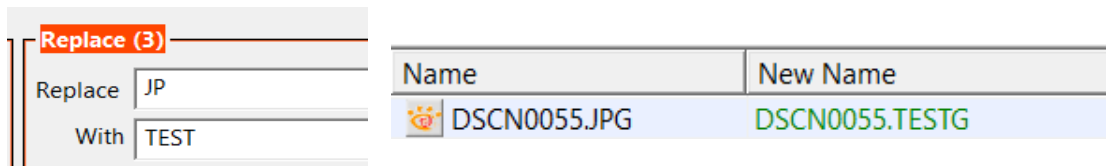


The extension of a file name is identified as a period or 'dot' followed by the ASCII characters that make up the extension name. In a file, this is used to identify the file type. If a dot notation appears as part of a folder name, it has no meaning to Windows and can be ignored.

- Rename File Extensions as being Part of File Name

This option will ignore the file name extension and treat the entire file name as a single string during renaming operations. Use this option with great care because the file name extension is used to identify the file type. If the file extension is renamed, Windows may no longer recognize how to open the file.

Example:

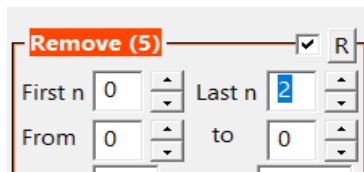


- Rename Folder Extensions as being Part of Folder Name

With this option selected, any dot notation "extensions" within folder names will be ignored - this is because folder extensions have no real meaning, unlike file name extensions which identify the file type. Often, a directory name may contain dot characters the program might otherwise interpret as an extension like a file extension.

Example:

Removing the last 2 characters -



Enabled (folder name all one string):

Name	New Name
Holidays.2004	Holidays.20

Last two characters of '04' are removed.

Not Enabled (folder name w/extension):

Name	New Name
Holidays.2004	Holida.2004

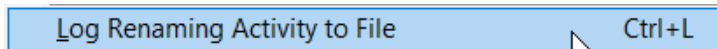
Last two characters of 'ys' are removed.

Notes:

1. This option is enabled by default.
2. Any criteria can be used to rename a folder without requiring '[Section #11: Extension](#)' or resorting to RegEx.

Renaming Options Menu

Log Renaming Activity to File (Ctrl + L)



This will direct BRU to log all of the files renamed including ‘Undo’ requests in a text file, ‘Bulk Rename Utility.log’ located at `..\Documents\Bulk Rename Utility\`.

Example of a log (lines edited for clarification):

```
1/6/2020 7:32:36 PM - Renamed "H:\Music\Victoria Summer\Truth Hurts (From Ratpocalypse).mp3"
to "h:\0\Truth Hurts (From Catpocalypse)_2.mp3"
```

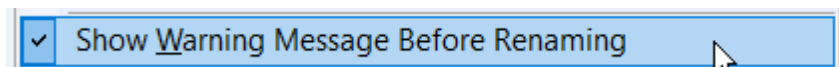
```
1/6/2020 7:33:06 PM - UNDO! Renamed "h:\0\Truth Hurts (From Catpocalypse)_2.mp3" back to
"H:\Music\Victoria Summer\Truth Hurts (From Ratpocalypse).mp3"
```

Notes:

1. Only gives an account of what files were renamed and does not account for how. The ‘Debug New Name’ is the only facility that provides this, however, limited, because it only displays the stages of the process and not the details. The other problem with the “Debug New Name’ is that the Debug Log file is not saved when you exit the dialog box.
2. Does not include physical changes to files, i.e. Changing file attributes or Timestamps are not logged entries.
3. Newer versions of BRU will also support Unicode names.

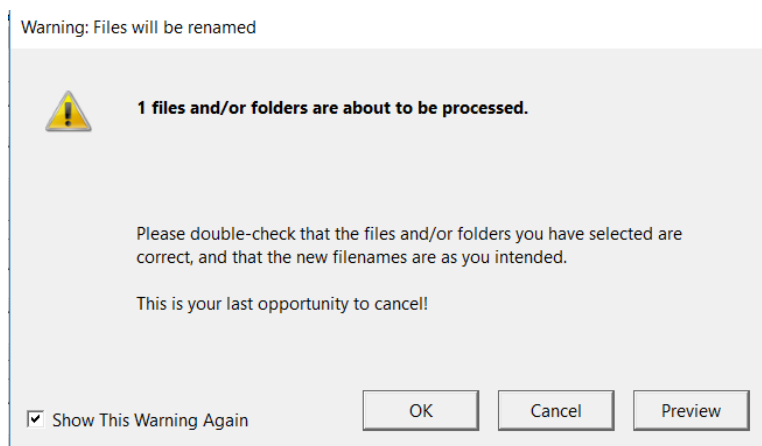
Renaming Options Menu

Show Warning Message Before Renaming



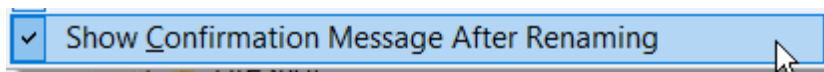
Look at the message carefully. This is your last chance to change your mind. It informs you of how many files and or folders are about to be included in the rename operation.

e.g.,

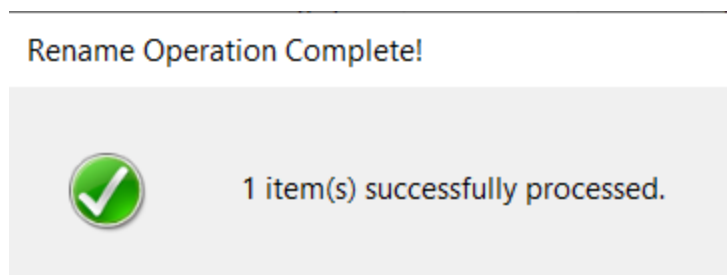


The default is enabled.

Show Confirmation Message After Renaming

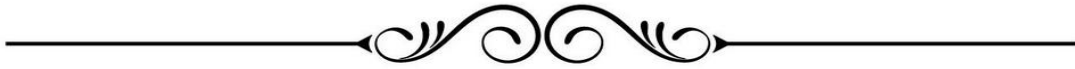


e.g.,



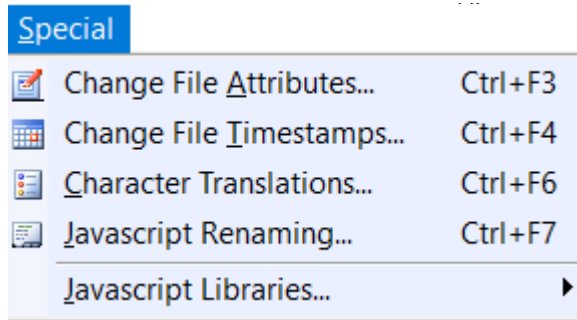
The default is enabled.

1. Is a good reminder that the number of files you wanted to include is the number of files that were processed.
2. This will alert you to any problems with files that did not get processed for whatever reason.
3. If a mistake has been made, you now have the opportunity to Undo the changes.



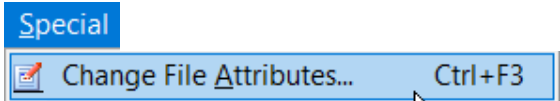
Special Menu:

Also see Section #14: Special



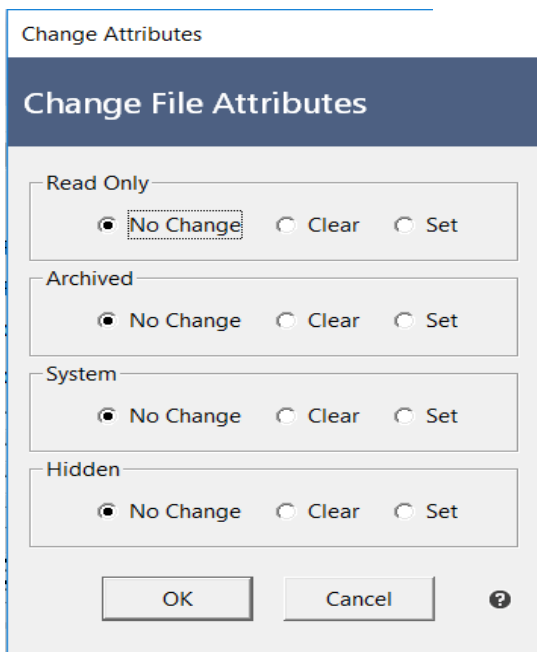
Special Menu

Change File Attributes



Directs Bulk Rename Utility to ‘set’ specified file attributes of the renamed files (does not affect directories). This operation is performed **after** processing completion. Refers to the File Attributes of [Archive](#), [System](#), [Hidden](#), and [Read-Only](#). This ‘bit’ can be set by Windows, a program, or the user.

This is useful if you need to flag all of the renamed files/folders as requiring archiving, or if you want to hide all the renamed files/folders.



Read-only - Allows a file to be read, but cannot be written or modified. The file is ‘write-protected’ but can be deleted.

Archive – Indicates that this file has been modified since a previous backup. Commonly used by Backup software for determining ‘Incremental’ Backup status. Files can be written, modified and deleted.

System – Indicates to Windows that this is a system-critical file that should not be tampered with (modified, renamed, moved, etc.) or deleted. These will normally be hidden unless authorized.

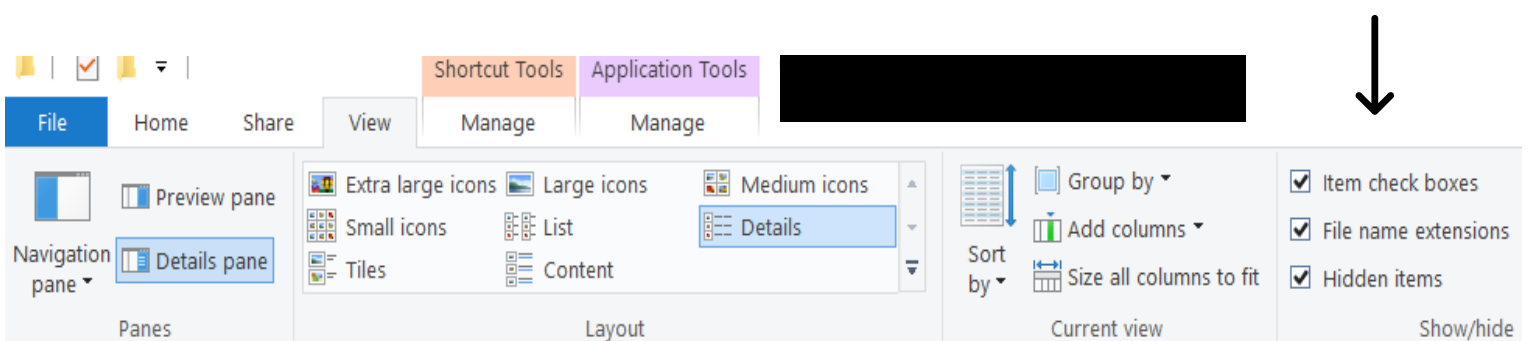
Hidden - File is not shown. It will display in Windows Explorer if you have the system set to show Hidden Files but System Files will not be included. They require a separate option for authorization.

The authorization can be overridden in BRU by enabling ‘Hidden’ in the ‘[Section #12: Filters](#)’. Hidden files will now display in the Content Pane.

Notes:

1. To Display Hidden Files (non-system) in Windows Explorer:

a. In Windows Explorer check ‘Hidden Items’ under the ‘Show/Hide’ section.

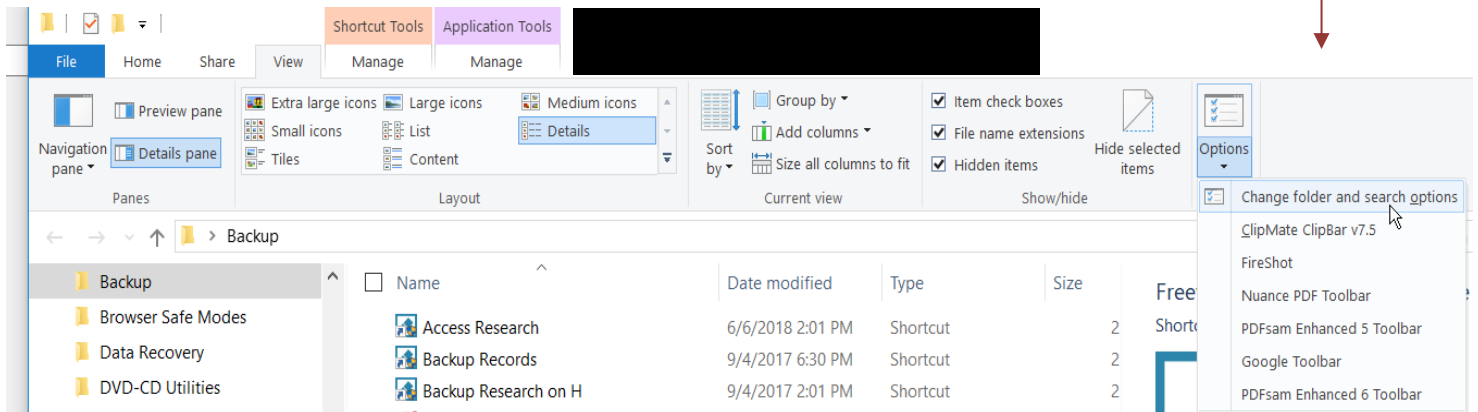


Special Menu

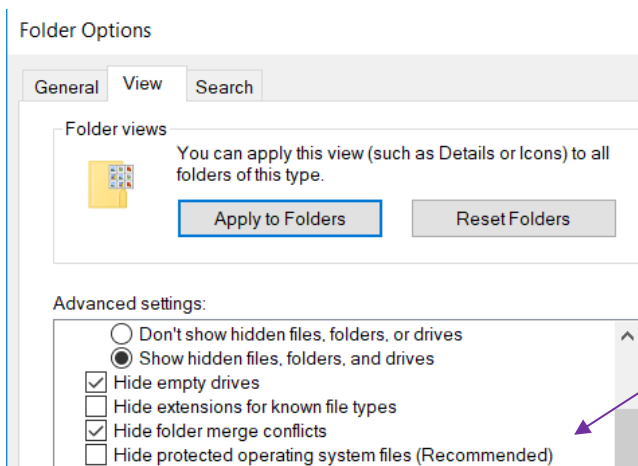
Notes

2. To Display System Files in Windows Explorer:

- a. In Windows Explorer select ‘Change Folder and Search Options’ under the ‘Options’ drop down menu of the ‘Show/Hide’ section of Windows Explorer.



- b. Under the ‘View’ tab of the ‘Folder Options’ dialog box, uncheck ‘Hide protected Operating System Files’.



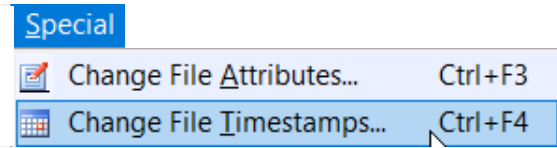
Once a File’s Attribute has been changed via BRU, the Attribute field in the Content Pane will reflect this:

Attributes	Name	New Name ▲
RA	HeLIO WoRID	HeLIO WoRID
	file_01_name.01	file_01_name.01
A	some text 8259420 some more.jpg	some text 8259420 some more.jpg
A	UK-PeterFord-JeremyJordan-LisaWendt.pdf	UK-PeterFord-JeremyJordan-LisaWendt.pdf

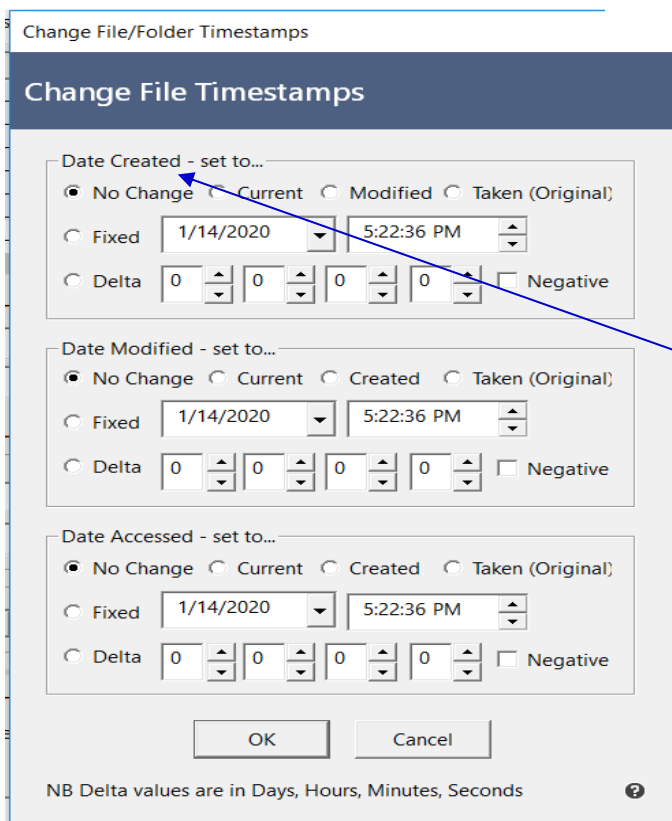
In the example above, the Read Attribute, ‘R’, has been set on the file, HeLIO WoRID.

Special Menu

Change File Timestamps



Change the Windows date timestamp properties, Date Created, Date Modified and Date Accessed of the selected files.



Looks more complicated than it is.

This is useful if you have certain applications which use a file's timestamp in order to identify if a file has been modified, or if you want to sort photographs in a particular sequence.

Some older applications even use the timestamp to identify a product version.

If you would like to set the Date Modified and Date Accessed timestamps to the same value as the Date Created timestamp, then choose the Date Created option.

Once a File's Timestamp has been changed, this will be reflected in the Content Pane under the appropriate date field:

Name ▲	New Name	Created
<input type="checkbox"/> HeLIO WoRID	HeLIO WoRID	10/31/2020 11:19:34 AM
<input type="checkbox"/> file_01_name.01	file_01_name.01	5/3/2020 10:58:39 AM

Notes:

1. Created in the Column Headings refers to the Windows Property, Date Created.
2. Modified in the Column Headings refers to the Windows Property, Date Modified.
3. Accessed in the Column Headings refers to the Windows Property, Date Accessed.

Special Menu

Essentially, Windows assigns a timestamp (Windows Property) to all files and directories. They are:

Date Created

The date the file was created in Windows, in Month Day Year Hours Minutes and seconds. If the file did not originate in Windows, then this property represents when the file was transferred into the Windows OS, but not when the file was created.

Date Modified

The date the file was last physically changed. Date Modified is only affected if the actual contents of the file changes, for example, editing and saving the file. Moving, copying the file has no effect typically.

Date Accessed

The date the file was last accessed.

1. Date Accessed includes a move, open, read or other simple access. By itself it is not very reliable because even system access will modify this date. This is probably the reason that Item Date does not take Date Accessed in consideration when determining its value.

For each timestamp, Created, Modified and Accessed, you have several options:

Set to:

No Change
 Current
 Modified
 Taken (Original)
 Created

No Change –	no modification.
Current –	set to current date and timestamp
Modified –	use the existing Modified Date as the new timestamp
Taken(Original) –	use the EXIF DateTimeOriginal as the new timestamp
Created -	use the existing Date Created as the new timestamp

Fixed

Fixed – use the specified date and time as the new timestamp

Delta

 Negative

Delta – individual timestamps can be adjusted by Days, Hours, Minutes and Seconds

Some further explanation is required:

Special Menu




Taken (Original) (aka Date Taken) –

For Taken (Original) to hold a value, the image must have been created in a device outside of the Windows OS capable of recording Metadata, e.g., a camera.

Taken(Original) maps directly from the EXIF item DateTimeOriginal. DateTimeOriginal represents the immediate timestamp of when the image was created in the device. Thus, Taken (Original) is more reliable and accurate than the Windows Property, Date Created, if available.

For Taken (Original) to extract properly, the EXIF data must exist for DateTimeOriginal. The problem becomes that EXIF DateTimeOriginal only applies to certain image file types. If the filetype is not supported, Taken (Original) will have no value. If DateTimeOriginal holds a **null** value, Taken (Original) will also be rendered **null**.

Below are three files, two of which are digital images with recorded EXIF Metadata that was taken with a camera. One of these Metadata items that was recorded is DateTimeOriginal. Because DateTimeOriginal holds a value, this value can be observed in the column ‘Taken (Original)’. The other file is an image file that has no EXIF data and so the ‘Taken (Original)’ column for that file subsequently **contains no value**.

Name ▲	New Name	Created	Modified	Accessed	Taken (Original)
 Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg	1/21/2012 4:56:04 AM	1/21/2012 1:44:24 AM	1/21/2012 4:56:04 AM	10/18/2009 2:34:59 PM
 Belvedere Plantation 6492.jpg	Belvedere Plantation 6492.jpg	1/21/2012 4:56:04 AM	1/21/2012 1:44:24 AM	1/21/2012 4:56:04 AM	10/18/2009 2:34:30 PM
 Becca BW.music	Becca BW.music	1/20/2012 6:55:52 PM	1/20/2012 3:34:34 PM	1/20/2012 6:55:52 PM	

Notes:

- BRU can change the Windows timestamps but cannot change the EXIF data but there are tools out there that can.
- For more information refer to ‘[Section #7: Add](#)’, ‘[Section #8: Auto Date](#)’ and ‘ID3 /EXIF Data / File Properties’ under the Renaming Options Menu.
- Additional EXIF data may be recorded, some of which can be accessed by BRU for the purpose of appending (only). These are: Taken (Digitized), Taken (Modified) and Taken (Recent). Refer to ‘[Section #8: Auto Date](#)’.
- If you direct the program to use the Taken(Original) date, and there is no data present, it will change nothing.
- ‘Extract EXIF Data (Photos)’ option from the ID3/ EXIF Data/ File Properties submenu of the Renaming Options Menu must first be enabled in order for the data to be extracted.
- Windows timestamps, EXIF data, ID3 data, are also referred to by their generic name, Metadata.

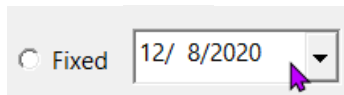
Special Menu

Fixed –

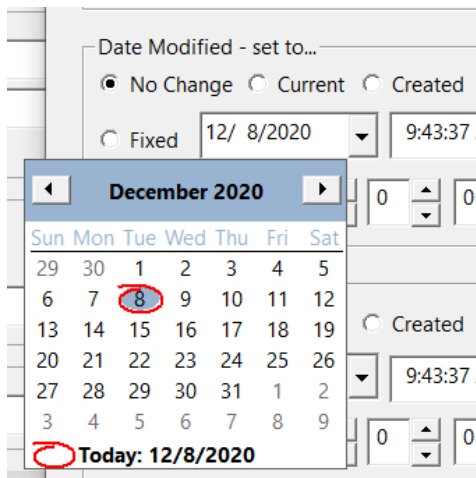
To use the Fixed option:

For the date –

You can choose to enter the date directly by clicking inside the data entry field:



Or by clicking on the Drop down arrow, you can select using a calendar.



For the time –

You can choose to enter the time directly by clicking inside the data entry field:



Or you can adjust it by clicking on the field and using the up and down arrows to increment and decrement.

Special Menu


Understanding Delta

Delta – The duration expressing the difference between two date-time instances. Using delta in BRU, current individual timestamps can be adjusted by Days, Hours, Minutes and Seconds using a positive value for increasing, and a negative value for decreasing. Also refer to ‘[Section #8: Auto Date](#)’ which discusses Offset, same as Delta..

e.g., if you were traveling and you took some pictures but forgot to account for time zones, you could rename the files with the corrected timestamps using a delta adjustment. This will NOT update the ‘Date Taken’ Metadata.

Example,

Here is a file with a Date Created of April 30, 2014 timestamped 11:36:55 PM

Name ▲	New Name	Created
 Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg	4/30/2014 11:36:55 PM

To increment this timestamp by 1 day 8 minutes-: enter, ‘1 0 8 0’ in the Delta data field;

To decrement the original timestamp by 1 day 8 minutes, just change the positive value to a negative:

Change File Timestamps

Date Created - set to...

No Change
 Current
 Modified
 Taken (Original)

Fixed

Delta

 Negative

Change File Timestamps

Date Created - set to...

No Change
 Current
 Modified
 Taken (Original)

Fixed


Delta

 Negative


Where:

1 = Day 0 = hours 8 = minutes 0 = Seconds

Here is the new Date Created of May 1, 2014 timestamped 11:44:55 pm exactly 24 hours 8 minutes ahead

Name ▲	New Name	Created
 Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg	5/1/2014 11:44:55 PM

Here is the new Date Created of April 29, 2014 timestamped 11:28:55 PM exactly 24 hours 8 minutes behind

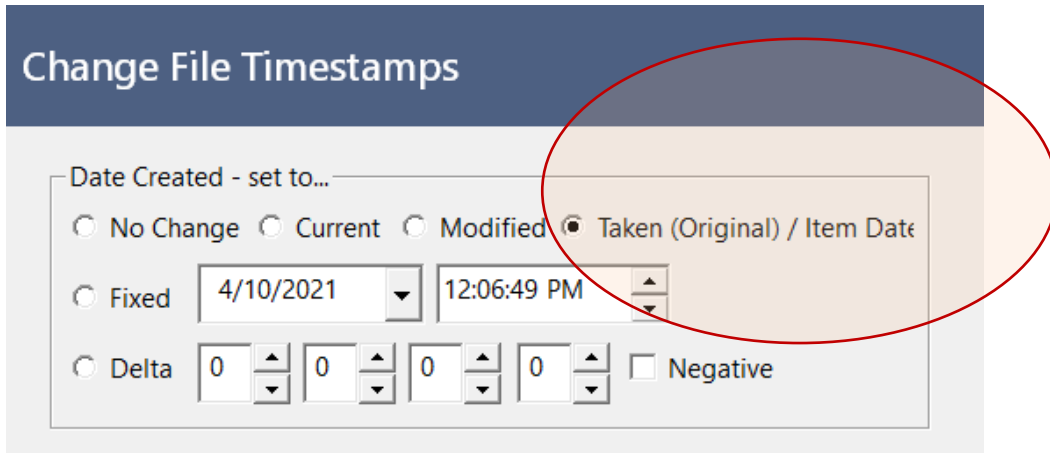
Name ▲	New Name	Created
 Belvedere Plantation 6490.jpg	Belvedere Plantation 6490.jpg	4/29/2014 11:28:55 PM

Special Menu

v3.42 New Additions

Change File Timestamps function Allows Item Date

Enhanced the Change File Timestamps function to use the file's 'Item Date' if the EXIF 'Taken (Original)' timestamp is not available, for example, for some unsupported video or image formats.



Example.

Here are two files. One with a Taken (Original) timestamp and one without. The filetype is not important. Item Date is available for most if not all filetypes. This is indeed a welcome addition.

Name	New Name	Taken (Original) ▲	Item Date	Created
Pat Child Picture06202017.tif	Pat Child Picture06202017.tif		6/20/2017 7:21 PM	7/15/2017 9:02:25 AM
SAM_1141.JPG	SAM_1141.JPG	2/7/2016 2:12:06 PM	2/7/2016 2:12 PM	1/13/2017 7:34:27 AM

After renaming:

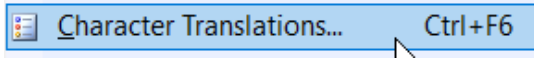
New Name	Taken (Original) ▲	Item Date	Created
Pat Child Picture06202017.tif		6/20/2017 7:21 PM	6/20/2017 7:21:00 PM
SAM_1141.JPG	2/7/2016 2:12:06 PM	2/7/2016 2:12 PM	2/7/2016 2:12:06 PM

You can see that the first file that did not have a Taken (Original) timestamp will have applied the Item Date as its new Created Date. The second file, on the other hand, does have the timestamp and will change its Created Date to the file's Taken (Original Date).

For further information, please refer to Item Date.

Special Menu

Character Translations



Defines a set of rules that will translate one specific character or sequence of characters into a different character or sequence of characters. These rules are set forth in the Character Translation Table.

There are three ways to enter the replacement data:

1. As a character e.g. A
2. As a hex value e.g. 0F
3. As a decimal value e.g. 065

Syntax:

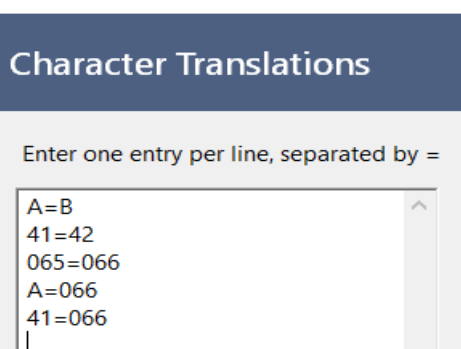
<character or string to be translated FROM> = <character or string to be translated TO>

Notes:

1. To differentiate between the values, decimal is entered as three characters, hex is entered as two characters and direct characters (literals) are entered as single characters.
2. If translating the equal sign character, it must be entered as either hex or decimal since it is **part of the syntax**.

In the following, every example is converting a capital "A" to a capital "B" -

Character Translations



where:

- | | |
|---------|---|
| A=B | direct (literal) expression of the characters to convert, A, B |
| 41=42 | two characters long, therefore hex value equivalents of A, B |
| 065=066 | three characters long, therefore decimal value equivalents of A, B |
| A=066 | a direct (literal) character, A and a decimal value equivalent of B |
| 41=066 | a hex value equivalent of A and a decimal value equivalent of B |

3. Strings are translated by using comma delimiters to separate individual values (characters) that make up the string.
e.g.,

41,066,C=D,E,070 converts ABC to DEF

4. Bulk Rename Utility identifies the type of value entered by its length.
 - a. If the value on either side of the equal sign is one character long (e.g. "A" or "1") then it's a literal character.
 - b. If the value is two characters long (e.g. "6F") it is automatically interpreted as a hex value.
 - c. If the value is three characters long, (e.g. '017') it is interpreted as a decimal value (see "Decimal" column).
 1. This allows you to perform replacements of non-keyboard characters. Refer to the ASCII Charts.

Special Menu

Notes: cont.

5. The Translation Table syntax is using a format similar to, From ... To

where e.g.,

From ... To
A = B


The **From... To..** in the Translation table is using the equal sign, e.g. **A=B**. If you wish to translate the equal sign as part of the Translation Table, then use the corresponding hex or decimal equivalent for the equal sign. This won't work: **= = A**.

The same principle is applied to translating the comma sign that is otherwise used as part of the syntax to indicate the conversion of multiple characters.

Here are some examples of translations:

1) convert SAM to DEF

S,A,M=D,E,F

 SAM_0030 \$.jpg DEF_0030 \$.jpg

2) convert ABC to DEF

41,066,C=D,E,070

 ABC this is a test.jpg DEF this is a test.jpg


This will not work using the ABC example:

 BeAcCa BW.music BeAcCa BW.music

.. because the value is 'ABC', and the letters must be in the filename in that order consecutively for translation to take place. If you wanted A, B, or C anywhere in the file to be translated, you would instead use this:

Table: Result:


41=D
066=E
C=F

 BeAcCa BW.music EeDcFa EW.music

Here's another one:


3) translate the \$ sign to the word DOLLAR

\$=D,O,L,L,A,R

 SAM_0030 \$.jpg SAM_0030 DOLLAR.jpg

4) .. or vice versa

D,O,L,L,A,R=\$

 SAM_0030 DOLLAR.jpg SAM_0030 \$.jpg

Special Menu

These are some additional examples by Forum member Stefan from the BRU Forum website:

4) Replace signs with a space:

(= <space>

[= <space>

!= <space>

5) Replace signs with an underscore:

+=_

;=_

6) Replace signs with nothing (**null**, meaning no entry, or "", is equivalent to removing the character):

Û=

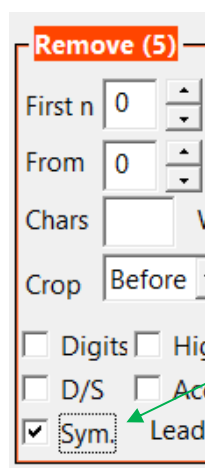
ü=

\$=

This is also useful when you have file names filled with junk symbols:

Test File 1 - !2# 3\$ 4(5) 6+ 7, 8- 9. 0; 1= 2@ 3] 4_ 5{ 6} 7~.txt

To remove these symbols you could also use 'Section #5: Remove' and enable 'Sym' option:



Special Menu

However, to have more control over which symbols gets removed and which do not, use Character Translation:

```
!=
#=#
$=#
(=#
)=#
+=#
-=#
.=#
;=#
@=#
]=#
_=#
{=#
}=#
~=#
```

When applied to the Test File, this results in a cleaned up filename:

```
Test File 1 - 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7.txt
```

7) Translate German umlautes:

```
Ü=U,e
ü=u,e
Ö=O,e
ö=o,e
Ä=A,e
ä=a,e
ß=s,s
```

8) Shorten file names:

```
f,e,a,t,u,r,i,n,g=f,e,a,t
B,u,l,k,R,e,n,a,m,e,U,t,i,l,i,t,y=B,R,U
R,e,m,i,x=R,M,X
R,e,m,a,s,t,e,r,e,d=R,E,M
I,m,p,o,r,t,a,n,t=!
V,e,r,y, ,I,m,p,o,r,t,a,n,t=!,!,!
2,0,1,3=1,3
```

Special Menu

9) Replace a group of signs or one sign with an another sign or group:

A=F,i,r,s,t
 J,a,n=0,1
 0,0,5=V
 0,0,6=V,I

10) Roman Numerals Translation List

0,0,1=I
 0,0,2=I,I
 0,0,3=I,I,I
 0,0,4=I,V
 0,0,5=V
 0,0,6=V,I
 0,0,7=V,I,I
 0,0,8=V,I,I,I
 0,0,9=I,X
 0,1,0=X
 0,9,8=X,C,V,I,I,I

11) Use Character Translations for URL encoded files decoding

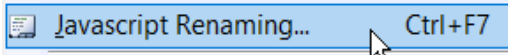
%,2,0=
 %,2,1=!
 %,2,3=#
 %,2,4=\$
 %,2,5=%
 %,2,6=&
 %,2,7='
 %,2,8=(
 %,2,9=)
 %,2,B=+
 %,2,C=,
 %,5,B=[
 %,5,D=]

Important –

BRU does save the Character Translation settings as part of a Favourite File, but they would be loaded in every time, which may be undesirable. The better solution is to copy and paste them to a text file, and when you need them, just paste them back. Make sure that no entries have a trailing space when pasting. Using 'Ctrl + Enter' at the end of each line, rather than simply pressing 'Enter', is the best method to ensure this. Ctrl + Enter in BRU inserts a Line Break.

Special Menu

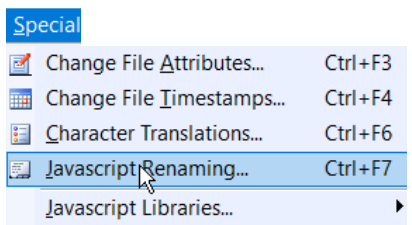
JavaScript Renaming (Ctrl + F7)



This is a powerful addition to the already splendid capabilities that BRU offers. The company decided to make this functionality available only to the commercial version and requires an inexpensive commercial license to unlock. It is well worth it even for home users. Think of it as registering the program.

JavaScript renaming gives you total flexibility and full control of your file renaming needs by using JavaScript code. Bulk Rename Utility uses the v8 JavaScript, Google's high performance JavaScript engine, that is also used in Google Chrome. v8 implements ECMAScript as specified in ECMA-262, 5th edition. Most standard JavaScript syntax and functions are supported in the BRU implementation.

The program provides two built-in JavaScript libraries, Sugar.js library and Date.js library. Additional libraries may be added using the functions, 'Include' and 'Require' (see Appendix – JavaScript Bulk Rename Utility Functions).



Brings up the JavaScript Code Entry Form

The Code Entry Form is where JavaScript code can be entered **and** tested.

Example:

This scripts adds a counter, padded with up to 5 zeros, after each selected file name:

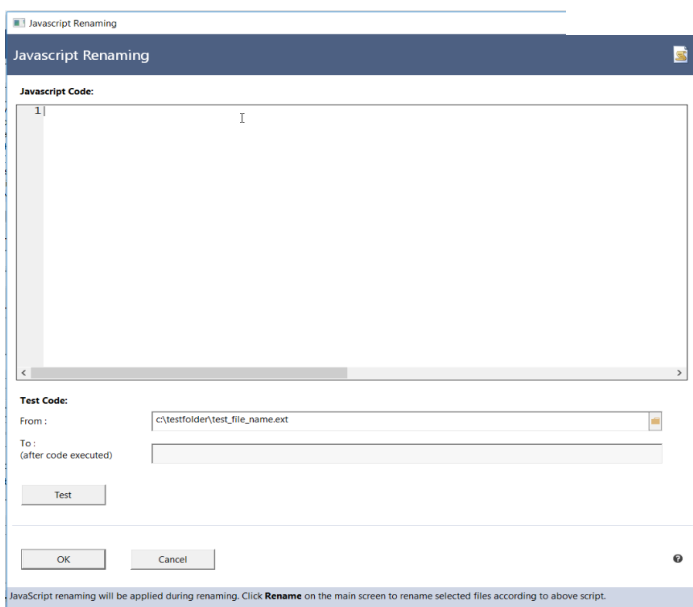
JavaScript Code:

```
1 function padleft(nr, n, str){
2   return Array(n-nr.toString().length+1).join(str||'0')+nr;
3 }
4
5 newName = name + '_' + padleft(counter, '5', '0') ;
```

This the text in case you want to enter it yourself:

```
function padleft(nr, n, str){
  return Array(n-nr.toString().length+1).join(str||'0')+nr;
}

newName = name + '_' + padleft(counter, '5', '0') ;
```



Quick Analysis:

1. The function padLeft is defined.
2. newName represents the new name of the file.
3. name represents the current name of the file.
4. counter is a constant that starts from 1 and it's incremented for each selected file.

Special Menu

The selected files showing the results of the JavaScript:

Name	New Name
DSCN0001.JPG	DSCN0001_00001.JPG
DSCN0029.JPG	DSCN0029_00002.JPG
DSCN0032.JPG	DSCN0032_00003.JPG
DSCN0055.JPG	DSCN0055_00004.JPG
DSCN0056.JPG	DSCN0056_00005.JPG
DSCN0057.JPG	DSCN0057_00006.JPG
DSCN0058.JPG	DSCN0058.JPG
DSCN0059.JPG	DSCN0059.JPG
DSCN0064.JPG	DSCN0064.JPG

Using the Test facility

After entering in your code, you can optionally test it using the 'Test' button. This will verify the syntax is correct.

Example of bad code entry:

Code entered as:

Javascript Renaming

Javascript Code:

```

1 function padleft(nr, n, str){
2     return Array(n-string(nr).length+1).join(str||'0')+nr;
3 }
4
5 newName = name + '_' + padleft(counter, '5', '0') ;
6
7

```

Test Code:

From :

To :

(after code executed)

Press the 'Test' button

Generates this error:

It does not debug the code – there are no breakpoints or anything similar – it just determines if it will run or not. It is also colour coded which indicates possible problems with syntax.

Bulk Rename Utility

Javascript Error:

ReferenceError: string is not defined in : return Array(n-string(nr).length+1).join(str||'0')+nr; [Line:2 Start:16 End:17]

Special Menu

Handling Syntax Errors

The following is taken from Bulk Rename Utility Manual Volume II. At the risk of repeating information found there, I feel it is important to include in order to have a better understanding of the Code Entry Form.

Understanding the JavaScript Code Entry Form Colour Coding

The colour coding system used is to help with syntax and errors (colour displayed is as close as I can get).

Within function

Item	Colour	Correct	Bad	Neutral
Keywords	blue	*	Black & White	
strings	Brown-red	*	“	
Variables	Black & White			*
Numerals	Black & White			*
Function name	Black & White			*

Outside Function

Item	Colour	Correct	Bad	Neutral
Variables	purple	*	Black & White	

All

Item	Colour	Correct	Bad	Neutral
Operators	silver-green	*	Black & White	
parenthese/brackets	silver-green	*	“	
Braces, Semicolon				

Avoid Common Errors: <spaces>

JavaScript, unlike RegEx, is very flexible when it encounters extra spacing in the code.

For example, you could have:

```
function myFunction() {
    var y = 5;
    var x = y + 2;
    return x
}
newName = myFunction();
```

Changed to:

```
function myFunction( ) {
    var y = 5 ;
    var x = y + 2
    return x
}
newName = myFunction();
```

Result: 7

.. and it will still produce the result: 7

Special Menu

Avoid Common Errors: CASE SENSITIVITY

Occasionally, you will encounter errors while entering the script into the JavaScript Code Entry form:

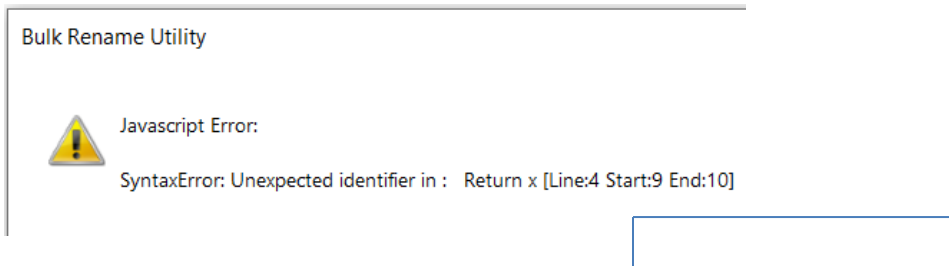
Javascript Code:

```

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 a keyword has been capitalized 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 colour depending on whether it is a keyword, string, or operator. To correct, all that is required is to change it to a lowercase, 'return'.

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

Test Code:

From :

To :

(after code executed)

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: **Belvedere Plantation 6493.jpg**

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

To: **Belvedere Plantation 6493_00001.jpg**

Notes:

1. This allows you to see an execution of the JavaScript virtually. Acts as a preview before committing to Rename.
2. The Syntax Testing and Test Code Data Entry **do not** require the Commercial license.
3. Execution of the actual renaming process does require the commercial license.
4. The result in photo, Belvedere Plantation_00001.jpg, is taken from 'padleft' script not the 'myFunction' script.

Special Menu

Avoid Common Errors: ASCII Mistakes

If you copy and paste code – be it RegEx or JavaScript from either volume, you may have to do a little fine editing if you discover the code is not working. Much of these errors has to do with inadvertent ASCII conversions.

1. Some text editors and word processing programs may convert <hyphens> into ‘dashes’. Microsoft Word converts a <hyphen> character into a dash character when you press ‘Enter; after a <hyphen>.
2. Microsoft Office products use High ASCII 145 and ASCII 146 for the right and left single quotes, instead of the ASCII 39 used for either side, typically found in plain text editors.
3. Microsoft Office products use high ASCII 147 and ASCII 148 for the right and left double quotes. Why? Because they look better than the ASCII 34 used for either side, typically found in plain text editors. Unfortunately, however, the JavaScript Code Entry Form requires ASCII 34.

If the script contains one of these characters, it will generate this error:

A clue to this problem can be found in the JavaScript Code Entry Form...

Bulk Rename Utility



Javascript Error:

SyntaxError: Unexpected token ILLEGAL in : newName = "is " + (b);
// = [Line:3 Start:10 End:11]

You have:

```
3 newName = "is "
```

instead of:

```
3 newName = "is "
```

Notice the colour difference? Black and white characters that should be reddish brown for the string 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. This is the cause of the error.

4. For more information, please refer to BRU JavaScript Reference Manual in the appendix of Volume II.

Special Menu

About Conditional Renaming

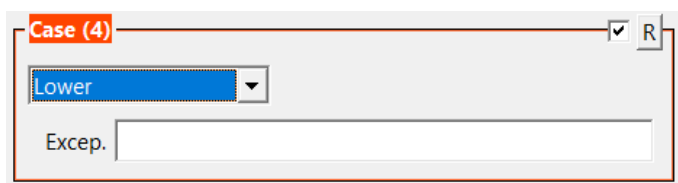
A Conditional in JavaScript is essentially an **If.. Then..** statement. **'If'** something is **true**, **'Then'** do this. Setting the variable, `newName`, back to the `origName` is like canceling the renaming of an object, as the new name is set back to the object's original name. This feature allows 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")`. The same would be valid if applied to `newExt` and `newLocation`.

Another Example –

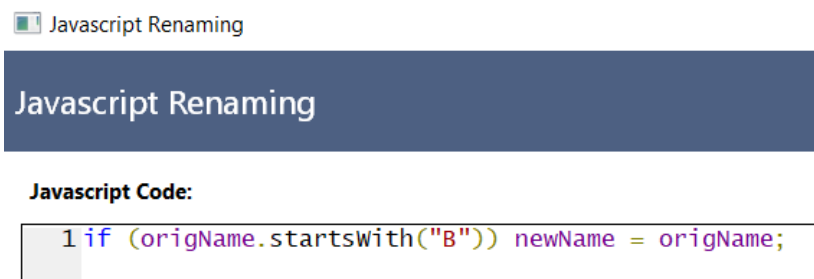
This JavaScript code conditionally renames files that do not start with the letter 'B'. If they do, then `newName` is reset back to `origName`, thereby canceling out the renaming operation on those filenames that matched.

1. 'Section #4: Case'



Case = Lower

2. 'Section #14: Special' : JavaScript



Code:
 if (origName.startWith("B")) newName =
 origName;

Analysis:

Variables:

`origName` The equivalent of Name in the Content Pane. This is the filename before the renaming operation.

`newName` The equivalent of New Name in the Content Pane. This is the filename in preview of the renaming operation. New Name represents the result after the renaming operation has been completed.

Simple one line script that tests 'If' `origName` begins with an uppercase letter, 'B'. If **true**, 'Then', for those filenames, the renaming operation would be canceled through the assignment of `origName` to `newName`.


Special Menu

In the Order of Evaluation, ‘**Section # 4: Case**’ comes before JavaScript, ‘**Section #14: Special**’. JavaScript will always be the last criteria applied before the actual Renaming is performed.

Example using Belvedere Plantation 6492.jpg with Debug log.

Belvedere Plantation 6492.jpg Belvedere Plantation 6492.jpg

Filename Generation Debug Results

 Original name: Belvedere Plantation 6492.jpg
Components: Name=Belvedere Plantation 6492, Extension=.jpg

Finished Group 1. Name=Belvedere Plantation 6492, Extension=.jpg
Finished Group 2. Name=Belvedere Plantation 6492, Extension=.jpg
Finished Group 3. Name=Belvedere Plantation 6492, Extension=.jpg
Finished Group 4. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 5. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 6. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 7. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 8. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 9. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 10. Name=belvedere plantation 6492, Extension=.jpg
Finished Group 11. Name=belvedere plantation 6492, Extension=.jpg
Finished Processing JavaScript. Name=Belvedere Plantation 6492, Extension=.jpg

Final Name: Belvedere Plantation 6492.jpg ←

If you look at Group 4, this represents ‘**Section #4: Case**’. In Group 4 the Name changes –

From:
Belvedere Plantation 6492.jpg
To:
belvedere plantation 6492.jpg


The final group, JavaScript, tests that the filename starts with an uppercase ‘B’ = **true**, rendering the renaming operation void. The name changes back –

From:
belvedere plantation 6492.jpg
To:
Belvedere Plantation 6492.jpg

Example using Tim Master.accdb with Debug log.

Tim Master.accdb tim master.accdb

Filename Generation Debug Results

 Original name: Tim Master.accdb
Components: Name=Tim Master, Extension=.accdb

Finished Group 1. Name=Tim Master, Extension=.accdb
Finished Group 2. Name=Tim Master, Extension=.accdb
Finished Group 3. Name=Tim Master, Extension=.accdb
Finished Group 4. Name=tim master, Extension=.accdb
Finished Group 5. Name=tim master, Extension=.accdb
Finished Group 6. Name=tim master, Extension=.accdb
Finished Group 7. Name=tim master, Extension=.accdb
Finished Group 8. Name=tim master, Extension=.accdb
Finished Group 9. Name=tim master, Extension=.accdb
Finished Group 10. Name=tim master, Extension=.accdb
Finished Group 11. Name=tim master, Extension=.accdb
Finished Processing JavaScript. Name=tim master, Extension=.accdb

Final Name: tim master.accdb ←

If you look at Group 4, this represents ‘**Section #4:** In Group 4 the Name changes –

From:
Tim Master.accdb
To:
tim master.accdb

The final group, JavaScript, tests that the filename starts with an uppercase ‘B’ = **false**. The renaming operation is performed.

Special Menu

3. Select the files in the Content Pane.

Before JavaScript is applied:

Name	New Name
Menu (food).tmd	menu (food).tmd
Belvedere Plantation 6492.jpg	belvedere plantation 6492.jpg
Tim Master.accdb	tim master.accdb
Belvedere Plantation 6479.jpg	belvedere plantation 6479.jpg
2020-12-14_181229.png	2020-12-14_181229.png
wildcrypt.exe	wildcrypt.exe
Belvedere Plantation 6493.jpg	belvedere plantation 6493.jpg
Menu (food).rtf	menu (food).rtf
Belvedere Plantation 6497.jpg	belvedere plantation 6497.jpg
Menu (food).txt	menu (food).txt
Portishead-Roads.mp3	portishead-roads.mp3
Portishead-Roads a.mp3	Portishead-Roads a.mp3
setup(free-chm-to-pdf)(1).exe	setup(free-chm-to-pdf)(1).exe
Menu (food).docx	Menu (food).docx
test BRU 3.rar	test BRU 3.rar

Filenames that are already in lowercase are ignored.

e.g., wildcrypt.exe

After JavaScript is applied:

Name	New Name
Menu (food).tmd	menu (food).tmd
Belvedere Plantation 6492.jpg	Belvedere Plantation 6492.jpg
Tim Master.accdb	tim master.accdb
Belvedere Plantation 6479.jpg	Belvedere Plantation 6479.jpg
2020-12-14_181229.png	2020-12-14_181229.png
wildcrypt.exe	wildcrypt.exe
Belvedere Plantation 6493.jpg	Belvedere Plantation 6493.jpg
Menu (food).rtf	menu (food).rtf
Belvedere Plantation 6497.jpg	Belvedere Plantation 6497.jpg
Menu (food).txt	menu (food).txt
Portishead-Roads.mp3	portishead-roads.mp3
Portishead-Roads a.mp3	Portishead-Roads a.mp3
setup(free-chm-to-pdf)(1).exe	setup(free-chm-to-pdf)(1).exe
Menu (food).docx	Menu (food).docx
test BRU 3.rar	test BRU 3.rar

All filenames beginning with an uppercase letter, 'B' look to be ignored, but are in fact, excluded from the renaming operation based on the JavaScript code.

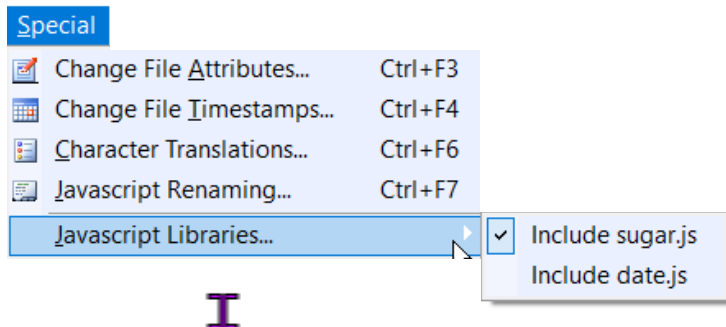
Special Menu

Notes:

1. Conditional Renaming under JavaScript can be as complex as needed, excluding and conditionally renaming a very complex set of files and conditions, including dates, windows properties, etc.
2. The simple example provided can just as easily have been accomplished using '[Section #12: Filters](#)'.

Special Menu

JavaScript (Extension) Libraries



Extension JavaScript libraries and files can be included using the functions [include](#) and [require](#). There are two JavaScript libraries that are already included in Bulk Rename Utility. They are saved in the js folder under the installation directory.

File names are [sugar.js](#) and [date.js](#).

To make them available to BRU, select libraries to include from 'JavaScript Libraries' option of the 'Special Menu',

Activating the option 'Include sugar.js' is equivalent to adding:

```
require('js/sugar.js')
```

.. at the start of your JavaScript code.

Activating the option 'Include date.js' is equivalent to adding:

```
require('js/date.js')
```

... at the start of your JavaScript code.

Using the menu options is much easier.

Include sugar.js

Sugar.js library

Sugar is a very powerful library. It adds many useful functions to work with dates, text, etc. in JavaScript.

Include date.js

Date.js library

Date.js is a very powerful library. It adds many useful functions to work with dates in JavaScript.

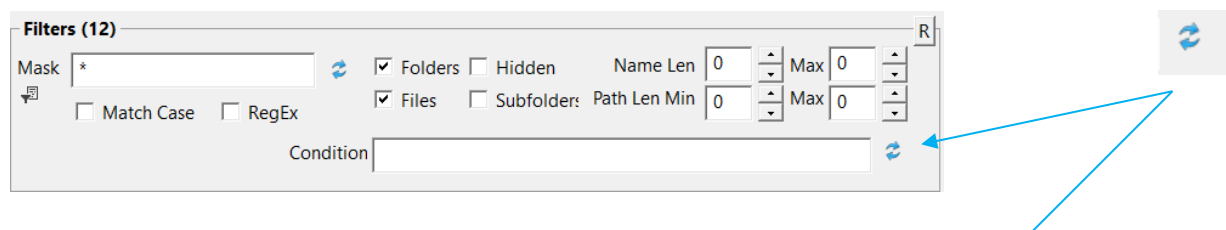
See Notes at the conclusion of this section.

Special Menu

JavaScript Filter Condition

The JavaScript filter ‘Condition’ is a data entry field found in ‘[Section #12: Filters](#)’ that allows you to enter JavaScript coding for the selection/inclusion of files/folders based on an objects' name and also on the date, size, time, EXIF, attributes, etc.

If the condition evaluates to **true** for an object in the File List, that object will be included, otherwise it is not. The JavaScript condition supports all the functions, variables and constants that are available for JavaScript Renaming.



The button next to the Condition data entry field will rescan the directory using the JavaScript condition. It will also display any error messages (similar to the ‘Test’ button on the JavaScript Code Entry form).

Notes:

1. This function requires the Commercial license.
2. Do not confuse this with ‘Conditional Renaming’. The Filter Condition is a feature of the ‘[Section #12: Filters](#)’ criteria. Conditional Renaming is a functionality of JavaScript.

Special Menu

Javascript Filter Condition Examples

Javascript Filter Condition	Requires sugar.js (*)	Requires date.js (**)	Result
<code>name.endsWith(/[q-z]/)</code>	Y	N	Include all objects that end with 'q' to 'z'.
<code>name.startsWith(/[a-d]/, null, false)</code>	Y	N	Include all objects that start with 'a' to 'd', case insensitive.
<code>ext.isBlank()</code>	Y	N	Include all objects that have no extension.
<code>object('modified').getTime() == exif('%d').getTime()</code>	N	N	Include all files that have the Windows modified timestamp matching the EXIF date taken.
<code>object('modified').getTime() != exif('%d').getTime() && exif('%d').getTime() != 0</code>	N	N	Include all files that have the Windows modified timestamp not matching the EXIF date taken and the EXIF date taken is not empty.
<code>object('size') == 0</code>	N	N	Include all objects with zero size. Disable option to include folders in Filters (12) to include only files.
<code>object('size') > 10000</code>	N	N	Include all files larger than 10000 bytes.
<code>object('modified').daysAgo() < 31</code>	Y	N	Include all objects with the Windows modified timestamp more recent than 31 days ago. daysAgo() requires sugar.js, see note below.
<code>object('modified').weeksAgo() < 4</code>	Y	N	Include all objects with the Windows modified timestamp more recent than 4 weeks ago. weeksAgo() requires sugar.js, see note below.
<code>object('modified').isBetween('yesterday', 'tomorrow');</code>	Y	N	Include all objects with the Windows modified timestamp between yesterday and tomorrow. isBetween() requires sugar.js, see note below.
<code>object('modified').isBetween('the beginning of last month', 'today');</code>	Y	N	Include all objects with the Windows modified timestamp between the beginning of last month and today. isBetween() requires sugar.js, see note below.
<code>exif('taken').isBetween('the beginning of last month', 'today');</code>	Y	N	Include all objects with the EXIF taken date between the beginning of last month and today. isBetween() requires sugar.js, see note below.

Special Menu

JavaScript

Notes:

1. JavaScript code is applied to each object and processed as the last step after all other criteria and options have been performed, just before the renaming operation.
2. More information available at <https://www.bulkrenameutility.co.uk/forum/>
3. The Sugar library is Copyright (c) 2014 Andrew Plummer. It is licensed under the MIT license.
4. The Date.js library is copyright (c) 2006-2007, Coolite Inc. all rights reserved, licensed under the MIT license.
5. The following are available to JavaScript :

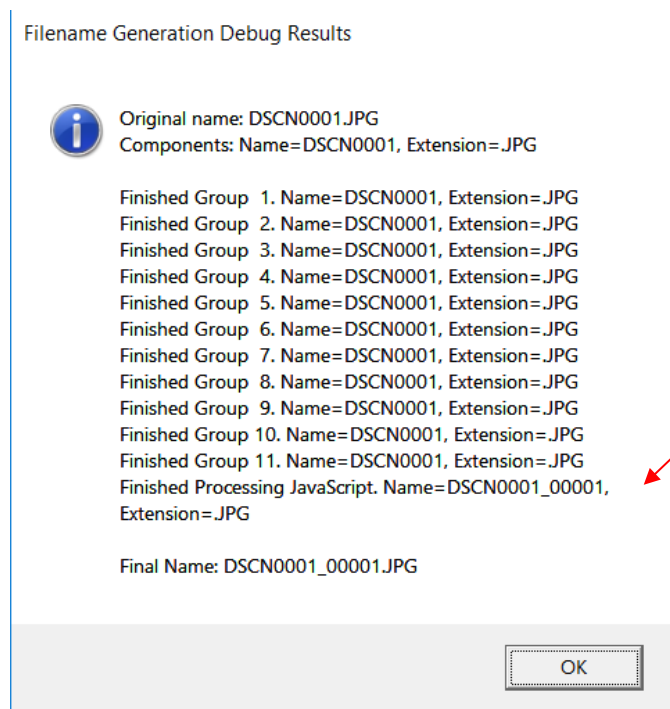
Windows Properties – see ‘Using Windows Properties’ under ‘**Section #7: Add**’

Substitution Tags – see ‘Using Substitution Tags’ under ‘**Section #7: Add**’

EXIF Tags – see ‘Using EXIF Tags’ under ‘**Section #7: Add**’

‘ID3 / EXIF Data / File Properties’ options must be set from the ‘Renaming Options Menu’ prior to use.

6. Processed JavaScript appears as the last line in the ‘Debug Results using the ‘Debug Name’ function from the ‘Actions Menu’. It may be noted it is not part of a group e.g., defined as Group 12, (‘**Section #14: Special**’), but its own entry.

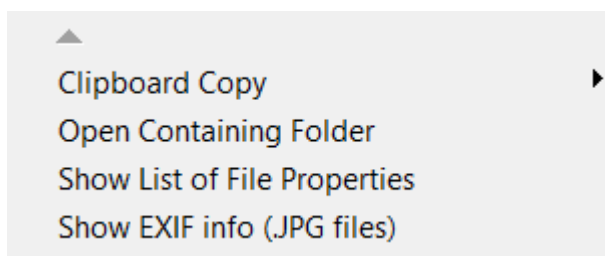


For more information, see ‘Debug Name’ option under the ‘Actions Menu’

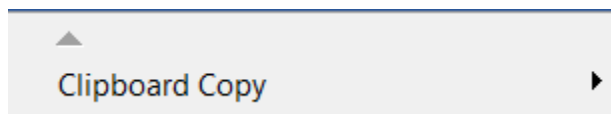


Context Menu

From Bulk Rename Utility: Content Panel



From Bulk Rename Utility: Navigation Panel



From Windows Explorer



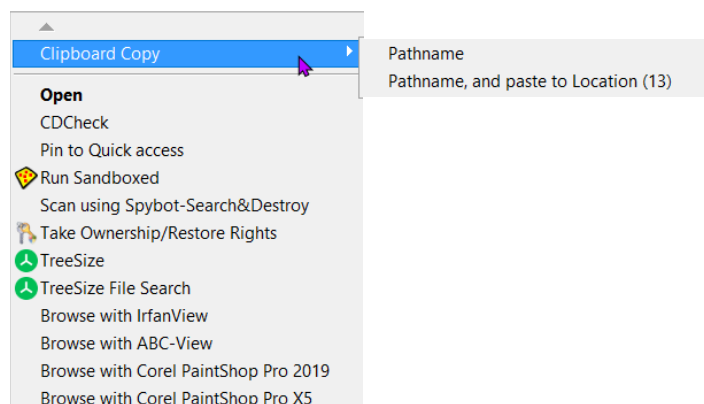
BRU Context Menu

Clipboard Copy

When you install BRU, it installs a Context Menu, 'Clipboard Copy', that can be accessed by right clicking either in the Navigation Pane or Content Pane of the Windows Explorer window.

Navigation Pane

In the Navigation Pane, right click on any directory or drive:

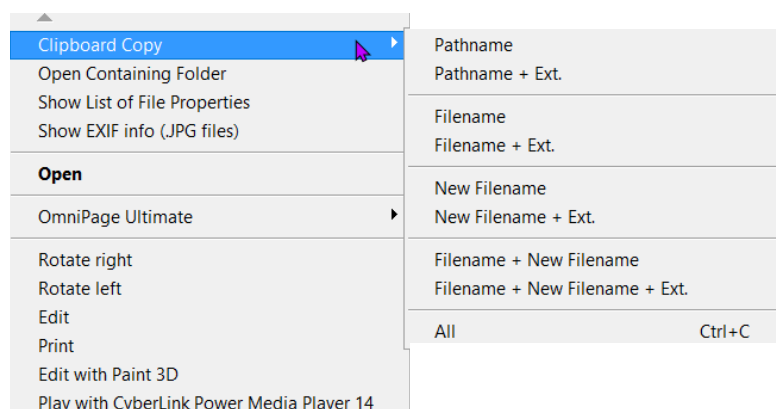


Your options are:

- **Pathname**
copy the directory pathname to the clipboard.
- **Pathname, and paste to Location (13)**
copy the path to the clipboard and set it as the new location (target destination) for a copy or move operation.

Content Pane

In the Content Pane, right click on a file or directory in the Name column:



Your options are:

- **Pathname**
copy the directory pathname (including filename) with or without file extension to the clipboard.
- **Filename**
copy just the filename with or without the extension to the clipboard.
- **New Filename**
copy the filename that appears in the New Name column with or without file extension to the clipboard.
- **Original Filename with New Filename**
copy the original filename along with the new filename with or without file extension to the clipboard.

Note:

The previous version had an additional option – 'Extended File Details' – this has been replaced by the 'All' option.

BRU Context Menu

Example of using Clipboard Copy from Navigation Pane:

Option	Clipboard Contents
Pathname:	F:\Music\Classical

Examples of using Clipboard Copy from Content Pane:

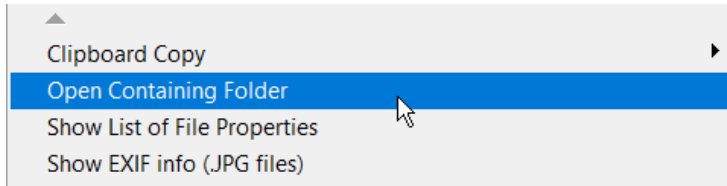
Option	Clipboard Contents							
Pathname:	F:\Music\Classical\Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro							
Pathname + Ext.:	F:\Music\Classical\Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro.mp3							
Filename:	Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro							
Filename + Ext.:	Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro.mp3							
New Filename:	220414Alvaro Cassuto- Sinfonia in B flat I- Allegro							
New Filename + Ext.:	220414Alvaro Cassuto- Sinfonia in B flat I- Allegro.mp3							
Filename + New Filename:	Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro 220414Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro							
Filename + New Filename + Ext.:	Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro.mp3 220414Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro.mp3							
Extended File Details:								
Filename	Parent Folder	Full Path	File Type	Size	Created	Modified	Accessed	
Alvaro Cassuto- Sinfonia in B	flat III- Minuet Allegro.mp3	Classical	F:\Music\Classical\Alvaro Cassuto- Sinfonia in B flat III- Minuet Allegro.mp3	VLC				
media file (.mp3)	3 MB	4/22/2014 9:23:28 AM	4/22/2014 9:23:38 AM	4/22/2014 9:23:28 AM				

Notes:

1. Fonts were changed in examples as required and do not reflect the font of the stored data.
2. In illustrating the Filename + New Filename selections, carriage returns were added for clarity and would not be part of the clipboard data. The data would be copied as one continuous line to the clipboard.
3. Box 13 refers to '[Section #13: Copy/ Move to Location](#)'
4. Extended File Details option no longer available as of v3.4 – Replaced by the 'All' option.

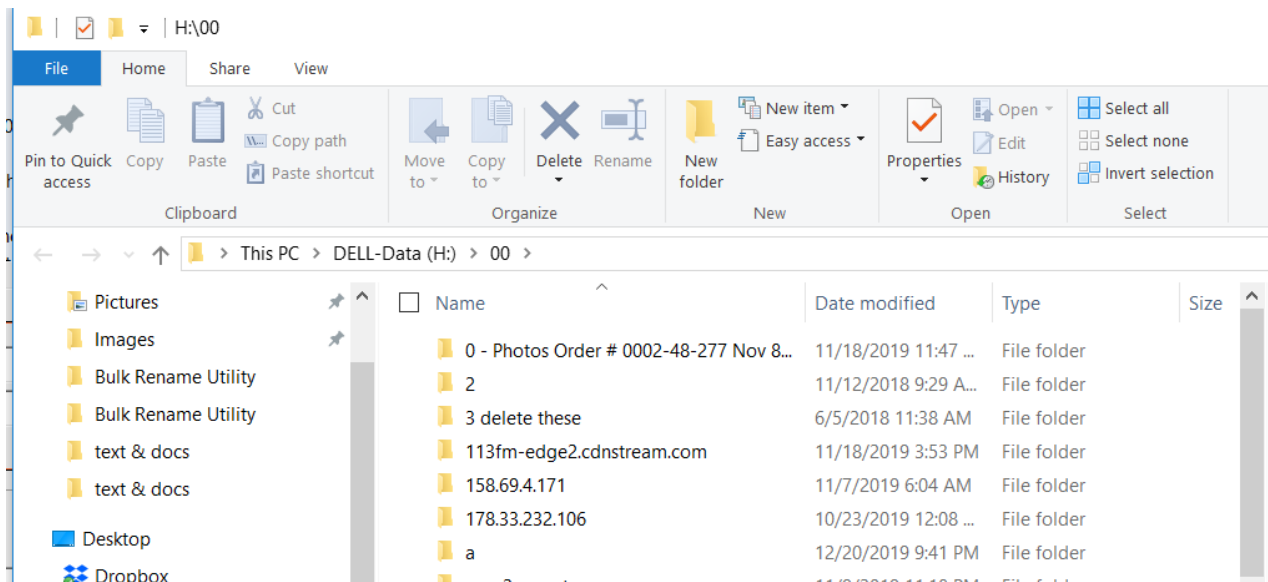
BRU Context Menu

In addition to Clipboard Copy, the Context Menu from the Content Pane provides several additional options:



Open Containing Folder

This initiates the standard Windows Explorer for the current directory.



Show List of File Properties

This has to do with ‘Using Windows Properties’ under ‘[Section #7:Add](#)’. It provides a complete list of all of the available Windows Properties for a selected file. BRU allows values assigned to these properties to be inclusive (appended) in ‘Add’ and JavaScript renaming operations. For more information, refer to these sections.

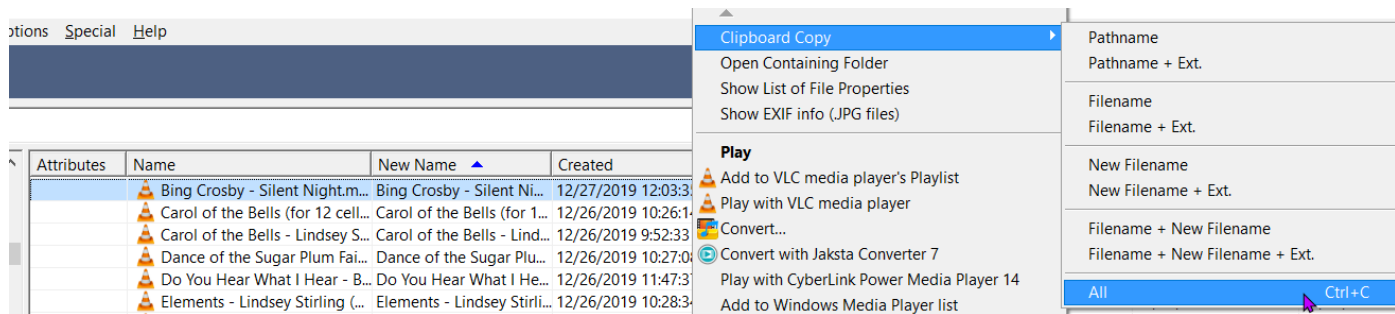
Show List of EXIF Info (.JPG Files)

This has to do with ‘Using EXIF Tags’ under ‘[Section #7:Add](#)’. It provides a complete list of all of the available EXIF Metadata for a selected (and supported) image filetype. BRU allows values assigned to these properties to be inclusive (appended) in ‘Add’ and JavaScript renaming operations. For more information, refer to these sections.

BRU Context Menu

v3.4 New Additions

Copy all available column data for highlighted files using the Clipboard Copy menu option 'All' or Ctrl + C



The contents of the Clipboard buffer are a bit of a jumble. This is how they would appear:

```

Filename      Parent Folder  Full Path      File Type      Size  Created Modified      Accessed      Length
Taken (Original)  Item Date      Attributes     Track  Status  New Name  Custom
Bing Crosby - Silent Night.mp3 Christmas      H:\Music\Christmas\Bing Crosby - Silent Night.mp3  MP3 Audio File
(VLC) 6 MB  12/27/2019 12:03:35 AM  12/27/2019 12:11:51 AM  12/27/2019 12:03:35 AM  30
      12/27/2019 12:03 AM      0      Bing Crosby - Silent Night.mp3 File Size is 6.17 MB - Modified Date
is 12/27/2019 12:11 AM)
    
```

I have clarified the contents of the Clipboard for better comprehension, but this is not how they would appear:

```

Attributes

Filename      Parent Folder  Full Path
Bing Crosby - Silent Night.mp3 Christmas      H:\Music\Christmas\Bing Crosby - Silent Night.mp3

Created      Taken (Original)  Item Date
12/27/2019 12:03:35 AM      12/27/2019 12:03 AM

Accessed      Modified
12/27/2019 12:03:35 AM      12/27/2019 12:11:51 AM

Custom
File Size is 6.17 MB - Modified Date is 12/27/2019 12:11 AM)

File Type      Size      Length      Status      Track
MP3 Audio File (VLC)  6 MB      30

New Name
Bing Crosby - Silent Night.mp3
    
```

BRU Context Menuv3.4 New Additions

The available column data can also be copied by selecting the files directly within BRU and then hitting Ctrl + C without having to use the Context Menu:

Attributes	Name	New Name ▲	Created	Taken (Original)	Item Date	Accessed	Modified
	Bing Crosby - Silent Night.m...	Bing Crosby - Silent Ni...	12/27/2019 12:03:35 AM		12/27/2019 12:03 AM	12/27/2019 12:03:35 ...	12/27/2019 12:11:51 AM
	Carol of the Bells (for 12 cell...	Carol of the Bells (for 1...	12/26/2019 10:26:14 PM		12/26/2019 10:26 PM	12/26/2019 10:26:14 ...	12/26/2019 11:06:42 PM
	Carol of the Bells - Lindsey S...	Carol of the Bells - Lind...	12/26/2019 9:52:33 PM		12/26/2019 9:52 PM	12/26/2019 9:52:33 P...	12/26/2019 10:24:39 PM
	Dance of the Sugar Plum Fai...	Dance of the Sugar Plu...	12/26/2019 10:27:08 PM		12/26/2019 10:27 PM	12/26/2019 10:27:08 ...	12/26/2019 11:08:23 PM
	Do You Hear What I Hear - B...	Do You Hear What I He...	12/26/2019 11:47:37 PM		12/26/2019 11:47 PM	12/26/2019 11:47:37 ...	12/26/2019 11:47:53 PM
	Elements - Lindsey Stirling (...)	Elements - Lindsey Stiri...	12/26/2019 10:28:34 PM		12/26/2019 10:28 PM	12/26/2019 10:28:34 ...	12/26/2019 11:14:38 PM
	God Rest Ye Merry Gentleme	God Rest Ye Merry Gent	12/27/2019 12:12:42 AM		12/27/2019 12:12 AM	12/27/2019 12:12:42	12/27/2019 12:15:38 AM

Dump of Clipboard buffer. This is how the contents would appear:

```

Filename      Parent Folder  Full Path      File Type      Size  Created Modified      Accessed      Length Taken
(Original)   Item Date     Attributes     Track Status   New Name      Custom
Bing Crosby - Silent Night.mp3 Christmas     H:\Music\Christmas\Bing Crosby - Silent Night.mp3  MP3 Audio File
(VLC) 6 MB 12/27/2019 12:03:35 AM 12/27/2019 12:11:51 AM 12/27/2019 12:03:35 AM 30
12/27/2019 12:03 AM 0 Bing Crosby - Silent Night.mp3 File Size is 6.17 MB - Modified Date
is 12/27/2019 12:11 AM)
Carol of the Bells - Lindsey Stirling.mp3 Christmas     H:\Music\Christmas\Carol of the Bells - Lindsey Stirling.mp3
MP3 Audio File (VLC) 7 MB 12/26/2019 9:52:33 PM 12/26/2019 10:24:39 PM 12/26/2019 9:52:33 PM 41
12/26/2019 9:52 PM 0 Carol of the Bells - Lindsey Stirling.mp3 File Size is
7.12 MB - Modified Date is 12/26/2019 10:24 PM)
Do You Hear What I Hear - Bing Crosby.mp3 Christmas     H:\Music\Christmas\Do You Hear What I Hear - Bing
Crosby.mp3 MP3 Audio File (VLC) 6 MB 12/26/2019 11:47:37 PM 12/26/2019 11:47:53 PM 12/26/2019
11:47:37 PM 41 12/26/2019 11:47 PM 0 Do You Hear What I Hear - Bing Crosby.mp3
File Size is 6.33 MB - Modified Date is 12/26/2019 11:47 PM)

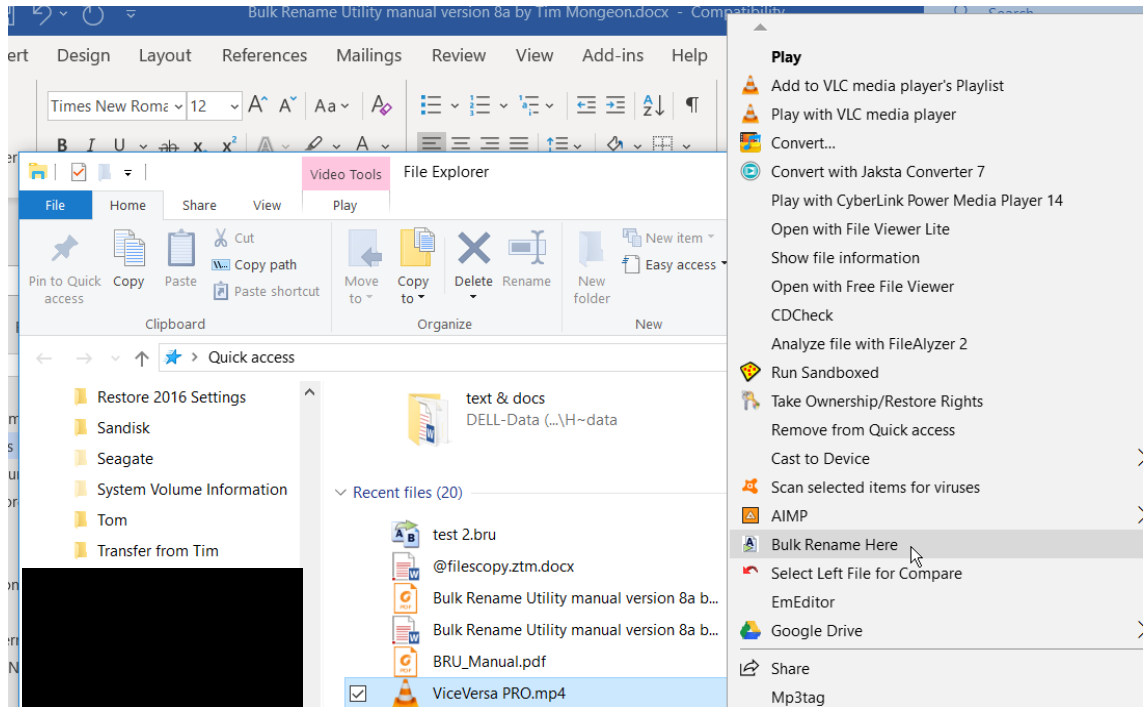
```

It is up to the user to sort through the contents and arrange it as needed, just as I did in the previous example.

Windows Explorer Context Menu

In addition to the Context Menu available through the Bulk Rename Utility Navigation and Content Pane, when you agree to install the Context Menu along with BRU at the time of program installation, it adds one item to the Windows Explorer Context menu.

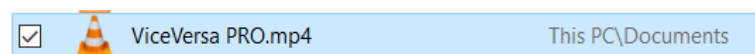
Bulk Rename Here



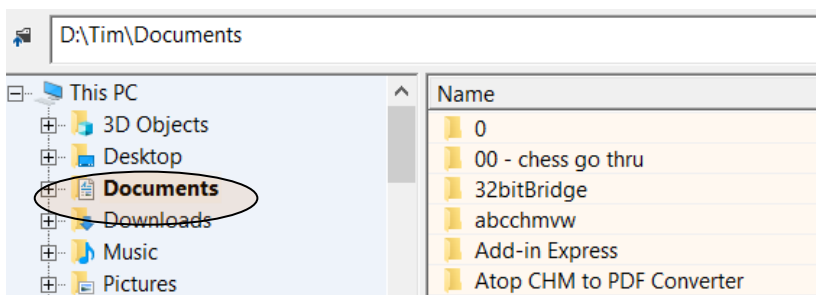
Selecting this option will launch Bulk Rename Utility, and will automatically select the folder you right-clicked (or the parent folder, if you right-clicked on a file).

Example:

This file was selected in Windows Explorer.



After selecting 'Bulk Rename Here' from the Context Menu:



BRU comes up with the directory already selected. This option only selects the directory, not the file.

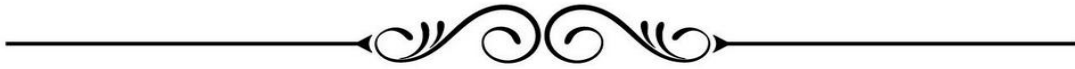
Windows Explorer Context Menu

Notes:

1. This will work regardless if BRU is already running.
2. If you have a Favourite file loaded and 'Save on Exit' option set, the current folder will replace the folder saved in the Favourites file when you exit the program. If you wish to avoid this, reload the Favourites file or use 'Revert All Criteria to Last Saved' option from the Actions Menu.
3. For reasons of performance, if you use the Bulk Rename Here right-click function, the Sub-folders flag (recursion) will not be enabled, regardless of its stored setting.

This is to prevent situations where you choose a high-level folder (e.g. C:\) without realizing that a recursive directory search is about to be performed. The note regarding Favourites above also applies here.

Appendix



Speeding up the Program

Speeding up the Program

Display Options Menu

List - Show Icons

Normally enabled, but displaying icons can sometimes slow down a long file listing.

Renaming Options menu

ID3 / EXIF Data / File Properties

Leave off unless absolutely required because extracting the data from each file during a scan can slow down the processing quite a bit.

Section #12: Filters

Subfolders option (Recursive Scan)

Recursive scan reads all the files not only in the current directory but in all subdirectories below it. Depending on the amount of files that have to be read, this can take a long time and you may have to wait awhile until the Content Pane displays anything.

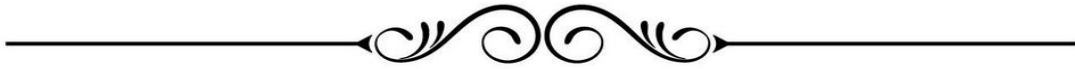
Remember to look at the status indicator at the bottom of the Interface window which will provide feedback as to why an operation is possibly taking longer than necessary. Typically, it is because the directories are in process of being scanned.

v3.4 New Additions

Section #1: RegEx

Enabling v2 (PCRE v2 with Boost)

I believe, and this is not verified, that just as with enabling certain features – JavaScript libraries, Extraction of Metadata, etc., performance could take a hit. So my thinking is if it is not warranted, don't enable it.



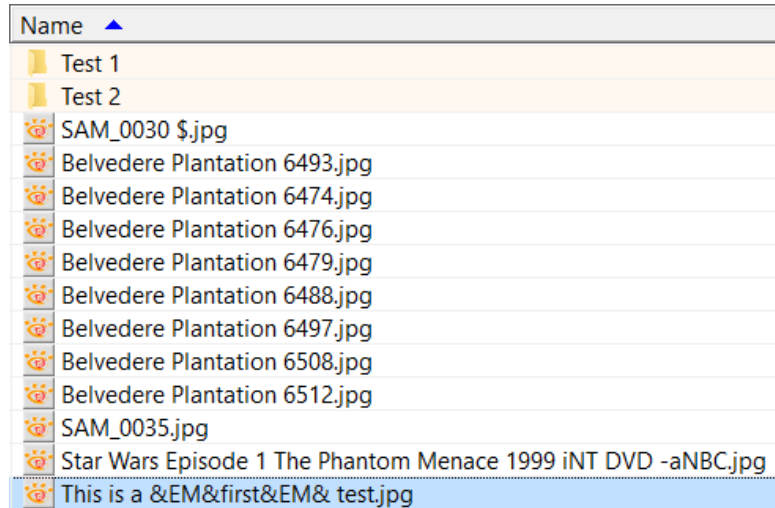
Regular Expressions (RegEx) Manual

Volume I

Regular Expressions (RegEx) Manual

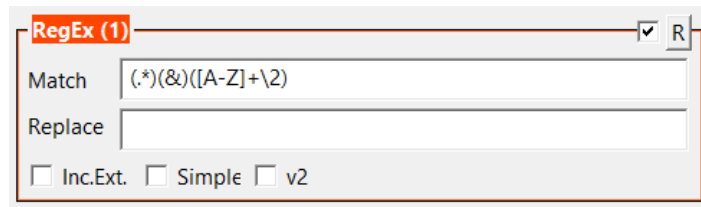
Applying example to BRU – Review Time

1. Highlight and select what you want to rename.

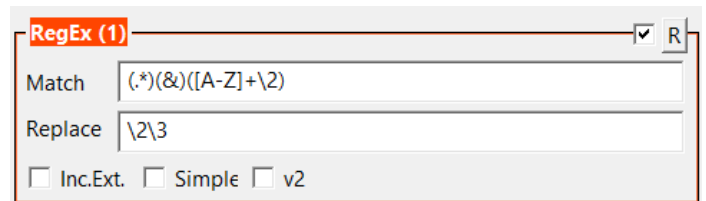


This example uses only one file.

2. Enter the RegEx expression in the Match data field.



3. Enter the replacement data in the Replace data field.



4. If the match is successful, the name under the New Name column will be in Green lettering. If the Match is unsuccessful, there will be no change in New Name. If there is an error in the expression, the status at the bottom will show 'Invalid'. If there are other kinds of problem, like having to do with the replacement data, New Name will be in red lettering due to illegal literals like a backslash displayed because of a non-existent Capture Group.

Successful match

&EM&.jpg

Match unsuccessful

This is a &EM&first&EM& test.jpg

Error in Expression

Regular Expression is invalid

Other Problem

\2\3.jpg

What's great about this program is that you can enter in different data and the New Name will immediately change to reflect this new data. This allows you to test and retest without committing to the rename action.

Notes:

1. Although the file extension shows up in all the New Name files, it is not a consideration in evaluating for a match unless explicitly checked off in the 'Include Ext.' field or enabled in the Renaming Options Menu.

Regular Expressions (RegEx) Manual

v3.4 New Additions

BRU Supports PCRE v2 with Boost

BRU now supports PCRE v2 with Boost. What this means is that most of the limitations, some which have been documented here in this volume and in Volume II, have been removed. Modifiers such as Global Switch, Case Insensitivity, and many other features that are part of the Perl Regular Expression Engine v2 are now available.

Some of these additional features of the language have been introduced in this book. If the user wishes to explore the newer capabilities of the language other than what has already been presented, I have provided some resources below.

I can understand why TGRMN has decided to include both v1 and v2 in the new version. The RegEx v2 with all of its capabilities does not replace v1 – only supplements it.

The majority of the RegEx Manuals in both volumes covers PCRE v1, but this is not to say that the RegEx manual in this volume has not undergone a complete rewrite. It has. A lot of new material and instruction has been added.

General information:

<https://perldoc.perl.org/perlre>

Information specific to the Match Equation:

https://www.boost.org/doc/libs/1_74_0/libs/regex/doc/html/boost_regex/syntax/perl_syntax.html

Information specific to the Replace String:

https://www.boost.org/doc/libs/1_74_0/libs/regex/doc/html/boost_regex/format/boost_format_syntax.html

Additional resources for using some of these new capabilities can be found at:

<https://www.rexegg.com/>

<https://www.regular-expressions.info>

<https://regexlib.com>

Regular Expressions (RegEx) Manual

Literal – a regular or normal character (that can be reproduced with the keyboard) includes all characters except for the following special metacharacters: `. | * ? + () { [^ $ \`

Special Characters (aka Metacharacters) – certain characters are reserved for special use. They cannot be used as a literal unless preceded by an escape. An escape Metacharacter is the backslash, e.g., `\[` is a literal left bracket.

Metacharacters include:

<code>\</code>	backslash	Escape character. Used to specify that the next character will be a literal even though this character is normally a reserved metacharacter. e.g. <code>\+</code> is a plus sign.
<code>.</code>	period or dot	Matches any single character.
<code> </code>	vertical bar or pipe symbol	Used to separate alternatives . Think of it like an OR operator. e.g. <code>cat dog</code>
<code>-</code>	hyphen	Specifies a range (usually within a class).

Word Boundaries

<code>\b...\b</code>	A Word Boundary is a word (alpha numeric or underscore) isolated with no characters before or after it except whitespace or punctuation (whole word) The characters between the ' <code>\b</code> ' establish the Word Boundary of the word to be matched. Equivalent to <code>[a-zA-Z0-9_]</code> . E.g. <code>\bcat\b</code> matches 'cat' but not 'catfish'. The Word Boundary can be used singularly, e.g. <code>\b</code> , or in pairs.
<code>\B...\B</code>	Requires the current character not be at a Word Boundary. The character must be inside a word and not at a Word Boundary. There can be no whitespace or punctuation before or after it, only other characters. <code>\Bice\B</code> matches 'priceless' but not 'price'. The non-Word Boundary can be used singularly, e.g. <code>\B</code> , or in pairs. Negated <code>\b</code> .

Word Characters

<code>\w</code>	A Word Character used to form words. Equivalent to <code>[A-Za-z0-9_]</code> .
<code>\W</code>	Matches against all characters that are not Word Characters. Negated <code>\w</code> .

Regular Expressions (RegEx) Manual

Special Characters (aka Metacharacters) cont.

Anchors

^	the caret	Requires any match to occur at the beginning of the string input. \A is same as ^ This is called the <u>BOL</u> or Beginning of Line.
\$	dollar sign	Requires any match to occur at the end of the string input. \Z is same as \$ This is called the <u>EOL</u> or End of Line, aka EOF or End of File.

Quantifiers

*	asterisk or star	Match the previous (character, metacharacter, Capture Group or class) zero or more times.
+	plus sign	Matches the previous (character or metacharacter, Capture Group or class) one or more times.
?	question mark	Matches the previous (character or metacharacter, Capture Group or class) zero or one time. The question mark makes the previous optional.

Range Quantifiers (Minimum, Maximum)

{...}	curly braces	Limits how many matches (or attempts at matches through iterations):
{ <i>n</i> }		Specifies exactly how many matches are allowed. Match previous (character, metacharacter, Capture Group or class) exactly <i>n</i> times. e.g., \d{4}
{ <i>n</i> ,}		Specifies that it can match <i>n</i> or more times. Match previous (character, metacharacter, Capture Group or class) at least <i>n</i> times with no upper limit. e.g., \d{3,}
{ <i>min</i> , <i>max</i> }		Limits how many times something can be repeated. Match previous (character, metacharacter, Capture Group or class) at least <i>min</i> times but no more than <i>max</i> times, e.g., \d{3,5}

Regular Expressions (RegEx) Manual

Special Characters (aka Metacharacters) cont.

Numeric Digits

<code>\d</code>	Matches any single numeric digit – equivalent to class <code>[0-9]</code> .
<code>\D</code>	Matches any single non-numeric digit – equivalent to class <code>[^0-9]</code> . Negated <code>\d</code> .

Class

<code>[...]</code> square brackets	A character class. Match characters contained in list or range.
<code>[^...]</code> caret inside brackets	A negating character class. Do Not match characters contained in list or range. Generally it searches to match any character that is not of this class. I think it is more accurate, however, to state that the characters contained in a negating class are seemingly ignored when the engine is moving forward or back through the string. This opinion is merely an observation.

Subpattern (Capture Group)

<code>(...)</code> parentheses	Creates a subpattern group also called grouping or Capture Group. In this document, I will refer to subpatterns as Capture Groups for simplicity. A sub-expression is enclosed between a left and right parenthesis. The RegEx Engine will evaluate this Capture Group and if the pattern is matched, the value produced is held or ‘captured’. This value can then be recalled in BRU’s Regular Expression ‘Replace’ data entry field, using the syntax: <code>\1</code> to represent Capture Group 1, <code>\2</code> to represent Capture Group 2, etc.
--------------------------------	---

Regular Expressions (RegEx) Manual

Special Characters (aka Metacharacters) cont.

Non-Printable Characters

<code>\n</code>	Matches new line (line break).
<code>\r</code>	Matches carriage return.
<code>\t</code>	Matches tab character.
<code>\s</code>	Matches any single Whitespace character – a space, tab, and newline.
<code>\S</code>	Matches any single non-Whitespace character. Negated <code>\s</code> . Will not match against space, tab and newline because these are Whitespace.

The only non-printable Whitespace character supported in BRU is the <space> character

\ The Escape (backslash)

If you want to use any of the following metacharacters as a literal, you must precede it with the escape:

`\ () [{ + * ? | ^ $`

For example, to include a `*` in the Match expression as a literal, you would use `*`

Metacharacters in Depth

Character class (or just class)

I like to think of this in terms of algebra. In algebra, groups of related characters are considered collected together when classified as a 'set'. When something is in a set, it is treated as one element in an equation. This is kind of what character classes are. Groups of characters that are placed between square brackets are called a 'character class'. A class can match any single one of the characters between the brackets, or any single character belonging to a range of characters. The range is specified using a <hyphen> (-) as a delimiter:

e.g., [0-9] is a range with the set of characters = 0 1 2 3 4 5 6 7 8 and 9.

Just like in sets, you can have what **to include** and what **not to include**. What not to include is referred to as a **negated character class**. What to include is identified by placing the characters between two square brackets []. What not to include (negated) is identified by adding a caret ^ as the first character, before any other characters.

Examples (with any ranges specified):

ho[ur]se - the characters that **are** included are u and r. [ur] is a character class.
 ho[^ur]se - the characters that are **not** included are u and r. [^ur] is a negated class (this is still a character class).

[a-z]	a class that includes any single lowercase character a-z range: a-z
[A-Z]	a class that includes any single uppercase character A-Z range: A-Z
[A-Za-z]	a class that includes any single alpha character range: upper and lowercase
[A-D]	a class that includes a range of uppercase characters, A,B,C,D
[ab3-]	a class that includes a, b 3 or <hyphen>. The <hyphen> must be the last character in the list.
[0-9]	a class that includes any single numeric (digit) character range: 0-9
[A-Za-z0-9]	a class that includes any single alpha numeric character range: uppercase, lowercase, numeric digits
[^0-9xyz]	a Negated class that matches any single character that is NOT a numeric or the letters x,y or z, or a numeric digit that falls within the range: 0-9
[^-]	a Negated class that matches a single character, numeric, Whitespace or punctuation but is NOT a <hyphen> character
h[aeiou][a-z]	a class that matches hat, hip, hit, hop, and hut range: a-z

Metacharacters in Depth

Character class (or just class) cont.

The only metacharacters allowed inside a class are.. (`[]`), (`\`), (`^`) and (`-`). All others are treated as a literal and do not require a backslash 'escape' character (to use a backslash as a literal, you use `\\` inside a class, e.g. the right bracket, `[abc\\]`).

Example:

`[.]` is treated as a dot literal and not a dot metacharacter

Notes:

1. Each class represents a **single** character in a search

Example:

`[A-Za-z] [a-zA-Z0-9]` represents two characters, an alpha character and an alpha numeric character

2. In a class containing a <hyphen>, the <hyphen> must appear at the end of the list, otherwise it will be mistakenly interpreted as a range delimiter.

Range delimiter:

`[0-9]`

defines a class made up of numeric digits, 0-9 or 0,1,2,3,4,5,6,7,8,9

Range delimiter with added <hyphen>:

`[0-9-]`

defines a class made up of numeric digits, 0-9, and a <hyphen>

Metacharacters in Depth

Posix Character Class (Bracket Expression)

POSIX	Description	Class Equivalent
<code>[:alnum:]</code>	Alphanumeric characters	<code>[a-zA-Z0-9]</code>
<code>[:alpha:]</code>	Alpha characters	<code>[a-zA-Z]</code>
<code>[:ascii:]</code>	ASCII characters	<code>[\x00-\x7F]</code>
<code>[:blank:]</code>	Space and tab	<code>[<space> \t]</code>
<code>[:cntrl:]</code>	Control characters	<code>[\x00-\x1F\x7F]</code>
<code>[:digit:]</code>	Numeric Digits	<code>[0-9]</code>
<code>[:graph:]</code>	Visible characters (anything except spaces and control characters)	<code>[\x21-\x7E]</code>
<code>[:lower:]</code>	Lowercase letters	<code>[a-z]</code>
<code>[:print:]</code>	Visible characters and spaces (anything except control characters)	<code>[\x20-\x7E]</code>
<code>[:punct:]</code>	Punctuation and symbols, including left and right brackets	<code>[!"#\$%&'()*+,-./:;<=>?@\[\]^_`{ }~]</code>
<code>[:space:]</code>	All whitespace characters, including line breaks	<code>[<space> \t\r\n\v\f]</code>
<code>[:upper:]</code>	Uppercase letters	<code>[A-Z]</code>
<code>[:word:]</code>	Word Characters (letters, numbers and underscores)	<code>[A-Za-z0-9_]</code>
<code>[:xdigit:]</code>	Hexadecimal digits	<code>[A-Fa-f0-9]</code>

To use with BRU, the Bracketed Posix Class must be enclosed in an additional square brackets. This is because it involves two different syntax, not just one. [The syntax of the Posix requires the keyword enclosed in brackets](#), but the addition of using it in a [Character Class requires that any value be enclosed between brackets](#); thus the two brackets:


Example-

Belvedere Plantation 6492.jpg



Match: `([:digit:])`

Replace: `\1`

Name	New Name
 Belvedere Plantation 6479.jpg	6.jpg

Metacharacters in Depth

Dot Metacharacter


.

Dot represents any single character



Example:

ab. Matches strings **abc** and **abz** and **ab_**
h.t Matches strings **hat**, **hit**, **hot** and **hut**.

Name	New Name
 abc	abc

Notes:

1. Bru under PCRE v1 only returns the value of the first match if there were more than one matched.
2. New Name only displays in **green** if there is a valid match and the original Name would be changed as a result in some form. It is a preview of what will happen before committing to the Rename operation.
3. In the example above, **ab.** matches the entire string, **abc**. Ordinarily, New Name would not be changed and would display as:

Name	New Name
 abc	abc

This is because, even if there is a match, which there is, Name would not be changed, therefore New Name reflects no change. In order to display a change, I added a <space> character in the Replace String.

New Name is now 'abc <space>'. I use this technique a lot in these volumes whenever I want to illustrate a change, that would otherwise not display.

Match: (ab.)
 Replace: \1 <space>

4. To display any changed values, the values have to be 'captured'. That is what the parentheses around **ab.** are for. They capture the evaluated value into the Capture Group. The RegEx designates the Capture Group as Capture Group 1 because it assigns each Capture Group as they are encountered from left to right.

In this example, there is only one, designated as Capture Group 1. The value of Capture Group 1 can be recalled in the Replace String using \1.


5. Additional photos are provided by the program, Regex Buddy.

Metacharacters in Depth

`\s` Matches any single **Whitespace character** – a <space>, tab, newline, among others. This can be used inside a character class. [] In BRU only the <space> is supported.

Match: (e\sP)

Replace: \1

Name	New Name
 Belvedere Plantation 6474.jpg	e P.jpg

1. e Match against a lowercase ‘e’ Capture Group 1 = e
This is the ‘e’ after ‘B’.
2. \s Match against a <space> character Capture Group 1 = e <space>
This changes the original ‘e’ captured from the ‘e’ after ‘B’, to the ‘e’ after ‘r’ of ‘Belvedere’.

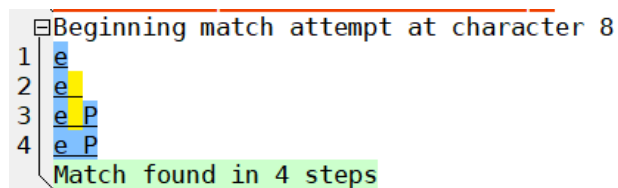
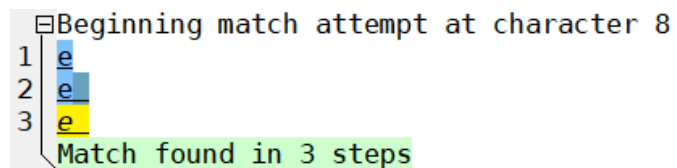
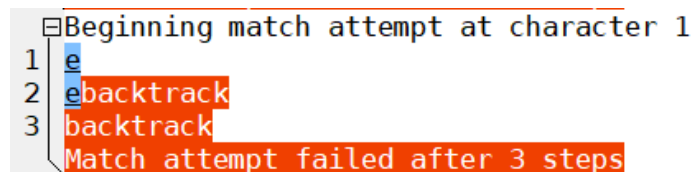
Why?

Because in order to match against the <space> it had to move forward testing each character for the <space> until it matched, which it did. However, as it moved forward after the initial match of the ‘e’ character, it also backtracked because the pattern was now ‘e’ with a <space> immediately following. The only pattern that matched was the – ‘e <space>’ following the ‘r’.

This can best be illustrated as:

Belvedere **e** Plantation 6474.jpg

Belvedere **e P**lantation 6474.jpg



The first photo shows the initial match, and the second shows the second match with the added <space> in the pattern, represented by the underscored blank character after the underscored ‘e’. The third photo shows the final pattern matched as ‘e<space>P’.


3. P Match against a ‘P’. Continues to move forward and matches against the ‘P’ following the <space> after ‘Belvedere’. Capture Group 1 = e<space> P
4. Will continue searching the remainder of the string looking for any additional matches that match the established pattern, ‘e<space>P’. Finding none, the RegEx fails at that point having reached the EOL, satisfying the match and exhausting the string (no more matches can be made).

Metacharacters in Depth

\S Matches any single non-Whitespace character. This is the Negated \s. Where \s matches against the Whitespace character, <space> in BRU, \S will match against anything that is not a <space> character. This can be used inside a character class. []

Match: (e\S)

Replace: \1

Name	New Name
 Belvedere Plantation 6474.jpg	el.jpg


1. e Match against the 'e'. Capture Group 1 = e
This is the e after 'B' in 'Belvedere'.

2. \S Match against the first non-Whitespace character. Capture Group 1 = el
This is the 'l' after 'e' in 'Belvedere'

Pretty straight forward.


What happens if I change it by adding 'e' to the pattern?

3. e Match against the 'e'. Capture Group 1 = ede
Changes the original 'e' matched after the 'B' to the 'e' after the 'v' in 'Belvedere'

Name	New Name
 Belvedere Plantation 6474.jpg	ede.jpg

The pattern becomes 'e' followed by any non-Whitespace character followed by a second 'e'. Therefore, 'el' no longer matches. The RegEx engine moves forward and matches the pattern against 'e' followed by 'd' followed by 'e'.

Likewise if I changed the match to (P\Sa), the pattern becomes, 'P' followed by a non-Whitespace character followed by an 'a'. The only possible match in the string becomes the 'Pla' of 'Plantation'.

Name	New Name
 Belvedere Plantation 6474.jpg	Pla.jpg

Capture Group1 = Pla

Notes:

In this section,

1. I used **green** whenever I am discussing the string.
2. I used **blue** when discussing matches.
3. Patterns and the RegEx and matches in these discussions are left as is.
4. Analysis is colour coded in **violet**.

Metacharacters in Depth


| The vertical bar is used to distinguish between alternatives. Think of it like an OR operator. You can have this or that or this or that.

Example:

gray|grey matches **gray** and **grey** which can also be accomplished by, gr(a|e)y

Match: (gray|grey)

Replace: \1<space>


Name	New Name
 gray	gray

There is that technique I use with the <space> added in the Replacement String, otherwise New Name would indicate no change because the entire string 'gray' is matched.

Let's change it to:

Match: (gray|grey)

Replace: \1

Name	New Name
 graygrey	gray

This is important regarding how a Conditional Alternative works. You will notice that in the string, both values would match the alternatives, but only 'gray' matches and not 'grey'.

Why?


The Conditional works on a first come, first served. In other words, only the first match is valid as far as BRU is concerned (under PCRE v1). When a match is found, the second alternative is never tested.

Notes:

1. The <space> technique in the Replace String is not needed because the entire string is not matched, only a portion.

What if I changed it to:

Match: (grey|gray)


Name	New Name
 graygrey	gray

hmmm.. no change in the result. I would have thought it would change to 'grey'. What happened?

Metacharacters in Depth

The vertical bar cont.

Match: (grey|gray)

Name	New Name
 graygrey	gray

This brings me to the second point. It isn't the alternates that are tested against the string, it is the string that is tested against the alternates. What I mean by this:

It is **NOT**

Grey, as an alternate, is first tested against 'gray' in the string, and failing that moves on to the second alternate, grey, that matches against 'grey' in the latter part of the string... no..

It **IS**

The first part of the string, 'gray' is tested against the alternate, grey. This fails, so it repeats the RegEx, testing the same portion of the string, 'gray', against the second alternate, gray = true. Match successful, Capture Group 1 = gray.

Notes:

1. This is under PCRE v1. Under PCRE v2, all matches would display, not just the first. The result would instead be the entire string, graygrey. The process, however, would still be the same. The first part of the string, 'gray' is tested against the first alternate, grey. This fails so the second alternate, gray, is tested against the same portion of the string, 'gray' = true. The Regex repeats looking for additional matches with the remainder of the string.

The second portion of the string, 'grey' is tested against the first alternate, 'grey' = true. EOL is reached, the RegEx has been satisfied, the string is exhausted. Two matches were made, 'gray' and 'grey'

string = graygrey

Match: (grey|gray)

Results:


Match 1:	gray
Group 1:	gray
Match 2:	grey
Group 1:	grey

Beginning match attempt at character 0	
1	g
2	gr
3	gr backtrack
4	g
5	gr
6	gra
7	gray
8	gray
Match found in 8 steps	
Beginning match attempt at character 4	
1	g
2	gr
3	gre
4	grey
5	grey
Match found in 5 steps	

Metacharacters in Depth

| The vertical bar cont.

Match: (grey|gray)

Name	New Name
 graygrey	gray


My analysis is a bit over simplified for clarity. What really happens is that each character in the string is tested against the pattern created by the alternate. This can be seen above where, 'g' then 'r' is matched in steps 1-2 in the first part of the photo above, but the 'e' tested fails at step 3, so it backtracks, this time testing the second alternate beginning with 'g' in step 4, matching again the 'r' in step 5, and this time looking to match against an 'a', and this matches in step 6. Step 8 completes the first match of 'gray'.

The second part of the photo illustrates the second match of 'grey'. It begins again at the beginning of the string testing against the first alternate, grey. Once again, each character is tested. Because 'grey' is in the first part of the string, it quickly matches moving forward with no backtracking necessary this time. The match is completed in step 5.

string = graygrey

Match: gr(a|e)y

Replace: \1 <space>

Name	New Name
 graygrey	a

Similar to before, in terms of the process.

1. gr Match against string 'gr' but not captured = 'gr'.
This is the 'gr' of 'gray'.
2. a Alternate.
Tests the next character after 'gr' for 'a' = true.
Second alternate not tested.
3. y Match against string 'y' but not captured = 'y'
Regex satisfied.
4. Tests remainder of string for additional matches
of the pattern. None found. String exhausted.

Capture Group 1 = a

If there had not been a 'y' that matched, the Regex would have failed prior to the Regex satisfied, and the previous match would have been 'given up' and subsequently there would be no match. Capture Group 1 would have no value and New Name would not have changed.

Metacharacters in Depth

| The vertical bar cont.

As already discussed, several expressions can be combined into one by using the parentheses () and | characters. The Parentheses creates a Capture Group, Capture Group 1:

Example:

```
(apple|^pear$)
```

a) This takes the expressions,

```
apple
^pear$
```

and places them into a Capture Group as defined by the surrounding parentheses.

b) The expressions remain as two separate expressions because they are alternates in a Conditional Group.

c) Because there is only one Capture Group defined, this is designated as Capture Group 1.

In summary:

Capture Group 1 contains a Conditional Alternate made up of two expressions, apple and ^pear\$.

Capture Group 1 matches strings containing the characters **‘apple’** or the word **‘pear’**.

In this example, it matches against any separate word **‘apple’**, or a word that contains the word **‘apple’** while the word, **‘pear’**, can only appear as a separate word and not contained within another word.

This is because of the caret, ‘^’ (**BOL**, Beginning of Line) which means that the alternate, pear, must be at the beginning of the string for a match to occur. The dollar sign, ‘\$’ (**EOL**, End of Line) further refines the match limiting the alternate, pear, not only to begin the start of the string, but additionally, it must be the last word in the string. In other words, to match against **‘pear’**, the word must be isolated with nothing before or after it; the only word in the string.

Match: (apple|^pear\$)

Replace: \1 <space>

Name	New Name
<input type="checkbox"/> two pears in the basket	two pears in the basket
<input type="checkbox"/> pearing into the night	pearing into the night
<input type="checkbox"/> They live on appleton avenue	apple
<input type="checkbox"/> pear	pear
<input type="checkbox"/> an apple a day	apple
<input type="checkbox"/> apple	apple

For more information, refer to the anchors, ^ and \$.

Metacharacters in Depth

These escape sequences are used to match against special characters:

<code>\f</code>	Match against the PAGE BREAK (form feed) character	(not supported in BRU)
<code>\t</code>	Match against the TAB character	(not supported in BRU)
<code>\r</code>	Match against the CARRIAGE RETURN character	(not supported in BRU)
<code>\n</code>	Match against the LINEFEED (new line) character	(not supported in BRU)

Used for matching expressions that span line boundaries. This cannot be followed by operators '*', '+' or {}, so you can only match an exact number of them (e.g. `\n\n` will match a single blank line.). Do not use this for constraining matches to the end of a line. It's much more efficient to use "\$".

Notes:

1. The reason these may not be supported in BRU is because they are ineffectual for matching a single Line of text, e.g. , a filename.

These Metacharacters are meant for multiline text segments. BRU may in fact accept them as valid, but since they do nothing to change the original string, they are ignored and no changes are reflected in New Name.

This theory is untested and I find no need to explore further.

Metacharacters in Depth

() Parentheses – used for:



1. **Create a set of optional characters.** Enclose the characters within parentheses followed by the question mark. This makes the previous expression optional. For more information, refer to the, ? Quantifier.

Examples:

- 1) Nov(ember)? matches **Nov** and **November**

Match: (Nov(ember)?)

Replace: \1 <space>








Name	New Name
 Nov	Nov
 November	November

‘**Nov**’ matches because the second part of the expression, ‘ember’ is made optional.

- 2) (Sun|Mon|Tues|Wednes|Thurs|Fri|Satur)day

Match: (Sun|Mon|Tues|Wednes|Thurs|Fri|Satur)day

Replace: \1

Name	New Name
 Monday	Mon
 Tuesday	Tues
 Wednesday	Wednes
 Thursday	Thurs
 Friday	Fri
 Saturday	Satur
 Sunday	Sun


Matches the name of any day. For example, ‘**Tuesday**’ will match because Tues is one of the alternates. After the RegEx Engine evaluates the expression through to Tues, it continues to process the next part of the expression and finds a secondary match for ‘**day**’.

2. **Grouping.** By placing part of the RegEx expression within parentheses, it groups that part of the expression.

This allows for locating and isolating a part of the input string. Data that matches is stored or ‘captured’ in these groups. These groups are referred to as ‘subpatterns’ because the RegEx expression is used to define a search pattern. Another common name for these are ‘Capture Groups’ and that is the term I will use to refer to them.

Example:

(B.*\sP)

Name	New Name
 Belvedere Plantation 6476.jpg	Belvedere P.jpg

Metacharacters in Depth

() Parentheses – used for: cont.

These Capture Groups are designated a number that is dependent on the order in which they occur. In a later discussion I will elaborate on just how these Capture Groups are assigned.

In the Replace data entry field, the resultant values of the data ‘held’ or ‘stored’ within these Capture Groups can be recalled by referencing them by their number in the form, `\n` where ‘n’ is the designated number of the Capture Group.

Example:

`\1 \2 \3`

Notes:

1. In the first example, the Replace String `\1` was used to recall the value of Capture Group 1 of `(B.*\sP)`.

2. The analysis is:

- | | | |
|-------|--|-------------------------------------|
| a. B | Match against the ‘B’ string. | Capture Group 1 = B |
| b. . | Match against any character. | Capture Group 1 = Be |
| c. * | Make it Greedy. | Capture Group 1 = <entire string> |
| | Current position = <u>EOL</u> . | |
| d. \s | Match against <space>. | Capture Group 1 = Belvedere <space> |
| | Backtracks to locate the first <space>. | |
| | Matches at <space> after the ‘e’ of ‘Belvedere’. | |

This changes Capture Group 1’s value to ‘Belvedere <space>’ because as the RegEx Engine backtracked and tested each character, it dropped those characters from the captured value.

- | | | |
|------|---|-------------------------------|
| e. P | Match against the ‘P’ string.
Moves forward to match against the ‘P’ of ‘Plantation’ | Capture Group 1 = Belvedere P |
| f. | Will continue to test moving forward looking for additional matches until it reaches <u>EOL</u> . | |

Metacharacters in Depth

() Parentheses – used for: cont.

3. **Nested Parentheses in Capture Groups** create a hierarchy that determines the order of evaluation much like that used in mathematics, only, the outer Capture Groups are evaluated *first* before the inner Capture Groups.

This is an example of a simple nested Capture Group.

(Nov(ember)?)

It breaks down as:

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

 Capture Group 2

(Nov (ember) ?)

Capture Group 1 contains the following expressions:

Capture Group 2 contains a single expression:

Nov (Capture Group 2) ?
ember

Using the string, **November** –

result =

Capture Group 1

\1 <space>

Name	New Name
<input type="checkbox"/> November	November

Capture Group 2

\2

Name	New Name
<input type="checkbox"/> November	ember

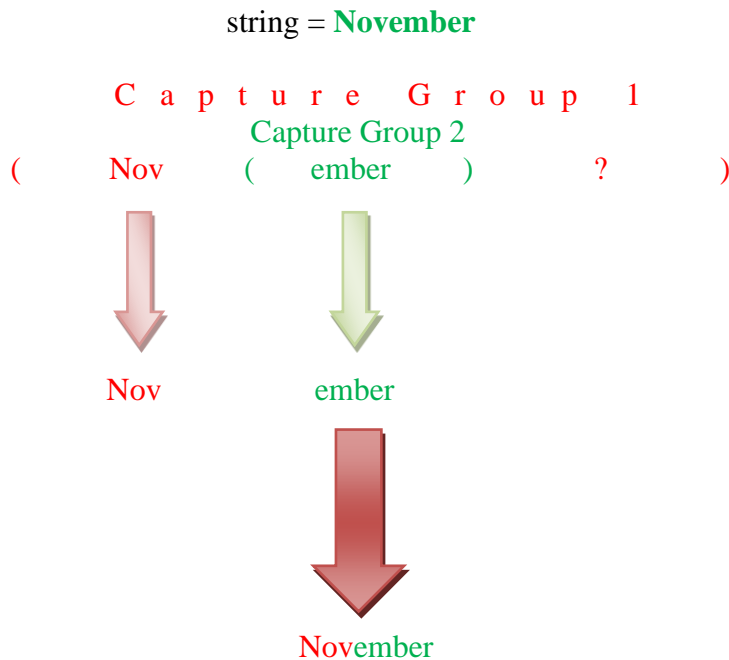
When a Capture Group is nested, the Outer Capture Group, Capture Group 1, holds the value of all nested Capture Groups, e.g., Capture Group 2, inclusive. This means that whatever values are evaluated from the expressions inside Capture Group 2 are aggregated (appended, concatenated, added with, etc., however you want to say it) - to any values resulting from the evaluation of expressions within Capture Group 1.

Thus, if Capture Group 2 evaluates to ‘ember’ and Capture Group 1 evaluates to ‘Nov’, then Capture Group 1’s value becomes. ‘November’. Capture Group 1, as the outer group, is evaluated first.

Metacharacters in Depth

() Parentheses – used for:

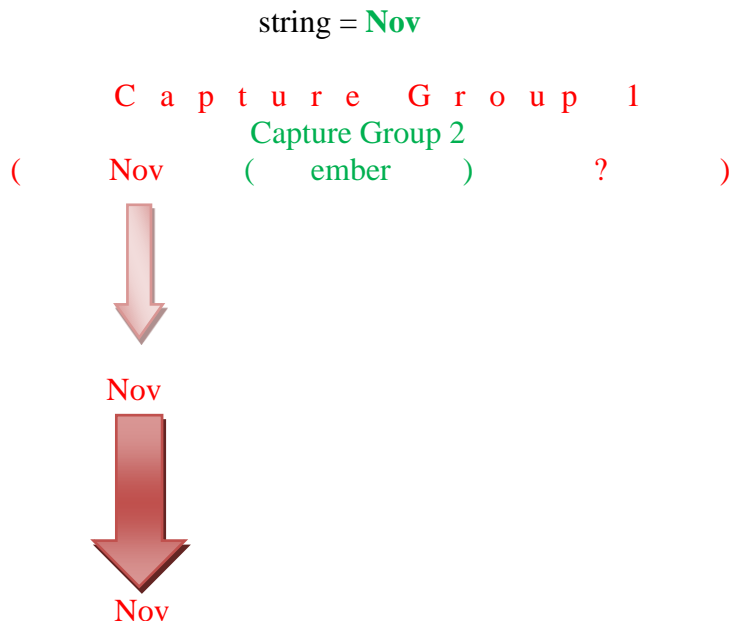
3. Nested Parentheses in Capture Groups cont.



Notes:

1. The ? Quantifier makes the (ember) expression optional, but because it matches against '**ember**', of '**November**' using the RegEx, (Nov(ember)?), the evaluation of the expression is performed in Capture Group 2.

If instead the string was, '**Nov**', The RegEx would exercise the option because the string '**ember**' does not match. Rather than fail the entire RegEx at this point, the expression, (ember), is optional, and the evaluation is allowed to proceed.



Metacharacters in Depth

() Parentheses – used for:

4. **When used with a Quantifier**, * ? + {*min,max*}, it matches the enclosed string against the input string (filename), zero, one, or more times, depending on the type of Quantifier used.

Where parentheses differentiate with a character class is that a character class, as defined between square brackets, represents a single character whereas parentheses represent more than one character in a string.

Example:

(abc)+ matches **abcabc**123 (two matches), but does not match **ab123** or **bc123**

Before delving into how this is evaluated, I want to take a few steps back.

Start with

Match: (abc)
Replace: \1 <space>

Name	New Name
abcabc123	abc

This captures the first match of 'abc' in the string. How do I know it is the first set and not the second set, abc? I use a simple trick to determine the current position. I use dot characters after the expression, abc:

(abc..)

New Name
abcab

If the current position after the immediate capture of 'abc' is an 'a', the value returned from the first dot, and the second is a 'b', then this confirms that the capture was abc of abcabc123 because if it was the second set that was captured, the first character would have been a '1'.

Back to the original expression.

String = abcabc123

Match: (abc)+
Replace: \1

Name	New Name
abcabc123	abc

By placing the Greedy Quantifier outside of the expression, the RegEx Engine will repeat the evaluation of (abc) in as many iterations required to match against the string until no matches can be made. At this point, the string is said to be exhausted.

BRU will *only* match against a single match (under PCRE v1) even if there are two matches. The '+' Greedy Quantifier doesn't change this behaviour. A repeated evaluation of a capturing group will only capture the **last** iteration. In this example, the RegEx Engine discards the first match and retains and captures the second match.

Notes:

1. The dot Metacharacter trick does not always work. Adding additional characters to match against, may change the expression drastically enough to affect any validity of the results, so take care.

Metacharacters in Depth

() Parentheses – used for:

String = `abcabc123`

Match: `(abc)+`

Replace: `\1`

Name	New Name
abcabc123	abc

Analysis:

The expression, `abc`, of Capture Group 1 in the RegEx, matches against the first set of `'abc'` in the string. The `'+'` Greedy Quantifier repeats the expression (iteration) and a second match is found. Capture Group 1 = `'abcabc'`. The RegEx continues moving forward. Finding no further matches, only the last match is kept and the first discarded. Capture Group 1 = `'abc'`. The string is exhausted.

One thing I should mention is that trick of positioning that I employed before, will not work with a Quantifier outside of the expression. Here's why.

`(abc..)+`

New Name
abcab

Looks the same. Provides the same result indicating that it is matching against the first set..., hey, wait a minute, I thought with the addition of using the Quantifier it only matches against the last set?

With the expression in a Capture Group and the Quantifier outside, it creates a pattern for which the expression has to be matched against in order to have a successful match. When I added the two dot Metacharacters, I changed that pattern from:

`abc` to `abc + two more characters`

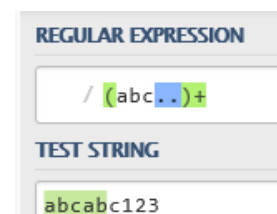
So each iteration has to begin with `abc` and end with two additional characters. When it begins to search the string, the RegEx Engine encounters the first set of:

`abcab` This is – `abc + a + b`

`..` and matches. Capture Group 1 = `abcab`. Current position is the 'last b' of `'abcab'`. After this, it continues to move forward and encounters,

`c123`

It is looking to match the first character 'a' of the pattern established as `'abc..'`. It tests the character 'c' of `'c123'` and failing to match the 'a', the RegEx fails, and the first and only match is retained. Capture Group 1 = `abcab`



Notes:

1. To avoid confusion I use the `+` sign above in blue to indicate concatenation, not as a Quantifier.

Metacharacters in Depth

() Parentheses – used for:

Therefore, the trick does not work with this example because it cannot accurately determine the current position of (abc)+ by adding the two dot characters, (abc..)+ because the pattern changed too much from the original pattern.

But you can analyze it in the same manner to verify the results.

Analysis:

1. abc is the pattern established.
2. The first iteration matches against the first set of 'abc' in the string, 'abcabc123'. Capture Group 1 = abc
3. The RegEx Engine moves forward with the second iteration and encounters, the second set, abc, and this is also captured. Capture Group 1 = abcabc.
4. The RegEx Engine moves forward to the 123 and tests for the pattern that begins with 'a' = false. Finding none, the last match of 'abc' is retained. Capture Group 1 = abc, the second match.

The screenshot shows a 'REGULAR EXPRESSION' field with the pattern `/ (abc)+` and a 'TEST STRING' field with the text `abcabc123`. To the right, the 'MATCH INFORMATION' section displays 'Match 1' with a 'Full match' of 'abcabc' at positions 0-6 and 'Group 1.' of 'abc' at positions 3-6. A blue arrow points from the match information to the second 'abc' in the test string.

In the above photos, the second one in particular, verifies that it is the second set of 'abc' that is captured in Capture Group 1 as indicated by the 3-6, which represents the character positions in the string. The second set of 'abc' begins at character position 3.

a	b	c	a	b	c	1	2	3
↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	7	8

If I change the RegEx to:

Match: `(abc)+(.*)`
 Replace: `\1 \2`

Name	New Name
abcabc123	abc 123

The second Capture Group, designated as Capture Group 2, holds the value of '123'. This indicates that after the capture of the second set of 'abc', because the RegEx has a second expression, `.*`, it moves forward, matching against and capturing the remainder of the string to EOL. Capture Group 2 = '123'. The RegEx Engine remains at EOL, the position after the last capture. String exhausted.

Capture Group 1 = abc

Capture Group 2 = 123

Metacharacters in Depth

() Parentheses – used for:

Under PCRE v2, the '+' Quantifier outside of the expression acts like a Global Switch, although not completely as I will illustrate later. It will match and retain all occurrences of the pattern established as 'abc'.

The same RegEx (abc)+ when v2 is enabled produces:

Match:	(abc)+	Name	New Name
replace:	\1	<input type="checkbox"/> abcabc123	abc123

I was expecting abcabc and why was the 123 captured?

Let's use the substitution Replace String syntax to explain this.

The match of the RegEx is in fact, abcabc. All occurrences of abc are matched and captured. It is the Replace String that is the problem. BRU displays the **matched string** as \1 or \$1 = abc, rather than the **entire match** of abcabc.

This accounts for the single set of abc in New Name.

The entire match including all instances of abc is represented using the substitute Replace String of \$0. BRU does not support a Replace String of \0. If you want to display the entire match, in this example using a Quantifier outside of an expression, you will need to use the \$0 syntax.

match = abcabc

Match:	(abc)+	Name	New Name
replace:	\$0 <space>	<input type="checkbox"/> abcabc123	abcabc 123

Now as to the explanation for the capturing of 123. It is *not* captured.

This was previously discussed in an earlier part of the book under the Global Switch topic. Under PCRE v1, only those parts of the string that are captured are retained and all other parts are excluded from New Name. Under PCRE v2, elements of the string such as those values that are captured will be **substituted** in a **copy** of the original string as New Name. This will occur even if those substituted values do not change the original elements of the string that have been replaced. The original string, with the exception of those substituted values, are retained and copied over to New Name.

Summary:

The original filename:	abcabc123
String matched	abc
Matched	abcabc (2 matches of 'abc' will substitute for the current value in the New Name string)
Not matched	123 (This is retained as part of the original filename and copied over to New Name)
New filename	abcabc123 (the replaced value of 'abcabc' and the retained value of '123')

Metacharacters in Depth

() Parentheses – used for:

BRU will only display the entire match using \$0, otherwise only the matched string of `abc` is displayed in New Name. While BRU only displayed `abc` originally using `\1`, the entire match of `abcabc` was actually substituted. That left `123` as the only part of the string that was not substituted, hence New Name became `abc123`.

`abcabc`123 2 matches of 'abc'. BRU using `\1` or `$1` only returns the first matched value of Capture Group 1.
`abcabc`123 2 matches of 'abc'. BRU using `$0` returns all matches in Capture Group 1 as a single value.

You may be better able to understand this if I change the string to:

String = now `abcabc`123

Match: `(abc)+`
 Replace: `$1 <space>`

Name	New Name
now <code>abcabc</code> 123	now <code>abc</code> 123

Using `$1`, only the matched string, `(abc)`, and not the entire match, `abcabc`, displays in New Name.

The matched elements of `abcabc` are substituted in the substring of `abcabc` even though this value is the same as the original. Using `$1`, only the first matched value is returned = '`abc`'. All other elements of the string that were not matched are retained and copied over to New Name. This includes, 'now' and '123'.

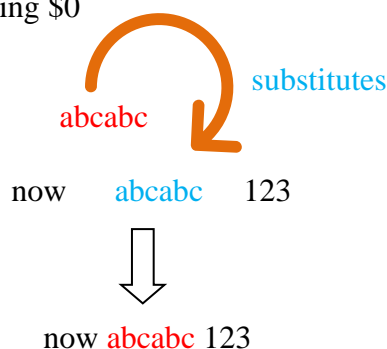
Match: `(abc)+`
 Replace: `$0 <space>`

Name	New Name
now <code>abcabc</code> 123	now <code>abcabc</code> 123

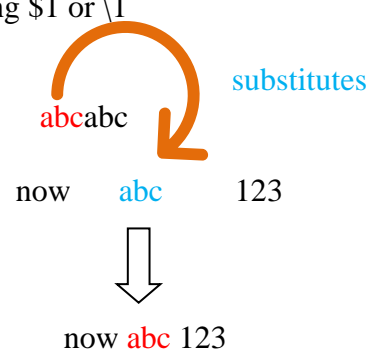
Using `$0`, the entire match of `(abcabc)` is displayed in New Name.

The matched elements of `abcabc` are substituted in the substring of `abcabc` even though this value is the same as the original. Using `$0`, both matches are returned as a single value = '`abcabc`'. All other elements of the string that were not matched are retained and copied over to New Name. This includes, 'now' and '123'. Comparison -

match: using `$0`



match: using `$1` or `\1`



Metacharacters in Depth

() Parentheses – used for:

Using the <space> in the Replace String is the needed change to provide New Name with a value, otherwise Name would remain the same and New Name would display no change.

Match:	(abc)+	Name	New Name
Replace:	\$0	<input type="checkbox"/> now abcabc123	now abcabc123

Now add the Global Switch to the RegEx in place of the Quantifier:

Match:	(abc)/g	Name	New Name
Replace:	\$1	<input type="checkbox"/> now abcabc123	now abc abc 123

Using the Global Switch in this manner without the Quantifier no longer requires \$0 to display all matches.

Whereas using the '+' Greedy Quantifier resulted in a single match made up of **abcabc**, using the Global Switch results in separate matches of 'abc' for each occurrence of the matched string, 'abc' within the same Capture Group.

In this example, Capture Group 1 = 'abc' 'abc'.

Now here is something interesting.

There is no Capture Group 2. Therefore, Capture Group 2 holds no value. If I were to express Capture Group 2 under PCRE v1, it would display as **Invalid** (invalid because it treats the backslash as a literal):

Match:	(abc)	Name	New Name
Replace:	\2	<input type="checkbox"/> now abcabc123	\2

Under PCRE v2, using the Global Switch, because Capture Group 2 holds no value, there is no match to display in New Name, but this doesn't change the original parts of the string that were retained. Displays as:

Match:	(abc)/g	Name	New Name
Replace:	\2	<input type="checkbox"/> now abcabc123	now 123

No **Invalid** flag.

Backslash is no longer part of NewName because it was not part of the original string and therefore, not copied over.

Remember that this is only what is **displayed**. Capture Group 1 still holds the matched value of 'abc' 'abc'

Metacharacters in Depth

() Parentheses – used for:

Notes:

1. **Invalid** flag displays as **red** in New Name. This indicates the problem has to do with the Replace String of Capture Group 2. Some errors in expressions won't display as red, and remain in black and white (no change in New Name) possibly meaning that the syntax is in error.

The reason that it displays in **red** is because of the non-existent Capture Group. New Name interprets both the backslash and the Capture Group Reference number as a literal. The **Invalid** flag is because the backslash is an illegal character in a Windows filename.

v2 RegEx doesn't display the flag because any changed or matched elements are substituted for the original elements in a copied New Name string. All other elements are retained in the copy. If a Capture Group is non-existent, it is ignored because it has no values to replace and nothing in which to copy over.

2. Under v1, BRU matches against all possible matches, but only the value of the **first** match is retained, with the exception of a Quantifier placed **outside** of a Capture Group. Only the match of the **last** iteration is retained. Under v2, BRU matches against all possible matches using a Greedy Quantifier **outside** of the Capture Group. All matched values are retained **in** the Capture Group, but it is dependent on the Replace String which values are returned. The backreferences, e.g., \1 or \$1 will return the **first** matched value of Capture Group 1 = **abcabc**123 (First match in yellow, secondary match in blue), while \$0 returns **all** matched values as a single value = **abcabc**123. Under v2, using the Global Switch, each match is retained separately **within** the Capture Group. They can be distinguished using a <space> delimiter but can't be referenced individually.

String = abcabc123

Match: (abc)/g
Replace: \1 <space>

Match the string, abc globally
Return the two matches of abc

Capture Group 1 first match

Capture Group 1 second match

abc

<space>

abc

<space> 123

I can't backreference the individual matched values of 'abc'

Changing the Match to:

(abc)(123)/g

This matches the second occurrence of abc only: abc**abc123** because the pattern becomes 'abc123' to match against and there is only one occurrence in the string.

3. This is my understanding of how this works. Remember, that this all new to me as well. I could have to revise this in a future edition.

Metacharacters in Depth

\d Matches any single numeric digit – equivalent to a character class of [0-9]
This can be used inside a character class []

Example:

`[\d.-]` (backslash, dot, followed by a <hyphen>) matches any numeric digit, dot or minus sign character.

A common example is to match against dates in a string.

String = 01-10-2021

By now you know of patterns. RegEx is based on pattern recognition. This string is an example of a simple pattern of a month, day of month and year expressed as 4 digits using a <hyphen> as a delimiter. The pattern can be expressed as:

##-##-####

When you know the pattern, you can design a RegEx that can match the pattern.

- (1) Where # = a numeric digit, substitute that with \d
- (2) The <hyphen> character can always be included or left out of the Replace String so I will decide to exclude it in the RegEx. I can do this by capturing those parts of the string I wish to retain and ignore the rest.

Now when I say ‘ignore the rest’, the <hyphens> are still a part of the pattern so I must include it in the RegEx BUT I won’t capture them.

So my RegEx becomes:

`(\d\d)-(\d\d)-(\d\d\d\d)`

If I did not include the <hyphen>, the RegEx would fail if I just had:

`(\d\d)(\d\d)(\d\d\d\d)`

1. Matches against the first two numeric digits
Current position = right after the last match of the ‘1’ of ‘01’
2. Fails to match against the next numeric digit. RegEx engine tests the character for a numeric digit = <hyphen> = **false**.
RegEx Fails.

Capture Group 1= 01

See?

So although I don’t want to include the <hyphen>, I still must recognize it as part of the pattern to match the string.

Metacharacters in Depth

`\d` Matches any single numeric digit cont.

Each Metacharacter corresponds to a string character. The following is the completed process.

String	01-10-2021									
Pattern	#	#	-	#	#	-	#	#	#	#
Pattern	0	1	-	1	0	-	2	0	2	1
RegEx	(\d	\d)	-	(\d	\d)	-	(\d	\d	\d	\d)
Captured	(0	1)		(1	0)		(2	0	2	1)
Value	01			10			2021			
			↑			↑				
			Not Captured			Not Captured				

Match: `(\d\d)-(\d\d)-(\d\d\d\d)`
 Replace: `Month = \1 Day = \2 Year = \3`

Name	New Name
<input type="checkbox"/> 01-10-2021	Month = 01 Day = 10 Year = 2021

This can also be expressed as:

.. to obtain the same result.

Match: `(\d{2})-(\d{2})-(\d{4})`
 Replace: `Month = \1 Day = \2 Year = \3`

Name	New Name
<input type="checkbox"/> 01-10-2021	Month = 01 Day = 10 Year = 2021

where:

{n} = number of iterations

For example, using Capture Group 3 of `(\d{4})`

{4} = repeat the previous expression or sub-expression, 4 times (iterations).

`\d {4}` = repeat the match for a numeric digit, 4 times (iterations).

2021 = `\d` first iteration = 2
 `\d` second iteration = 0
 `\d` third iteration = 2
 `\d` forth iteration = 1

Capture Group 3 = 2021

Metacharacters in Depth

\D Matches any single non-numeric digit – equivalent to the negated class `[^0-9]`. Negated `\d`. This can be used inside a character class `[]`

The `\D` is the negated (opposite) of the `\d`. The above uses the equivalent, `[^0-9]`, where the `^` carrot sign in a class negates the class thus `[^0-9]` is the same thing as saying that `\D` represents any character that is NOT 0-9 (numeric digits).

Some of the Metacharacters do have a ‘negated’ counterpart. The `\s` and `\S` Metacharacters previously discussed are an example of this.

string = `two apples in 1 basket -02`

If I want to isolate the second part of the string, where it begins with a numeric, from the first section, I can use this:

pattern: `text <space> text <space> text <space> # <space> text <space> <hyphen> ##`

(\D+)
two apples in <space>

(.*)
1 basket -02

You do not have to account for each item in the pattern separately for a successful match to occur, just as long as they are accounted for in the RegEx.

In the above RegEx, the `\D` will search to match against any character that is not a numeric digit. The ‘+’ Greedy Quantifier will repeat the `\D` for as many iterations as required until EOL or the string is exhausted, whichever comes first.

The RegEx continues to match until the encounter with the ‘1’. The tested character fails to match, at which point, the string is exhausted because no more matches can be made by this sub-expression.

Capture Group 1 = two apples in <space>

The next sub-expression is evaluated. The dot Metacharacter captures any character. The Greedy Quantifier, `*`, repeats the match until EOL, capturing the remainder of the string.

Capture Group 2 = 1 basket -02

Match: `(\D+)(.*)`
 Replace: `Group 1 = \1 Group 2 = \2`

Name	New Name
two apples in 1 basket 02	Group 1 = two apples in Group 2 = 1 basket 02

Metacharacters in Depth

Anchor Metacharacters

- Placed at the beginning of a pattern to **require any match to occur at the beginning of the input string**. In BRU the string input will always be a file or folder name. This is the BOL or Beginning of String. The BOL is the position before the first character of the string. It is not a character but a position and has zero length.

Example:

`^abc` matches 'abc123' but not '123abc' because none of the characters, 'a b c', appear at the beginning of the string input.

Match:	<code>(^abc)</code>							
Replace:	<code>\1</code>							
		<table border="1"> <thead> <tr> <th>Name</th> <th>New Name</th> </tr> </thead> <tbody> <tr> <td>abc123</td> <td>abc</td> </tr> <tr> <td>123abc</td> <td>123abc</td> </tr> </tbody> </table>	Name	New Name	abc123	abc	123abc	123abc
Name	New Name							
abc123	abc							
123abc	123abc							

In this example, only the first string matches because it begins with 'abc'.

as opposed to:

Match:	<code>(abc)</code>							
Replace:	<code>\1</code>							
		<table border="1"> <thead> <tr> <th>Name</th> <th>New Name</th> </tr> </thead> <tbody> <tr> <td>abc123</td> <td>abc</td> </tr> <tr> <td>123abc</td> <td>abc</td> </tr> </tbody> </table>	Name	New Name	abc123	abc	123abc	abc
Name	New Name							
abc123	abc							
123abc	abc							

With this example, the match of 'abc' can occur anywhere in the string so both strings match.

Another Example:

`^fun` matches first 'fun' only in "fun function"

Match:	<code>(^fun)</code>					
Replace:	<code>\1</code>					
		<table border="1"> <thead> <tr> <th>Name</th> <th>New Name</th> </tr> </thead> <tbody> <tr> <td>fun function</td> <td>fun</td> </tr> </tbody> </table>	Name	New Name	fun function	fun
Name	New Name					
fun function	fun					

.. and how can this be verified, other than with the knowledge gained from reading this book? ☺

By employing that trick I demonstrated earlier. It wouldn't work, if you recall with a Quantifier outside of the expression, because it changed the pattern too drastically, but with this example, it will.

Match: `(^fun..)`
 Replace: `\1`

Name	New Name
fun function	fun f

With the two added characters, it captures the <space> after the first word, 'fun' and the following 'f' as the beginning of the word, 'function', thus verifying that (^fun) captured the first word in the string.



Metacharacters in Depth

- \$** Placed at the end of a pattern to **require any match to occur at the end of the input string**. Use this for restricting matches to characters at the end of a line. In BRU the string input will always be a file or folder name. This is the EOL or End of String. The EOL is the position just after the last character of the string and before the file extension if any. It is not a character but a position and has zero length.

Example:

abc\$ matches '123abc' but not 'abc123' because none of the characters, ' a b c', appear at the end of the string input.

Match: abc\$
Replace: \1


Name	New Name
 abc123	abc123
 123abc	abc

In this example, only the second string matches because it ends with 'abc'.

Another Example:

on\$ matches last 'on' only, in 'onto my fun function'

Match: (on\$)
Replace: \1

Name	New Name
 onto my fun function	on

Notes:

1. **^** and **\$** are referred to as anchors because they fall at either end of the pattern.
2. **end\$** only matches "end" when it's the last word on a line, and **^end** only matches "end" when it's the first word on a line.
3. Anchors return a **null** value.

Metacharacters in Depth





Both the Beginning of Line (BOL) Anchor, ^, and the End of Line (EOL) Anchor, \$, can be used together:

Example:

```
^abc$
```

Match: (^abc\$)

Replace: \1 <space>

Name	New Name
 abc	abc
 123abc	123abc
 abc123	abc123
 now abcabc123	now abcabc123

Matches only 'abc'

In order for a match to occur there must be no other characters before or after in the string input. In other words, this produces an exact match.

Metacharacters in Depth

\w Matches any single Word Character.

A Word Character is an alpha numeric character or an `_` (underscore). Equivalent to `[a-zA-Z0-9_]` that can be used to form words. This can be used inside a character class `[]`.

Example:


`\w` would match against any of these single characters, a, b, c, d, e, and f in the string, 'abc def', however the pattern must account for the `<space>` character.

Example:

String = This is a word 123 Another word

Match: `(\w*) (\w*) (\w*) (\w*)\s(d*)`

Replace: `|1 |2 |3 |4 |5`

Name	New Name
 This is a word 123 Another word	This is a word 123

Analysis:

- Each `\w*` will capture a word. This is done by using the `\w` to match against a single Word Character and then making it Greedy using a Greedy Quantifier that captures to the next non-Word Character, the `<space>` character.
- `<spaces>` are not Word Characters, therefore they must be accounted for but not captured (at least I don't need them to be captured). I use two representations for the `<spaces>` above just for fun and for purposes of demonstrating there is more than one way to specify a `<space>`. One, of course, is the `<space>` character found in between some of the Capture Groups, and the other is using the Whitespace Metacharacter, `\s`.
- I also capture the numeric digits in the string using, `\d*`

Notes:

- This is a pretty good RegEx, no backtracking at all, but it does use too many Capture Groups, 5 out of 9 available. If I wanted to parse the entire string, I would need 7 Capture Groups total using this method. Here is the debugging output taken from Regex Buddy showing no backtracking:

```

Beginning match attempt at character 0
1 This
2 This
3 This
4 This is
5 This is
6 This is
7 This is a
8 This is a
9 This is a
10 This is a word
11 This is a word
12 This is a word
13 This is a word 123
14 This is a word 123
Match found in 14 steps

```

Metacharacters in Depth

\W Matches any single non-Word Character


Examples of non-Word Characters are whitespace (<space>), beginning and end of strings (BOL, EOL), punctuation and symbols. Negated \w. This can be used inside a character class [].

This is the negated (opposite) counterpart to the Word Metacharacter, \w.

String = This is a word 123 Another word

Match: (.*)(\W..)(\W..)(\w*)

Replace: |1|2|3|4

Name	New Name
 This is a word 123 Another word	This is a word

Analysis:

- | | | |
|-------|--|--|
| 1. .* | Starts out by matching the entire string to EOL. | Capture Group 1 = <entire string> |
| 2. \W | Match against a non-Word Character. Already at <u>EOL</u> , so it backtracks to the <space> following 'Another'. | Capture Group 2 = <space>
Changes Capture Group 1 = This is a word 123 Another |
| 3. .. | Match against any two characters. Moves forward and matches 'wo' of the second occurrence of 'word'. | Capture Group 2 = <space>wo |
| 4. \W | Match against a non-Word Character. This is the <space> after 'is'. | Capture Group3 = <space>
Changes Capture Group 2 = <space> is
Changes Capture Group 1 = This |

At this point, it is moving backwards testing each character for the following pattern established by, **.*** **\W..** **\W**

.*, character followed by a **\W**, <space> followed by any two characters, **..** followed by a **\W**, <space>

- Backtracks to test current character against the sub expression, **(.*)**. This will match of course.
- Moving forward to test for the **<space>** (**\W**) in the next sub-expression. If this matches, continue.
- Moving forward again to test for the two characters, **..**, part of the same sub-expression. This will match of course.
- and finally moving forward to test for the **<space>** (**\W**) in the next sub-expression.

If this fails, it backtracks one character to repeat the whole process again.

The only test that matches in the string is currently:

.*	.*	= 's' of 'This'
followed by a <space>	\W	= the <space> after 'This'
followed by two characters	= 'is'
followed by a <space>	\W	= the <space> after 'is'

Capture Group 3 = <space> after 'is'

Metacharacters in Depth

\W Matches any single non-Word Character cont.

Notes:

1. Capture Group 2 changes its value.
2. Capture Group 2's value prior = <space>wo.
3. Because Capture Group 3 moved backwards (backtracking) past the value captured by Capture Group 2, Capture Group 2 was forced to give up its value, and proceed to Capture the <space> + two characters that **precedes** Capture Group 3's capture of the <space> **after** 'is'.

4. Capture Group 1 changes its value.

Correspondingly, Capture Group 1 was also forced to give up its value of, 'This is a word 123 Another', because Capture Group 3 captured the <space> after 'is', that is included as part of the value of Capture Group 1, and Capture Group 2 captured the '<space> is', that was also included in the value of Capture Group 1.

5. Capture Group 1's value is based on the pattern of **.*** meaning any characters. Capture Group 1 changes to include all characters prior to the value of Capture Group 2 = This.
6. In (a), **.*** will match against any character. Similarly, in (c) the **..** will match against any two characters.





Analysis continues.

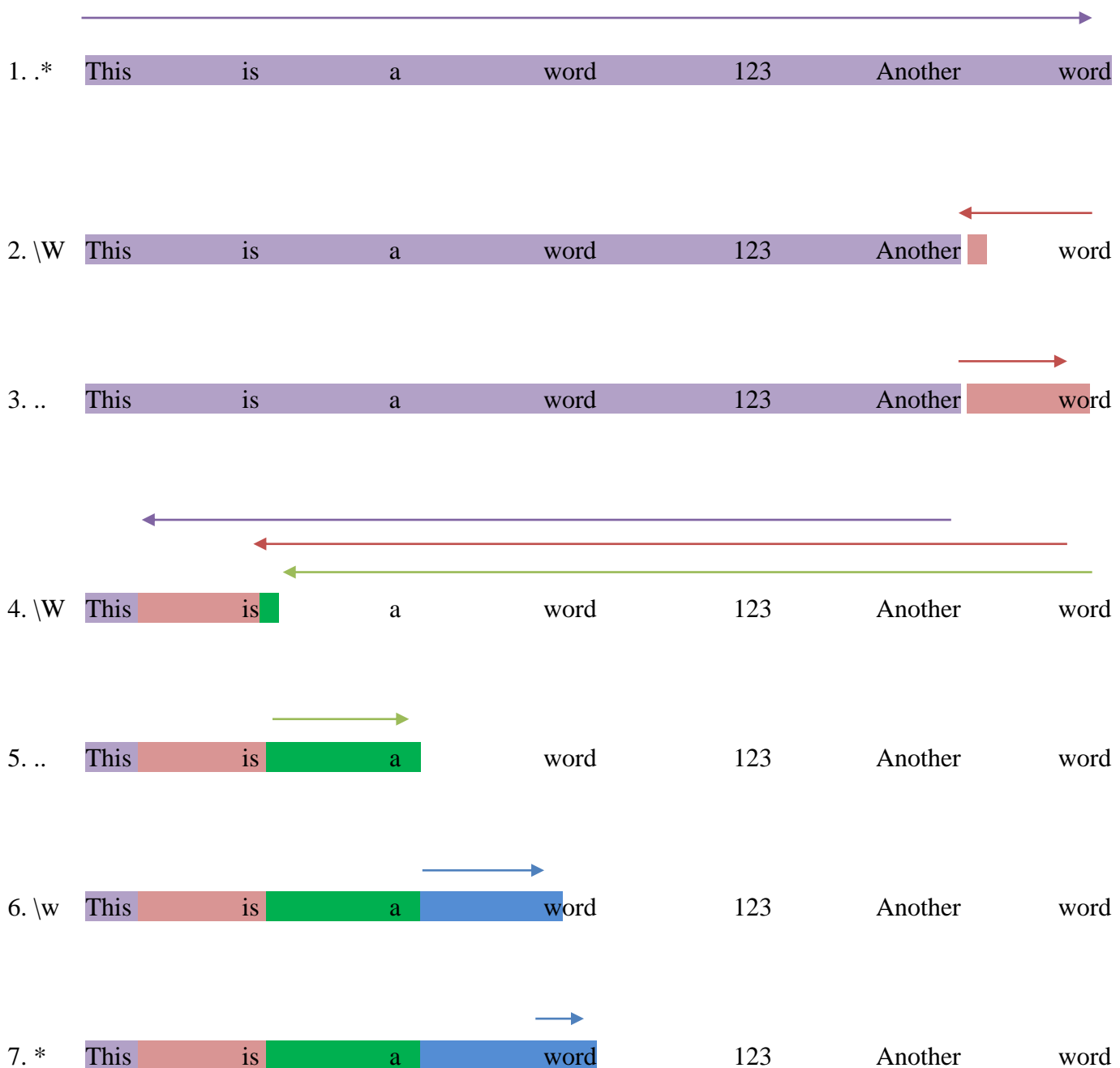
- | | | |
|---------|---|------------------------------------|
| 5. .. | Match against any two characters.
Moves forward and matches 'a<space>' preceding the first occurrence of 'word'. | Capture Group 3 = <space> a<space> |
| 6. (\w) | Match against a Word Character.
Matches against the 'w' of the first occurrence of 'word' in the string. | Capture Group 4 = w |
| 7. * | Make it Greedy.
Matches against the first occurrence of 'word' in the string. | Capture Group 4 = word |

Metacharacters in Depth

\W Matches any single non-Word Character cont.

Here is a more visual view of the analysis.

Capture Group 1 =  Violet
 Capture Group 2 =  Red
 Capture Group 3 =  Green
 Capture Group 4 =  Blue



Metacharacters in Depth

\W Matches any single non-Word Character cont.

Plenty of backtracking in this example, not my best work, but it does work.

```

Beginning match attempt at character 0
1 This is a word 123 Another word
2 This is a word 123 Another word
3 This is a word 123 Another word backtrack
4 This is a word 123 Another word backtrack
5 This is a word 123 Another word
6 This is a word 123 Another word
7 This is a word 123 Another word backtrack
8 This is a word 123 Another word backtrack
9 This is a word 123 Another wo
10 This is a word 123 Another wo
11 This is a word 123 Another wo backtrack
12 This is a word 123 Another wo backtrack
13 This is a word 123 Another w
14 This is a word 123 Another w
15 This is a word 123 Another w backtrack
16 This is a word 123 Another w backtrack
17 This is a word 123 Another
18 This is a word 123 Another
19 This is a word 123 Another backtrack
20 This is a word 123 Another backtrack
21 This is a word 123 Another
22 This is a word 123 Another
23 This is a word 123 Another
24 This is a word 123 Another w
25 This is a word 123 Another wo
26 This is a word 123 Another wo
27 This is a word 123 Another wo backtrack
28 This is a word 123 Another wo backtrack
29 This is a word 123 Another backtrack
30 This is a word 123 Anothe
31 This is a word 123 Anothe
32 This is a word 123 Anothe backtrack
33 This is a word 123 Anothe backtrack
34 This is a word 123 Anoth
35 This is a word 123 Anoth
36 This is a word 123 Anoth backtrack

```

```

106 This is a
107 This is a w
108 This is a wo
109 This is a wo
110 This is a wo backtrack
111 This is a wo backtrack
112 This is a backtrack
113 This is
114 This is
115 This is backtrack
116 This is backtrack
117 This is
118 This is
119 This is
120 This is a
121 This is a
122 This is a
123 This is a backtrack
124 This is a backtrack
125 This is backtrack
126 This i
127 This i
128 This i backtrack
129 This i backtrack
130 This
131 This
132 This backtrack
133 This backtrack
134 This
135 This
136 This
137 This i
138 This is
139 This is
140 This is
141 This is
Match found in 141 steps

```

Metacharacters in Depth

`\b` A Word Boundary

Quick Review:

Word Characters:

A word is made up of Word Characters. A Word Character is an alpha numeric character that also includes the underscore character. If you recall the discussion on `\w`, the Word Metacharacter, it includes the class of `[A-Za-z0-9_]`.

Whitespace:

Whitespace is defined as `<space>` characters. Whitespace are non-Word Characters.

non-Word Characters:

non-Word Characters include Whitespace, EOL, BOL, and any characters that are part of the negated class, `[^A-Za-z0-9_]`, e.g., punctuation marks, symbols, #, \$, !, etc.

So what is a Word Boundary?

I have seen all manner of explanations regarding the Word Boundary. Here are a couple of them:

‘A Word Boundary is a position that is either preceded by a Word Character and not followed by one or followed by a Word Character and not preceded by one.’

‘Word Boundaries match before the first and after the last Word Characters in a string, as well as any place where before it is a Word Character or non-Word Character, and after it is the opposite.’

These explanations are certainly not wrong, but they do seem complex, and perhaps that is because Word Boundaries are complex. In my research, I have found this topic to be one of the most difficult endeavors of this volume to date. And it is not just myself, but many others who are perplexed over what appears seemingly and deceptively an easy subject matter.

I will ‘attempt’ to simplify it.

A Word Boundary, `\b`, is matched if it occurs...

- before, if used at the beginning of the word, `\btest`
- or after, if used at the end of a word, `test\b`
- or both, if used on either side of the word, `\btest\b`

but it is not that simple.

Metacharacters in Depth

`\b` A Word Boundary cont.

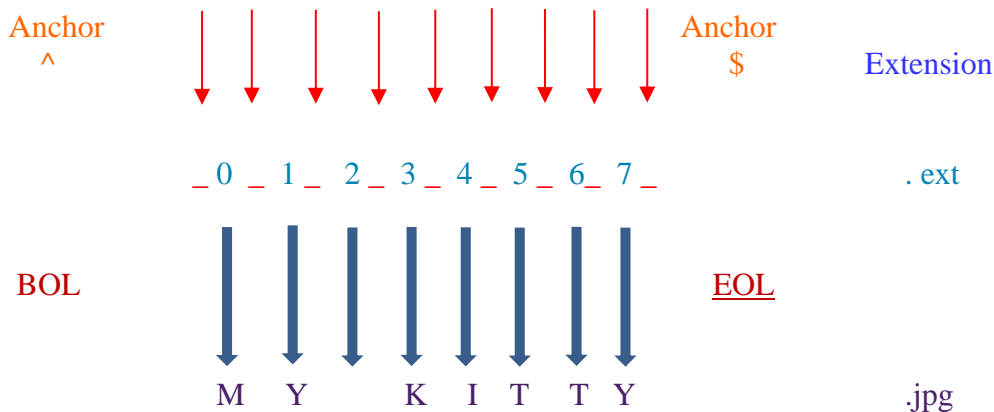
The RegEx Engine treats a word as a sequence of one or more Word Characters, `\w`, as defined by the class, `[A-Za-z0-9_]`. When testing a character, the RegEx Engine cannot perceive whole words, it can only perceive the character preceding and the character following. You can think of this as standing at a street corner and you want to walk across the street. So what do you do? You look left and then you look right to determine if it is safe to move forward. Your perspective, however, is limited to what you can see on either side. The RegEx Engine also has a perspective. From the current position, It can examine the character to the left and the character to the right.

When discussing Word Boundaries, the focus is not on the character so much as it is on the position. Please don't misunderstand me. The characters still determine a match or not, but those characters are to the left and to the right of the current position and that position of the Word Boundary is between the characters, and not necessarily positioned on the characters themselves.

In Volume II, I discuss briefly Character Position vs Movement Position. Note that I updated the diagram below to reflect movement that occurs between the BOL and EOL, something that was omitted from the original Volume II diagram.

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

String = MY KITT.Y.jpg



The RegEx Engine can move between the characters when searching to match against Word Boundaries.

Word Boundary Metacharacters are Zero-Length Assertions. They are called Zero-Length because they match without consuming any characters, meaning they do not retain the values captured. The match is an 'assertion' that what has been searched for has been found. This will be explained in more detail under the Lookarounds discussion.

A Word Boundary uses a set of rules by which to match:

`\b` asserts the position at a word boundary defined as: `(^\w|\w$|\W\w|\w\W)`

Diagram illustrating the definition of `\b` using the word "this":

`this` `this` `this` `this`
 `or` `or` `or`
 ↓ ↓ ↓
 `(^\w|\w$|\W\w|\w\W)`

Still confused? Read on.

Metacharacters in Depth

\b A Word Boundary cont.

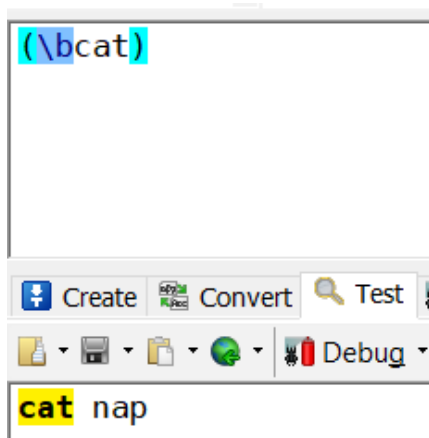
The Rules:

A Word Boundary matches ...

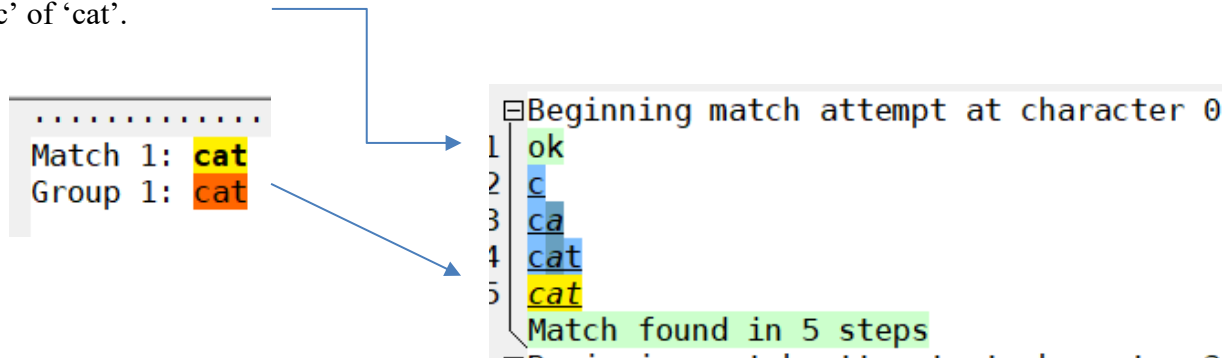
^  \w at the beginning of string (a Word Character preceded by BOL)

String = cat nap

e.g. \bcat




The Word Boundary matches between the position of the non-Word Character, BOL, and the Word Character, 'c' of 'cat'.



BOL (Beginning of Line) ^, is a non-Word Character because it falls into the negated class of a non-Word Character, `\W`, as defined by `[^A-Za-z0-9_]`. The alpha character, 'c', is a Word Character because it falls into the class of a Word Character, `\w`, as defined by `[A-Za-z0-9_]`.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters.

Metacharacters in Depth

\b A Word Boundary cont.

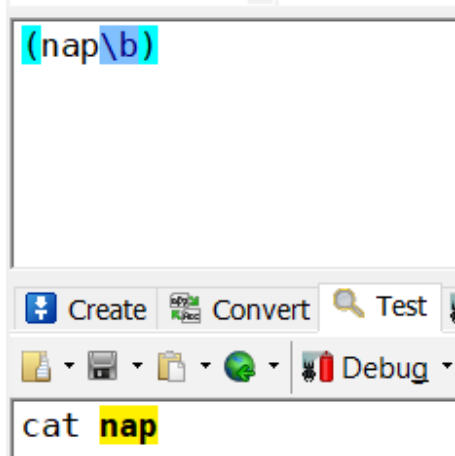
The Rules:

A Word Boundary matches ... cont.

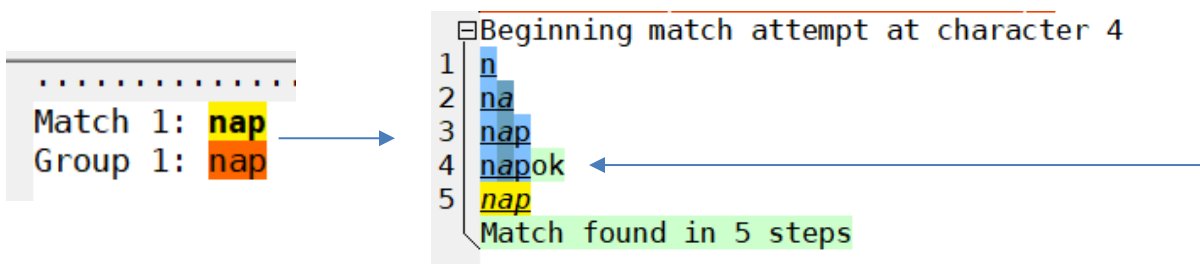
\w  **\$** at the end of a string (a Word Character followed by EOL)

String = cat nap

e.g. nap\b




The Word Boundary matches between the position of the Word Character, 'p' of 'nap', and the non-Word Character, EOL.



EOL (End of Line) **\$**, is a non-Word Character because it falls into the negated class of a non-Word Character, **\W**, as defined by `[^A-Za-z0-9_]`. The alpha character, 'p', is a Word Character because it falls into the class of a Word Character, **\w**, as defined by `[A-Za-z0-9_]`.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters.

Metacharacters in Depth

`\b` A Word Boundary cont.

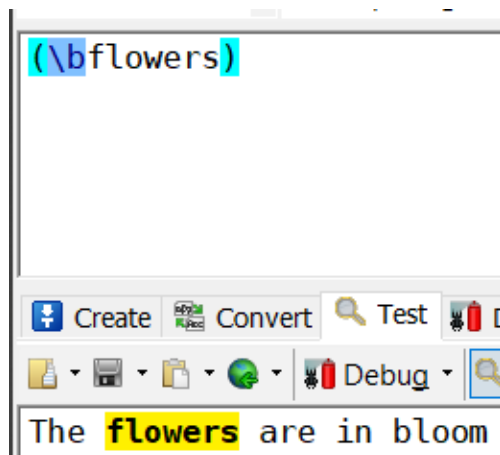
The Rules:

A Word Boundary matches ... cont.

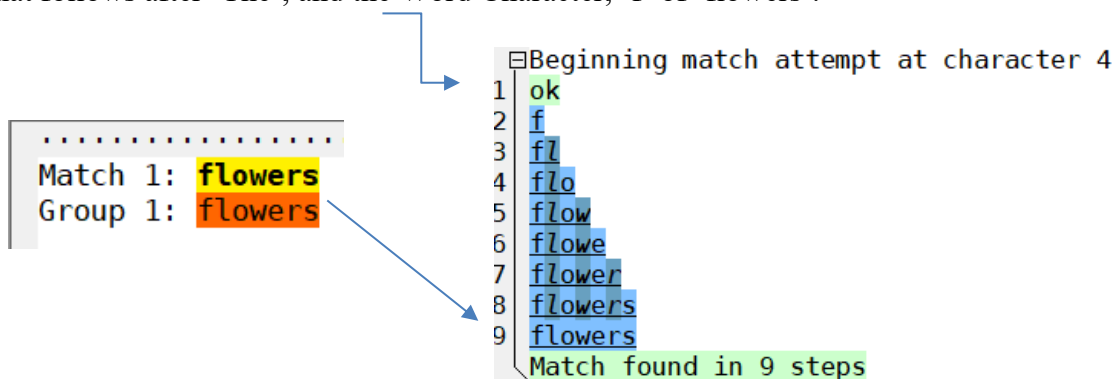
`\W`  `\w` a position between a non-Word Character and a Word Character (beginning of word within string)

String = The flowers are in bloom

e.g. `\bflowers`




The Word Boundary matches between the position of the non-Word Character, `<space>`, a Whitespace Metacharacter that follows after 'The', and the Word Character, 'f' of 'flowers'.



The alpha character, 'f', is a Word Character because it falls into the class of a Word Character, `\w`, as defined by `[A-Za-z0-9_]`. The Whitespace character, `<space>`, is a non-Word Character because it falls into the negated class of a non-Word Character, `\W`, as defined by `[^A-Za-z0-9_]`.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters.

Metacharacters in Depth

`\b` A Word Boundary cont.

The Rules:

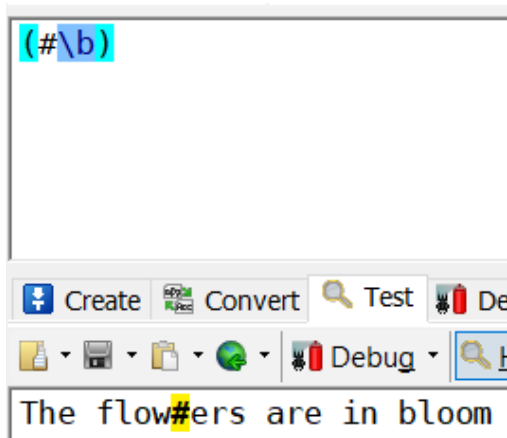
A Word Boundary matches ... cont.

`\W`  `\w` a position between a non-Word Character and a Word Character (beginning of word within string) cont.

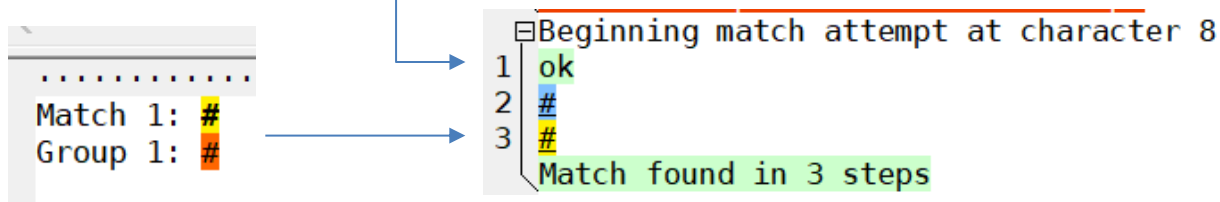
This can also apply when matching non-Word Characters including punctuation and symbol characters that appear within the text of the string, even within the same word.

String = The flow#ers are in bloom

e.g. `#\b`




The Word Boundary matches between the position of the non-Word Character, #, the Numeric Sign, and the Word Character, 'e' of 'flow#ers'.



The Numeric Sign, '#', is a non-Word Character because it falls into the negated class of a non-Word Character, `\W`, as defined by `[^A-Za-z0-9_]`. The alpha character, 'e', is a Word Character because it falls into the class of a Word Character, `\w`, as defined by `[A-Za-z0-9_]`.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters.

Metacharacters in Depth

`\b` A Word Boundary cont.

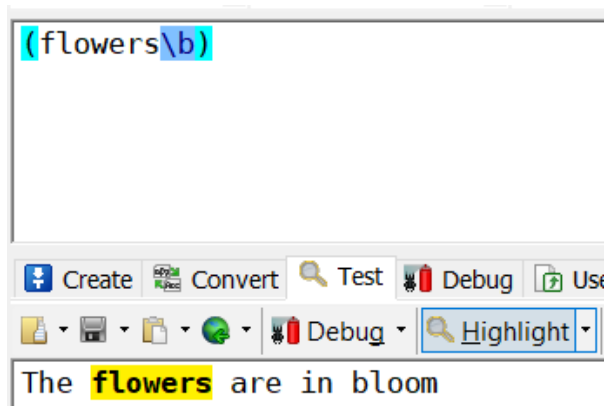
The Rules:

A Word Boundary matches ... cont.

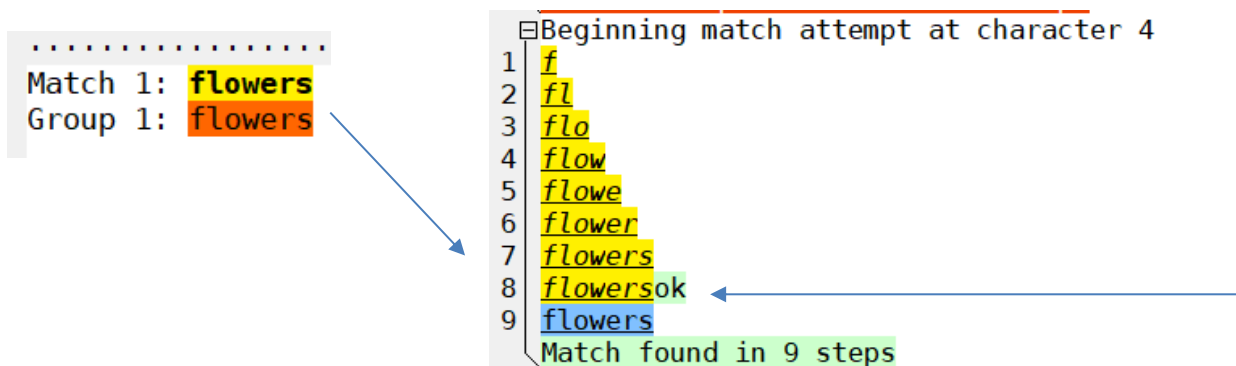
`\w`  `\W` a position between a Word Character and a non-Word Character (end of word within string)

String = The flowers are in bloom

e.g. flowers**\b**




The Word Boundary matches between the position of the Word Character, 's' of 'flowers', and the non-Word Character, <space>, a Whitespace Metacharacter, that follows.



The alpha character, 's', is a Word Character because it falls into the class of a Word Character, `\w`, as defined by `[A-Za-z0-9_]`. The Whitespace character, <space>, is a non-Word Character because it falls into the negated class of a non-Word Character, `\W`, as defined by `[^A-Za-z0-9_]`.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters.

Metacharacters in Depth

\b A Word Boundary cont.

The Rules:

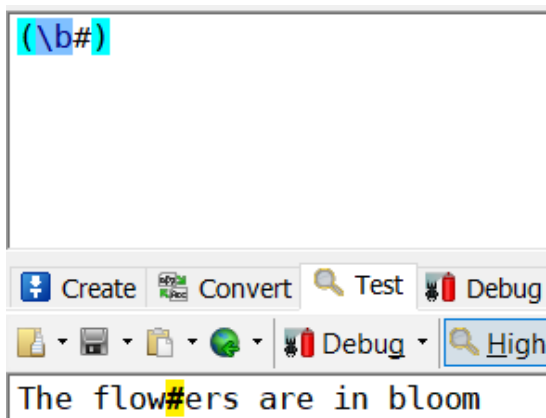
A Word Boundary matches ... cont.

\w  **\W** a position between a Word Character and a non-Word Character (end of word within string) cont.

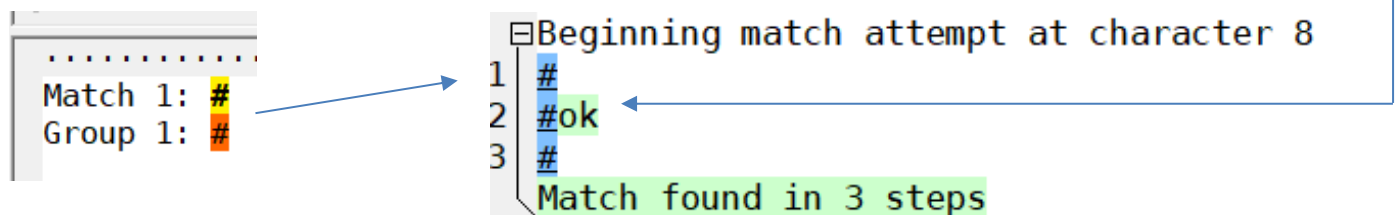
This can also apply when matching non-Word Characters including punctuation and symbol characters that appear within the text of the string, even within the same word.

String = The flow#ers are in bloom

e.g. `\b#`




The Match is made because the Word Boundary is between the position of the Word Character, 'w' of 'flow#ers' and the non-Word Character, '#', the Numeric Sign, of 'flow#ers'.



The alpha character, 'w', is a Word Character because it falls into the class of a Word Character, `\w`, as defined by `[A-Za-z0-9_]`. The Numeric Sign is a non-Word Character because it does not fall into the class of a Word Character as defined by `[A-Za-z0-9_]`.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters.

Metacharacters in Depth

`\b` A Word Boundary cont.

This island is beautiful

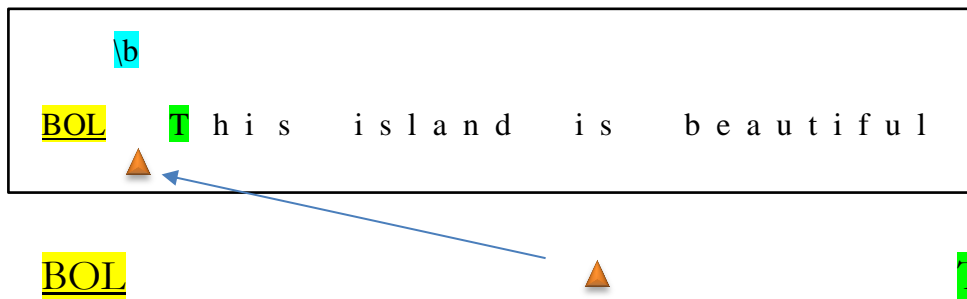
It may be easier to think of it this way.

When testing for a Word Boundary, it is the position between characters that is matched, not the string character. When testing for literal characters, the string character is matched. Let's take as an example,

String = This island is beautiful

Match: `\bis\b`

This island is beautiful



looks to the left for a non-Word Character
= true

position

looks to the right for a Word Character
= true

`\b` is matched.

The RegEx Engine advances to the character, the 'T' of 'This' and tests for the literal string character 'i' of 'is'.

T h i s i s l a n d i s b e a u t i f u l

T = i = false.

The RegEx Engine continues to search the string for a Word Boundary.

Notes:

1. The  icon used in these pages represents the Word Boundary position between characters currently being tested.

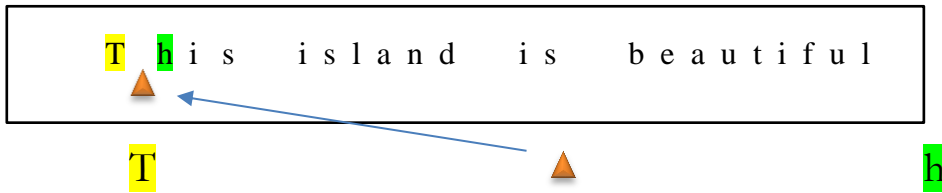
Metacharacters in Depth

`\b` A Word Boundary cont.

This island is beautiful cont.

Match: `\bis\b`

`\b` cannot match at the position between the T and the h.



looks to the left for a non-Word Character

position

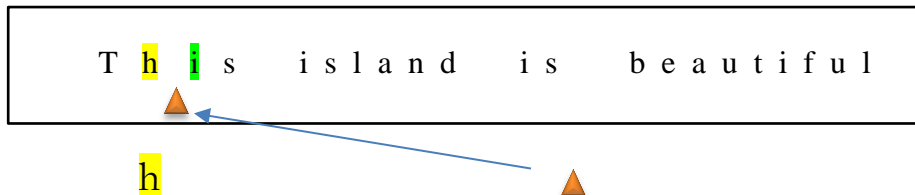
looks to the right for a Word Character

= false

= true

Doesn't match because there is a word character to the left of the tested position.

It cannot match between the h and the i either,



looks to the left for a non-Word Character

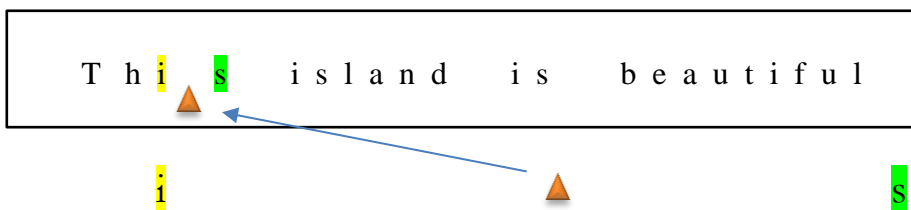
position

looks to the right for a Word Character

= false

= true

.. and neither between the i and the s.



looks to the left for a non-Word Character

position

looks to the right for a Word Character

= false

= true

Metacharacters in Depth

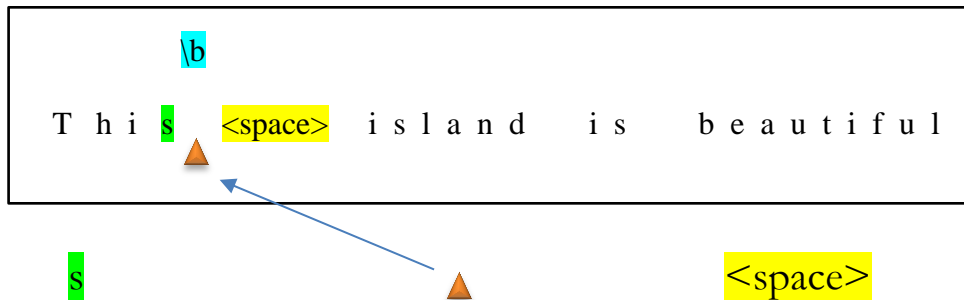
`\b` A Word Boundary cont.

This island is beautiful cont.

Match: `\bis\b`

`\b` does match at the position between the `s` and the `<space>` but fails when testing for the string characters.

Up to now matches have been attempted using the rule of a non-Word Character, `\W`, followed by a Word Character, `\w`. However, the RegEx Engine also attempts to match for a Word Character, `\w`, followed by a non-Word Character, `\W`, i.e., 's' of 'this' and the following `<space>`.



looks to the left for a Word Character

position

looks to the right for a non-Word Character

= true

= true

`\b` is matched.

The RegEx Engine advances to the `<space>` character following 'this', and tests for the literal string character 'i' of 'is'.

T h i s `<space>` i s l a n d i s b e a u t i f u l

`<space>` = i = false

The RegEx Engine continues to search the string for a Word Boundary.

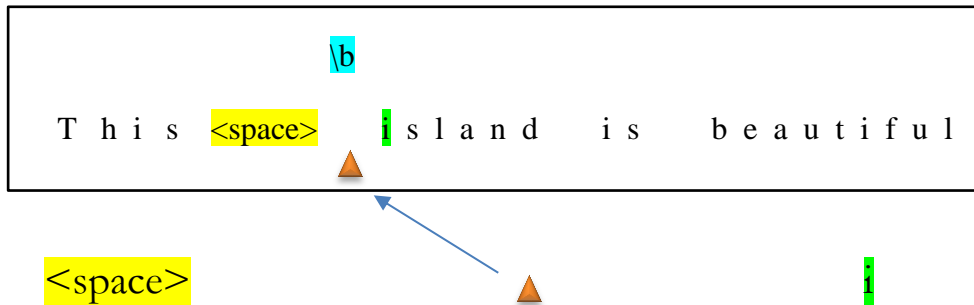
Metacharacters in Depth

`\b` A Word Boundary cont.

This island is beautiful cont.

Match: `\bis\b`

The Word Boundary matches between the `<space>` character that follows after 'This' and the 'i' of 'island'.



looks to the left for a non-Word Character

position

looks to the right for a Word Character

= true

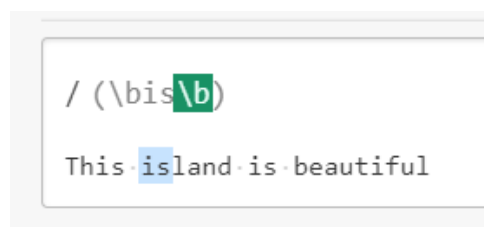
= true

`\b` is matched.

The RegEx Engine advances to the character, the 'i' of 'island' and tests for the literal string characters 'i' and the 's' of 'is'.



Matches.



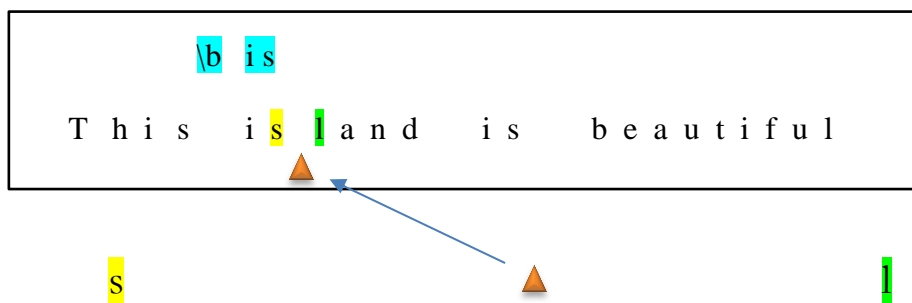
Metacharacters in Depth

`\b` A Word Boundary cont.

This island is beautiful cont.

Match: `\bis\b`

Looking to match against the final `\b` in the RegEx but fails to match.



looks to the left for a non-Word Character position looks to the right for a Word Character

= false

= true

`\b` doesn't match because there is a word character to the left of the current position.

The RegEx Engine continues to search the string for a Word Boundary.

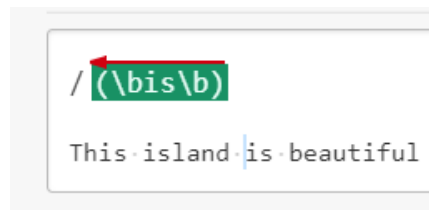
Metacharacters in Depth

`\b` A Word Boundary cont.

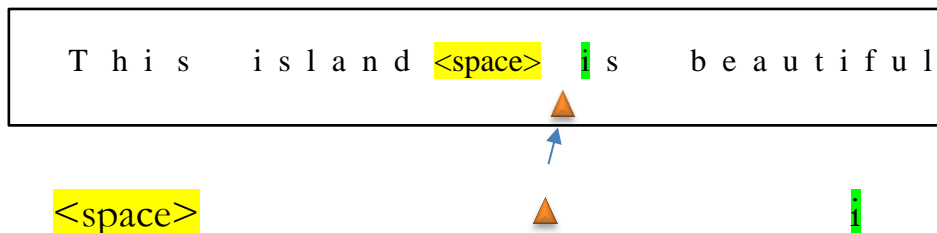
This island is beautiful cont.

Match: `\bis\b`

To save time, additional intermediary steps have been omitted.



The Word Boundary matches between the `<space>` character that follows after 'island' and the 'i' of 'is'.



looks to the left for a non-Word Character

position

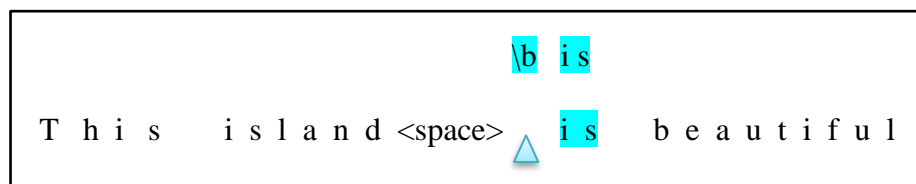
looks to the right for a Word Character

= true

= true

`\b` is matched.

The RegEx Engine advances to the character, the 'i' of 'is' and tests for the literal string characters 'i' and the 's' of 'is'.



Matches.

Metacharacters in Depth

`\b` A Word Boundary cont.

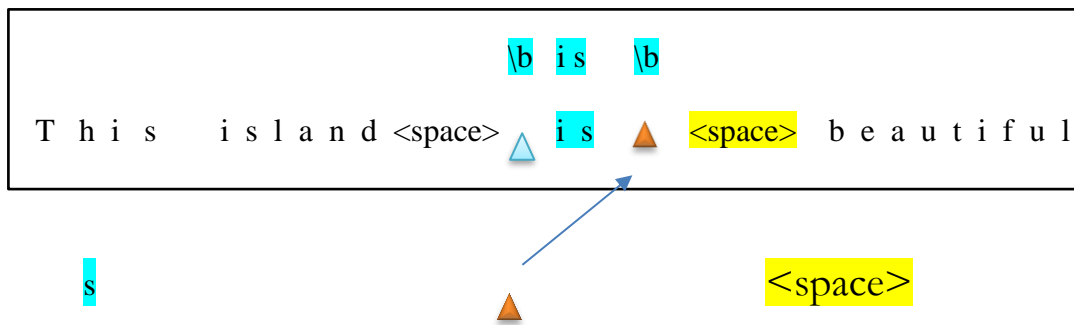
This island is beautiful cont.

Match: `\bis\b`

Looking to match against the final `\b` in the RegEx.

```

/ (\bis\b)
This·island·is·beautiful
  
```



looks to the left for a Word Character

position

looks to the right for a non-Word Character

= true

= true

The Word Boundary matches between the 's' of 'is' and the `<space>` character that follows after 'is'.

Matches. RegEx satisfied.

```

/ (\bis\b)
This·island·is·beautiful
  
```

Match 1		
Full match	12-14	is
Group 1.	12-14	is

```

REGULAR EXPRESSION
: / (\bis\b)
TEST STRING
This·island·is·beautiful
  
```

After the match of the 's', the current position is just before the `<space>` preceding 'beautiful'. The last Word Boundary begins the test at this position. It looks to the left to match against the 's' that was just matched as a string literal, as a Word Character because it belongs to the class of `[A-Za-z0-9_]`. It looks to the right and matches against the `<space>` as a non-Word Character because it falls into the class of `[^A-Za-z0-9_]`. The match is complete and the RegEx is satisfied. The search for additional matches continues, but finding none, the string exhausts.

Metacharacters in Depth

`\b` A Word Boundary cont.

This island is beautiful cont.


Match: `\bis\b`

In BRU, it is represented as:

String = This island is beautiful

Match: `(\bis\b)`

Replace: `\1`

Name ▲	New Name
 This island is beautiful	is

Analysis:

- `\b` Word Boundary.
Matches between `<space>` after 'island' and 'i' of 'is'.
- is Match against string 'is' = true
- `\b` Word Boundary.
Matches between 's' of 'is' and the `<space>` preceding 'beautiful'.

Capture Group 1 = is

Metacharacters in Depth

`\b` A Word Boundary cont.

This island is beautiful cont.

Match: `\bis\b`

When it comes to debugging using the Regex Buddy program, there isn't too much to go on:

T h i s i s l a n d i s b e a u t i f u l

Character Position 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

1. This first photo below shows the match of the Word Boundary between BOL and 'T' of 'This' and subsequent failure when matching against the string characters, 'is'.

```

Beginning match attempt at character 0
1 ok
2 backtrack
3 backtrack
Match attempt failed after 3 steps

```

2. This second photo shows the match of the Word Boundary between the <space> following 's' of 'This' and the 'i' of 'island' and subsequent failure of the Word Boundary test between the 's' of 'island' and the 'l' of 'island', although this is not apparent in the photo, nor does it display the match of the string characters, 'is' of 'island'.

When it comes to documenting RegEx, most often, failures are not documented well or at all, only the successful matches. That's where I differ in my opinion. I do attempt to document the pertinent failures as well as the successes so that you can see what went wrong and why.

```

Beginning match attempt at character 4
1 ok
2 backtrack
3 backtrack
Match attempt failed after 3 steps

```

3. This third photo shows the successful match of the Word Boundary between the <space> following 'island' and the 'i' of 'is' at step 1. Steps 2 to 3 show the match of the literal string characters, 'is' against the 'is' in the string. Step 4 is the final match of the Word Boundary between the 's' of 'is' and the <space> following.

```

Beginning match attempt at character 12
1 ok
2 i
3 is
4 isok
5 is
Match found in 5 steps

```

Metacharacters in Depth

`\b` A Word Boundary cont.

Now that you understand how it works, let's see it in action.

Using Word Boundaries on either side of a word affords the opportunity to allow 'whole word only' searches using a Regular Expression.

Example:

`\bcat\b` Looking to match against the word 'cat'.

Matches 'cat' if there are non-Word Characters (including whitespace) before the 'c' and after the 't'.

Match: `(\bcat\b)`
 Replace: `\1 <space>`

Name	New Name
catfish	catfish
catamaran	catamaran
cat o' nine tails	cat
bobcat	bobcat
hello kitty cat	cat

It will not match 'catfish' since an 'f' character follows after the 't' instead of punctuation or whitespace. In other words, 'catfish' is a separate word from 'cat'. For each of the above strings, if 'cat' is a separate word, it will match, otherwise, no.

String = cat o' nine tails

Analysis:

- `\b` Word Boundary.
Matches at position between BOL and the 'c' of 'cat'.
- cat Matches the literal character string of 'cat' Capture Group 1 = cat
- `\b` Word Boundary.
Matches at position between the 't' of 'cat' and the `<space>` following.

String = hello kitty cat

Analysis:

- `\b` Word Boundary.
Matches at position between `<space>` after 'kitty' and the 'c' of 'cat'.
- cat Matches the literal character string of 'cat' Capture Group 1 = cat
- `\b` Word Boundary.
Matches at position between the 't' of 'cat' and EOL.

Metacharacters in Depth

`\b` A Word Boundary cont.

In the string, 'cat o' nine tails', used in the samples on the previous page, 'cat' is matched because the character that precedes 'cat' is BOL. BOL is considered a non-Word Character so `\b` will match the position between BOL and the 'c' of 'cat', a Word Character. The second `\b` matches the position between the `<space>` after 'cat', a non-Word Character, and the 'o', a Word Character.

In the same manner, 'cat' of 'hello kitty cat' is matched because the preceding character is a `<space>`, a non-Word Character, followed by the 'c' of 'cat' matching the first `\b`. The second `\b` is matched using a different rule because the 't' of 'cat', a Word Character, is followed by EOL, a non-Word Character.

String = catfish

Analysis:

1. `\b` Word Boundary.
Matches at position between BOL and the 'c' of 'cat.'
2. `cat` Matches the literal character string of 'cat' Capture Group 1 = cat
3. `\b` Word Boundary.
Fails to match at position between 't' of 'cat' and 'f' of 'fish'.

RegEx Fails

Metacharacters in Depth

`\b` A Word Boundary cont.

Example:

If I want to search out words that end with ‘sand’,

Match: `(sand\b)`

Replace: `\1`

Name	New Name
<input type="checkbox"/> This is quicksand more text	sand
<input type="checkbox"/> This is sand more text	sand
<input type="checkbox"/> This is sandquick more text	This is sandquick more text
<input type="checkbox"/> This is sandwich more text	This is sandwich more text
<input type="checkbox"/> This is quick-sand more text	sand

‘Sand’ is matched in these samples where the ‘d’ of ‘sand, a Word Character,’ is followed by a <space>, a non-Word Character, matching the `\b`.

Example:

If I want to search out words that begin with ‘sand’,

Match: `(\bsand)`

Replace: `\1`

Name	New Name
<input type="checkbox"/> This is quicksand more text	This is quicksand more text
<input type="checkbox"/> This is sand more text	sand
<input type="checkbox"/> This is sandquick more text	sand
<input type="checkbox"/> This is sandwich more text	sand
<input type="checkbox"/> This is quick-sand more text	sand

‘Sand’ is matched in these samples where the ‘s’ of ‘sand, a Word Character,’ is preceded by a <space>, a non-Word Character, matching the `\b`.

The ‘quick-sand’ sample is also matched because the <hyphen> is a non-Word Character. Therefore, the `\b` matches the position between the ‘k’ of ‘quick-sand’ followed by the ‘-’ of ‘quick-sand’.

Metacharacters in Depth

`\b` A Word Boundary cont.

Example:

If I want to match against the entire word that includes 'sand' as the ending part of a word,

Match: `(\w+sand\b)`

Replace: `\1`

Name	New Name
<input type="checkbox"/> This is quicksand more text	quicksand
<input type="checkbox"/> This is sand more text	This is sand more text
<input type="checkbox"/> This is sandquick more text	This is sandquick more text
<input type="checkbox"/> This is sandwich more text	This is sandwich more text
<input type="checkbox"/> This is quick-sand more text	This is quick-sand more text

The string 'quicksand' is the only match because that is the only string that includes 'sand' as the ending part of a word. The 'quick-sand' string is not included because the <hyphen> is considered a non-Word Character and would therefore does not match against the `\w+`.

Analysis:

1. `\w` Word Character.
Matches against a Word Character, e.g., the 'q' of 'quicksand'.
2. `+` Made Greedy using the '+' Quantifier, it matches the subsequent Word Characters, 'u,' 'i,' 'c', and 'k', until..
3. `sand` ..it encounters the 's' beginning the word, 'sand'.
The 's' matches against 's' in 'sand' in the next evaluation of the RegEx, `sand`. The string 'sand' is matched.
4. `\b` Word Boundary.
After 'sand' is matched, it searches for a Word Boundary. Using these samples, this would include those strings that have a <space>, a non-Word Character, followed by a Word Character.

The 'quicksand' sample is the only string that matches. Strings that have 'sand' as the beginning of the word, or by itself will not match because there are no Word Characters that precede, thus precluding the `\w+` match.

Metacharacters in Depth

`\b` A Word Boundary cont.

Example:

If I want to match against the entire word that includes 'sand' as the beginning part of a word,

Match: `(\bsand\w+)`

Replace: `\1`

Name	New Name
<input type="checkbox"/> This is quicksand more text	This is quicksand more text
<input type="checkbox"/> This is sand more text	This is sand more text
<input type="checkbox"/> This is sandwich more text	sandwich
<input type="checkbox"/> This is sandwich more text	sandwich
<input type="checkbox"/> This is quick-sand more text	This is quick-sand more text

Only those samples that have 'sand' as the beginning part of the word followed by additional text (Word Characters) are matched. For example, 'sand-quick' would not match because the <hyphen> '-' is not a Word Character and would not match against the `\w+`.

Analysis:

- `\b` Word Boundary.
In the samples above, matches...
- `sand` But this is refined by 'sand' which indicates that the only valid match of Word Boundaries are those that are followed by the string, 'sand', or to be more specific, a non-Word Character followed by, 's' of 'sand', a Word Character. In the sample group, those strings that qualify have a <space> or other non-Word Character preceding 'sand', <hyphen> included.
- `\w` This is further refined by the inclusion of the `\w`. This limits the match to only those strings that have a <space> preceding 'sand' followed by a Word Character. This precludes any strings that have 'sand' isolated and disallows the <hyphen> previously matched.
- `+` Make it Greedy.
Include all Word Characters that follow for the samples that were matched in step 3.

```

This is quicksand more text
This is sand more text
This is sandwich more text
This is sandwich more text
This is quick-sand more text
  
```

```

This is quicksand more text
This is sand more text
This is sandwich more text
This is sandwich more text
This is quick-sand more text
  
```

```

This is quicksand more text
This is sand more text
This is sandwich more text
This is sandwich more text
This is quick-sand more text
  
```

```

This is quicksand more text
This is sand more text
This is sandwich more text
This is sandwich more text
This is quick-sand more text
  
```

Metacharacters in Depth

`\b` A Word Boundary cont.

Example:

Match: `(\bcat)`
 Replace: `\1 <space>`

Name	New Name
catfish	cat
catamaran	cat
cat o' nine tails	cat
bobcat	bobcat
hello kitty cat	cat

Searches out any words that have a non-Word Character before the 'c' followed by an 'a' and 't'. In the samples, all strings that have the word, 'cat', preceded by BOL or have a preceding <space> character, are matched. The string, 'bobcat' is not matched because there is a Word Character, 'b' (as in 'bob'), before the Word Character, 'c'.

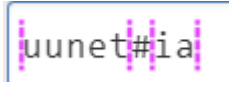
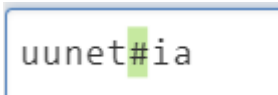
Example:

String = `uunet#ia`

Match: `(\b#)`
 Replace: `\1`

Name	New Name
uunet#ia	#

In this example I will provide a little more insight into what the RegEx Engine is doing when it matches using Word Boundaries.

- `\b` Matches: 
- `#` Matches: 

In step 1, potential matches include the position before BOL, the position preceding EOL, as well as those positions both before and after the Numeric Sign, '#'.

In step 2, by the addition of the '#', the match in step1 becomes limited to only a Word Character followed by a non-Word Character, the Numeric Sign, '#'. This matches the position between the 't' and the '#' to satisfy the `\b` in step 1 and the match of the literal string character, the Numeric Sign, '#', satisfies step 2.

Metacharacters in Depth

`\b` A Word Boundary cont.

Example:

String = `uunet#ia` cont.

Match: `(\b#)`

The Word Boundary searches to match using the rules:

`^\w` Position preceding Beginning of String followed by a Word Character

`\w$` Position following End of String preceded by a Word Character

`\W \w` Position between a non-Word Character followed by a Word Character

`\w \W` Position between a Word Character followed by a non-Word Character

The RegEx Engine first matches at the position between BOL, and the first 'u' of 'uunet#ia'. It next tests for the '#' character. Failing that, it moves forward and matches the `\b` at the position between the 't' of 'uunet#ia' and the '#' character. The 't' is a Word Character because it falls into the class defined as `[A-Za-z0-9_]`. The '#' is a non-Word Character because it falls into the negated class defined as `[^A-Za-z0-9_]`.

Thereby, there is a valid match of the Word Boundary because it adheres to the rule of a Word Character followed by a non-Word Character.

The RegEx moves to the next character position, the '#', to test for the string character, the Numeric Sign, '#' and this also matches, satisfying the RegEx expression.

Remember what I said earlier when discussing perspective about how the RegEx Engine examines the character both preceding and following that of the current position? It has to physically move to those positions to accomplish this. The result of which is a constant back and forth movement as each character is tested.

Metacharacters in Depth

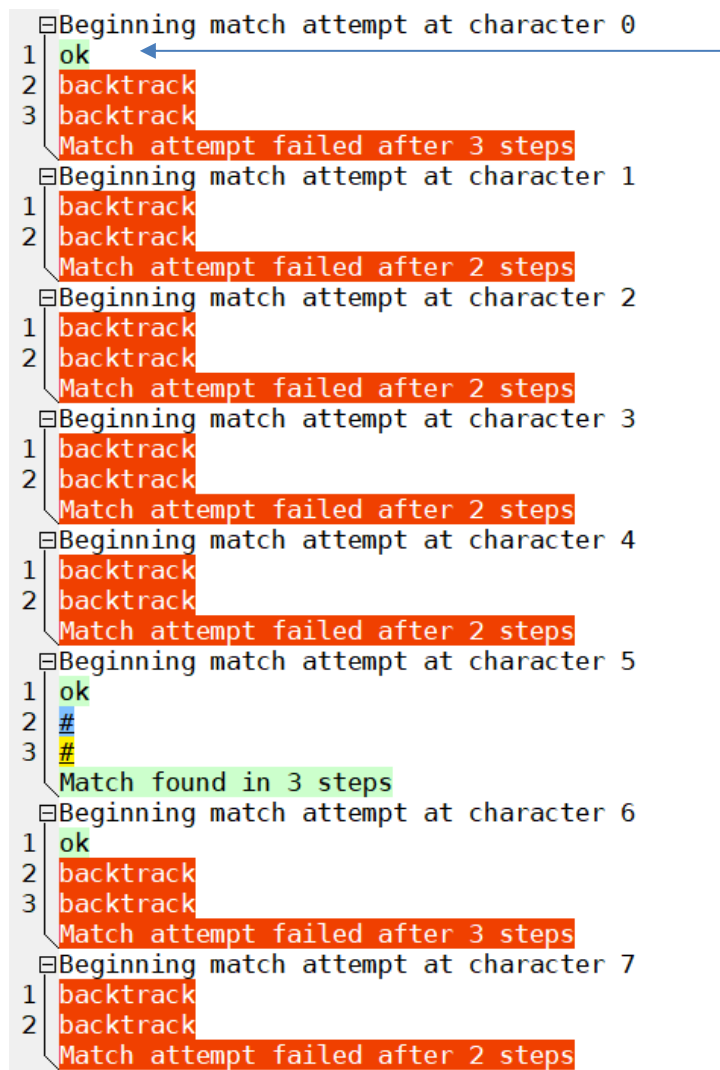
`\b` A Word Boundary cont.

Example:

String = `uunet#ia` cont.

Match: `(\b#)`

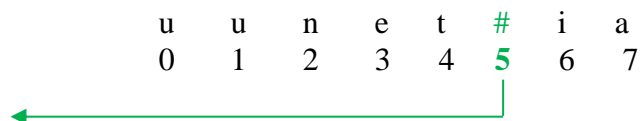
This back and forth of testing can be represented as:



Matches `\b` at the position between BOL and the first 'u', but fails to match against the string character, the Numeric Sign, '#':

`'#' = 'u' = false`

The expression is repeated and tested against each character that makes up the string. Only character position 5 will have a successful match. Character position 5 is the '#' in the string,



The photo also illustrates in the match attempts at character 7 and 8 respectively, how even after the successful match of the string character, the Numeric Sign, '#', the RegEx Engine continues to search for any additional matches until EOL or the string is exhausted, whichever comes first.

Metacharacters in Depth

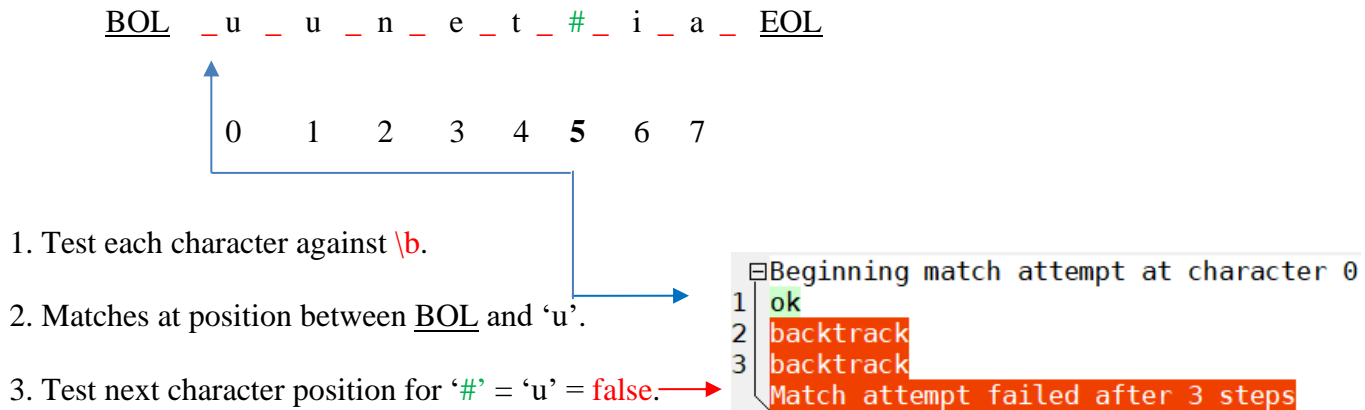
`\b` A Word Boundary cont.

Example:

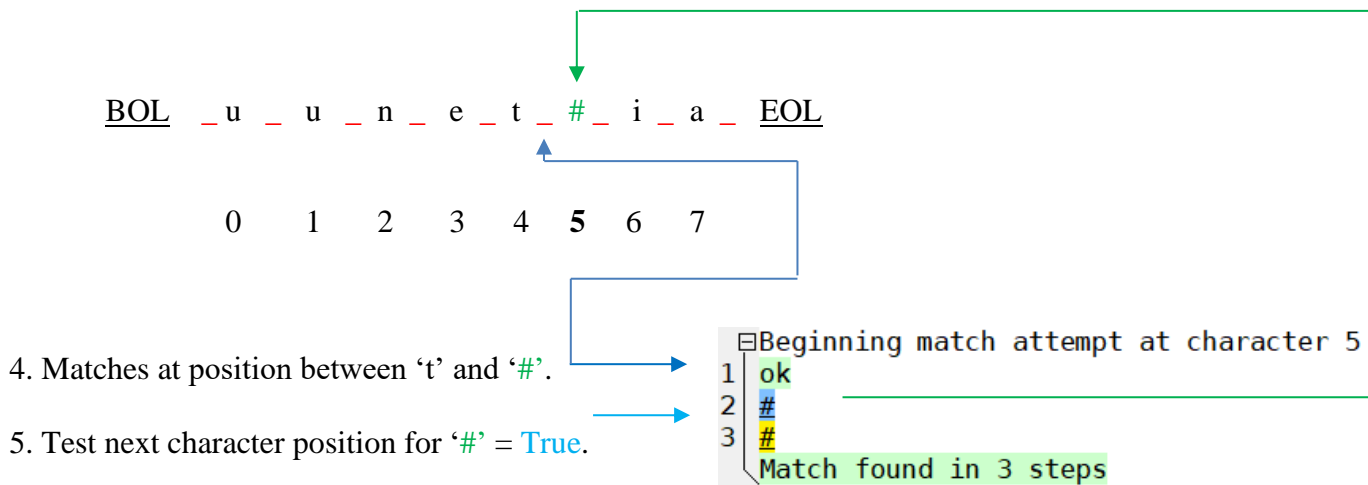
String = `uunet#ia` cont.

Match: `(\b#)`

Examining the match more closely –



Move through remainder of string, test for `\b`.



Metacharacters in Depth

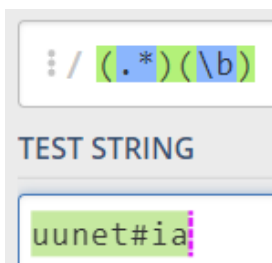
`\b` A Word Boundary cont.

That's fine if the RegEx Engine is testing moving forward in the string, but what happens if it is Backtracking?

Example:

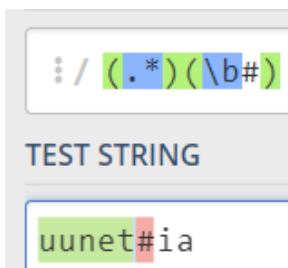
String = `uunet#ia`

I will determine if there is any difference in matching against the Word Boundary:



According to this, it matches the entire string using `(.*)`, then backtracks to match the `\b` to the position between the 'a' and EOL.

Adding the string character to match against results in:



No difference. The `\b` is still matched to the position between the 't' and the '#'. The string character, '#', is also matched in the same manner as before.

In BRU, this can be seen as:

Match: `(.*)(\b#)`

Replace: `Group 1 = \1 Group 2 = \2`

Name	New Name
uunet#ia	Group 1 = uunet Group 2 = #

- `.*` Matches to EOL. Capture Group 1 = <entire string>
- `\b` Match against Word Boundary.
Current position = EOL, so it Backtracks.
Matches position between the 't' and the '#'.
Capture Group 2 = #
- `#` Match against string character, '#'.
Changes Capture Group 1 = uunet

This is a simplified analysis. The RegEx Engine as previously mentioned, is testing each position and character against the expression, matching at EOL as well as the '#' character for the `\b`. I prefer in both volumes to illustrate only the pertinent data and leave out much of the intermediary steps.

Metacharacters in Depth

`\b` A Word Boundary cont.

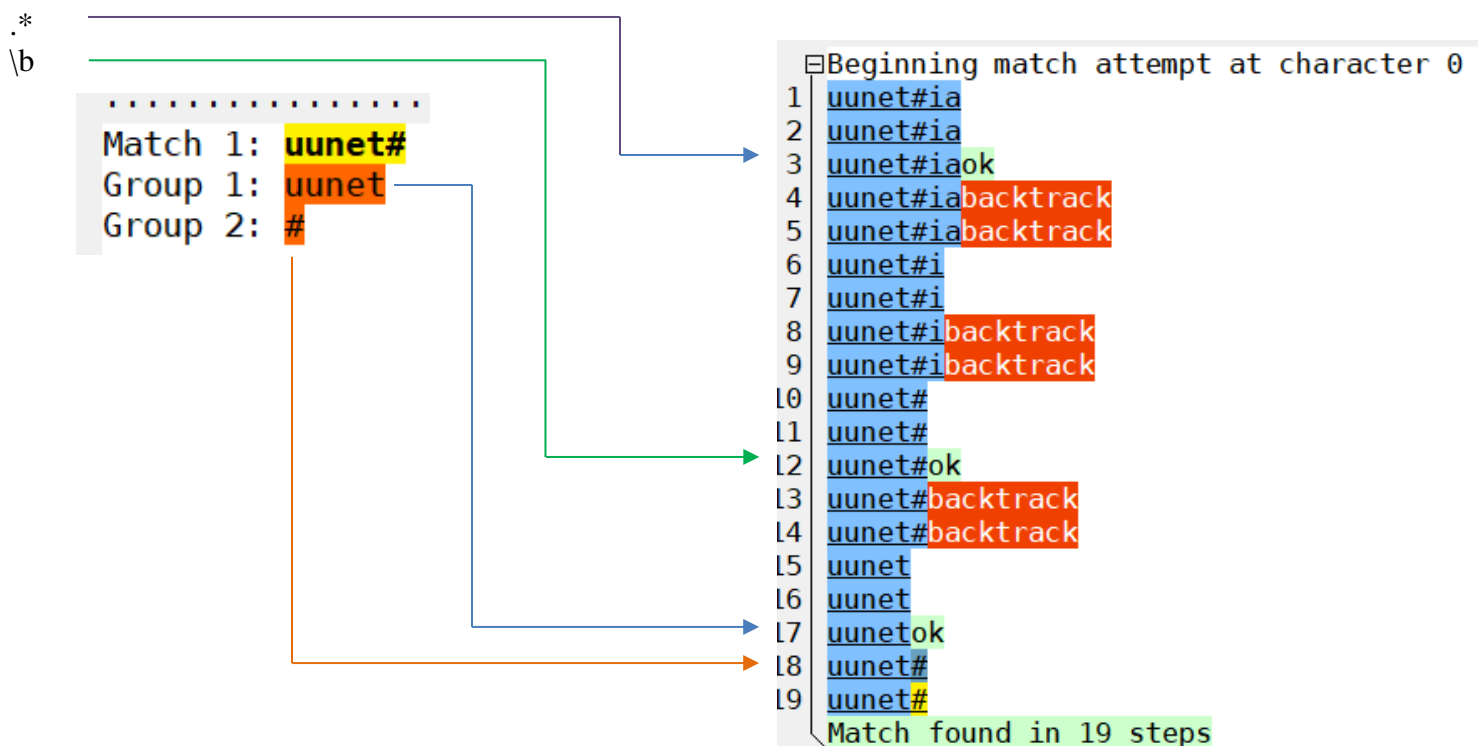
That's fine if the RegEx Engine is testing moving forward in the string, but what happens if it is Backtracking? cont.

Example:

String = `uunet#ia` cont.

Match: `(.*)(\b#)`

Here is the Debug output from Regex Buddy program:



Conclusion:

Moving forward or Backtracking has no effect on how the matches are conducted. The only difference is that in Backtracking, Capture Group 1 captures the entire string and later gives up characters to account for Capture Group 2's capture of the `#`.

Metacharacters in Depth

`\b` A Word Boundary cont.

Word Boundaries are not just applicable to alpha characters, but numeric digits as well.

I'll change this around so that the character string is not a non-Word Character, but a Word Character, the numeric digit, '6'. Remember that Word Characters consist of a class, `[A-Za-z0-9_]`, so numeric digits fall into this class under the range, 0-9.

Example:

String = uunet 6ia

Match: `(\b6)`
 Replace: `\1`

Name	New Name
uunet 6ia	6

```
.....
Match 1: 6
Group 1: 6
```

Analysis:

- `\b` Match against Word Boundary. Matches between the `<space>` after 'uunet', as a non-Word Character and the '6' of '6ia' as a Word Character.
- 6 Evaluates next character position for the numeric digit, '6'. = `true`. RegEx Satisfied.

```
Beginning match attempt at character 6
1 ok
2 6
3 6
Match found in 3 steps
```

On to more complex examples.

Metacharacters in Depth

\b A Word Boundary cont.

Example:

The following provide a range of numbers to match against.

```
\b[1-9][0-9]{3}\b
```

let's break it down to determine the range:

lowest value

match the [1-9] =	n	lowest number is 1	(1 numeric place)
match the [0-9] 3 times =	nnn	lowest number is 000	(3 numeric places)
therefore, the lowest value is	nnnn	= 1000	(4 numeric places)
(n + nnn)			
1 + 000			








highest value

match the [1-9] =	n	highest number is 9	(1 numeric place)
match the [0-9] 3 times =	nnn	highest number is 999	(3 numeric places)
therefore, the highest value is	nnnn	= 9999	(4 numeric places)
(n + nnn)			
9 + 999			

so, the above would match a number between 1000 and 9999

Match: `(\b[1-9][0-9]{3}\b)`

Replace: `\1`

Name	New Name
 Belvedere Plantation 6512.jpg	6512.jpg
 Belvedere Plantation 1002.jpg	1002.jpg
 Belvedere Plantation 12.jpg	Belvedere Plantation 12.jpg
 Belvedere Plantation 564.jpg	Belvedere Plantation 564.jpg
 Belvedere Plantation 6497.jpg	6497.jpg
 Belvedere Plantation 650.jpg	Belvedere Plantation 650.jpg
 Belvedere Plantation 6508.jpg	6508.jpg

Those strings that do not match are because there are not enough numeric places in the value.

For example, '564' value is three numeric places. The Regex calls for one specified by [1-9] and three specified by [0-9]{3}. This means that in order to match, there must be exactly four decimal places in the value.

Metacharacters in Depth

\b A Word Boundary cont.

Analysis:

String = Belvedere Plantation 6512.jpg

Match: `(\b[1-9][0-9]{3}\b)`

I won't bother analyzing the failed intermediary attempts against BOL and the <space> after 'Belvedere'. I rarely do that anyway. And when I did, it was to show you how I believe the RegEx Engine works. My analysis is in no way a complete representation of the process. Much of the time, only pertinent steps that indicate changed values are presented and this is only my interpretation of the events.

- | | | |
|----------|--|------------------------|
| 1. \b | Match against Word Boundary.
Matches between the 'n' of 'Plantation' and the following <space>. | |
| 2. [1-9] | Match against a class consisting of numeric digits,
1-9 = '6' | Capture Group 1 = 6 |
| 3. [0-9] | Match against a class consisting of numeric digits,
0-9 = '5' | Capture Group 1 = 65 |
| 4. {3} | Repeat last sub-expression three iterations
(two more additional iterations) = '12' | Capture Group 1 = 6512 |
| 5. \b | Match against Word Boundary.
Matches between the '2' of '6512' and <u>EOL</u> . | |

Notes:

1. The reason I tend to repeat certain information is to drive home those concepts I feel may help you to better understand this material.

Metacharacters in Depth

`\b` A Word Boundary cont.

Example:

The following provide a range of numbers to match against.

```
\b[1-9][0-9]{2,4}\b
```

Break it down to determine the range:

lowest value

match the [1-9] =	n	lowest number is 1	(1 numeric place)
match the [0-9] for a minimum 2 times =	nn	lowest number is 00	(2 numeric places)
therefore, the lowest value is	nnn	= 100	(3 numeric places)
(n + nn)			
1 + 00			








highest value

match the [1-9] =	n	highest number is 9	(1 numeric place)
match the [0-9] for a maximum 4 times =	nnnn	highest number is 9999	(4 numeric places)
therefore, the highest value is	nnnnn	= 99999	(5 numeric places)
(n + nnnn)			
9 + 9999			

so, the above would match a number between 100 and 99999

Match: `(\b[1-9][0-9]{2,4}\b)`

Replace: `\1`

Name	New Name
 Belvedere Plantation 6512.jpg	6512.jpg
 Belvedere Plantation 1002.jpg	1002.jpg
 Belvedere Plantation 12.jpg	Belvedere Plantation 12.jpg
 Belvedere Plantation 564.jpg	564.jpg
 Belvedere Plantation 6497.jpg	6497.jpg
 Belvedere Plantation 650.jpg	650.jpg
 Belvedere Plantation 6508.jpg	6508.jpg

Those strings that do not match are because there are not enough numeric places in the value.

For example, the '12' value is two numeric places. The Regex calls for one specified by [1-9] and a minimum of 2 and a maximum of four specified by [0-9]{2,4}. This means that in order to match, there must be at least three but less than six decimal places in the value.

Metacharacters in Depth

\b A Word Boundary cont.

Analysis:

String = Belvedere Plantation 650.jpg

Match: `(\b[1-9][0-9]{2,4}\b)`

I won't bother analyzing the failed matches against BOL, and the <space> after 'Belvedere'.

- | | | |
|----------|--|-----------------------|
| 1. \b | Match against Word Boundary.
Matches between the 'n' of 'Plantation' and the following <space>. | |
| 2. [1-9] | Match against a class consisting of numeric digits,
1-9 = '6' | Capture Group 1 = 6 |
| 3. [0-9] | Match against a class consisting of numeric digits,
0-9 = '5' | Capture Group 1 = 65 |
| 4. {2,4} | Repeat last sub-expression between 2 and 4 iterations.
(one minimum, 3 maximum additional iterations) = 0 | Capture Group 1 = 650 |
| 5. \b | Match against Word Boundary.
Matches between the '0' of '650' and <u>EOL</u> . | |

Metacharacters in Depth

`\b` A Word Boundary cont.


Example:

`\b[A-Z0-9]+\b` matches first set of numbers, '1987894' in string...

String = This is a 1987894, 3219800, 234567, 345261. test.jpg

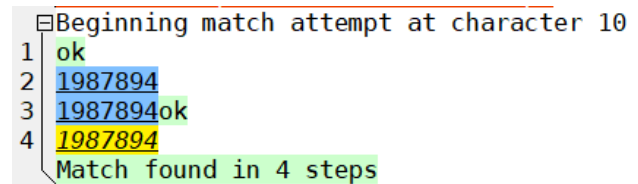
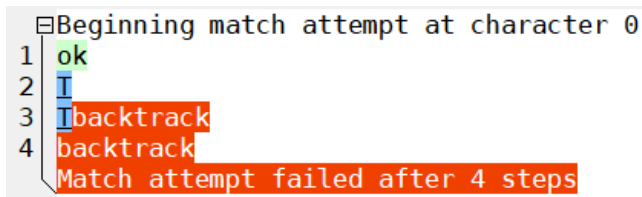
Match: `(\b[A-Z0-9]+\b)`

Replace: `\1`

Name	New Name
 This is a 1987894, 3219800, 234567, 345261. test.jpg	1987894.jpg

Question:

Using the sub-expression of `[A-Z0-9]+`, why isn't the word, 'This' matched? Why does the first match not take place until the first numeric sequence?



The answer is simple when you notice the class consists of UPPERCASE letters and Numeric Digits. Following this is the Greedy Quantifier, '+'. It is this that will match against EITHER a word, as defined by the surrounding Word Boundaries, `\b<text>\b`, made up of all capital letters, OR a word that begins and ends with a numeric digit; a numeric sequence.

In the sample string, there are no words that are made up of all capital letters. In the first photo, you will notice that the position between BOL and the 'T' (it appears as an 'I' because of the underline) of 'This' satisfies the first Word Boundary match. The character position, 'T' is tested against the class of `[A-Z0-9]` and matches because it is an uppercase letter, falling into the class `[A-Z]`, but fails to match the next character position, 'h', because it is a lowercase letter and does not fall into either class of `[A-Z]` or `[0-9]`. The 'h' is tested because the class is made greedy by the adjoining '+' Quantifier.

Could the next character have been a numeric digit and match? In other words, could the sample have matched on a word made up of alpha-numeric characters, e.g. N1WT3J? Yes. That would have matched because the match only has to be a character that falls within the class defined as either uppercase letters or numeric digits. The only other requirement is that the surrounding Word Boundaries will only permit a match if the word has both a preceding and following non-Word Character.

The first Word Boundary matches at the position between the `<space>` and the '1' beginning the numeric sequence of '1987894'. It continues to match the numeric digits, '1', '9', '8', '7', '8', '9', and '4' that fall into the class, `[0-9]`, made Greedy by the '+' Quantifier. The RegEx is satisfied when the second Word Boundary matches between the '4' of '1987894' and the `<space>` following. Although the RegEx will continue to match all of the numeric sequences, under PRE v1, only the first numeric sequence is retained in BRU. New Name returns, '1987894'.

Metacharacters in Depth


\b A Word Boundary cont.

Example:

This next example is a variation on the previous. This RegEx takes in consideration the repeated ‘comma <space>’ between each numeric sequence and limits the captures to 4 groups of the five numeric sequences available..

String = This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg

Match: (\b[A-Z0-9]+\b)(,)(\b[A-Z0-9]+\b)(,)(\b[A-Z0-9]+\b)(,)(\b[A-Z0-9]+\b)
 Replace: \1 \3 \5 \7

Name ▲	New Name
 This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg	1987894 3219800 234567 345261.jpg

Where:

\1 = 1987894 \2 = , <space> \3 = 3219800 \4 = , <space> \5 = 234567 \6 = , <space> \7 = 345261

Breaks down:

(\b[A-Z0-9]+\b)	1987894	Sequence #1
(,)	, <space>	
(\b[A-Z0-9]+\b)	3219800	Sequence #2
(,)	, <space>	
(\b[A-Z0-9]+\b)	234567	Sequence #3
(,)	, <space>	
(\b[A-Z0-9]+\b)	345261	Sequence #4

Nothing but repetition when you see it broken down. It is using the same RegEx as before, with the addition of capturing the ‘comma<space>’.

The fifth numeric sequence, ‘895645’ in the string is ignored because there is not a capture group that exists in the RegEx that can capture the sequence. This would require two more Capture Groups for the maximum of 9 allowed. If you wanted fewer Capture Groups, a simple redo would remove the Capture Groups from the ‘comma<space>’, reducing the maximum used to 5.

Metacharacters in Depth

`\b` A Word Boundary cont.


Example:

Expounding on this, I can select to capture the **first** numeric sequence that ends with a ‘comma<space>’ or the **last**:

String = This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg

Match: `(\b[A-Z0-9]+(,)\b)`

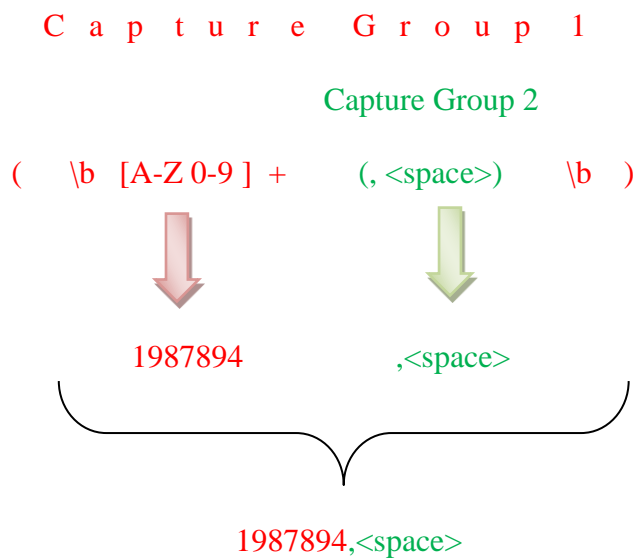
Replace: `\1`

Name ▲	New Name
 This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg	1987894, .jpg

Captures the first numeric sequence that ends with a ‘comma <space>’ combination.

Quick Analysis:

Same as before, it matches against the first numeric sequence isolated from the other words through the inclusion of the surrounding Word Boundaries on either side. However, after this, it must match against both the comma and the <space> in Capture Group 2 ‘nested’ within Capture Group 1.




Metacharacters in Depth

`\b` A Word Boundary cont.

Match: `(\b[A-Z0-9]+(,)\b)+`

Replace: `\1`

Name ▲	New Name
 This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg	345261, .jpg

Captures the last numeric sequence that ends with a ‘comma <space>’ combination. It does this by the addition of the Greedy Quantifier, ‘+’ placed **outside** of the expression making the entire expression repeat throughout the string until the string is exhausted or EOL whichever comes first. In this example, it matches against the last numeric sequence that ends with the ‘comma <space>’. The numeric sequence, ‘895645’ is not matched because it doesn’t end with a ‘comma <space>’. String is exhausted. RegEx satisfied.

There you have it.

I hope I have explained a rather difficult topic in the most logical and simple terms.

Notes:

1. The `\b` is considered (in some of my research) an anchor similar to the `$` and the `^` because it matches at a position called a ‘Word Boundary’, rather than a character.

2. Word Boundaries are probably the best way to do a simple search in a search and replace operation

Example:

`\bCapella\b`matches

Bohdan Warchal; Capella Istropolitana- Water Music Suite No- 1 for orchestra in F major, HWV 348 Minuet 1.mp3

3. `\b` as demonstrated with the ‘#’ series of examples, can be used to capture some punctuation.

4. Word Boundaries are Zero-Length Assertions that do not capture but only ‘assert’ that a match can be made. It is the other sub-expressions of the RegEx that must do the work of matching and capturing for the expression to be successfully evaluated.

Metacharacters in Depth

`\B` A non-Word Boundary

`\B` is the negated version of the Word Boundary, `\b`. `\B` matches at every position where `\b` does not. Effectively, `\B` matches at any position between two word characters as well as at any position between two non-word characters. `\B` is considered (in some of my research) an Anchor Metacharacter and is a Zero Length Assertion handled in the same manner as `\b` during evaluation. It can also match against punctuation, #,\$,!, etc., as long as the character following is not a Word Character.

Quick Review:

Word Characters:

A word is made up of Word Characters. A Word Character is an alpha numeric character that also includes the underscore character. If you recall the discussion on `\w`, the Word Metacharacter, it includes the class of `[A-Za-z0-9_]`.

Whitespace:

Whitespace is defined as `<space>` characters. Whitespace are non-Word Characters.

non-Word Characters:

non-Word Characters include Whitespace, EOL, BOL, and any characters that are part of the negated class, `[^A-Za-z0-9_]`, e.g., punctuation marks, symbols, #, \$, !, etc.

If you haven't already, please also review the Word Boundary section because this will explain in detail about the positioning between characters, etc. Information found there is applicable to this section, much of which is not repeated herein, and this will help you to comprehend the following text.

A non-Word Boundary, `\B`, is matched if it occurs...

- after a string of text, `test\B`
- on either side of a string of text, `\Btab\B`
- Between punctuation and other non-Word Characters, `\B.\B`

Metacharacters in Depth

\B A non-Word Boundary

The Rules:

\w  **\w** a position between two Word Characters (A character, or substring within a word)

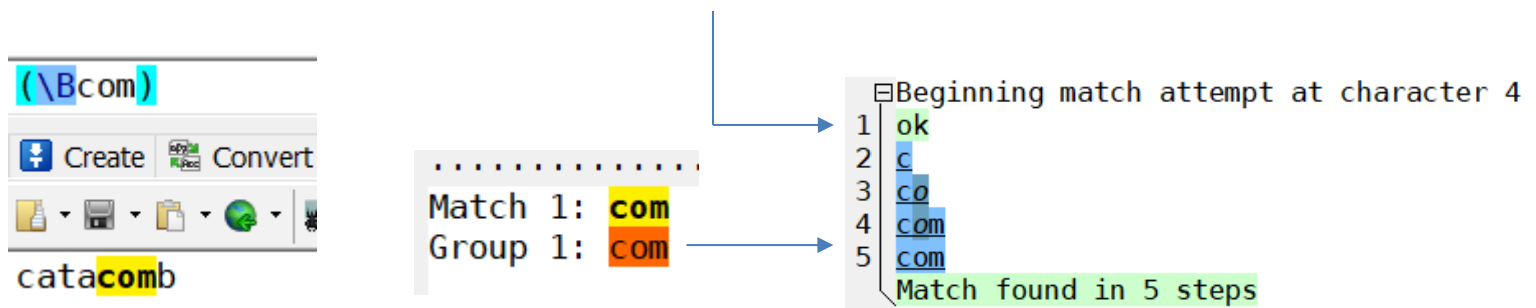
String = catacomb

Match: (\Bcom)


Replace: \1

Name	New Name
 catacomb	com

The non-Word Boundary matches between the position of the second occurrence of the Word Character, 'a' and the second occurrence of the Word Character, 'c' of 'catacomb'.



Notes:

1. The  icon used in these pages represents the non-Word Boundary position between characters.
2. Word Boundaries and non-Word Boundaries are not well documented in Regex Buddy except for acknowledging the match by 'ok'.

Metacharacters in Depth

\B A non-Word Boundary cont.


The Rules:

\W  **\W** a position between two non-Word Characters

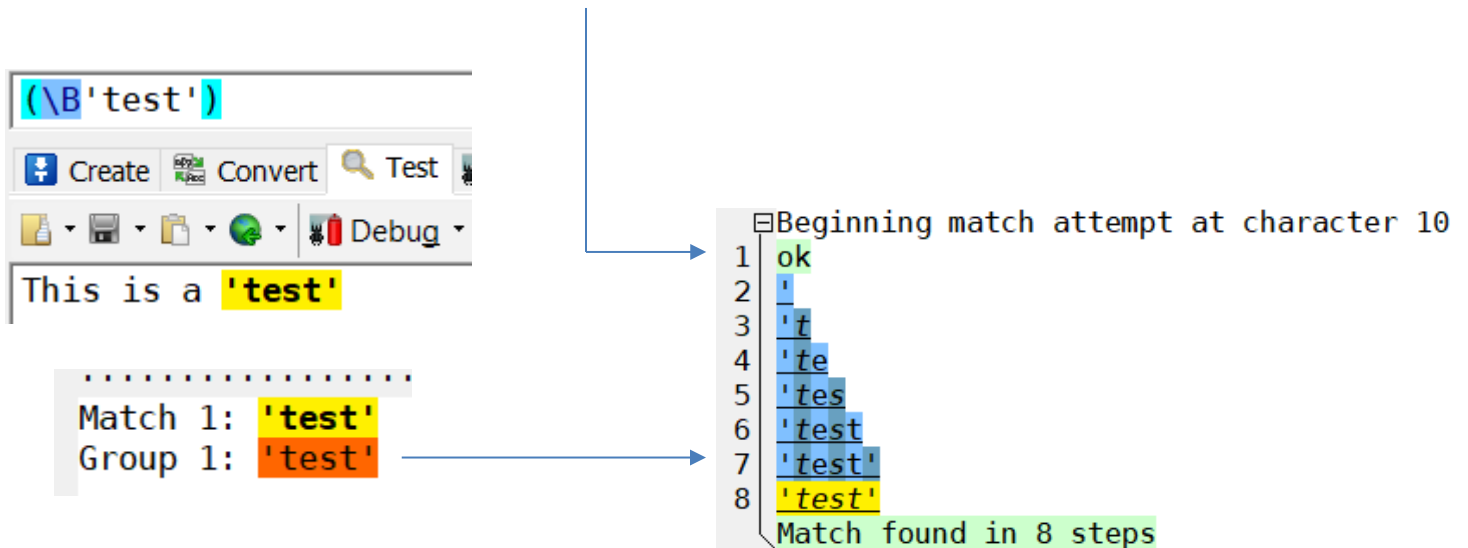
String = This is a 'test'

Match: (\B'test')

Replace: \1

Name	New Name
 This is a 'test'	'test'

The non-Word Boundary matches between the position of the non-Word Character, <space>, following 'a', and the first occurrence of the single quotation non-Word Character of ' 'test' '.



The screenshot shows the Regex Buddy interface. The search pattern is `(\B'test')`. The target string is `This is a 'test'`. The match is found at character 10. The match is highlighted in yellow. The match details are shown as:

```
Match 1: 'test'
```


```
Group 1: 'test'
```

The match attempt is shown in a detailed view, starting at character 10. The match is found in 8 steps:

```
1 ok
2 '
3 't
4 'te
5 'tes
6 'test
7 'test'
8 'test'
```

Match found in 8 steps

Notes:

1. The  icon used in these pages represents the non-Word Boundary position between characters.
2. Word Boundaries and non-Word Boundaries are not well documented in Regex Buddy except for acknowledging the match by 'ok'.

Metacharacters in Depth

\B A non-Word Boundary cont.

The Rules:

\W  **\W** a position between two non-Word Characters cont.

In the same manner I can match at the position between the single quotation and the EOL.

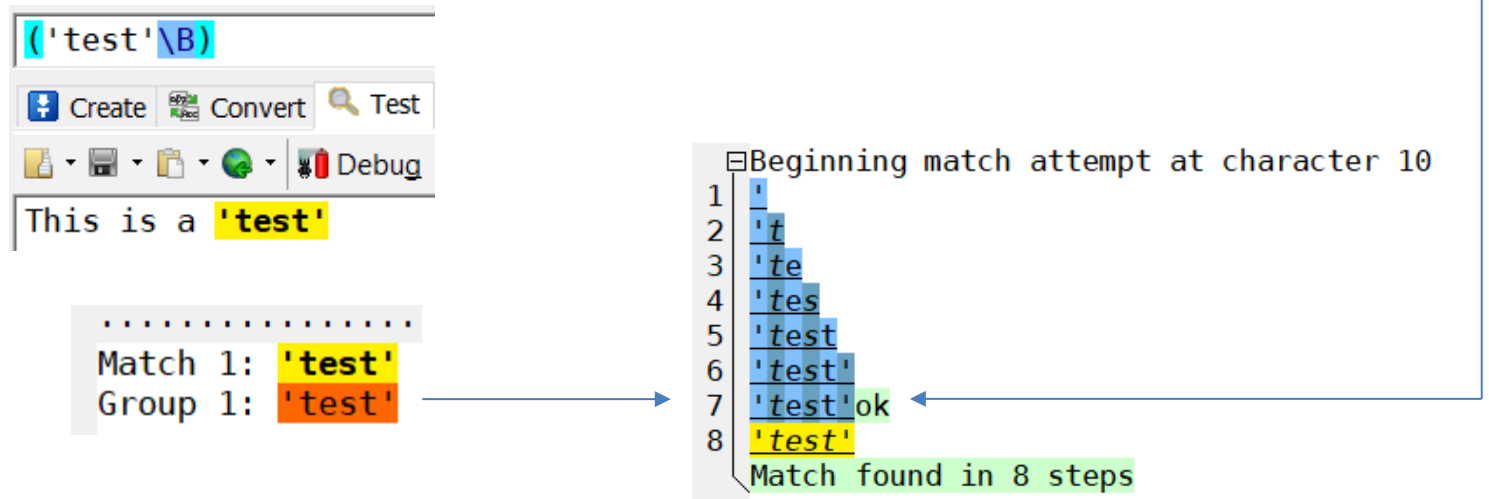
String = This is a 'test'

Match: ('test'\B)

Replace: \1

Name	New Name
 This is a 'test'	'test'

The non-Word Boundary matches between the position of the last occurrence of the single quotation non-Word Character of ' ' and the non-Word Character, EOL.



The screenshot shows the Regex Buddy interface. The regex `('test'\B)` is entered in the search field. The string `This is a 'test'` is shown below. The match attempt starts at character 10. The match is found in 8 steps, resulting in the match `'test'` and group 1 `'test'`.

Match 1: 'test'
Group 1: 'test'

Beginning match attempt at character 10

Match found in 8 steps

Notes:

1. Word Boundaries and non-Word Boundaries are not well documented in Regex Buddy except for acknowledging the match by 'ok'.

Metacharacters in Depth

\B A non-Word Boundary cont.

More Examples:

Samples –

Jim has a test morning

Giving testimony today

Charlie says Midtesting will take place

The evidence will be used to establish guilt or innocence

This will match any string of text that includes 'est'.

Match: (\Best)

Replace: \1

Name ▲	New Name
<input type="checkbox"/> Jim has a test morning	est
<input type="checkbox"/> Giving testimony today	est
<input type="checkbox"/> Charlie says Midtesting will take place	est
<input type="checkbox"/> The evidence will be used to establish guilt or innocence	The evidence will be used to establish guilt or innocence

String = Giving testimony today

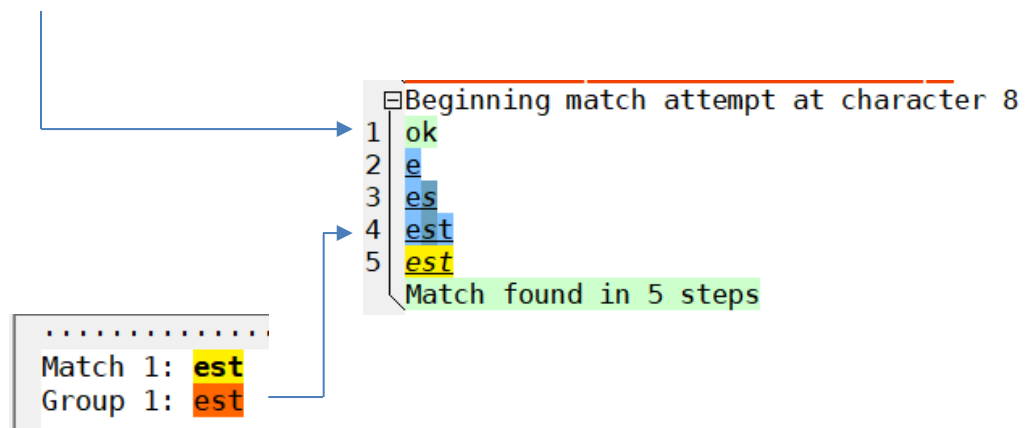
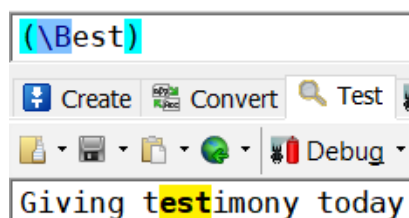
1. **\B** non-Word Character.
Matches the position between a Word Character and another Word Character in the sample string.

Giving testimony today

In the photo to the right, this will match the positions between all of the characters (violet parallel lines) that make up the words in the sample string with the exceptions of the first and last character of each word because the **BOL** and **EOL** and **<space>** characters are all non-Word Characters that are either followed or preceded by a Word Character.

2. **est** Match against the string, 'est'. Capture Group 1 = est

The RegEx Engine moves forward through the string testing for the non-Word Boundary followed by 'est'. This matches at 'est' of 'testimony'. The **\B** matches at the position between the first occurrence of the 't' of 'testimony' and the following 'e'.



Metacharacters in Depth

\B A non-Word Boundary cont.

More Examples:

➤ after a string of text, test\B

Samples –

Jim has a test morning

Giving testimony today

Charlie says Midtesting will take place

The evidence will be used to establish guilt or innocence

This will match only if 'test' has text immediately following. 'test' cannot be at the end of a word.

Match: (test\B)

Replace: \1

Name ▲	New Name
<input type="checkbox"/> Jim has a test morning	Jim has a test morning
<input type="checkbox"/> Giving testimony today	test
<input type="checkbox"/> Charlie says Midtesting will take place	test
<input type="checkbox"/> The evidence will be used to establish guilt or innocence	The evidence will be used to establish guilt or innocence

String = Giving testimony today

1. test Match against string, 'test'.
Matches the 'test' of 'testimony'. Capture Group 1 = test
2. \B non-Word Character.
Matches the position between a Word Character and another Word Character in the sample string.

After matching 'test', it evaluates for the non-Word Character either followed and preceded by a Word Character. The \B matches at the position between the second occurrence of 't' of 'testimony' and the following 'i'.

The screenshot shows the Bulk Rename Utility interface. The search pattern `(test\B)` is entered in the search field. Below it, the string "Giving testimony today" is shown with "test" highlighted in yellow. A detailed view of the matching process is shown to the right, titled "Beginning match attempt at character 7". It shows a step-by-step breakdown of the string "testimony" character by character:

- 1 t
- 2 te
- 3 tes
- 4 test
- 5 testok
- 6 test

A green highlight under "test" in step 6 indicates the match found in 6 steps. Below this, the results are shown as:

```
Match 1: test
Group 1: test
```

Metacharacters in Depth

\B non-Word Boundary cont.

➤ on either side of a string of text, `\Btab\B`

Samples –

The evidence will be used to establish guilt or innocence

Does anyone drink tab anymore

Must tabulate the results

This will match only if the string of text falls within a larger string of text.

Match: `(\Btab\B)`

Replace: `\1`

Name ▲	New Name
<input type="checkbox"/> The evidence will be used to establish guilt or innocence	tab
<input type="checkbox"/> Does anyone drink tab anymore	Does anyone drink tab anymore
<input type="checkbox"/> Must tabulate the results	Must tabulate the results

String = The evidence will be used to establish guilt or innocence

1. **\B** non-Word Character.
Matches the position between a Word Character and another Word Character in the sample string.

This will match the positions between all of the characters that make up the words in the sample string with the exceptions of the first and last character of each word because the **BOL** and **EOL** and <space> characters are all non-Word Characters that are either followed or preceded by a Word Character.

2. **tab** Match against the string, 'tab'. Capture Group 1 = tab

The RegEx Engine moves forward through the string testing for the non-Word Boundary followed by 'tab'. This matches at 'tab' of 'establish'. The **\B** matches at the position between the first occurrence of the 's' of 'establish' and the following 't'.

3. **\B** non-Word Character.
Matches the position between a Word Character and another Word Character in the sample string.

In order to match, there must be a successive non-Word Character after the match of 'tab'. The string, 'tab' must be a substring within another word and not at the start or end of a word. The **\B** matches at the position between the 'b' of 'establish' and the following 't'.

Metacharacters in Depth

\B non-Word Boundary cont.

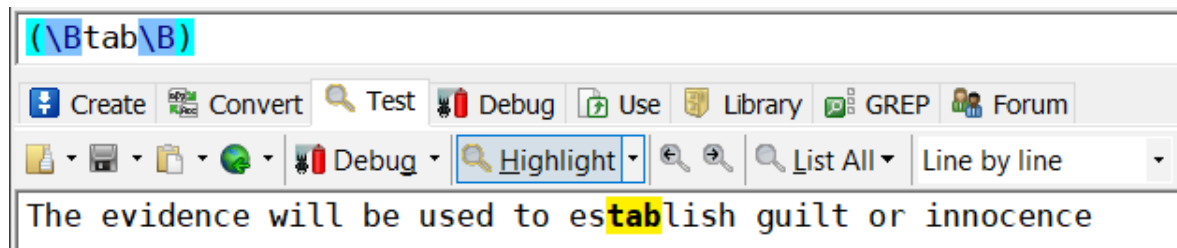
➤ on either side of a string of text, `\Btab\B`

Match: `(\Btab\B)`

Replace: `\1`

Name ▲	New Name
The evidence will be used to establish guilt or innocence	tab
Does anyone drink tab anymore	Does anyone drink tab anymore
Must tabulate the results	Must tabulate the results

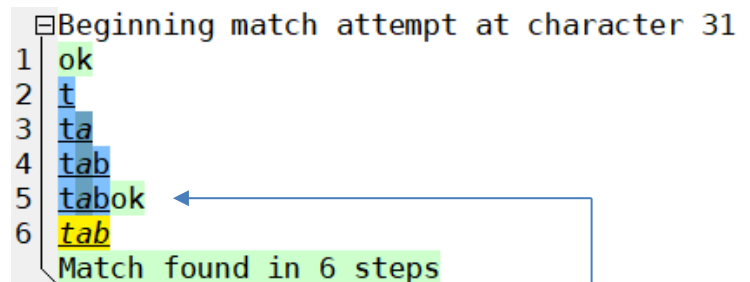
String = The evidence will be used to establish guilt or innocence



The first `\B` matches at the position between the first occurrence of the 's' of 'establish' and the following 't'.

The string, 'tab' matches at the 'tab' substring within the word, 'establish'.

The second `\B` matches at the position between the 'b' of 'establish' and the following 'l'.



Metacharacters in Depth

\B non-Word Boundary cont.

➤ Between punctuation and other non-Word Characters,

\B.\B

This will match only if the non-Word Character is preceded or followed by another non-Word Character. non-Word Characters fall within the negated class of `[^A-Za-z_]`. This includes punctuation, like the literal dot character above representing a period, EOL (End of Line or `$`), BOL (Beginning of Line or `^`), symbols, etc., and whitespace including the `<space>` character.

Examples:

\B<space>

This is a test

	Start	Length	
Match 1:	0	1	^{CL} _{RF}
Group 1:	0	1	

Placing a `<space>` before the start of the string will match the position between BOL and the `<space>`. The same cannot be said for placing a `<space>` at the end of a string even though EOL is a non-Word Character, excess trailing `<spaces>` are (truncated) ignored in BRU. This will work, however, inside of programs like Regex Buddy because these programs are meant for multi-line text and not just single filenames or folder names.

Multiple `<space>` characters will also match:

This is a test

	Start	Length	
Match 1:	8	1	^{CL} _{RF}
Group 1:	8	1	

!\B

This will match at EOL.

This is a test!

	Start	Length	
Match 1:	14	1	^{CL} _{RF}
Group 1:	14	1	

Now that you have the hang of it, here are some more:

\B#

\B'

!\B (secondary match in blue)

\B!\B

This is a test of '#2'

This is a test of '#2'

This is a test of '#2'

This is a test of '#2'

<space>\B

This is a test of '#2'

These won't match. Can you tell me why?

\B#\B

\B<space>\B

#\B

Metacharacters in Depth

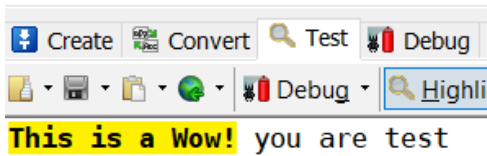
\B non-Word Boundary cont.

More Examples: cont.

String = This is a Wow! you are test

Match: (.*)(!\B)

`(.*)(!\B)`



```

.....
Match 1: This is a Wow!
Group 1: This is a Wow
Group 2: !

```

Beginning match attempt at character 0

- 1 This is a Wow! you are test
- 2 This is a Wow! you are test
- 3 This is a Wow! you are test backtrack
- 4 This is a Wow! you are test backtrack
- 5 This is a Wow! you are tes
- 6 This is a Wow! you are tes
- 7 This is a Wow! you are tes backtrack
- 8 This is a Wow! you are tes backtrack
- 9 This is a Wow! you are te
- 10 This is a Wow! you are te
- 11 This is a Wow! you are te backtrack
- 12 This is a Wow! you are te backtrack
- 13 This is a Wow! you are t
- 14 This is a Wow! you are t
- 15 This is a Wow! you are t backtrack
- 16 This is a Wow! you are t backtrack
- 17 This is a Wow! you are
- 18 This is a Wow! you are
- 19 This is a Wow! you are backtrack
- 20 This is a Wow! you are backtrack
- 21 This is a Wow! you are
- 22 This is a Wow! you are
- 23 This is a Wow! you are backtrack
- 24 This is a Wow! you are backtrack
- 25 This is a Wow! you ar
- 26 This is a Wow! you ar
- 27 This is a Wow! you ar backtrack
- 28 This is a Wow! you ar backtrack
- 29 This is a Wow! you a
- 30 This is a Wow! you a
- 31 This is a Wow! you a backtrack
- 32 This is a Wow! you a backtrack
- 33 This is a Wow! you
- 34 This is a Wow! you
- 35 This is a Wow! you backtrack
- 36 This is a Wow! you backtrack
- 37 This is a Wow! you
- 38 This is a Wow! you
- 39 This is a Wow! you backtrack
- 40 This is a Wow! you backtrack
- 41 This is a Wow! yo
- 42 This is a Wow! yo
- 43 This is a Wow! yo backtrack
- 44 This is a Wow! yo backtrack
- 45 This is a Wow! y
- 46 This is a Wow! y
- 47 This is a Wow! y backtrack
- 48 This is a Wow! y backtrack
- 49 This is a Wow!
- 50 This is a Wow!
- 51 This is a Wow! backtrack
- 52 This is a Wow! backtrack
- 53 This is a Wow!
- 54 This is a Wow!
- 55 This is a Wow! backtrack
- 56 This is a Wow! backtrack
- 57 This is a Wow
- 58 This is a Wow
- 59 This is a Wow!
- 60 This is a Wow! ok
- 61 This is a Wow!

Match found in 61 steps

The `\B` matches at the position between the `!` of `Wow!` and the `<space>` that follows.

Metacharacters in Depth

\B non-Word Boundary cont.

➤ Between punctuation and other non-Word Characters,

\B.\B

More Examples:

Simplified string = is a Wow! r

Match: `(.*)(\B!)`

Replace: `Group 1 = \1 Group 2 = \2`

This won't match because the pattern is now **\B** followed by the exclamation point. This requires that a non-Word Character precede the exclamation point. It doesn't. The Word Character, 'w' of 'Wow' precedes.

Metacharacters in Depth

Quantifiers

Before I begin.

This section may be especially difficult to understand the RegEx components, e.g., a, b, c, from the string characters, e.g., a, b, c, because in many cases, they are the same or similar characters. To help differentiate between them, I have the string characters and any matched values in **bold light blue**. I did not bother with this in the analysis sections because the meaning of the two is not as confused. I did, however, apply this to any commentary.

Metacharacters in Depth

Quantifiers cont.

A quantifier specifies how many instances of the previous character or Metacharacter must be present for the match to be successful.

- * Match the previous character, expression or metacharacter **zero or more times**
(As many repetitions as needed - Greedy)

Examples:

1) a* matches **ab**:

Match: (a*)
Replace: %1 <space>

Name ▲	New Name
ab	a

ab

2) a* matches **aaab**:

Match: (a*)
Replace: %1 <space>

Name ▲	New Name
aaab	aaa

aaab

3) a* matches: **baaa**. Zero Length Match returns a **null** value

Match: (a*)
Replace: %1 <space>

The first character is not a match to the **a** but the Greedy Quantifier makes the match to the beginning of the string where there is no value resulting in what is called a Zero Length Match.

Name ▲	New Name
baaa	

Zero Length Match returns a **null** or empty value of "" and because the Replace String specifies a <space> literal character, New Name displays with the <space> character only, otherwise New Name would reflect no change:

Name ▲	New Name
baaa	baaa

Metacharacters in Depth

Quantifiers cont.

This will require some explanation.

A Zero Length Match, e.g., Zero Occurrence Match, is where the match is made, but there is no value or the value is empty, resulting in the length of the match as zero. Examples of a Zero Length Match include matching at EOL or BOL, or matching at positions between characters in the case of a Word Boundary.

A Word Boundary is a Zero Length Assertion where the matched value is not retained anyway.

A Zero Occurrence Match, as some refer to it, is typically when a previous value of the match is given up but not **Invalidated**, so the match exists with no value. An **Invalidated** match on the other hand, would be as if the match were never made, e.g., using an 'Optional' or Lazy Quantifier, ' ? '.

If the Global Switch was used, there would be three matches in this example.

- * Using the Greedy Quantifier in this manner will always match against the beginning and end of string.
- aaa The a in the Match String will initially match against the first **a** in the sample string, but when made Greedy by the *, will capture any and all other **a** characters in the sample string.

These matches can be expressed as:

The screenshot shows a regex tool interface. The pattern `(a*)` is entered. The test string is `baaa`. Three matches are shown:

- Match 1:** Full match 0-0, Group 1. 0-0
- Match 2:** Full match 1-4, Group 1. 1-4
- Match 3:** Full match 4-4, Group 1. 4-4

Arrows indicate the start and end of each match on the string 'baaa'.

```

Beginning match attempt at character 0
1 ok
2 ok
Match found in 2 steps
    
```

```

Beginning match attempt at character 1
1 aaa
2 aaa
Match found in 2 steps
    
```

	Start	Length	...
Match 1:	0	0	<small>C L R F</small>
Group 1:	0	0	<small>C L R F</small>
Match 2: aaa	1	3	<small>C L R F</small>
Group 1: aaa	1	3	<small>C L R F</small>
Match 3:	4	0	<small>C L R F</small>
Group 1:	4	0	

Zero Length Matches

Metacharacters in Depth

Quantifiers cont.

Examples:

5) abc.*123 matches: **abcAnything123**

This requires a discussion concerning .*

Typically the * Greedy Quantifier is used most often in combination with the dot Metacharacter. The dot Metacharacter, if you recall, can match against any single character. By adding the Greedy Quantifier, it will match against all characters from the matched character of the dot to the end of the string.

Used as:

Match: (.*)

Will match against the entire string. This moves the RegEx Engine to the end of the string. The purpose of doing this may be to capture any remaining portion of the string after previous matches by other sub-expressions:

String = This is quicksand more text

Match: (This <space>)(.*)

Replace: Group 1 = \1 Group 2 = \2

Name ▲	New Name
<input type="checkbox"/> This is quicksand more text	Group 1 = This Group 2 = is quicksand more text

Another method is after matching to capture the entire string, any sub-expression that follows will cause the RegEx Engine to backtrack in an attempt to satisfy the RegEx. If a successful match is made, that value is removed from the previous capture. This is referred to as giving up or giving back characters of the match. Myself, I like to also refer to it as re-evaluating the match, in order to simplify the explanation for new learners.

String = This is quicksand more text

Match: (.*)(text)

Replace: Group 1 = \1 Group 2 = \2

Name ▲	New Name
<input type="checkbox"/> This is quicksand more text	Group 1 = This is quicksand more Group 2 = text

- | | | |
|---------|--|--|
| 1. .* | matches entire string | Capture Group 1 = This is quicksand more text |
| 2. text | Already at <u>EOL</u> , so RegEx Engine backtracks (moves backwards) to match against 'text'. Because 'text' was part of the first capture, those characters are removed from Capture Group 1. | Capture Group 2 = text
Changes Capture Group 1 = This is quicksand more <space> |

Metacharacters in Depth

Quantifiers cont.

With this in mind, the example –

Match: (abc.*123)
Replace: %1 <space>

Name	New Name
abc123	abc123
abcAnything123	abcAnything123

abcAnything123

Analysis:

1. abc Matches against the string characters in the sample string. Capture Group 1 = abc
2. . Matches against any character. Capture Group 1 = abcA
3. * Make it Greedy. Capture Group 1 = abcAnything123
4. 123 Match against the string characters, '123'. Already at EOL, so RegEx Engine backtracks to match against the '123'. Capture Group 1 = **Unchanged**

```
.....
Match 1: abcAnything123
Group 1: abcAnything123
```

```
Beginning match attempt at character 0
1 a
2 ab
3 abc
4 abcAnything123
5 abcAnything123 backtrack
6 abcAnything12
7 abcAnything12 backtrack
8 abcAnything1
9 abcAnything1 backtrack
0 abcAnything
1 abcAnything1
2 abcAnything12
3 abcAnything123
4 abcAnything123
Match found in 14 steps
```

Notes:

1. Although Capture Group 1 already contained the entire string value, in order to match against the **123** in step 4, it had to backtrack. If this match had been captured in a second Capture Group, then the characters of **123** in Capture Group 1 would have been given back. Because it was the same Capture Group, the value remained the same.
2. The <space> is a descriptive reference to an actual space character in the Match String. You would enter it using the space bar on your keyboard and **not** the characters of '<', 'space', and '>'
3. I will tend to repeat those topics I wish to emphasize based on their importance. That is why you may find several passages where I explained Zero Occurrence Matches, for example. This term is just something I came across that really is just a better explanation for a type of Zero Length Match.

Metacharacters in Depth

Quantifiers cont.

Examples:

6) abc.*123 matches: **abc123**

Match: (abc.*123)

Replace: \1 <space>

Name ▲	New Name
abc123	abc123

7) ho*p matches **hp**, **hop** and **hoop**

Match: (ho*p)

Replace: \1 <space>

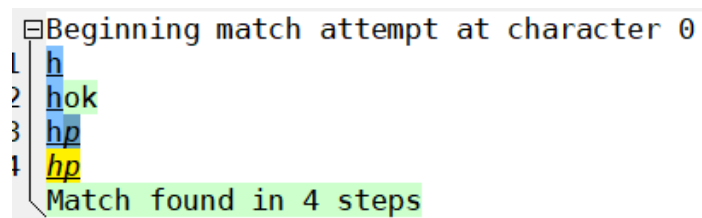
Name ▲	New Name
hp	hp
hop	hop
hoop	hoop

Analysis:

1. h Matches to the 'h' at the beginning of the string Capture Group 1 = h
2. o The 'o' **fails** to match.
3. * Tests p = o = **false** Despite this ...
4. p .. the Greedy Quantifier will continue to match until Capture Group 1 = hp

Because of the Quantifier, although the **o** was not matched, the Greedy Quantifier did match, and if there were additional characters in the string, which there were not, would continue matching until the EOL or another subexpression. In example #5, it captured until the EOL and then had to backtrack in order to match the sub-expression, but in this example, there was no backtracking. The RegEx engine just moved forward and matched against the **p**.

You can see this clearly here:



This is because the Greedy Quantifier in the fifth example was Quantifying the dot character that matches against any character. This took the RegEx Engine to the end of the string before it even got to the sub-expression of 123. But in this example it does not. It matches the **h**, and then moves forward only until it can satisfy the match against the **p**. In order to match, there has to be **p** following the **h**. That is why **hp** matches even though there is no **o** to match against.

Metacharacters in Depth

Quantifiers cont.

Examples:

7) `(.*)` matches **def**, **ghi** in string, **abc 'def' 'ghi' jkl**

Match: `('def' 'ghi')`

Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	'def' 'ghi'

Punctuation is part of the literal characters that must be matched against in order to satisfy the RegEx.

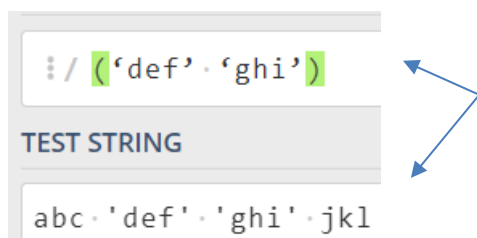
Analysis:

- | | | |
|------------|--|---------------------------------|
| 1. 'def' | Matches against string characters of 'def' in the sample string, ignoring the 'abc'. | Capture Group 1 = 'def' |
| 2. <space> | Matches against the <space> character that follows after 'def'. | Capture Group 1 = 'def' <space> |
| 3. 'ghi' | Matches against string characters of 'ghi' in the sample string. | Capture Group 1 = 'def' 'ghi' |
| 4. | Continues to search for additional matches until the string is exhausted or EOL whichever comes first. | |

Notes:

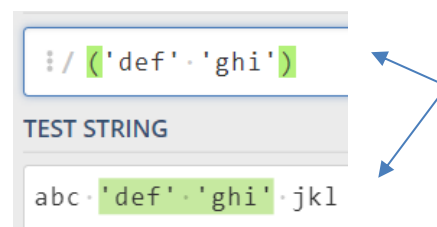
1. Punctuation counts. In **Microsoft Word**, different ASCII characters are used for the **apostrophe characters**. If you notice the little loop on the apostrophe is different for the left and right side. `' '` In **most editors**, the **apostrophe character** is not different and doesn't have the loops. It is not the same ASCII value. If I were to copy and paste this into RegEx101.com, it would not match:

Doesn't match:



Different single apostrophe characters

Matches:



Same single apostrophe characters

Remember this when copying and pasting some of the examples from this volume or from Volume II. If something doesn't work, and it most likely should, this may be the cause.

2. Be aware that there are <space> characters both before and after each grouping of text. This may not be noticeable.
3. Not really ignoring the **abc**. Just easier to think of it that way. The RegEx Engine is testing each and every character against the expression. **abc** doesn't match, so it moves forward continually searching for a match.

Metacharacters in Depth

Quantifiers cont.

- + Matches the previous character, expression or metacharacter **one or more** times
(As many repetitions as needed - Greedy)

The ‘ + ’ is no less Greedy a Quantifier than the ‘ * ’, but instead of matching zero or more times, there has to be at least one match to be successful using the ‘ + ’ Quantifier.

Example:

- 1) a+ matches the strings **ab** and **aaab**. In fact it will match against any string that contains one or more lowercase **a**'s including those found within words.

Match: (a+)
Replace: \1 <space>

Name ▲	New Name
<input type="checkbox"/> baaa	aaa
<input type="checkbox"/> ab	a
<input type="checkbox"/> aaab	aaa
<input type="checkbox"/> abab	a
<input type="checkbox"/> ABC-123	ABC-123
<input type="checkbox"/> eeeAiiZuuuuAoooZeeee	eeeAiiZuuuuAoooZeeee
<input type="checkbox"/> is this all there is or more	a
<input type="checkbox"/> The evidence will be used to establish guilt or innocence	a
<input type="checkbox"/> This is quick-sand more text	a
<input checked="" type="checkbox"/> This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg	a .jpg
<input type="checkbox"/> ABC	ABC
<input type="checkbox"/> ABC-123-DEF#456	ABC-123-DEF#456

String = baaa

Match: (a+)
Replace: \1 <space>

Analysis:

1. a Match against literal lowercase ‘a’. Capture Group1 = a
Will continually move through the string searching for a match of the lowercase ‘a’. If found it will then match against the Quantifier, +, which will capture any additional consecutive lowercase ‘a’s.
Matches against the lowercase ‘a’ of ‘baaa’.
2. + Make it Greedy. Capture Group 1= aaa
Matches against the ‘aaa’ of ‘baaa’

Metacharacters in Depth

Quantifiers cont.

String = eeeAiiZuuuuAoooZeeee

Match: (a*)
 Replace: \1 <space>

Analysis:

1. a Match against literal lowercase 'a'.
 Will continually test each character searching to match against the 'a', but this string does not contain any lowercase 'a' string characters so it will fail at each test.
2. * Make it Greedy.
 By adding the Greedy Quantifier to the 'a' sub-expression, it means to match the 'a' zero or more times. When the test for the lowercase 'a' fails, the zero repetition of the Quantifier matches, but because there is no lowercase 'a', it captures a **null** value resulting in a Zero Occurrence Match with a length of zero. This will evaluate the same for each character in the string.

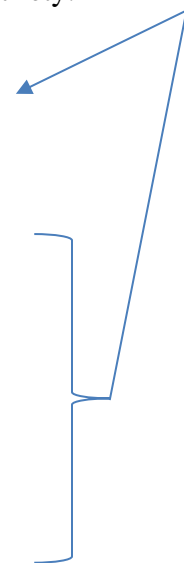
Capture Group 1 = **null**

	Start	Length	
Match 1:	0	0	^C _R F
Match 2:	1	0	^C _R F
Match 3:	2	0	^C _R F

etc.

Because of the Replace String that includes the literal <space> character, the resulting New Name will be the <space> along with any extension associated with the sample filename, for each string that does not contain a lowercase **a** as the first character of the string or for any string that does not contain a lowercase **a** in its entirety.

Name	New Name
baaa	
ab	a
aaab	aaa
abab	a
ABC-123	
eeeAiiZuuuuAoooZeeee	
is this all there is or more	
The evidence will be used to establish guilt or innocence	
This is quick-sand more text	
This is a 1987894, 3219800, 234567, 345261, 895645. test.jpg	.jpg
ABC	
ABC-123-DEF#456	



For samples, e.g., **baaa** or **is this all there is or more**, this is the same result of example #3 under the discussion of the '*' Quantifier where a Zero Occurrence Match results when the first character of the string, the **b** of **baaa** or the **i** of **is**, does not match against the sub-expression. In both of these examples it is the lowercase **a**.

Metacharacters in Depth

Quantifiers cont.

Example:

2) `ho+p` matches **hop**, and **hoop**, but not **hp**

Match: `(ho+p)`

Replace: `\1 <space>`

Name ▲	New Name
<input type="checkbox"/> hoop	hoop
<input type="checkbox"/> hop	hop
<input type="checkbox"/> hp	hp

String = hoop

Analysis:

- | | | |
|--------------------|---|------------------------|
| 1. <code>ho</code> | Match against the literal string characters, 'ho'.
Matches against the 'ho' of 'hoop'. | Capture Group 1 = ho |
| 2. <code>+</code> | Make it Greedy. | Capture Group 1 = hoop |
| 3. <code>p</code> | Match against literal string character, 'p'.
Matches against the 'p' of 'hoop'. | Capture Group 1 = hoop |

The difference between `(ho*p)` and `(ho+p)` is that using the '`*`' Quantifier, it would have matched the entire string in step 2 (zero or more) and have to backtrack in order to match against the **p** in step 3. The '`+`' will only match multiple consecutive occurrences (one or more) of the specified sub-expression, **o**.

If, for example, the string had been:

Name ▲	New Name
<input type="checkbox"/> hoho	ho

The initial **ho** was matched but not the second occurrence of **ho** (unless the global switch was used under v2), or even the second occurrence of the **o** that would have been evaluated in step 2, because it is not consecutive.

String = hp

- | | | |
|-------------------|---|---------------------|
| 1. <code>h</code> | Match against the literal string character, 'h'.
Matches against the 'h' of 'hp'. | Capture Group 1 = h |
| 2. <code>o</code> | Match against the literal string character, 'o'.
<code>p = o = false.</code> | |
| 3. <code>+</code> | Make it Greedy.
<u>EOL</u> reached. No Matches, String Exhausted.
Fails. | |

Think of it this way. Using '`*`', all characters are matched and not just the specified sub-expression, whereas the '`+`' limits itself to the specified sub-expression, **o** in the RegEx, `(ho+p)`.

Metacharacters in Depth

Quantifiers cont.

Example:

- 3) `ab.+` Will not match the string `ab` because there has to be at least one match. Although `'ab'` exists in the string, the dot character calls for a third character, and there is none. The string, `aba` would match.

Name ▲	New Name
ab	ab
aba	aba

String = `aba`

Analysis:

1. `ab` Match against literal string characters, `'ab'`. Capture Group 1 = `ab`
Matches against the `'ab'` of `'aba'`.
2. `.` Dot Metacharacter Matches against any character. Capture Group 1 = `aba`
Matches against the second occurrence of `'a'` in `'aba'`
Current position = EOL.
3. `+` Make it Greedy.
Already at EOL. String Exhausted. No More Matches can be made.

Because the Sub-expression using the dot Metacharacter is made Greedy using the `'+'`, it would behave the same as if I had used `(.*)`. This is because in essence the search is for any characters that are consecutive multiples of the dot Metacharacter. The dot Metacharacter can match against any character. This means that it will match against any characters that follow the initial match of the dot Metacharacter.

Name ▲	New Name
abc123	abc123

If I add a sub-expression after the Quantifier, `+.ab`, the EOL is reached with the `+`, so it backtracks in an attempt to match the new sub-expression, `'ab'`, but it also has to account for the dot metacharacter as well. When backtracking it re-evaluates the expression because of the `'+'` Quantifier. Thus, each character of the string is tested to match against the `.` <dot Metacharacter> and then the `a`, and the `b`, of the sub-expression, `ab`.

Match: `(.+ab)`

Name ▲	New Name
abc123	abc123

The `+` moves to EOL. The sub-expression, `ab` will **fail**. Why? because the Dot Metacharacter has to account for at least one character BEFORE the `ab` of `abc123` in order to match. When it backtracks and finally reaches the `a` in the string, the dot Metacharacter assumes the `a` value. When it tests for `a`, the position is at the `b`, and `b = a = false = Fails`.

Metacharacters in Depth

Quantifiers cont.

For our purposes, case closed. The match fails. But what really happens behind the scenes is that at this point, the RegEx Engine will move forward and attempt to test each subsequent character in the same manner. For example, the match attempt beginning with character 1, the **b** of **abc123**, evaluates the `.+` value as the remaining string, **bc123** and backtracks from there to attempt a match against `.ab` just as it did in the first attempt. This will continue until the string is exhausted and no more matches can be made. This happens in the match attempt at character 5, the **3** of **abc123**.

This can be seen using these debug logs from Regex Buddy:

```

Beginning match attempt at character 0
1 abc123
2 abc123backtrack
3 abc12
4 abc12backtrack
5 abc1
6 abc1backtrack
7 abc
8 abcbacktrack
9 ab
10 abbacktrack
11 a
12 abacktrack
13 backtrack
Match attempt failed after 13 steps

Beginning match attempt at character 1
1 bc123
2 bc123backtrack
3 bc12
4 bc12backtrack
5 bc1
6 bc1backtrack
7 bc
8 bcbacktrack
9 b
10 bbacktrack
11 backtrack
Match attempt failed after 11 steps

Beginning match attempt at character 2
1 c123
2 c123backtrack
3 c12
4 c12backtrack
5 c1
6 c1backtrack
7 c
8 cbacktrack
9 backtrack
Match attempt failed after 9 steps

Beginning match attempt at character 3
1 123
2 123backtrack
3 12
4 12backtrack
5 1
6 1backtrack
7 backtrack
Match attempt failed after 7 steps

Beginning match attempt at character 4
1 23
2 23backtrack
3 2
4 2backtrack
5 backtrack
Match attempt failed after 5 steps

Beginning match attempt at character 5
1 3
2 3backtrack
3 backtrack
Match attempt failed after 3 steps

```

So, it isn't just as simple as my analysis shows, but the analysis is provided to give you an idea of how the values are obtained and the matches are made, not a blow by blow description of every tiny step involved in the process.

Metacharacters in Depth

Quantifiers cont.

On the other hand, if I have:

string = abc123

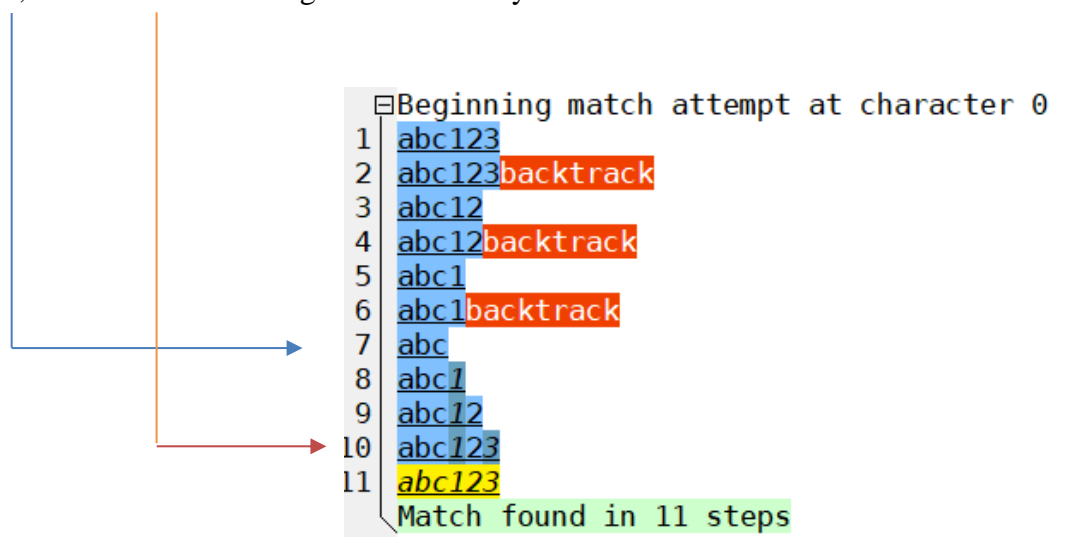
Match: (.+123)
 Replace: \1 <space>

Name ▲	New Name
abc123	abc123

Analysis:

- | | | |
|--------|--|---|
| 1. . | Match the dot Metacharacter to any character. | Capture Group 1 = a |
| 2. + | Make it Greedy . | Capture Group 1 = <entire string> |
| | Matches to <u>EOL</u> capturing the entire string. | |
| 3. 123 | Match against the literal string characters, '123'. | Capture Group 1 = abc123 |
| | Already at <u>EOL</u> . Backtracks to match against the '123' of 'abc123'. | Still remains as entire string after re-evaluation. The <space> in the Replace String permits New Name. |

What is happening here is that once it reaches the EOL in step 2, it backtracks, BUT it still has to account for the initial dot Metacharacter of step 1 to match against the character PRIOR to the **123**. In other words, in order to match, there has to be at least one character in the position before the match of **123** can take place. In this example, the dot Metacharacter matches against the **c**, and the **123** following are successfully matched.



Metacharacters in Depth

Quantifiers cont.

- ? Matches the previous (character or metacharacter, Capture Group or class) **zero or one time**
(The question mark makes the previous optional. - Lazy)

This is referred to as Lazy, Non-Greedy, Reluctant and Optional. It means to do as little work as possible to get the job done. Unlike other Quantifiers, which take and take as much as possible and give back only when they have to, a Lazy Quantifier starts off with very little and adds if required. In Regular Expressions, a Greedy Quantifier can *Decrease* the number of already matched characters and a Lazy Quantifier can *Expand* to include additional characters to match against. Zero or One Time refers to the Quantifier match as optional or Zero, or one time, a single successful match. If the match cannot be made, then the optional path is taken. The ' ? ' Quantifier is Greedy in the sense that it will attempt to match overall at least once rather than zero.

Example:

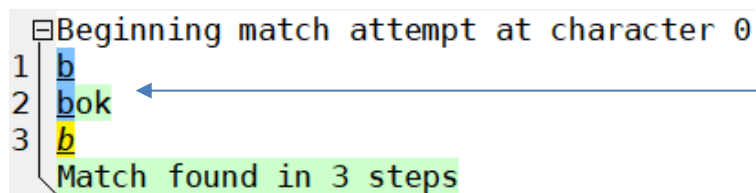
1) `ba?` matches **b** or **ba**.

Match: `(ba?)`
Replace: `\1 <space>`

Name ▲	New Name
b	b
ba	ba

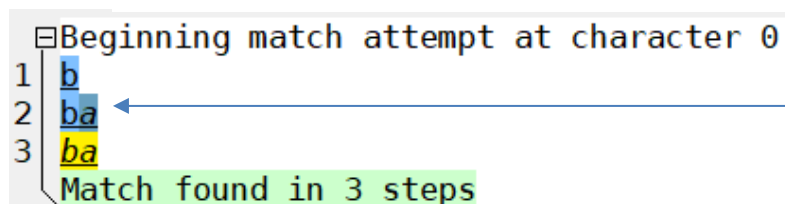
String = b

Looking for a b followed by an a, but the a is made optional and therefore **if b is found but the a is not, that's okay**. The match still succeeds.



String = ba

Looking for a b followed by an optional a. **If the b is found and so is the a, because it does exist, then what the hell**. The match succeeds.



Metacharacters in Depth

Quantifiers cont.

Example:

2) `def?` matches `def` in string, `abc 'def' 'ghi' jkl`

Match: `(def?)`
 Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	def

looking for `de` followed by an optional `f`.

Example:

3) `(def)?` matches all in string, `abc 'def' 'ghi' jkl`

Match: `(def?)`
 Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	<code>\1</code>

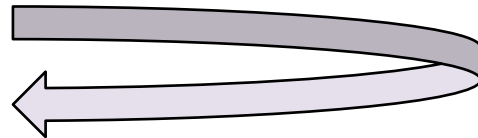
```

Beginning match attempt at character 0
1 backtrack
2 ok
Match found in 2 steps
    
```

looking for an optional `def`.

BRU returns the first match which matches against the `abc`. How does it match at all? When it attempts to match against the sub-expression, `def`, made **optional**, it matches `abc` but it returns a **null** value as a Zero Occurrence Match. Capture Group 1 doesn't exist at this point and this is why the Replace String of `\1 <space>` returns `\1` as **Invalid**.

.....Start·Length·...		
Match 1:	0	0 C L R F
Match 2:	1	0 C L R F
Match 3:	2	0 C L R F
Match 4:	3	0 C L R F
Match 5:	4	0 C L R F
Match 6:	5	3 C L R F
Match 7:	8	0 C L R F
Match 8:	9	0 C L R F
Match 9:	10	0 C L R F
Match 10:	11	0 C L R F
Match 11:	12	0 C L R F
Match 12:	13	0 C L R F



```

Beginning match attempt at character 5
1 d
2 de
3 def
4 def
Match found in 4 steps
    
```

etc.

It is not until character 5 that 'def' is matched.

Metacharacters in Depth

Quantifiers cont.

Notes:

1. A little more explanation may be required.

The match of every character in the string is accomplished because the RegEx is saying, match **def** if it exists. If it does exist, then capture the value into Capture Group 1. If it doesn't exist, then technically this is still a match, a Zero Occurrence Match. Nothing is captured in a Capture Group because Capture Group 1 will only exist when a successful match is made for the **def** characters and this is not until character 5 under the sixth match in the string.

BRU only concerns itself with the first match performed at character 0. The value was **null** because of the Zero Occurrence Match but Capture Group 1 won't exist until the match of def, so nothing, not **null** or anything else, was captured. The value may have been **null**, but this was not captured, so the returned value to BRU is 'nothing'.

Because Capture Group 1 does not exist at the first match, the reference to Capture Group 1 is void in the Replace String and instead all of the characters are evaluated as literal string characters. This means that New Name has a value of the characters, ' \ ' ' 1 ' and <space>.

The backslash character is an illegal character in a filename, therefore the name is flagged as **Invalid**. To us, though, because the Group Reference literally appears in New Name, it is an indication that the fault with the RegEx may lie with the Group Reference. When troubleshooting, this gives a place to start.

Name ▲	New Name
abc 'def' 'ghi' jkl	\1

2. The reason that it displays in **red** is because of the non-existent Capture Group. If the Replace String references it as it does here using \1, New Name will display both the backslash and the Capture Group Reference number '1' as a literal. Because the backslash is an illegal character in a Windows filename, thus the **Invalid** red flag.

If I change the Replacement String and remove the backslash character:

Name ▲	New Name
abc 'def' 'ghi' jkl	1

You will see that New Name is no longer **Invalid**, even though there is no value for Capture Group 1 and without the backslash, the Replace String is no longer referencing a Capture Group, but a literal character, '1' followed by a <space>.

Metacharacters in Depth

Quantifiers cont.

Example:

4) `abc?` matches `abc` in string, `abc 'def' 'ghi' jkl`

Match: `(abc?)`

Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	abc

looking for an `ab` followed by an optional `c`.

```

Beginning match attempt at character 0
1 | a
2 | ab
3 | abc
4 | abc
  | Match found in 4 steps
Beginning match attempt at character 3
  | Match attempt failed after 2 steps
Beginning match attempt at character 4
  | Match attempt failed after 2 steps
Beginning match attempt at character 5
  | Match attempt failed after 2 steps
Beginning match attempt at character 6
  | Match attempt failed after 2 steps
Beginning match attempt at character 7
  | Match attempt failed after 2 steps

```

etc.

all other matches fail.

Metacharacters in Depth

Quantifiers cont.

Example:

5) (abc)? matches (all) abc in string, `abc 'def' 'ghi' jkl`

Match: (abc)?
Replace: \1 <space>

Name ▲	New Name
abc 'def' 'ghi' jkl	abc

looking for an optional abc.

Because `abc` is in the first part of the string, it is immediately matched and captured by BRU.

```

Beginning match attempt at character 0
1 a
2 ab
3 abc
4 abc
  Match found in 4 steps
Beginning match attempt at character 3
  Match found in 2 steps
Beginning match attempt at character 4
  Match found in 2 steps
Beginning match attempt at character 5
  Match found in 2 steps
Beginning match attempt at character 6
  Match found in 2 steps
Beginning match attempt at character 7
  Match found in 2 steps

```

All other characters are matched as well, but BRU only displays the first match under v1.

Metacharacters in Depth

Quantifiers cont.

Example:

5) `colou?r` matches both **colour** and **color** because the u becomes **optional**

Match: `(colou?r)`
 Replace: `\1 <space>`

Name ▲	New Name
colour	colour
color	color

String = colour

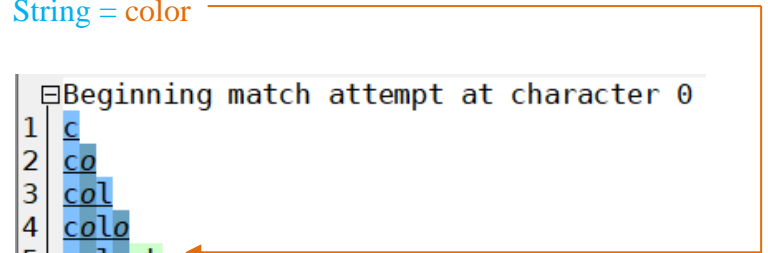
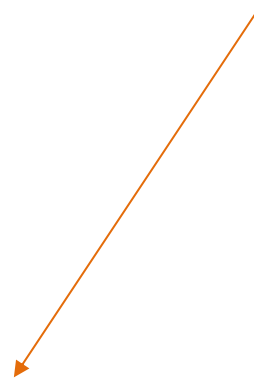
```

Beginning match attempt at character 0
1 c
2 co
3 col
4 cola
5 colou
6 colour
7 colour
Match found in 7 steps
  
```

String = color

```

Beginning match attempt at character 0
1 c
2 co
3 col
4 cola
5 colook
6 color
7 color
Match found in 7 steps
  
```



Metacharacters in Depth

Quantifiers cont.

Example:

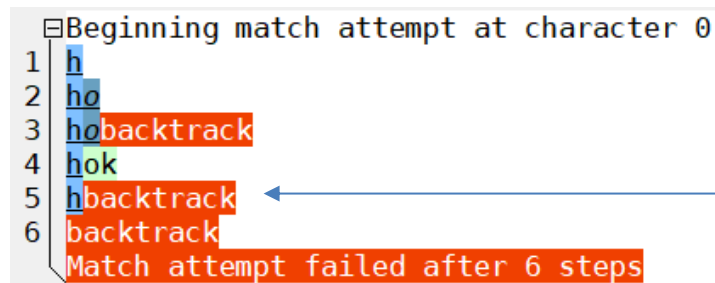
- 6) `ho?p` matches `hp`, and `hop`, but not `hoop`

Match: `(ho?p)`
 Replace: `\1 <space>`

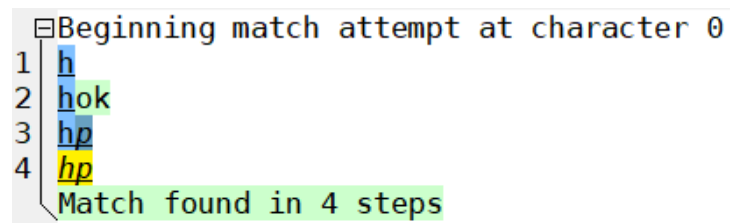
Name ▲	New Name
hp	hp
hop	hop
hoop	hoop

Look for an `ho` optionally followed by a `p`.

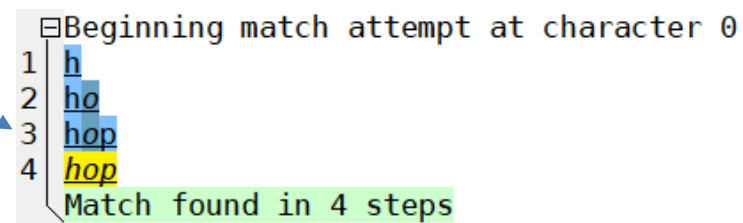
`hoop` doesn't match because there must be 1 character, the `h`, followed or not by a second character, an `o`, that must immediately precede a `p`. It is the third character of `hoop`, the second `o`, that fails the Regex because it is not a `p`.



`h <optional o> p` matches `hp`



`h <optional o> p` matches `hop`



Metacharacters in Depth

Quantifiers cont.

Multiple Lazy Quantifiers can be used within one RegEx expression to match more than one alternative.

Example:

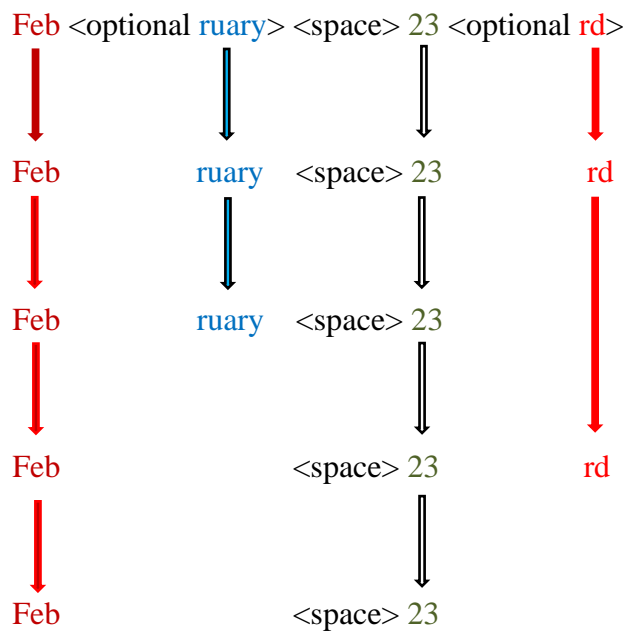
Feb(ruary)? 23(rd)? matches February 23rd, February 23, Feb 23rd and Feb 23

Match: (Feb(ruary)? 23(rd)?)

Replace: \1 <space>

Name ▲	New Name
February 23rd	February 23rd
February 23	February 23
Feb 23rd	Feb 23rd
Feb 23	Feb 23

looks for Feb optionally followed by ruary, immediately followed by a <space> 23, followed optionally by rd.



Notes:

1. The Lazy Quantifier can also be used with a Greedy Quantifier, ‘ *? ’ or ‘ +? ’. This will make the Greedy Quantifier Lazy. This will be discussed a little later.

Metacharacters in Depth

Quantifiers cont.

?? Matches the previous (character or metacharacter, Capture Group or class) **zero or one time** (The double question marks makes the previous optional. – Lazy or Non-Greedy)

The difference between using the ‘?’ and the ‘??’ is that the ‘?’ is Greedy in that it will strive to match at least once whereas ‘??’ is really lazy in that it will strive to match zero rather than attempt to match once and even then only if required for the match. Meaning, that while using ‘?’ would match the optional character if available in the string, ‘??’ will not.

Example:

1) `abc??` matches `ab` in string `ab` or `abc`

Match: `(abc??)`
Replace: `\1 <space>`

Name ▲	New Name
ab	ab
abc	ab

Looks to match `ab` followed immediately by an optional `c`, but because of the Lazy Quantifier, even if `c` exists in the string, the optional path will be taken and only `ab` will match.

```

Beginning match attempt at character 0
1 a
2 ab
3 abok ←
4 ab
Match found in 4 steps
  
```

So what would force the RegEx to consider the ‘c’ necessary enough to match?

One thing I can think of off hand is to use an End of Line Anchor, \$.

Match: `(abc???)$`
Replace: `\1 <space>`

Name ▲	New Name
ab	ab
abc	abc

```

Beginning match attempt at character 0
1 a
2 ab
3 abok
4 ab backtrack
5 abc
6 abcok
7 abc
Match found in 7 steps
  
```

`ab` still matches because the `c` is still optional. If the `c` doesn't exist in the string, even with the anchor, it will still match, but if the `c` does exist then in order to successfully match, the `c` must be included in the match if in fact the `ab` along with the `c` is at the end of the string. If the `c` does exist but is not at the end of the string, the RegEx will fail.

Metacharacters in Depth

Quantifiers cont.

Example:

2) `def??` matches `de` in string, `abc 'def' 'ghi' jkl`

Using `(def?)` matched `def` where using `(def??)` matches `de`.

Match: `(def??)`
 Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	de

`(def??)` looks to match `de` optionally followed by `f`. However, if all three characters are available, then the optional path is still taken matching only the first two characters. Really Lazy.

The same with this -

Match: `(jkl??)`
 Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	jk

... but not this

Match: `(jkl??)$`
 Replace: `\1 <space>`

Name ▲	New Name
abc 'def' 'ghi' jkl	jkl

The End of String anchor, `'$'` forces the match against the third character `l`, because the match cannot be made unless the last character of the string is an `l`. Therefore, although `jk` is matched, so must the `l` be as well in order for the RegEx to successfully match.

Metacharacters in Depth

Quantifiers cont.

Just as:

Match: (jkl)??
 Replace: \1 <space>

... results in –

Name ▲	New Name
abc 'def' 'ghi' jkl	\1

Looks for an optional jkl, but even though the string contains **jkl**, the Lazy Quantifier dismisses it and takes the optional path, creating a Zero Occurrence Match.

.....	Start	Length	...
☒ Match 1:	0	0	^C _R ^L _F
☒ Match 2:	1	0	^C _R ^L _F
☒ Match 3:	2	0	^C _R ^L _F
☒ Match 4:	3	0	^C _R ^L _F
☒ Match 5:	4	0	^C _R ^L _F
☒ Match 6:	5	0	^C _R ^L _F

... adding the anchor, \$

Match: (jkl)??\$
 Replace: \1 <space>

... forces the match –

Name ▲	New Name
abc 'def' 'ghi' jkl	jkl

Looks for an optional jkl that must be positioned at the end of the string in order to match.

☒ Beginning match attempt at character 16	
1	ok
2	backtrack
3	j
4	jk
5	jkl
6	jkl
7	jkl ok ←
Match found in 7 steps	

Metacharacters in Depth

Quantifiers cont.

***?** Match the previous character, expression or metacharacter **zero or more times**
(As few repetitions as possible – Made Lazy or Non-Greedy)

Example:

1) A.*?Z matches **AiiZ**, and if Global set (PCRE v2), **AoooZ** in string, **eeeAiiZuuuuAoooZeeee**

In many respects, Lazy can also mean that the previous sub-expression is made optional. If, during the evaluation of a sub-expression that captured values, that sub-expression is made optional, then the previous values of that match are given up, but the match still exists but with no value. This is referred to as a Zero Occurrence Match with a **null** value and a zero length. This is not one of those examples.

Normally when the **dot Metacharacter** is paired with the *** Greedy Quantifier**, **.***, it matches from that point on to the end of the string, but when made Lazy by the Non-Greedy Metacharacter, **.*?**, the sub-expression is made optional. The RegEx engine returns to the point just after the last match and instead matches only until the successful match of the next sub-expression.

Match: (A.*?Z)
Replace: \1

Name	New Name
eeeAiiZuuuuAoooZeeee	AiiZ

1. A Match against the uppercase letter, 'A'.
Matches against the first occurrence of the 'A' of 'eeeAiiZuuuuAoooZeeee'.
2. . Match against any character.
Matches against the first occurrence of the 'i' of 'eeeAiiZuuuuAoooZeeee'.
3. * Make it Greedy.
4. ? But not too Greedy.

Capture Group 1 = A

Capture Group 1 = Ai

Capture Group 1 = AiiZuuuuAoooZeeee

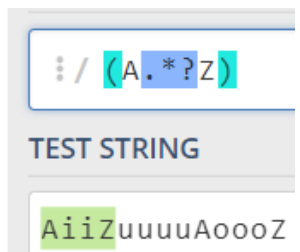
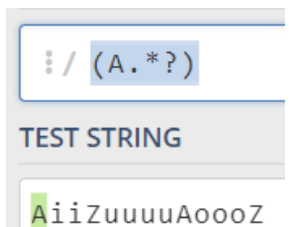
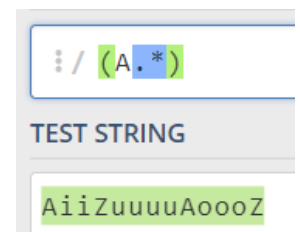
Capture Group 1 = A

What happened here is when the sub-expression **.*** was made Lazy, it made the sub-expression optional as if it never existed. The RegEx Engine returned to the point just before the **.*** was evaluated, and this would be just after the match of the uppercase **A**.

5. Z Match against the uppercase letter, 'Z'.
Matches against the first occurrence of the 'Z' of 'eeeAiiZuuuuAoooZeeee'.

Notes:

1. Although the **.*** was made optional, there is no Zero Occurrence Match that results because Capture Group 1 already holds the **A** value previously captured. Zero Occurrence Matches are more likely with using the Greedy Quantifier **+** made Lazy using **.*?** than with ***?**, but I will provide an example that does.



Metacharacters in Depth

Quantifiers cont.

Example:

String = eeeAiiZuuuuAoooZeeee

If the A was removed from the RegEx ...

Match: (.?*Z)

Replace: \1

Name ▲	New Name
eeeAiiZuuuuAoooZeeee	eeeAiiZ

Analysis:

- . Match against any character.
Matches against the first occurrence of the lowercase 'e' of 'eeeAiiZuuuuAoooZeeee'
- * Make it Greedy.
Matches against the entire string.
- ? But not too Greedy.
The .* is made optional. Therefore the RegEx Engine handles the last sub-expression, .*, as if it never existed. This changes the position to right before the beginning of the string resulting in a **null** value with a zero length.

Capture Group 1 = e

Capture Group 1 = <entire string>

Capture Group 1 = **null**

	Start	Length	...
Match 1:	0	0	<small>CL RF</small>
Group 1:	0	0	<small>CL RF</small>

This is a Zero Occurrence Match as some refer to it. Capture Group 1 previously held the value of the entire string. When the sub-expression that originally matched and created this value was made optional, any value held by the Capture Group as a result of this was removed, BUT the Capture Group was not **Invalidated**, only the match was given back. Thus, Capture Group 1 exists but holds no value. It is empty, or **null** with a match that is **zero length**.

- Z Match against an uppercase 'Z'.
Matches against the first occurrence of 'Z' of 'eeeAiiZuuuuAoooZeeee'.

Capture Group 1 = eeeAiiZ

The current position is at the beginning of the string. The RegEx Engine moves forward until it encounters and matches against the first occurrence of the uppercase **Z**, capturing all characters as it proceeds in Capture Group 1.

Notes:

- Zero Occurrence Match is a term I came across that really is just a better explanation for a type of Zero Length Match.
- In step 3, the position just before the first character is the **BOL** (Beginning of Line).

Metacharacters in Depth

Quantifiers cont.

Notes:

1. When comparing against greedy, `A.*Z` matches **AiiZuuuuAoooZ**. It doesn't stop when matching for an occurrence of an uppercase `A` with some text following and ending with an uppercase `Z`. It will continue until the last match, meaning no more matches are available, or what is referred to as the string being 'exhausted', or until it reaches EOL, whichever comes first.
2. BRU using the Match String of `(A.*Z)` would only be able to return the first value in Capture Group – `\1 = AiiZ`. To see the second result, you would have to account for the 'u' characters so the Match String would instead be:

Match: `(A.*Z)uuuu(A.*Z)`
 Replace: `Group 1 = \1 Group2 = \2`

Name ▲	New Name
eeeAiiZuuuuAoooZeeee	Group 1 = AiiZ Group2 = AoooZ

or

Match: `(A.*Z)u{4}(A.*Z)`
 Replace: `Group 1 = \1 Group2 = \2`

Name ▲	New Name
eeeAiiZuuuuAoooZeeee	Group 1 = AiiZ Group2 = AoooZ

where:

`/1 = AiiZ` `\2 = AoooZ`.

Metacharacters in Depth

Quantifiers cont.

Example:

2) a.*?b matches the first match of **ab** in string **abab** as compared with the Greedy equivalent of a.*b that matches the entire string, **abab**.

Match: (a.*?b)
 Replace: \1

Name ▲	New Name
abab	ab

Analysis:

1. a Match against the literal lowercase 'a'.
Matches against the first lowercase 'a' of 'abab'.
2. . Match against any character.
Matches against the 'b' of 'abab'.
3. * Make it Greedy.
4. ? But not too Greedy.
the * is made optional. The RegEx Engine returns to the point after the last successful match, the 'a' of 'abab' in step 1.
5. b Match against the literal lowercase 'b'.
Matches against the first lowercase 'b' of 'abab'

Capture Group 1 = a

Capture Group 1 = ab

Capture Group 1 = <entire string>

Capture Group 1 = a

Capture Group 1 = ab

Match: (a.*b)
 Replace: \1 <space>

Name ▲	New Name
abab	abab

1. a Match against the literal lowercase 'a'.
Matches against the first lowercase 'a' of 'abab'.
2. . Match against any character.
Matches against the 'b' of 'abab'.
3. * Make it Greedy.
Current position = EOL
5. b Match against the literal lowercase 'b'.
Already at EOL. **Backtracks** to match against the second occurrence of the lowercase 'b' of 'abab'.

Capture Group 1 = a

Capture Group 1 = ab

Capture Group 1 = <entire string>

Capture Group 1 = abab

☐ Beginning match attempt at character 0

```

a
abab
abab backtrack
aba
abab
abab

```

Match found in 6 steps

Metacharacters in Depth

Quantifiers cont.

+? Matches the previous character, expression or metacharacter **one or more** times
(The question mark makes the previous optional. - Lazy)

Lazy, so the optional item is excluded in the match if possible.

Example:

1) ".+?" matches "def" and "ghi" in string, abc "def" "ghi" jkl

abc "def" "ghi" jkl

To understand this, let's first change it to Greedy:

“.+”

Now this matches the entire value of...

“def” <space> “ghi”

abc "def" "ghi" jkl

String = abc 'def' 'ghi' jkl

Match: ‘.+’

Replace: \1 <space>

Name ▲	New Name
abc 'def' 'ghi' jkl	'def' 'ghi'

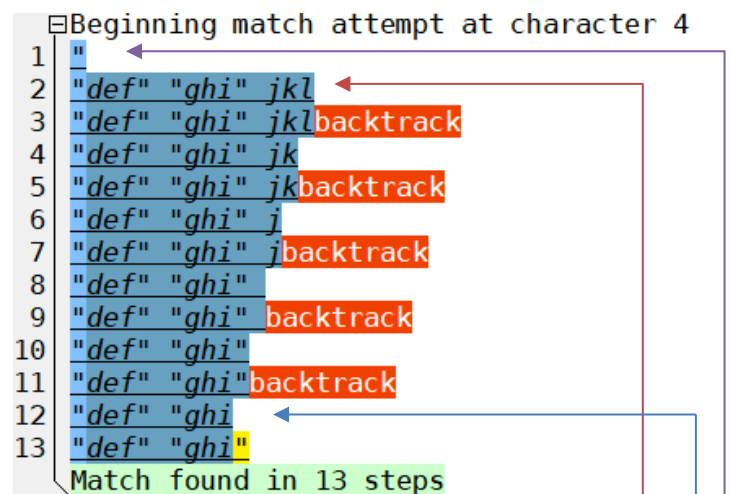
1. ‘ Match against the single quotation mark.
Matches the first single quotation after ‘ abc <space> ’
2. . Dot Metacharacter. Match against any character.
Matches against the ‘d’ of ‘ ‘def’ ’
3. + Make it Greedy.
Matches to EOL.
Current position = EOL
4. ’ Match against the single quotation mark.
Already at EOL. Backtracks to match against the single quotation after ‘ghi’

RegEx satisfied. Single Match = "def" "ghi"

When it matches against the final single quotation mark in step 4, it gives up the characters, **jkl** of the previous match.

Notes:

1. Changed to single quotation marks because double quotation marks are illegal characters in Windows File Names.



Capture Group 1 = ‘

Capture Group 1 = ‘d

Capture Group 1 = ‘def’ ‘ghi’ jkl

Capture Group 1 = ‘def’ ‘ghi’

Metacharacters in Depth

Quantifiers cont.

Now let's examine the current example, Lazy.

String = abc 'def' 'ghi' jkl

Match: ('.+?')

Replace: \1

Name ▲	New Name
abc 'def' 'ghi' jkl	'def' 'ghi' jkl

Analysis:

- | | |
|--|-----------------------------------|
| 1. ' Match against the single quotation mark
Matches against the first single quotation mark of 'def'. | Capture Group 1 = ' |
| 2. . Match against any character = 'd' | Capture Group 1 = 'd |
| 3. + Make it Greedy.
Current position = <u>EOL</u> | Capture Group 1 = 'def' 'ghi' jkl |
| 4. ' Match against single quotation mark.
Already at <u>EOL</u> . Backtracks to match against the second single quotation mark of 'ghi'. | Capture Group 1 = 'def' 'ghi' |
| 5. ? Makes the previous sub-expression optional.
The match of the single quotation in step 4 is given up.
The RegEx Engine returns to the last successful match of step 3. | Capture Group 1 = 'def' 'ghi' jkl |

When the sub-expression in step 4 is made optional, it is handled as if it was never evaluated. Any value resulting from the match is also removed. The previous captured value is restored from step 3.

Metacharacters in Depth

Quantifiers cont.

String = abc 'def' 'ghi' jkl

Match: (.+?)
 Replace: \1 <space>

Name ▲	New Name
abc 'def' 'ghi' jkl	'def'

Analysis:

- ‘ Match against the single quotation mark
Matches against the first single quotation mark of ‘def’.
Capture Group 1 = ‘
- . Match against any character = ‘d’
Capture Group 1 = ‘d
- + Make it Greedy.
Current position = EOL
Capture Group 1 = ‘def’ ‘ghi’ jkl
- ? But not too Greedy.
Makes the previous statement optional. The Quantifier, ‘ + ’
is made ‘Lazy’ or optional. The RegEx engine throws out the
match in step 3 and the RegEx Engine returns to the point of the
last successful match in step 2.
Capture Group 1 = ‘d
- ’ Match against the single quotation mark.
Current position is the ‘d’ of ‘def’.
Moves forward to match against the second single quotation
mark of ‘def’. First match is satisfied.
Capture Group 1 = ‘def’

Remember, at this point, BRU under v1, will only return the first match. This doesn’t stop the behaviour of the RegEx engine that will continue to move forward testing each character searching for any additional pattern matches until it hits EOL, and exhausts the string.

6. Finds a second match of ‘ ‘ghi’ ’.
7. Moves forward continually testing and reaches EOL without any further matches.
String is exhausted.

RegEx satisfied. Two separate matches = "def" and "ghi"

Notes:

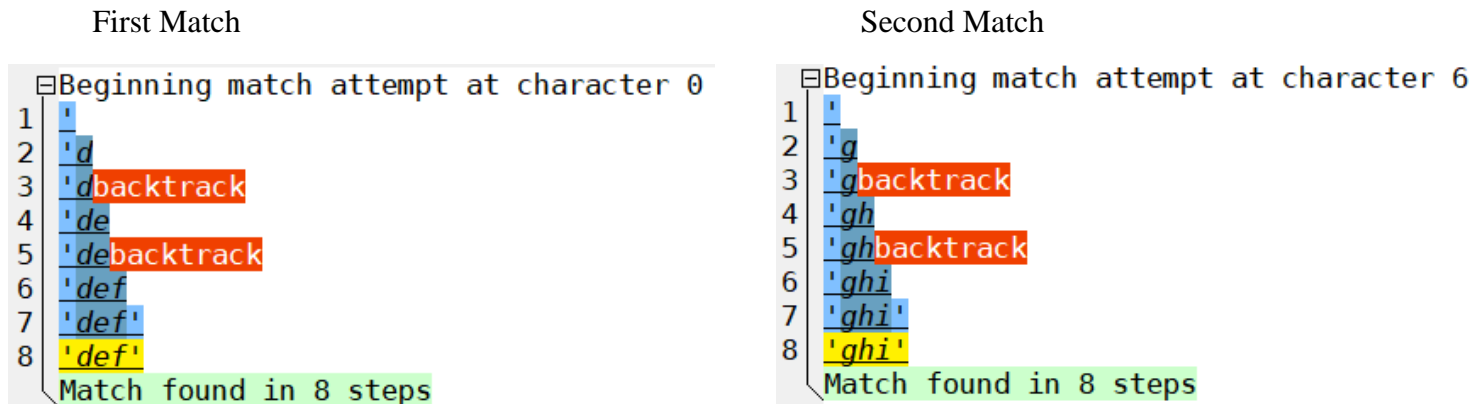
1. Exhausting the string refers to no more matches can be made. The string has reached the EOL but can no longer backtrack. The RegEx evaluation is considered finished, match or no match.

Metacharacters in Depth

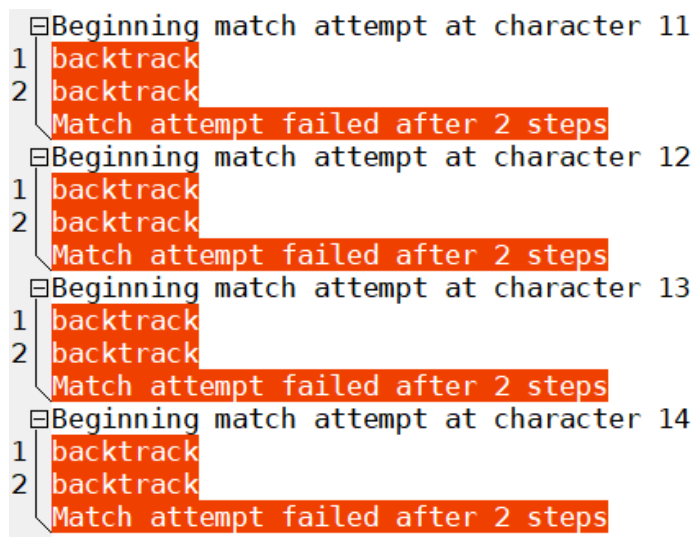
Quantifiers cont.

Notes:

1. The RegEx engine can move to positions between each character, but to simplify things, I use the character positions in the descriptions. Further information can be found in Volume II.
2. Actually, the RegEx engine does not simply move forward, it looks for the first character, the single quotation, finds it and moves forward and then back, tests against each additional character until it matches against the second single quotation mark. It looks like this:



It will continue to test each character looking to match against the same pattern as it did in the first two matches, but it will not match and it will fail when it reaches EOL in the match attempt at character 14, as demonstrated:



I am not going to address all of this background minutia on a regular basis. Instead, I will concentrate on only pertinent changes.

Metacharacters in Depth

Quantifiers cont.

Range Quantifiers

Also called repetition or Minimum Maximum Quantifiers, a Range Quantifier specifies a limitation of the number of consecutive matches that can be made using a minimum-maximum range where one value specified would limit the range to an exact number of matches, and two values would specify a range where the first value is the minimum number of matches and the second value would represent the maximum number of matches. Range Quantifiers are regarded as Greedy.

`{n}` specifies exactly how many matches are allowed. Match previous character or metacharacter exactly *n* times. When only one value is specified, this value represents both the minimum and maximum values.


Example:

1) `a{4}` matches four 'a' s

String = `aaaaa`

Match: `(a{4})`

Replace: `\1 <space>`

Name ▲	New Name
 aaaaa	aaaa

1. `a` Match against a literal 'a' string character. Matches the first 'a' of 'aaaaa'
2. `{4}` Range Quantifier. Repeat the Match for a total of 4 times. Matches three additional consecutive 'a' characters.


Capture Group 1 = a

Capture Group 1 = aaaa

The matches must be consecutive. This will not work:

Match: `(a{4})`

Replace: `\1 <space>`

Name ▲	New Name
 a a a a a	a a a a a

1. `a` Match against a literal 'a' string character. Matches the first 'a' of 'a a a a a'
2. `{4}` Range Quantifier. Repeat the Match for a total of 4 times. `<space> = a = false`. **Fails.**

Capture Group 1 = a

Metacharacters in Depth

Quantifiers cont.

{n} specifies exactly how many matches are allowed cont.

Example:

2) (Ho){3} matches the string, 'Ho' for three matches.

String = Santa laughs HoHoHo

Match: ((Ho){3})

Replace: \1 <space>

Name ▲	New Name
<input type="checkbox"/> Santa laughs HoHoHo	HoHoHo

Analysis:

1. Ho Match against the string, 'Ho'.
Matches the first 'Ho' of 'HoHoHo'.
2. {3} Range Quantifier.
Repeat the match for a total of 3 times.
Matches two additional consecutive 'Ho' strings.

Capture Group 1 = Ho

Capture Group 1 = HoHoHo

Example:

3) a{2} matches 'a' for two matches

Match: (a{2})

Replace: \1 <space>

Name ▲	New Name
<input type="checkbox"/> aa	aa
<input type="checkbox"/> aaa	aa
<input type="checkbox"/> aaaaa	aa
<input type="checkbox"/> aaab	aa
<input type="checkbox"/> aaaab	aa
<input type="checkbox"/> baaa	aa
<input type="checkbox"/> baaac	aa

An exact value of 2 is specified. It is important to note that by specifying an exact number does not preclude matching against strings that consecutively contain more than the required number of potential matches.

In other words, this RegEx is not a method by which to limit the match against only those strings that contain the fixed number of matches represented by the minimum value.

What a{2} really means is to match against strings that contain at minimum 2 matches of the literal character string, 'a'. And because there is only one value specified, this value represents both the minimum and maximum values. The equivalent would be {2,2}. This is why all the above sample strings that contain at least two or more consecutive **a** characters will match. I cannot limit the match to only the sample string, **aa**, not using this RegEx, anyway.

Metacharacters in Depth

Quantifiers cont.

{*min*, *max*} limits how many times a character or character set can be repeatedly matched. Match previous character or Metacharacter at least *min* times but more than *max* times (cannot be higher than 65536).

where:

min – the minimum number of matches

max – the maximum number of matches

Example:

`a{3,4}` Matches no less than three and no more than 4 consecutive matches.

Match: `(a{3,4})`
 Replace: `\1 <space>`

Name ▲	New Name
<input type="checkbox"/> aa	aa
<input type="checkbox"/> aaa	aaa
<input type="checkbox"/> aaaaa	aaaa
<input type="checkbox"/> aaaab	aaaa
<input type="checkbox"/> aaab	aaa
<input type="checkbox"/> ab	ab
<input type="checkbox"/> baaa	aaa
<input type="checkbox"/> baaac	aaa

Looks to match the literal string character, a, a minimum of 3 matches and a maximum of 4 matches. This matches string samples that have a minimum of three consecutive **a's** in the string. In the RegEx, it doesn't matter if the a's appear at the beginning, middle or end of the string, as long as there are three or more consecutive matches it will be satisfied. If there are more than four potential matches, only the first four will be matched. If there are less than three potential matches, the RegEx will fail.

For example, the string, **aaaaa** is made up of 5 potential matches, but New Name only recognizes the first four, **aaaa**.

Notes:

1. If I want to limit the number of 'a's matched in a string and have that string restricted to that number of characters, I can use this:

Match: `(^a{2}$)`

This will match the string, **aa** but not **aaa**, because the Anchors limit the characters to the value specified by the Range Quantifier. The Anchors limit the match to 2 characters that must match at both the beginning and end of the string. For this to happen, the first **a** must match at the BOL and the second consecutive **a** matches at the EOL.

Metacharacters in Depth

Quantifiers cont.

`{n,}` specifies that it can match *n* or more times. Match previous character or metacharacter at least *n* times.

Example:

`a{3,}` Match 'a' three or more times.

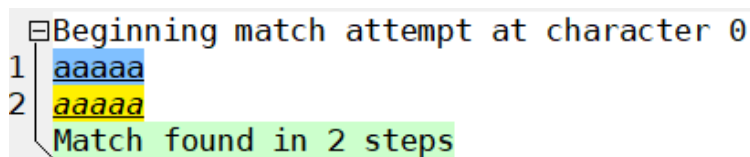
The syntax is specifying a minimum and maximum, but the maximum is only referenced and not specified. This is the same as saying that there is no upper limit on the range.

Match: `(a{3,})`
 Replace: `\1 <space>`

Name ▲	New Name
<input type="checkbox"/> aa	aa
<input type="checkbox"/> aaa	aaa
<input type="checkbox"/> aaaaa	aaaaa
<input type="checkbox"/> aaab	aaa
<input type="checkbox"/> aaaab	aaaa
<input type="checkbox"/> baaa	aaa
<input type="checkbox"/> baaac	aaa

Looks to match a literal string character, a, that in order to match, must be matched a minimum of three times with no upper limit on the maximum value.

String = aaaaa



1. a Match against literal string character, 'a'. Capture Group 1 = a
Matches against the first 'a' of 'aaaaa'
2. {3,} Range Quantifier. Capture Group 1 = aaaaa
Match a minimum of three times with no upper limit.
Matches against the remaining consecutive 'a' characters of 'aaaaa'.

Notes:

1. I cannot use this RegEx to only match against the **aa** sample string. For example, this will not work: `(a{2})` because it only specifies a minimum number of the matches and does not indicate that the string cannot contain more than 2 **a** characters. It consequently has no effect on **aaa**, **aaaa**, **aaab**, etc. They will always return the value of aa. See previous notation 1 under the discussion of `{min, max}` for a RegEx that will accomplish this.

Metacharacters in Depth

Quantifiers cont.

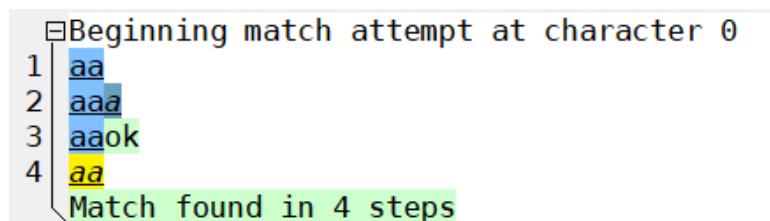
Oh, I could use a Lookahead in combination with a negated class that would disallow strings that have only two matches:

Match: `(a{2}(?!=[^$]))`
 Replace: `\1 <space>`

Name ▲	New Name
aa	aa
aaa	aa
aaaaa	aa

Look for 2 literal a string characters not immediately followed by the EOL. This will preclude the sample string, **aa** because it does have the EOL after the two **a**'s.

String = aaa



But I cannot use the RegEx to only match against the **aa** sample string (see previous notation for a RegEx that will).

Analysis:

- | | | |
|---------------|---|----------------------|
| 1. a | Match against literal string character, 'a'. | Capture Group 1 = a |
| 2. {2} | Range Quantifier.
Repeat match for one additional time.
Matches against the second 'a' of 'aaa' | Capture Group 1 = aa |
| 3. (?!=[^\$]) | Positive Lookahead.
Look ahead if the next character is <i>NOT</i> <u>EOL</u> .
a = EOL = false = True. | |

This uses a Positive Lookahead in conjunction with a negated class that consist of the end of string anchor represented by the '\$' character.

The current position is the second **a** of **aaa** from Step 2. The Lookahead in step 3 looks to the right testing the next character for the end of the string. Because the next character is the third **a** and not the EOL, the test for EOL is **false**, thereby proving the test as **true** in the statement represented by the negated class.

The Range Quantifier is used to disallow or preclude those strings that only have a single **a** character. The Lookahead is used test is to disallow or preclude strings that only have 2 consecutive **a** characters in the string.

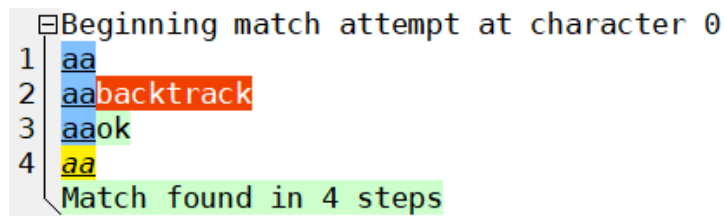
Metacharacters in Depth

Quantifiers cont.

In the previous example, I used a Lookahead with a negated class, but really all I had to do was use a Negative Lookahead:

Match: `(a{2}(?!$))`
 Replace: `\1 <space>`

Name ▲	New Name
aa	aa
aaa	aa
aaaaa	aa



However, using the negated class resulted in no backtracking, so there are *madness* to my methods ☺ I wasn't trying to impress, but just to illustrate that there are usually different methods available to accomplish the same task using Regular Expressions.

For more information on Lookaheads, refer to the sections on Lookarounds here and in the appendix of Volume II.

More Examples:

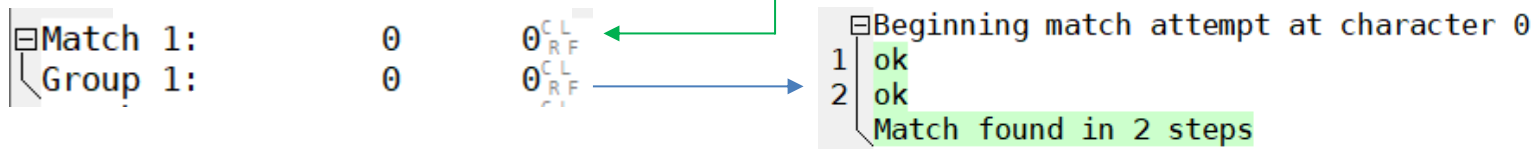
Match: `(a{0,2})`
 Replace: `\1 <space>`

Name ▲	New Name
aa	aa
aaa	aa
aaaaa	aa
aaab	aa
aaaab	aa
baaa	
baaac	

Match: `(a{2,2})`
 Replace: `\1 <space>`

Name ▲	New Name
aa	aa
aaa	aa
aaaaa	aa
aaab	aa
aaaab	aa
baaa	aa
baaac	aa

The strings, **baaa** and **baac** resulted in a Zero Occurrence Match in the example on the left because, although it matched, the minimum was set at zero, meaning match even if the 'a' is not found. Because the first character was a **b**, it matched but the value returned was **null**.



Metacharacters in Depth

Quantifiers cont.

Here's another similar example.

String = aabcz

Match: (a{0,}z)
 Replace: \1 <space>

Name ▲	New Name
aabcz	z

Analysis:

1. a Match against literal string character, 'a'. Capture Group 1 = a
2. {0,} Range Quantifier. Repeat match 0 (optional) to infinity (no upper limit). Capture Group 1 = aa
3. z Match against literal string character, 'z'. Capture Group 1 = z

In step #2, it doesn't just match against the two **a**'s, but because of the specified lower limit of 0 in the Quantifier, this is the same as making it optional to match or not, meaning that it will match against any character whether or not it is an **a**. Because of this, when the Range Quantifier is done being evaluated, the current position is at EOL. At this point in Step #3, the RegEx Engine has to backtrack to match against the z and in doing so, it gives up the **aa**, the characters of the first match in Step #1 and Step #2, leaving Capture Group 1 with a remaining value of **z**. It looks like this:

The image displays four debug log windows for the regex engine processing the string 'aabcz'. A yellow arrow points from the first character 'a' to the final successful match log.

- Beginning match attempt at character 0:** Shows steps 1-7. Step 1: 'aa' (ok), Step 2: 'aa' (backtrack), Step 3: 'a' (ok), Step 4: 'a' (backtrack), Step 5: 'ok', Step 6: 'backtrack', Step 7: 'backtrack'. Match attempt failed after 7 steps.
- Beginning match attempt at character 1:** Shows steps 1-5. Step 1: 'a' (ok), Step 2: 'a' (backtrack), Step 3: 'ok', Step 4: 'backtrack', Step 5: 'backtrack'. Match attempt failed after 5 steps.
- Beginning match attempt at character 2:** Shows steps 1-3. Step 1: 'ok', Step 2: 'backtrack', Step 3: 'backtrack'. Match attempt failed after 3 steps.
- Beginning match attempt at character 4:** Shows steps 1-3. Step 1: 'ok', Step 2: 'z' (ok), Step 3: 'z' (ok). Match found in 3 steps.

Match attempt character 0, matches first **a**, character 1 matches second **a**, character 2 matches the **b** and character 3, matches the **c**. These Matches in the Debug Logs above are indicated by 'ok'. All of the attempts fail because of the z.


Metacharacters in Depth

Quantifiers cont.

Range Quantifiers are also very useful is in matching multiple times against sub expressions.


For example if I wanted to match against four numeric digits in a string I could use:

Match: `(\d\d\d\d)`
 Replace: `\1 <space>`

Name ▲	New Name
 This is a test 0123 passed	0123

Or I could simplify it by using:

Match: `(\d{4})`
 Replace: `\1 <space>`


Name ▲	New Name
 This is a test 0123 passed	0123

Searches for a numeric digit. Matches against the **0** of the sequence, **0123**. The `{4}` instructs that the match is to be repeated 3 additional times for a total of 4 consecutive numeric digits. Matches against the **123** of **0123**. Capture Group 1 holds the final value of **0123**. The RegEx Engine will continue to move through the string looking for further matches. Finding none, it will reach the EOL and exhaust the string.

Another example:

String = This is a 1234 test 5678

Match: `(\d{4})`
 Replace: `\1 <space>`

Name ▲	New Name
 This is a 1234 test 5678	1234

Searches for a numeric digit. Matches against the **1** of the sequence, **1234**. The `{4}` instructs that the match is to be repeated 3 additional times for a total of 4 consecutive numeric digits. Matches against the **234** of **1234**. Capture Group 1 holds the final value of **1234**. The RegEx Engine will continue to move through the string and match a second time at the **5678**, but BRU only captures the first match under `v1`.

Metacharacters in Depth

Quantifiers cont.

This next example uses an Alternate.

String = test 1234

Match: `(\d{2}|\d{4})`

Replace: `\1 <space>`

Name ▲	New Name
test 1234	12

Selects between a match of two consecutive numeric digits or 4 consecutive numeric digits.

This illustrates a dilemma that I referenced at the beginning of this discussion:

To limit the match against only those strings that contain the fixed number of matches represented by the minimum value, you must use anchors (see notation below).

Given the first alternate of `\d{2}` in the RegEx, it will always match against the first two consecutive numeric digits in the string and therefore never get to evaluate the second alternative of the 4 consecutive numeric digits even, as they do here, exist in the string. The only value that can be returned is **12**.

This, however, is the expected behavior of using alternates. The first alternate is always evaluated first against the string, and if matched, any additional alternates would not be evaluated. First come, first served.

If I reverse the alternates, you will see this:

Match: `(\d{4}|\d{2})`

Replace: `\1 <space>`

Name ▲	New Name
test 1234	1234

The first alternate of `\d{4}` is matched to the value, **1234 = true**. The RegEx is satisfied but the second alternate, also **true**, is never evaluated.

Notes:


1. I have previously shown you a method that will successfully limit the match by using anchors. See notation 1 under the discussion of `{min, max}` for a RegEx that will accomplish this.

Metacharacters in Depth


Quantifiers cont.

String = The value of pi to 18 digits is 3.141592653589793238 on to infinity

Match: (3.\d{3,})
 Replace: \1 <space>

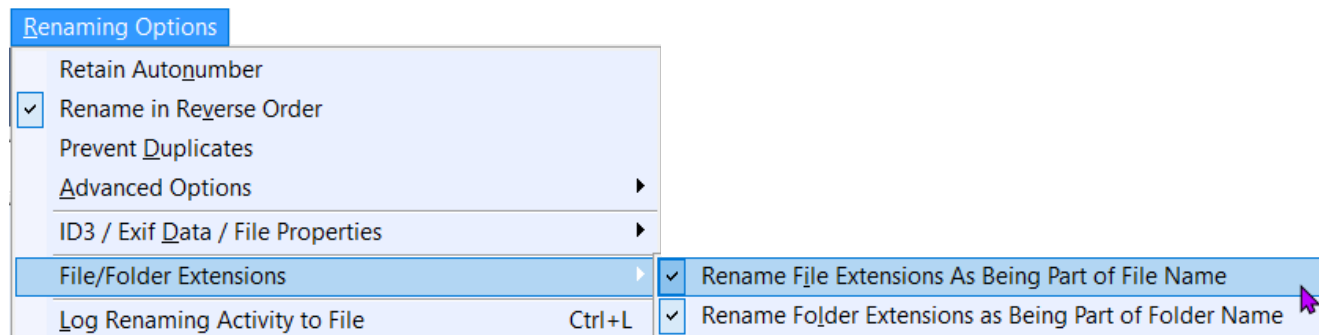
Name ▲	New Name
 The value of pi to 18 digits is 3.141592653589793238 on to infinity	3.141592653589793238

First you will notice that the string contains a dot character. Because this string in BRU is a filename, and not a folder name, it is interpreted as a file extension dot with everything else following as part of the file extension. This will not be evaluated correctly. It will result in:

Name ▲	New Name
 The value of pi to 18 digits is 3.141592653589793238 on to infinity	The value of pi to 18 digits is 3.141592653589793238 on to infinity

Doesn't match. In step 3, the value matched will be the **3** of **3.14..**, because to the RegEx, the string ends at the dot character right before what it interprets as the EOL. Step 4 will **fail**, because when it backtracks, the next character tested will be the <space> prior to the **3**.

To remedy this, 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.



Metacharacters in Depth

Quantifiers cont.

String = The value of pi to 18 digits is 3.141592653589793238 on to infinity

Match: (3.\d{3,})
 Replace: \1 <space>

Analysis:

- | | | |
|------|--|----------------------|
| 1. 3 | Match against a literal numeric digit, '3'.
Matches against the '3' of '3.14..' | Capture Group 1 = 3 |
| 2. . | Dot Metacharacter matches against any character.
Matches against the dot of '3.14 ..' | Capture Group 1 = 3. |

I could have also used this:

Match: (3\\.d{3,})

.. and obtained the same results. Why? Because instead of using a Dot Metacharacter that matches any character, I could have specified the literal dot character by preceding it with an Escape Metacharacter, the forward slash. The Escape indicates to the RegEx Engine not to treat the dot as a Metacharacter but as a literal character. After the match of the 3, it would have searched for the literal dot and matched.

- | | | |
|--------|---|--|
| 3. \d | Match against numeric digit.
Matches against the '1' of '3.14..' | Capture Group 1 = 3.1 |
| 4. {3} | Range Quantifier.
Repeat match for a minimum of 2 additional times with no upper limit.
Matches against the remaining numeric list of '41592653589793238' and stops when it encounters the <space> after the final '8'. | Capture Group 1 = 3.141592653589793238 |

The value of pi to 18 digits is **3.141592653589793238** on to infinity

```

Beginning match attempt at character 32
1 3
2 3.
3 3.141592653589793238
4 3.141592653589793238
Match found in 4 steps
  
```

Metacharacters in Depth

Quantifiers cont.

String = The value of pi to 18 digits is 3.141592653589793238 on to infinity

Match: `(.*\d{3,})`
 Replace: `\1 <space>`

Name ▲	New Name
<input type="checkbox"/> The value of pi to 18 digits is 3.141592653589793238 on to infinity	The value of pi to 18 digits is 3.141592653589793238

Analysis:

- | | | |
|--------|---|--|
| 1. . | Dot Metacharacter match any character.
Matches the 'T' of 'The'. | Capture Group 1 = T |
| 2. * | Make it Greedy.
Current position = <u>EOL</u> . | Capture Group 1 = <entire string> |
| 3. \d | Match against Numeric Digit.
Backtracks to match against the last numeric digit of the numeric list, the '8' | Capture Group 1 =
The value of pi to 18 digits is <space>
3.141592653589793238 |
| 4. {3} | Range Quantifier.
Repeat match for a total of three times to infinity.
Matches the numeric digit still Backtracking, eighteen additional two times, for the '23'. | Capture Group 1 = Unchanged |

Some interesting notes.

In step 2, the entire string is captured, so in Step 3, when it begins to backtrack, characters are being removed from the first capture and added back in the current capture. The characters that are removed are:

`<space> on to infinity`

The value of pi to 18 digits is 3.141592653589793238 on to infinity

← backtracking

This happens as it backtracks to the **8**.

The next thing that happens is as it backtracks through step 4, those characters are also being removed from the first capture and added back in the current capture as before, but because there is only one Capture Group, this is not visible because the final result in step 4 remains unchanged from step 3.

The next point of interest is the match of step 3 through 4. At first glance, you may think that {3,} would backtrack and capture all the way to the dot character, but you would be wrong. Instead only the minimum needed to match are captured and this includes the **8** from step 3 and the **23** from step 4. The resulting match of just step 3 through 4 = **238** and not **141592653589793238**.

Metacharacters in Depth

Quantifiers cont.

Look at the Debug log from Regex Buddy.

```

Beginning match attempt at character 0
1 The value of pi to 18 digits is 3.141592653589793238 on to infinity
2 The value of pi to 18 digits is 3.141592653589793238 on to infinitybacktrack
3 The value of pi to 18 digits is 3.141592653589793238 on to infinit
4 The value of pi to 18 digits is 3.141592653589793238 on to infinitbacktrack
5 The value of pi to 18 digits is 3.141592653589793238 on to infini
6 The value of pi to 18 digits is 3.141592653589793238 on to infinibacktrack
7 The value of pi to 18 digits is 3.141592653589793238 on to infin
8 The value of pi to 18 digits is 3.141592653589793238 on to infinbacktrack
9 The value of pi to 18 digits is 3.141592653589793238 on to infi
10 The value of pi to 18 digits is 3.141592653589793238 on to infibacktrack
11 The value of pi to 18 digits is 3.141592653589793238 on to inf
12 The value of pi to 18 digits is 3.141592653589793238 on to infbacktrack
13 The value of pi to 18 digits is 3.141592653589793238 on to in
14 The value of pi to 18 digits is 3.141592653589793238 on to inbacktrack
15 The value of pi to 18 digits is 3.141592653589793238 on to i
16 The value of pi to 18 digits is 3.141592653589793238 on to ibacktrack
17 The value of pi to 18 digits is 3.141592653589793238 on to
18 The value of pi to 18 digits is 3.141592653589793238 on to backtrack
19 The value of pi to 18 digits is 3.141592653589793238 on to
20 The value of pi to 18 digits is 3.141592653589793238 on tobacktrack
21 The value of pi to 18 digits is 3.141592653589793238 on t
22 The value of pi to 18 digits is 3.141592653589793238 on tbacktrack
23 The value of pi to 18 digits is 3.141592653589793238 on
24 The value of pi to 18 digits is 3.141592653589793238 on backtrack
25 The value of pi to 18 digits is 3.141592653589793238 on
26 The value of pi to 18 digits is 3.141592653589793238 onbacktrack
27 The value of pi to 18 digits is 3.141592653589793238 o
28 The value of pi to 18 digits is 3.141592653589793238 obacktrack
29 The value of pi to 18 digits is 3.141592653589793238
30 The value of pi to 18 digits is 3.141592653589793238 backtrack
31 The value of pi to 18 digits is 3.141592653589793238
32 The value of pi to 18 digits is 3.141592653589793238backtrack
33 The value of pi to 18 digits is 3.14159265358979323
34 The value of pi to 18 digits is 3.14159265358979323backtrack
35 The value of pi to 18 digits is 3.1415926535897932
36 The value of pi to 18 digits is 3.1415926535897932backtrack
37 The value of pi to 18 digits is 3.141592653589793
38 The value of pi to 18 digits is 3.141592653589793238 ←
39 The value of pi to 18 digits is 3.141592653589793238
Match found in 39 steps

```

See the resulting match of **238**?

I'll make this more clear to you by separating the values out in two Capture Groups:

Match: `(.*)(\d{3,})`

Replace: `\1 \2`

Name ▲	New Name
<input type="checkbox"/> The value of pi to 18 digits is 3.141592653589793238 on to infinity	The value of pi to 18 digits is 3.141592653589793 238

Capture Group 2 captures the **238** and this in turn is removed from Capture Group 1's value. You can also see how this could be used to isolate the end of the string for as many characters as required simply by changing the minimum value in the Range Quantifier.




Metacharacters in Depth

Quantifiers cont.

One more example to close this out.

Match: `(\d{2})-(\d{2})-(\d{2,4})`

Replace: `|1.|2.|3`

Name ▲	New Name
 The date is 10-24-1990	10. 24 .1990
 The date is 07-10-89	07. 10 .89
 The date is 5-6-2021	The date is 5-6-2021

The first two Capture Groups capture the month and day using a pretty straightforward single value fixed length of 2 repeated matches. The specified 2 though, also precludes the string, **The date is 5-6-2021**, because both the **5** and the **6** will not match.

The third Capture Group is more interesting because it specifies a range between 2 and 4 numeric digits. This will match both the 4 digit year of **1990** in the string, **The date is 10-24-1990**, and the 2 digit year of **89** in the string, **The date is 07-10-89**.

The RegEx does not capture the <hyphen> characters used as delimiters between the dates. This allows me to replace them if I so desire with something else, which I did using dot (literal) characters in the Replace String instead.

Metacharacters in Depth

Quantifiers cont.

Possessive Quantifiers.

When using a Possessive Quantifier the RegEx will not perform additional permutations to satisfy a match even if the remainder of the RegEx fails. Unlike a typical Greedy Quantifier where the RegEx will recalculate in an attempt, or permutation, to get a successful match, the Possessive Quantifiers afford it one chance, and one chance only to get it right. It moves forward in one direction, from left to right and it will not perform backtracking. As a result, Possessive Quantifiers run faster. This is important when you have a large number of files to process. Why not always use Possessive Quantifiers? Because there are many times that you want to match against a pattern that would otherwise be too restrictive using only Possessive Quantifiers.

After the evaluation of a Possessive Quantifier, the RegEx position is at the end of the string, or EOL. Since it will not backtrack to give back characters even if there are additional sub-expressions left to evaluate (this is what was meant in the statement, ‘even if the remainder of the RegEx fails’), the RegEx is finished whether it matched or failed to match. This behaviour also classifies the Possessive Quantifiers as Greedy.

For each ‘regular’ Quantifier, there is a Possessive Equivalent with the exception of ‘??’

***+** Match the previous character, expression or metacharacter **zero or more times**

As many items as possible will be matched, without trying any permutations with less matches even if the remainder of the RegEx fails. This is the possessive equivalent of using *

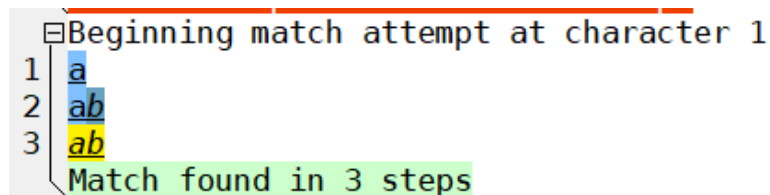
Example:

1) (ab*+) matches **ab** in string **rabcc** **rabcc**

Match: (ab*+)

Replace: \1 <space>

Name ▲	New Name
rabcc	ab



No Backtracking. That is the difference between making a Quantifier Possessive or not.

Notes:

1. Permutations refer to the RegEx performing different calculations in an attempt to satisfy the match. Unlike the human brain, the logic recognizes one definition of insanity is trying the same thing and expecting different results.

Metacharacters in Depth

Quantifiers cont.

Example:

String = aabc

Match: (.#+abc)
 Replace: \1 <space>

Name ▲	New Name
aabc	aabc

Analysis:

- | | | |
|--------|--|------------------------------|
| 1. . | Dot Metacharacter.
Matches against any character. | Capture Group 1 = a |
| 2. * | Make it Greedy. | Capture Group 1 = aabc |
| 3. + | Make it Possessive.
Current position = <u>EOL</u> . | Capture Group 1 = Unchanged. |
| 4. abc | Match against literal string characters, 'abc'.
Already at <u>EOL</u> .
Can't backtrack because it is Possessive.
Fails. | |

If this had been non-Possessive, it would have backtracked and matched against the **abc**.

Match: (.#abc)
 Replace: \1 <space>

Name ▲	New Name
aabc	aabc

Analysis:

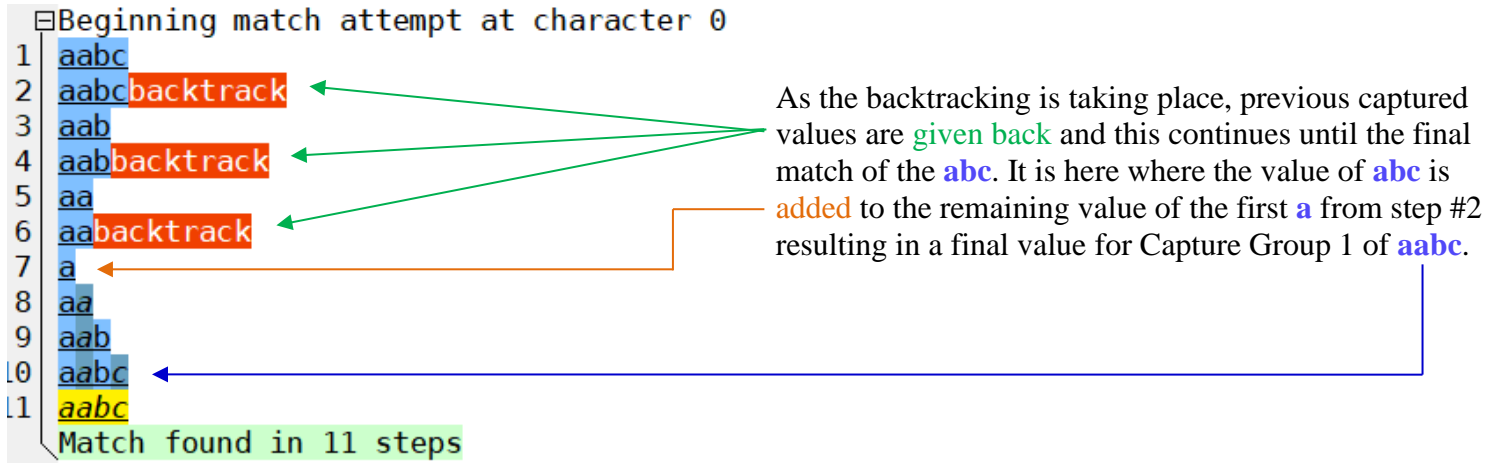
- | | | |
|--------|--|------------------------|
| 1. . | Dot Metacharacter.
Matches against any character. | Capture Group 1 = a |
| 2. * | Make it Greedy.
Current position = <u>EOL</u> . | Capture Group 1 = aabc |
| 3. abc | Match against literal string characters, 'abc'.
Already at <u>EOL</u> .
Backtracks to match against the 'abc'. | |

Metacharacters in Depth

Quantifiers cont.

Because Capture Group 1 already had a value of **aabc** resulting from Step 2, the user only sees the final value of **aabc** as unchanged, but behind the scenes there was backtracking.

This is what it looks like:



This behaviour can be clearly seen if I add a second Capture Group.

Match: `(.*)(abc)`

Replace: `Group 1 = \1 Group 2 = \2`

Name ▲	New Name
aabc	Group 1 = a Group 2 = abc

Analysis:

- | | | |
|--------|--|--|
| 1. . | Dot Metacharacter.
Matches against any character.
Matches against the first 'a' of 'aabc'. | Capture Group 1 = a |
| 2. * | Make it Greedy.
Current position = <u>EOL</u> . | Capture Group 1 = aabc |
| 3. abc | Match against literal string characters, 'abc'.
Already at <u>EOL</u> .
Backtracks to match against the 'abc'. | Capture Group 2 = abc
Changes Capture Group 1 = a |

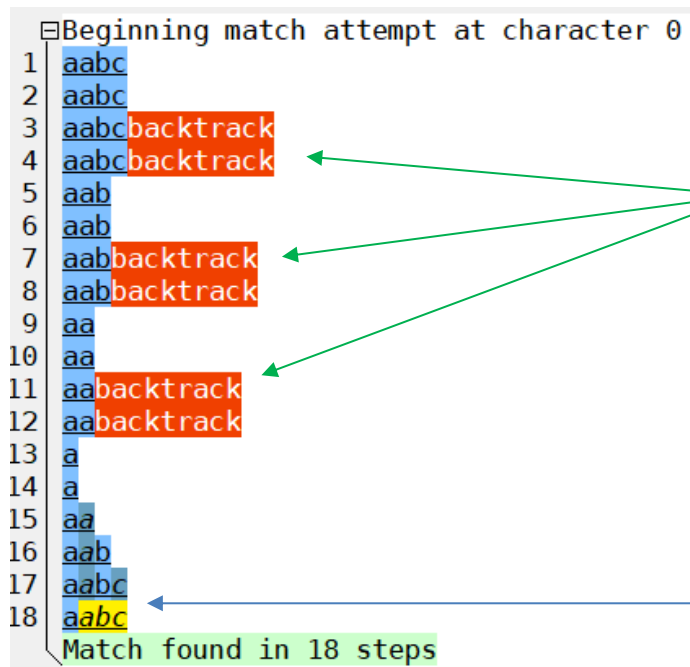
Metacharacters in Depth

Quantifiers cont.

Where:

\1 = a \2 = abc

	Start	Length
Match 1: aabc	0	4 ^{CL} _{RF}
Group 1: a	0	1 ^{CL} _{RF}
Group 2: abc	1	3



As the **a**, **b** and **c** are captured in Capture Group 2 in Step 3, these values are subsequently **given back** from Capture Group 1 leaving the remaining value the first **a** of **aabc** resulting from Step 2. Capture Group 2's value is the **'abc'** while Capture Group 1's value has been changed to **'a'**.

Metacharacters in Depth

Quantifiers cont.

++ Matches the previous character, expression or metacharacter **one or more times**

Possessive. As many items as possible will be matched, without trying any permutations with less matches even if the remainder of the regex fails. Possessive quantifiers do not use backtracking so they can run faster than using other quantifiers. This is important when you have a large list of files to process. ++ is the possessive equivalent of using +

A Tale of Two Strings:

XXXX
XXXXXZ

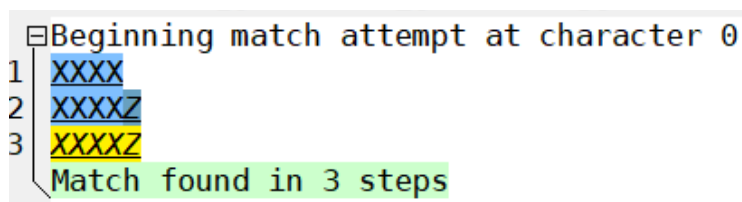
Lets' start with the non-possessive form, ' + '.

Match: (X+[A-Z])
Replace: \1 <space>

Name ▲	New Name
XXXXZ	XXXXZ
XXXX	XXXX

Analysis:

1. X Match against literal uppercase string character, 'X'. Capture Group 1 = X
Matches against the first 'X' of 'XXXXZ'.
2. + Make it Greedy... until Capture Group 1 = XXXX
The Match is made for one or more of the 'X' characters. This matches against all consecutive uppercase 'X' string literals.
3. Z ... Match against a class consisting of the literal uppercase string characters, A-Z. Capture Group 1 = XXXXZ
Matches against an uppercase 'Z'.



Straightforward, even though it is non-possessive, it required no Backtracking since it matched against the **X** characters ending with the **Z** character. Remember that if the ' * ' was used this would have matched the entire string and then would have backtracked to match against the **Z**.

Metacharacters in Depth

Quantifiers cont.

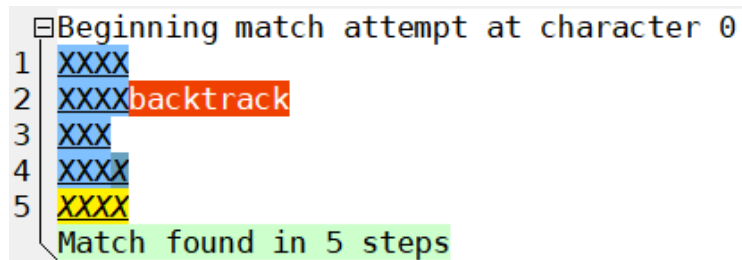
String = XXXX

Match: (X+[A-Z])
 Replace: \1 <space>

Name ▲	New Name
XXXXZ	XXXXZ
XXXX	XXXX

Analysis:

1. X Match against literal uppercase string character, 'X'. Capture Group 1 = X
 Matches against the first 'X' of 'XXXXZ'.
2. ++ Make it Greedy Possessive... until Capture Group 1 = XXXX
 The Match is made for one or more of the 'X' characters. This matches against all consecutive uppercase 'X' string literals.
 Current position = EOL.
3. Z ... Match against a class consisting of the literal uppercase string characters, A-Z. Capture Group 1 = XXXX
 Already at EOL. Backtracks to match against 'X'.



Metacharacters in Depth

Quantifiers cont.

Now I'll make it possessive usng, '++'.

String = XXXXZ

Match: (X++[A-Z])

Replace: \1 <space>

Name ▲	New Name
XXXXZ	XXXXZ
XXXX	XXXX

Analysis:

1. X Match against literal uppercase string character, 'X'.
Matches against the first 'X' of 'XXXXZ'.
Capture Group 1 = X
2. ++ Make it Greedy Possessive... until
The Match is made for one or more of the 'X' characters. This matches against all consecutive uppercase 'X' string literals.
Capture Group 1 = XXXX
3. Z ... Match against a class consisting of the literal uppercase string characters, A-Z.
Matches against an uppercase 'Z'.
Capture Group 1 = XXXXZ

No difference from before, but what about this?

String = XXXX

Analysis:

1. X Match against literal uppercase string character, 'X'.
Matches against the first 'X' of 'XXXXZ'.
Capture Group 1 = X
2. ++ Make it Greedy Possessive... until
The Match is made for one or more of the 'X' characters. This matches against all consecutive uppercase 'X' string literals.
Current position = EOL.
Capture Group 1 = XXXX
3. Z ... Match against a class consisting of the literal uppercase string characters, A-Z.
Already at EOL. Because of the Possessive Quantifier, it cannot Backtrack to match against 'X'
Fails.

Metacharacters in Depth

Quantifiers cont.

- ?+** Matches the previous (character or metacharacter, Capture Group or class) **zero or one time** (The question mark makes the preceding item optional. - Possessive quantifier.)

Possessive. If the optional item can be matched, then the quantifier won't give up its match even if the remainder of the regex fails. Possessive quantifiers do not use backtracking so they can run faster than using other quantifiers. This is important when you have a large list of files to process. ?+ is the possessive equivalent of using ?

Example:

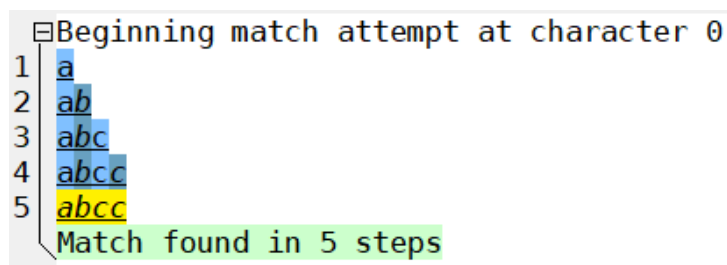
1) `abc?+c` matches `abcc` but not `abc`

Match: `(abc?+c)`
 Replace: `\1 <space>`

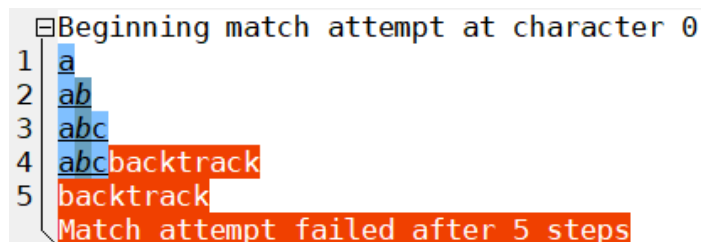
Name ▲	New Name
abcc	abcc
abc	abc

Looks to match `ab` followed optionally by `c` and immediately followed by a second `c`.

`abcc` matches because it matches against `ab`, immediately followed by a `c` that is in turn immediately followed by a second `c`. Because the first `c` exists in the string, the optional path is not taken.



The string `abc` does not match. Although `ab` is matched, and immediately followed by a `c`, the second `c` in the RegEx is not matched. In step 4 below, the attempt to backtrack is disallowed. The RegEx fails because Possessive Quantifiers do not allow characters to be given back from the first match, which would be required for the match of the second `c`.



Attempt to backtrack fails. Use of a Possessive Quantifier does not permit backtracking.

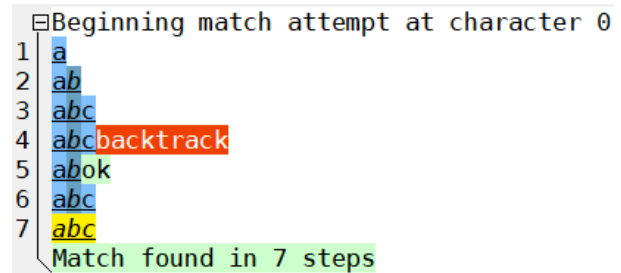
Metacharacters in Depth

Quantifiers cont.

Compare that against the non-Possessive equivalent of the RegEx:

Match: (abc?c)
Replace: \1 <space>

Name ▲	New Name
abc	abc



ab is matched, followed immediately by a **c**, but because there is no second **c** in the string, it backtracks to match against the second **c** in the RegEx (the **c** following the Lazy Metacharacter, '?'), giving up the first **c** value and taking the optional path, meaning that although there was a first **c**, in order to match, the RegEx Engine will instead treat the first **c** as optional even though in the first capture it was not.

Here is a full analysis:

- | | | |
|-------|--|-----------------------|
| 1. ab | Match against the string literals, 'ab'. | Capture Group 1 = ab |
| 2. c | Match against the string literal, 'c'. | Capture Group 1 = abc |
| 3. ? | Make the 'c' in step 2 optional. | Capture Group 1 = abc |

In step 3, although the capture of the **c** is made optional, because it does exist, the optional path is not taken. Capture Group 1's value remains unchanged. Current position = EOL.

- | | | |
|------|--|-----------------------|
| 4. c | Match against the string literal, 'c'. | Capture Group 1 = abc |
|------|--|-----------------------|

In step 4, a second **c** is searched for. The RegEx Engine is at the EOL so it backtracks. As it backtracks, it finds a **c** character in the string. This **c** character has previously been captured in step 2, but because step 3 makes this **c** capture optional, in order to satisfy the match, Capture Group 1 gives up the previous match of the **c** and applies it to the capture of the second **c** of the RegEx instead. Because the optional path is taken, the capture of the first **c** never existed, satisfying the match of the literal **c** in step 4.

When the RegEx is made Possessive, the same steps are taken except that the **c** value of step 2 is not given back, and the RegEx fails because it cannot satisfy the match of the second **c** in the evaluation.

A Possessive Quantifier runs faster as a result of not having to recalculate the expression (permutation) to try and satisfy the match. Either it matches or it doesn't, and backtracking will not take place. When you have a lot of files that require renaming, this can save some time. The trick is that you have to know when to use what you have to its advantage. That comes with experience.

Metacharacters in Depth

Quantifiers cont.

There are also Possessive equivalents of Range Quantifiers:

`{#, #}+` or `{#, }+` or `{#}+`

Example:

String = abcdz

Match: `(a.{0,}+z)`

Replace: `\1 <space>`

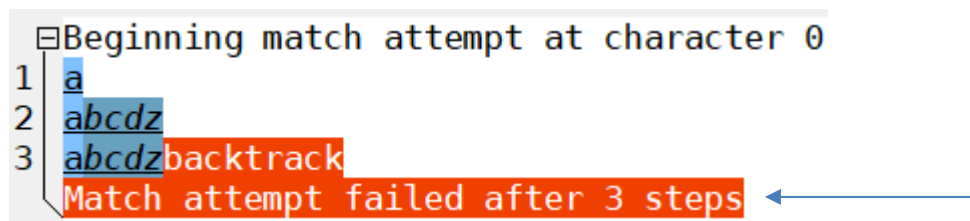
Name ▲	New Name
abcdz	abcdz

Analysis:

1. a Match against the literal string character, 'a'
2. . Match against any character.
3. {0,} Perform the match a minimum of zero (optional) with no upper limit.
Matches the remainder of the string.
4. + Make the `.{0,}` Possessive.
Current position = EOL.
5. z Match against the literal string character, 'z'.
Already at EOL. Because of the Possessive nature, cannot backtrack to match against the 'z'.

Capture Group 1 = a
Capture Group 1 = ab
Capture Group 1 = abcdz

FAILS.



Metacharacters in Depth

Quantifiers cont.

Compare this against the non-Possessive equivalent.

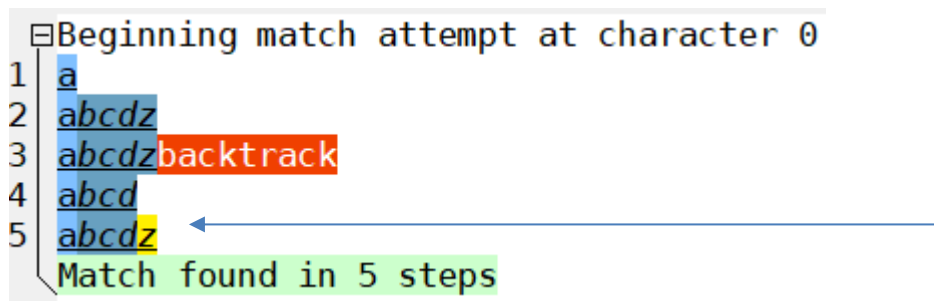
String = abcdz

Match: (a.{0,}z)
 Replace: \1 <space>

Name ▲	New Name
abcdz	abcdz

Analysis:

- | | | |
|---------|---|-------------------------|
| 1. a | Match against the literal string character, 'a'
Matches against the 'a' of 'abcdz'. | Capture Group 1 = a |
| 2. . | Match against any character. | Capture Group 1 = ab |
| 3. {0,} | Perform the match a minimum of zero (optional)
with no upper limit.
Matches the remainder of the string.
Current position = <u>EOL</u> . | Capture Group 1 = abcdz |
| 4. z | Match against the literal string character, 'z' of
'abcdz'.
Already at EOL. Backtracks to match against
the 'z' of 'abcdz'. | |



Summary Notes:

1. + is the shorthand for {1,}
2. * is the shorthand for {,}
3. ? is the shorthand for {,1}
4. I will not be emphasizing any differences between string, value, and RegEx characters using the **light blue bold** anywhere else but within this section that pertained to Quantifiers. I only did it here because I felt that it could be confusing for the reader given the similarities between the values and the RegEx components.

Greed and Lazy

Greed

The Greedy Quantifiers are -

match zero or more (match all)

*

match 1 or more

+

specified no. of matches

{*min,max*}

Possessive

e.g. ++

These Quantifiers are greedy because they consume (repeatedly testing the pattern against..) every character in the input string no matter if a match has been found and will do so until it reaches the end of the input string (**E**{nd}**O**{f}**L**{ine}) regardless if there is more of the pattern left to process (more sub-patterns to be evaluated).
e.g., ‘ .* ’

If you would rather have the Quantifier stop at the first possible match, follow it with a question mark. This is called ‘Lazy’, e.g., ‘ .+? ’. Lazy is also referred to as Non-Greedy or Reluctant. A Lazy Quantifier can also make the match, or if there is not a match, make the sub-expression optional as it was never evaluated, thereby allowing the rest of the RegEx to be evaluated even without a match of the sub-expression that was made Lazy.

String = Album1987

Match: .+(\d\d)

Replace: \1 <space>

Name ▲	New Name
Album1987	87

= 87 because ‘.+’ is greedy, it matches to the EOL, then backtracks to match against the ‘87’.

Match: .+?(\d\d)

Replace: \1 <space>

Name ▲	New Name
Album1987	19

= 19 because the ‘?’ on ‘.+?’ makes that expression Lazy and matches only until it finds the first two digits. In the analysis, the ‘.+’ matches against the entire string just as in the first example, but the ‘?’ makes the match optional, so the RegEx engine repositions back to the start of the string where it then tests each character moving forward against the \d\d, matching against the ‘19’.

Greed and Lazy

Greed cont.

Example:

using the ‘+’ Quantifier...

String = This is a first test

This input string is a piece of an html tag. HTML is an embedded language that uses code enclosed between < >.

I want to locate the first occurrence of in the string. I include the literal characters of ‘<’ and ‘>’ in the RegEx to accomplish this.

If looking for characters in ‘<>’, we indicate to first find a ‘<’ character and then find what is inside by using this pattern:

<.+> (the sub-expression, dot character and a plus sign, enclosed between angular brackets)

Match: <.+>

This is a first test

	Start	Length
Match 1: first	10	14

Analysis:

- | | | |
|------|---|---------------------------------------|
| 1. < | Match against the literal string character, ‘<’.
Matches the first ‘<’ of the first occurrence of ‘’. | Capture Group 1 = < |
| 2. . | Dot Metacharacter. Match against any character.
Matches against the first ‘E’ of the first occurrence of ‘’. | Capture Group 1 = <E |
| 3. + | Make it Greedy.
Current position = <u>EOL</u> . | Capture Group 1 = first test |
| 4. > | Match against the literal string character, ‘>’
Already at EOL. Backtracks to match against the last ‘>’ in the string, first test. | Capture Group 1 = first |

Notes:

1. Some examples use an HTML tag for reasons of illustration only. BRU is only used to rename files and folders, not text within a document or file.
2. Because the values returned in these examples may contain characters, e.g., < or > that are illegal in filenames, I was unable to show direct results in BRU. Instead I used results from the Regex Buddy program.
3. I felt that using these HTML text strings best illustrated what I wanted to explain and at the same time reflected some examples from source material that was used in this section.

Greed and Lazy

Greed cont.

How does it end up '`first`' and not '`first test`' ?

When the next part of the pattern '`>`' is processed, the end of the Input String (EOL) has already been reached because of the greedy '`+`'. The PCRE engine (aka RegEx Engine) tries to correct itself by processing the Input String *backwards* – testing from the end of the string moving to the front of the string looking for the next character '`>`' until it either finds a match or the beginning of the string has been reached. This is called **backtracking**.

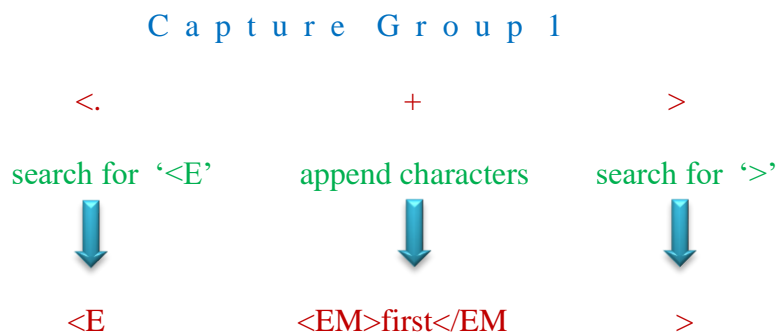
In backtracking, it gives up the characters, '`<space> test`' leaving the end result value as '`first`'.

```

Beginning match attempt at character 10
1  <
2  <EM>first</EM> test
3  <EM>first</EM> test backtrack
4  <EM>first</EM> tes
5  <EM>first</EM> tes backtrack
6  <EM>first</EM> te
7  <EM>first</EM> te backtrack
8  <EM>first</EM> t
9  <EM>first</EM> t backtrack
10 <EM>first</EM>
11 <EM>first</EM> backtrack
12 <EM>first</EM>
13 <EM>first</EM> backtrack
14 <EM>first</EM>
15 <EM>first</EM>
Match found in 15 steps

```

Basically the pattern is:



Notes:

1. Regular Expression concepts such as backtracking have previously been discussed.

Greed and Lazy

Lazy

Lazy refers to, ‘give em’ an out and they’ll take it’. When it comes to Quantifiers, Lazy instructs the RegEx Engine to do the least amount of matches to satisfy the sub-expression. A non-Greedy Quantifier is one that will match the shortest possible string in the least iterations of a permutation. An iteration is simply a repeat of the sub-expression. A permutation is a recalculated evaluation of the sub-expression to provide an alternative method in order to find an overall match. Where one fails, another is attempted. If you really want to see all the ‘visible’ permutations of a RegEx, I encourage you to visit regex101.com where you can see the animation of the RegEx play out.

To make a Greedy Quantifier lazy, add a question mark following the ‘repeat’, in this case the ‘+’, as in: ‘.+?’


The Question mark makes the ‘+’ lazy because instead of repeating the ‘.’ as many times as it can (until the end of the input string, EOL), it will now repeat it in as few times as possible. The ‘+’ requires a minimum match of 1 to make the match succeed, but the addition of the ‘?’ makes the sub-expression including the Greedy ‘+’ optional.

Greedy Example:

String = This is a &EM&first&EM& test

Match: (.*)(&)([A-Z]+2)

Replace: \2\3

Name ▲	New Name
 This is a &EM&first&EM& test	&EM&

Analysis:

- | | | |
|----------|---|--|
| 1. . | Dot Metacharacter. matches against any character.
Matches against the ‘T’ of ‘This’. | Capture Group 1 = T |
| 2. * | Make it Greedy.
Current position = <u>EOL</u> . | Capture Group 1= <entire string> |
| 3. & | Match against literal string character, ‘&’.
Already at EOL. Backtracks to match against the last ‘&’ that precedes the <space> before ‘test’. | Capture Group 2 = &
Changes Capture Group 1 =
This is a &EM&first&EM |
| 4. [A-Z] | Class consisting of an uppercase letter. | Capture Group 3 = E
Changes Capture Group 1 =
This is a &EM&first |

Attempts to move forward but cannot match the pattern ‘& uppercase letter’ against the ‘<space> test’ so it backtracks still testing to match the pattern, until it matches the ‘& Uppercase letter’ to the ‘E’ that precedes ‘first’. This also reevaluates the value of Capture Group 2 to the ampersand following ‘first’ and consequently changes Capture 1’s value to drop the characters, ‘&EM’. For a full explanation, please refer to the sub-section, ‘If I want to capture both of those values of ‘&EM& ...’ under the section, ‘Greedy and Lazy Examples’.

- | | | |
|-------|--|-----------------------|
| 5. + | Make it Greedy. | Capture Group 3 = EM |
| 6. \2 | Match against the value held in Backreference \2.
Backreference \2 references Capture Group 2.
Capture Group 2’s value is ‘&’.
Moves forward to match against the ‘&’ preceding the ‘<space> test’. | Capture Group 3 = EM& |

Greed and Lazy

Lazy cont.

This is one of those RegEx where the whole is better explained than the breakdown of its components.

What the `.*&[A-Z]+\2` is really searching for is to test each character from the end of the string for the pattern:

& <two uppercase letters> &

For each permutation, the search is always conducted from the end of the string because that is what the `.*` directs.

1. Evaluate `.*`, go to end of string.
2. Begin testing for pattern at current character, moving backwards.
3. If fail, repeat until pattern matched or string exhausted.

It looks like this:

```

Beginning match attempt at character 0
1 This is a &EM&first&EM& test
2 This is a &EM&first&EM& test
3 This is a &EM&first&EM& test backtrack
4 This is a &EM&first&EM& test backtrack
5 This is a &EM&first&EM& tes
6 This is a &EM&first&EM& tes
7 This is a &EM&first&EM& tes backtrack
8 This is a &EM&first&EM& tes backtrack
9 This is a &EM&first&EM& te
10 This is a &EM&first&EM& te
11 This is a &EM&first&EM& te backtrack
12 This is a &EM&first&EM& te backtrack
13 This is a &EM&first&EM& t
14 This is a &EM&first&EM& t
15 This is a &EM&first&EM& t backtrack
16 This is a &EM&first&EM& t backtrack
17 This is a &EM&first&EM&
18 This is a &EM&first&EM&
19 This is a &EM&first&EM& backtrack
20 This is a &EM&first&EM& backtrack
21 This is a &EM&first&EM&
22 This is a &EM&first&EM&
23 This is a &EM&first&EM& backtrack
24 This is a &EM&first&EM& backtrack
25 This is a &EM&first&EM
26 This is a &EM&first&EM
27 This is a &EM&first&EM&
28 This is a &EM&first&EM&
29 This is a &EM&first&EM& backtrack
30 This is a &EM&first&EM& backtrack
31 This is a &EM&first&EM& backtrack
32 This is a &EM&first&E
33 This is a &EM&first&E
34 This is a &EM&first&E backtrack
35 This is a &EM&first&E backtrack
36 This is a &EM&first&
37 This is a &EM&first&
38 This is a &EM&first& backtrack
39 This is a &EM&first& backtrack
40 This is a &EM&first
41 This is a &EM&first
42 This is a &EM&first&
43 This is a &EM&first&
44 This is a &EM&first&EM
45 This is a &EM&first&EM&
46 This is a &EM&first&EM&
Match found in 46 steps

```

The lines in Yellow represent each permutation beginning with the evaluation of `.*`. This moves the RegEx Engine to the end of the string positioning itself by backtracking to the current character. The current character is the character to the left of the last character tested in the previous permutation. It then tests this current character for the pattern. It is not until line 42 that the pattern is matched.

The Lines in Yellow represent the value captured in capture Group 1. Capture Group 2's value is seen in line 27 and Capture Group 3's value is in line 45 when it finally matches for the pattern in line 42 and moves forward. The Lines in Yellow also illustrate how characters from a previous capture are given back. This final value for Capture Group 1 can be seen in line 40.


Greed and Lazy

Lazy cont.

Notes:

1. A Backreference references only the captured value not the original expression.
2. If I had wanted to capture the last word 'test' I could have used: `(.*)&)(.+)` and only include Capture group `\4` in the Replacement String:

Match: `(.*)&)(.+)`
 Replace: `\4`

Name	New Name
 This is a &EM&first&EM& test.jpg	test.jpg

Where:

`\1` = This is a &EM&first&EM `\2` = & `\3` = <space> `\4` = test

After backtracking to match the last '&' prior to the <space> before 'test', the RegEx Engine moves forward to capture the <space> for Capture Group 3, and 'test' for Capture Group 4.

Match 1: **This is a &EM&first&EM& test**

Group 1: This is a &EM&first&EM

Group 2: &

Group 3:

Group 4: **test**

Beginning match attempt at character 0

1	This is a &EM&first&EM& test	
2	This is a &EM&first&EM& test	
3	This is a &EM&first&EM& test	backtrack
4	This is a &EM&first&EM& test	backtrack
5	This is a &EM&first&EM& tes	
6	This is a &EM&first&EM& tes	
7	This is a &EM&first&EM& tes	backtrack
8	This is a &EM&first&EM& tes	backtrack
9	This is a &EM&first&EM& te	
10	This is a &EM&first&EM& te	
11	This is a &EM&first&EM& te	backtrack
12	This is a &EM&first&EM& te	backtrack
13	This is a &EM&first&EM& t	
14	This is a &EM&first&EM& t	
15	This is a &EM&first&EM& t	backtrack
16	This is a &EM&first&EM& t	backtrack
17	This is a &EM&first&EM&	
18	This is a &EM&first&EM&	
19	This is a &EM&first&EM&	backtrack
20	This is a &EM&first&EM&	backtrack
21	This is a &EM&first&EM&	
22	This is a &EM&first&EM&	
23	This is a &EM&first&EM&	backtrack
24	This is a &EM&first&EM&	backtrack
25	This is a &EM&first&EM	
26	This is a &EM&first&EM	
27	This is a &EM&first&EM	
28	This is a &EM&first&EM	
29	This is a &EM&first&EM	
30	This is a &EM&first&EM	
31	This is a &EM&first&EM& test	
32	This is a &EM&first&EM& test	

Match found in 32 steps

Greed and Lazy

Lazy cont.


Lazy Example:

By making the ‘ .* ’ Lazy, we change the result to capturing the first part of the string rather than the last part of the string.

String = This is a &EM&first&EM& test.jpg

Match: (.?)(&)([A-Z]+2)


Replace: \2\3

Name ▲	New Name
 This is a &EM&first&EM& test	&EM&

1. . Dot Metacharacter. matches against any character. Capture Group 1 = T
Matches against the ‘T’ of ‘This’.
2. * Make it Greedy. Capture Group 1= <entire string>
Current position = EOL.
3. ? But not too Greedy. Capture Group 1 = **null**
Current position = BOL.

By making the Greedy Quantifier in step 2 Lazy, it makes what would have been the entire match of the string optional. When used with the ‘ * ’, what this essentially does is give up the match as if it never was evaluated, and return to the position before the match. In this example, it is at the BOL. Because the * can match against zero or more times, it matches at the BOL. The BOL has a **null** value, and this results in a Zero Occurrence Match for Capture Group 1. This is a Zero Length Match, so although Capture Group 1 does exist, it is **empty** at this point.

	Start	Length
Match 1:	0	0
Group 1:	0	0



If I had instead used ‘ .+? ’, Capture Group 1 would currently hold the value of ‘T’. Why? Although the match is still given up, the RegEx Engine returns to the point before the match, the BOL again, but the ‘ + ’ must match at least one time, so it moves forward to match against the ‘T’ of ‘This’.

4. & Match against literal string character, ‘&’. Capture Group 2 = &
Moves forward to match against the first ‘&’ after the <space> following ‘is’
Changes Capture Group 1 = This is a <space>
5. [A-Z] Class consisting of an uppercase letter. Capture Group 3 = E
6. + Make it Greedy. Capture Group 3 = EM
7. \2 Match against the value held in Backreference \2. Capture Group 3 = EM&
Backreference \2 references Capture Group 2.
Capture Group 2’s value is ‘&’.
Matches against the second ‘&’ prior to ‘first’

Notes:

1. EOL is the End of the String after the last character position, prior to the extension, if any, and BOL is the Beginning of the String before the first character position.

Greed and Lazy

Lazy cont.

Lazy Example:

My analysis, you must remember, is a simplified observation based on changing values, and not an accurate record that documents all of the background permutations. I do not want you to think that my interpretations are in stone. Far from it.

For example, when I state that the RegEx Engine moves forward – I imply that it moves forward only – it does not. It is actually testing each character as did the previous RegEx, looking for that pattern and backtracking as required, although the amount of backtracking is minimal compared with the previous example. The backtracking occurs when the `.*` is applied, matching or not matching, and if not matching, making the sub-expression optional, moving back to the previous match.

```

Beginning match attempt at character 0
1 ok
2 ok
3 backtrack
4 backtrack
5 I
6 I
7 Ibacktrack
8 Ibacktrack
9 Th
10 Th
11 Thbacktrack
12 Thbacktrack
13 Thi
14 Thi
15 Thibacktrack
16 Thibacktrack
17 This
18 This
19 Thisbacktrack
20 Thisbacktrack
21 This
22 This
23 This backtrack
24 This backtrack
25 This i
26 This i
27 This ibacktrack
28 This ibacktrack
29 This is
30 This is
31 This isbacktrack
32 This isbacktrack
33 This is
34 This is
35 This is backtrack
36 This is backtrack
37 This is a
38 This is a
39 This is a backtrack
40 This is a backtrack
41 This is a
42 This is a
43 This is a &
44 This is a &
45 This is a &EM
46 This is a &EM&
47 This is a &EM&
Match found in 47 steps

```

Another fact you should be aware of is that I base the analysis on typically a single component added at a time, and then I reflect back any changes that occurred as a result. But more often than not, the RegEx is testing using patterns that may be made up of sub-expressions of more than one component. I do document some of these. For example, a string portion may be tested against a pattern of a lowercase letter followed by a numeric digit, followed by a `<space>` character before it is allowed to match rather than just testing for a lowercase letter. The former may be required to reach the EOL and backtrack for the match. The latter may just move forward and match against the first lowercase letter found. I account for these discrepancies where possible and ensure that the final values obtained are not affected by my methodology. I also validate my findings from debug output from programs like Regex Buddy or websites like Regex101.com.

Greed and Lazy Examples

#1. Isolate both 'EM&'s

String = This is a &EM&first&EM& test

Match: (.*)(&)([A-Z]+\2)(.*)(&)([A-Z]+\2)

Replace: \2\3 <space> <space>\5\6

Name ▲	New Name
This is a &EM&first&EM& test	&EM& &EM&

Notice the use of the extra <spaces> that display in the New Name. Any character can be entered – spaces, etc. in the Replace String and it will display as a literal. This is because with the exception of the \1.. \9 backreferences to represent the values held by the Capture Groups, the Replace field, generally speaking, does not use the Regular Expression language syntax, only the Match field does.

What this example does is to capture both occurrences of 'EM&' in the filename. It does this by simply repeating the entire sequence over. There is currently no way to apply one expression on multiple match occurrences in BRU otherwise under v1. More on this a little later.

Under v2, it might look something like this:

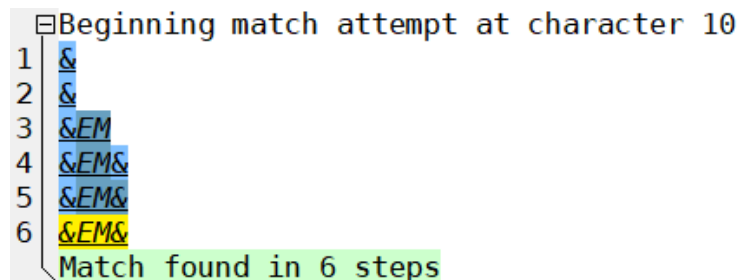
Match: ((&)([A-Z]+\2))/g

Replace: <space> } <5 spaces> 1 = \1 <3 spaces> 2 = \2 <3 spaces> 3 = \3 <5 spaces> {

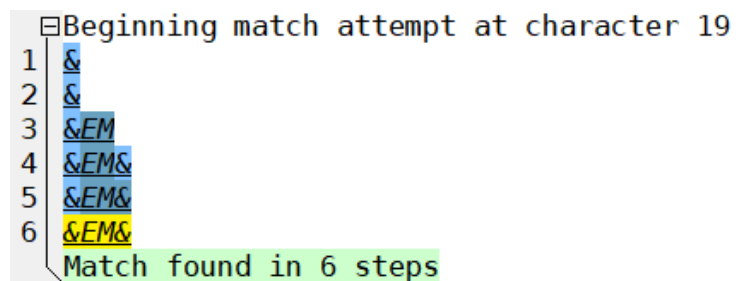
Name ▲	New Name
This is a &EM&first&EM& test	This is a } 1 = &EM& 2 = & 3 = EM& {first} 1 = &EM& 2 = & 3 = EM& { test

This is a &EM&first&EM& test

Match	Start	Length	CL	RF
Match 1: &EM&	10	4	CL	RF
Group 1: &EM&	10	4	CL	RF
Group 2: &	10	1	CL	RF
Group 3: EM&	11	3	CL	RF



Match 2: &EM&	19	4	CL	RF
Group 1: &EM&	19	4	CL	RF
Group 2: &	19	1	CL	RF
Group 3: EM&	20	3		



Greed and Lazy Examples


#1. Isolate both 'Em&'s cont.

String = This is a &EM&first&EM& test.jpg

I'll change the Replace String to make it a little more legible.

Match: ((&)([A-Z]+\2))/g

Replace: } \2 \3 {

Name ▲	New Name
 This is a &EM&first&EM& test	This is a } & EM& {first} & EM& { test

You can see more clearly where I have isolated parts of the original string from the matched values using the braces. Although the matched values have not changed from the original string, they were still matched successfully in the RegEx and therefore substituted for those same values in this copy of the original string.

Notes:

1. New Name is always a new string value. In v2, it creates a copy of the original string and substitutes values per the RegEx. It does not isolate those values from the original parts of the string as is done under v1. Under v1, in fact, the RegEx is used to indicate which parts of the original string are to be kept by Capturing those values and any parts that are not captured are discarded in the New Name.
2. v1 refers to the older PCRE Engine v5.x used under previous versions of BRU, as opposed to the newer engine implemented with BRU v3.4 with the Boost Regular Expressions Library, referred to as v2.

Greed and Lazy Examples

#1. Isolate both 'EM&'s cont.

Back to v1.

If I want to capture both of those values of '&EM&', I have to use two instances of the RegEx.

String = This is a &EM&first&EM& test

Match: `(.*)&([A-Z]+\2)(.*)&([A-Z]+\2)`

Replace: `\2\3 <space> <space> \5\6`

Analysis:

`(.*)&([A-Z]+\2)`

`(.*)&([A-Z]+\2)`

Where:

\1 =	This is a<space>	\2 =	&	\3 =	EM& (first occurrence)
\4 =	first	\5 =	&	\6 =	EM& (second occurrence)

- | | | |
|----------|---|--|
| 1. . | Dot Metacharacter. Matches against any character.
Matches against the 'T' of 'This'. | Capture Group 1 = T |
| 2. * | Make it Greedy.
Current position = <u>EOL</u> . | Capture Group 1 = <entire string> |
| 3. & | Match against literal string character, '&'.
Already at <u>EOL</u> . Backtracks to match against the ampersand preceding '<space> test'. | Capture Group 2 = &
Changes Capture Group 1 =
This is a &EM&first&EM |
| 4. [A-Z] | Match against a class consisting of uppercase letter.
Matches against the 'E' of the second occurrence of the '&EM&'. | Capture Group 3 = E
Changes Capture Group 2 = &
Changes Capture Group 1 =
This is a &EM&first |

Attempts to move forward but cannot match the pattern '& uppercase letter' against the '<space> test' so it backtracks still testing to match the pattern, until it matches the '& Uppercase letter' to the 'E' that precedes 'first'.

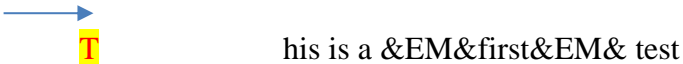
Capture Group 2's value changes although this is not apparent because it is the same character as before. What has changed is which character has been matched. When the 'E' is captured, Capture Group 2's value changes from the ampersand preceding '<space> test' to the ampersand that follows after 'first'. This is because when Capture Group 3 captures the 'E' value, Capture Group 2 is forced to give up its value and the sub-expression in step 3 is reevaluated as a result. Capture Group 1 is forced to drop the characters, '&EM'. Why? Because the ampersand of '&EM' is now held by Capture Group 2 and all following characters are discarded.

- | | | |
|-------|--|-----------------------|
| 5. + | Make it Greedy. | Capture Group 3 = EM |
| 6. \2 | Backreference to value held in Capture Group 2.
The value in Capture Group 2 is '&'.
Moves forward to match against the ampersand. | Capture Group 3 = EM& |


Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

Let's see if I can make sense of it for you.

Step 1. **Group 1**


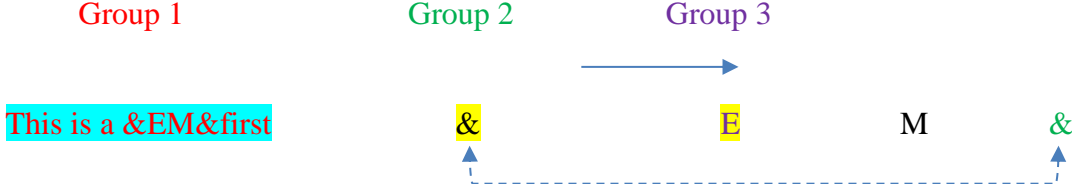
RegEx Engine moves forward. Capture Group 1 matches against the 'T'.

Step 2. **Group 1**


RegEx Engine moves forward to EOL. Capture Group 1 matches against the remaining string.

Step 3. **Group 1** **Group 2**


RegEx Engine backtracks. Capture Group 2 matches against the ampersand. Capture Group 1 gives up the characters, '& test'.

Step 4. **Group 1** **Group 2** **Group 3**


RegEx Engine moves forward. Capture Group 3 matches against the 'E'. Capture Group 2 gives up its value and reevaluates to match against the ampersand following 'first'. Capture Group 1 gives up the characters, '&EM'.

Step 5. **Group 1** **Group 2** **Group 3**


RegEx Engine moves forward. Capture Group 3 adds the 'M' to its previous value, 'E'. All other values unchanged.

Step 6. **Group 1** **Group 2** **Group 3**


RegEx Engine moves forward. Capture Group 3 adds the '&' to its previous value, 'EM'. All other values unchanged.

Greed and Lazy Examples

#1. Isolate both 'EM&'s cont.

String = This is a &EM&first&EM& test

Match: `(.*)&([A-Z]+\2)(.*)&([A-Z]+\2)`

Where:

\1 =	This is a<space>	\2 =	&	\3 =	EM& (first occurrence)
\4 =	first	\5 =	&	\6 =	EM& (second occurrence)

It gets a bit complicated from here.

7. . Dot Metacharacter. Matches against any character. Capture Group 4 = <space>
Matches against the <space> preceding 'test'.

Step 7.

Group 3	Group 4		
		→	
EM&	<space>	test	<u>EOL</u>

RegEx Engine moves forward. Capture Group 4 matches the <space> preceding 'test'. All other values are unchanged.

8. * Make it Greedy. Capture Group 4 = <space> test
Current position = EOL.

Step 8.

Group 3	Group 4		
		→	
EM&	<space> test	<u>EOL</u>	

RegEx Engine moves forward. Capture Group 4 matches the remaining string to EOL.

9. & Match against literal string character, '&'. Capture Group 5 = &
Already at EOL. Backtracks to match against the ampersand preceding '<space> test'. This results in reevaluating the values in all previous Capture Groups. Changes Capture Group 4 = first&EM
Changes Capture Group 3 = EM&
Changes Capture Group 2 = &
Changes Capture Group 1 = This is a <space>

Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

String = This is a &EM&first&EM& test

Match: `(.*)&([A-Z]+\2)(.*)&([A-Z]+\2)`

Where:

\1 =	This is a<space>	\2 =	&	\3 =	EM& (first occurrence)
\4 =	first	\5 =	&	\6 =	EM& (second occurrence)

Capture Group 5 Backtracks to match against the ampersand that precedes '<space> test'. Because the '<space> test' value was previously held by Capture Group 4 and the ampersand that precedes it was part of the value held by Capture Group 3, this creates a domino effect where all the other Capture Groups have to be reevaluated.

Before Step 9, the evaluations are:

Group 1	Group 2	Group 3	Group 4	
This is a &EM&first	&	EM&	<space> test	<u>EOL</u>

After Step 9, all of the other Capture Groups are reevaluated:

Group 1	Group 2	Group 3	Group 4	Group 5	
This is a <space>	&	EM&	first&EM	&	<space> test EOL
.*	&	[A-Z]/2 &	.*	&	

I used a colour code to visually display the overlaps.

Capture Group 2 has to match against the pattern of an ampersand followed by an uppercase letter. It fails to match against the '&' that precedes 'f' of 'first'. It does, however, match against the '&' that precedes the <space> of 'a <space>' because of the uppercase letter of 'E' that follows. As a consequence, this forces Capture Group 1 to discard the characters, '&EMfirst'. All of the other Capture Groups give up their value and are reevaluated.

Greed and Lazy Examples

#1. Isolate both 'EM&'s cont.

String = This is a &EM&first&EM& test

Match: `(.*)&([A-Z]+\2)(.*)&([A-Z]+\2)`

Where:

\1 =	This is a<space>	\2 =	&	\3 =	EM& (first occurrence)
\4 =	first	\5 =	&	\6 =	EM& (second occurrence)

Step 9. cont.

Group 1	Group 2	Group 3	Group 4	Group 5	
This is a <space>	&	EM&	first&EM	&	<space> test <u>EOL</u>

Capture Group 3 has to match against one or more uppercase letters followed by an ampersand and this matches against the 'EM&', that follows after the capture of the, '&', held by Capture Group 2.

Capture Group 4 captures the remainder of the string up to the value captured in Capture Group 5, the ampersand that precedes, '<space> test'.

After the reevaluations, the result is:

Group 1	Group 2	Group 3	Group 4	Group 5	
This is a <space>	&	EM&	first&EM	&	<space> test <u>EOL</u>

The '<space> test' are not captured.

Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

String = This is a &EM&first&EM& test

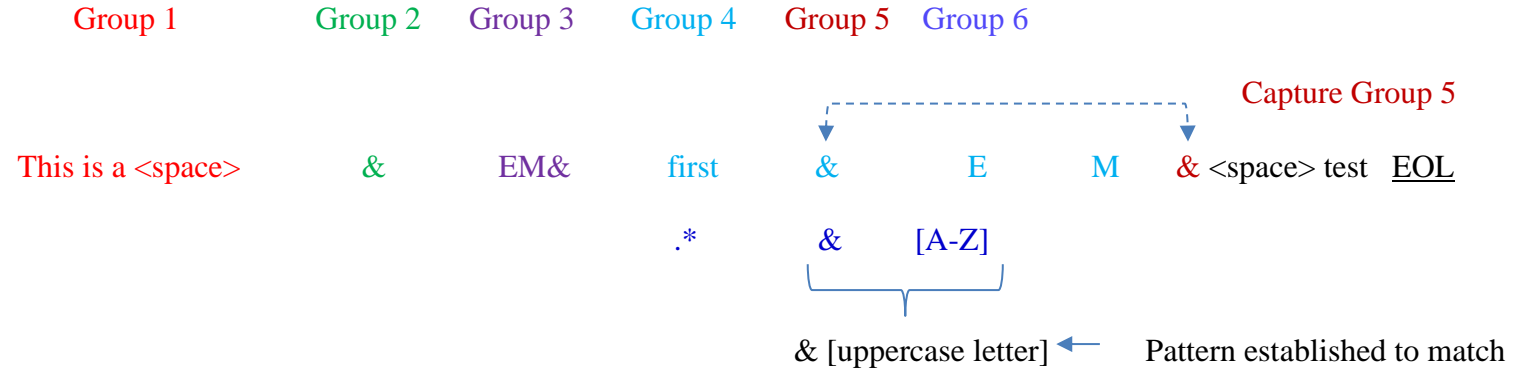
Match: `(.*)&([A-Z]+\2)(.*)&([A-Z]+\2)`

Where:

\1 =	This is a<space>	\2 =	&	\3 =	EM& (first occurrence)
\4 =	first	\5 =	&	\6 =	EM& (second occurrence)

10. [A-Z]	Match against a class consisting of uppercase letter. Matches against the 'E' of the second occurrence of the '&EM&.	Capture Group 6 = E Changes Capture Group 5 = & Changes Capture Group 4 = first
-----------	---	---

The current position after the '&' that was captured by Capture Group 5 is the <space> that precedes 'test'. Cannot match against the <space>, so after hitting the EOL, it backtracks to match against the pattern of '&', established in step 9, **followed** by an uppercase letter' in the class of [A-Z]. This fails to match against the 'M' of 'first&EM' because of the 'E' that precedes the 'M' but it does match against the 'E' of 'first&EM'.



Attempts to move forward but cannot match the pattern '& uppercase letter' against the '<space> test' so it backtracks still testing to match the pattern, until it matches the '& Uppercase letter' to the 'E' that precedes 'first'.

Capture Group 5's value changes although this is not apparent because it is the same character as before. What has changed is which character has been matched. When the 'E' is captured, Capture Group 5's value changes from the ampersand preceding '<space> test' to the ampersand that follows after 'first'.

When Capture Group 6 captures the 'E' value, Capture Group 5 is forced to give up its value and the sub-expression in step 9 is re-evaluated as a result. Capture Group 4 is forced to drop the characters, '&EM'. Why? Because the ampersand of that value of '&EM' is now held by Capture Group 5 and all the following characters are discarded.

11. +	Make it Greedy.	Capture Group 6 = EM
12. \2	Backreference to value held in Capture Group 2. The value in Capture Group 2 is '&'. Moves forward to match against the ampersand.	Capture Group 6 = EM&

Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

Using just one of the two expressions of the RegEx, results in this:

String = This is a &EM&first&EM& test

Match: (.*)(&)([A-Z]+2)

Beginning match attempt at character 0

.* → Capture Group 1

& → Capture Group 2

[A-Z]+ → Capture Group 3

\2

Match found in 46 steps

1	This is a &EM&first&EM& test
2	This is a &EM&first&EM& test
3	This is a &EM&first&EM& test backtrack
4	This is a &EM&first&EM& test backtrack
5	This is a &EM&first&EM& tes
6	This is a &EM&first&EM& tes
7	This is a &EM&first&EM& tes backtrack
8	This is a &EM&first&EM& tes backtrack
9	This is a &EM&first&EM& te
10	This is a &EM&first&EM& te
11	This is a &EM&first&EM& te backtrack
12	This is a &EM&first&EM& te backtrack
13	This is a &EM&first&EM& t
14	This is a &EM&first&EM& t
15	This is a &EM&first&EM& t backtrack
16	This is a &EM&first&EM& t backtrack
17	This is a &EM&first&EM&
18	This is a &EM&first&EM&
19	This is a &EM&first&EM& backtrack
20	This is a &EM&first&EM& backtrack
21	This is a &EM&first&EM&
22	This is a &EM&first&EM&
23	This is a &EM&first&EM& backtrack
24	This is a &EM&first&EM& backtrack
25	This is a &EM&first&EM
26	This is a &EM&first&EM
27	This is a &EM&first&EM&
28	This is a &EM&first&EM&
29	This is a &EM&first&EM& backtrack
30	This is a &EM&first&EM& backtrack
31	This is a &EM&first&EM& backtrack
32	This is a &EM&first&E
33	This is a &EM&first&E
34	This is a &EM&first&E backtrack
35	This is a &EM&first&E backtrack
36	This is a &EM&first&
37	This is a &EM&first&
38	This is a &EM&first& backtrack
39	This is a &EM&first& backtrack
40	This is a &EM&first
41	This is a &EM&first
42	This is a &EM&first&
43	This is a &EM&first&
44	This is a &EM&first&EM
45	This is a &EM&first&EM&
46	This is a &EM&first&EM&

Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

String = This is a &EM&first&EM& test

Match: (.*)(&)([A-Z]+\2)(.*)(&)([A-Z]+\2)

	Start	Length
Match 1: This is a &EM&first&EM&	0	23
Group 1: This is a	0	10
Group 2: &	10	1
Group 3: EM&	11	3
Group 4: first	14	5
Group 5: &	19	1
Group 6: EM&	20	3

Because of all of the numerous changes that are a result of the addition of the second expression, there are numerous backtracking steps. This may slow down processing if you have a lot of files.

Steps 1 through 46 are basically the same as before. After step 46, it begins the second expression:

Step 8 .*
Capture Group 4

46	This is a &EM&first&EM&
47	This is a &EM&first&EM& test
48	This is a &EM&first&EM& test
49	This is a &EM&first&EM& test backtrack
50	This is a &EM&first&EM& test backtrack
51	This is a &EM&first&EM& tes
52	This is a &EM&first&EM& tes
53	This is a &EM&first&EM& tes backtrack
54	This is a &EM&first&EM& tes backtrack
55	This is a &EM&first&EM& te
56	This is a &EM&first&EM& te
57	This is a &EM&first&EM& te backtrack
58	This is a &EM&first&EM& te backtrack
59	This is a &EM&first&EM& t
60	This is a &EM&first&EM& t
61	This is a &EM&first&EM& t backtrack
62	This is a &EM&first&EM& t backtrack
63	This is a &EM&first&EM&
64	This is a &EM&first&EM&
65	This is a &EM&first&EM& backtrack
66	This is a &EM&first&EM& backtrack
67	This is a &EM&first&EM&ok
68	This is a &EM&first&EM&ok

Step 9 &
Capture Group 5

Notes:

1. My indicators for the Debug Logs of Regex Buddy program are only estimates on where the final change takes place.

Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

String = This is a &EM&first&EM& test

Match: (.*)&([A-Z]+&2)(.*)&([A-Z]+&2)

	Start	Length
Match 1: This is a &EM&first&EM&	0	23
Group 1: This is a	0	10
Group 2: &	10	1
Group 3: EM&	11	3
Group 4: first	14	5
Group 5: &	19	1
Group 6: EM&	20	3

```

69 This is a &EM&first&EM&backtrack
70 This is a &EM&first&EM&backtrack
71 This is a &EM&first&EM&backtrack
72 This is a &EM&first&E
73 This is a &EM&first&Ebacktrack
74 This is a &EM&first&backtrack
75 This is a &EM&firstbacktrack
76 This is a &EM&firs
77 This is a &EM&firs
78 This is a &EM&firsbacktrack
79 This is a &EM&firsbacktrack
80 This is a &EM&fir
81 This is a &EM&fir
82 This is a &EM&firbacktrack
83 This is a &EM&firbacktrack
84 This is a &EM&fi
85 This is a &EM&fi
86 This is a &EM&fibacktrack
87 This is a &EM&fibacktrack
88 This is a &EM&f
89 This is a &EM&f
90 This is a &EM&fbacktrack
91 This is a &EM&fbacktrack
92 This is a &EM&
93 This is a &EM&
94 This is a &EM&backtrack
95 This is a &EM&backtrack
96 This is a &EM
97 This is a &EM
98 This is a &EM&
99 This is a &EM&
    
```

```

100 This is a &EM&backtrack
101 This is a &EM&backtrack
102 This is a &EMbacktrack
103 This is a &E
104 This is a &E
105 This is a &Ebacktrack
106 This is a &Ebacktrack
107 This is a &
108 This is a &
109 This is a &backtrack
110 This is a &backtrack
111 This is a
112 This is a
113 This is a &
114 This is a &
115 This is a &EM
116 This is a &EM&
117 This is a &EM&
    
```

Step 9 Reevaluate Capture Group 1 →

Step 9 Reevaluate Capture Group 2 →

Step 9 Reevaluate Capture Group 3 →

Group 1 Group 2 Group 3

This is a <space>

&

EM&



Greed and Lazy Examples

#1. Isolate both 'Em&'s cont.

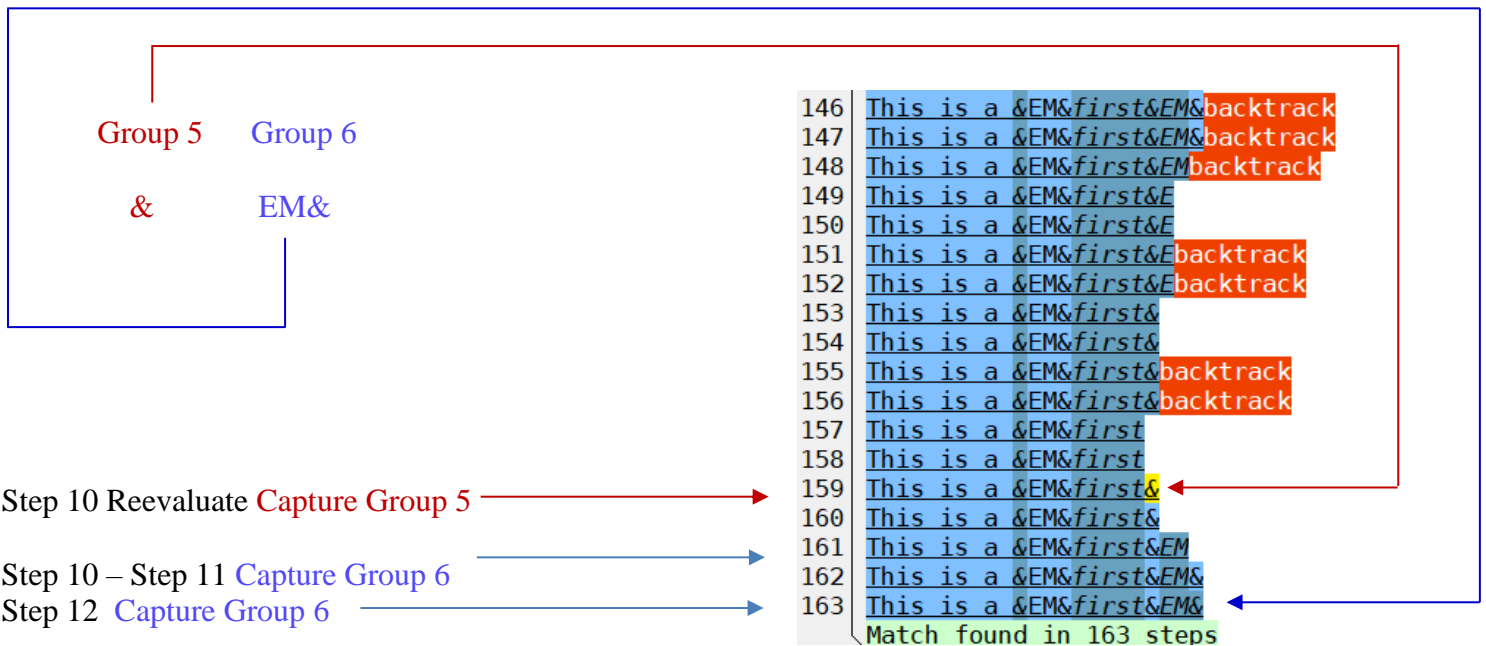
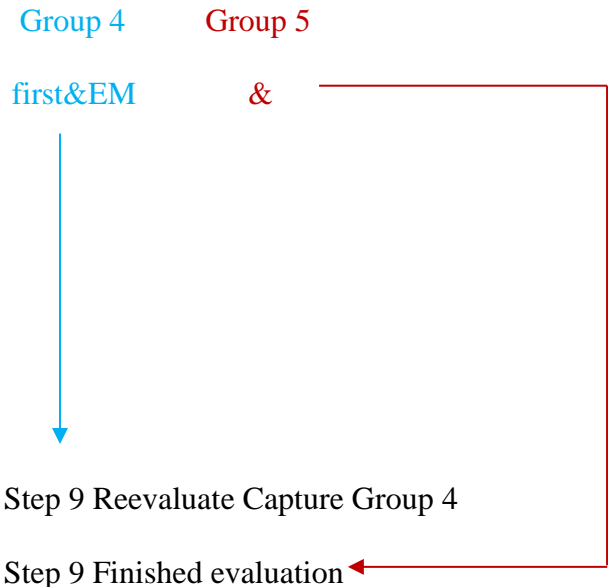
String = This is a &EM&first&EM& test

Match: `(.*)(&)([A-Z]{+2})(.*)(&)([A-Z]{+2})`

	Start	Length
Match 1: This is a &EM&first&EM&	0	23
Group 1: This is a	0	10
Group 2: &	10	1
Group 3: EM&	11	3
Group 4: first	14	5
Group 5: &	19	1
Group 6: EM&	20	3

```

118 This is a &EM&first&EM& test
119 This is a &EM&first&EM& test
120 This is a &EM&first&EM& test backtrack
121 This is a &EM&first&EM& test backtrack
122 This is a &EM&first&EM& tes
123 This is a &EM&first&EM& tes
124 This is a &EM&first&EM& tes backtrack
125 This is a &EM&first&EM& tes backtrack
126 This is a &EM&first&EM& te
127 This is a &EM&first&EM& te
128 This is a &EM&first&EM& te backtrack
129 This is a &EM&first&EM& te backtrack
130 This is a &EM&first&EM& t
131 This is a &EM&first&EM& t
132 This is a &EM&first&EM& t backtrack
133 This is a &EM&first&EM& t backtrack
134 This is a &EM&first&EM&
135 This is a &EM&first&EM&
136 This is a &EM&first&EM& backtrack
137 This is a &EM&first&EM& backtrack
138 This is a &EM&first&EM&
139 This is a &EM&first&EM&
140 This is a &EM&first&EM& backtrack
141 This is a &EM&first&EM& backtrack
142 This is a &EM&first&EM
143 This is a &EM&first&EM
144 This is a &EM&first&EM&
    
```




Greed and Lazy Examples

#2. isolate 'This is a test'

String = This is a &EM&first&EM& test.jpg

2 methods:

(1). Match using generic pattern

Name ▲	New Name
 This is a &EM&first&EM& test.jpg	This is a test.jpg

Match: `(.*)((.*)(((.*))`

Replace: `\1\4\5`

\1 = This is a


\2 = <space>

\3 = &EM&first&EM&

\4 = <space>

\5 = test

(2). Match by isolating words

Name ▲	New Name
 This is a &EM&first&EM& test.jpg	This is a test.jpg

Match: `^(\\D+?\\b)(\\b\\D+?\\b)(\\b\\D+?\\b)(\\b\\D+?\\b)(\\b\\D+?\\b)(.*)\\s(.*)`

Replace: `\1\2\3\4\5\2\7`

Where:

\1 = This

\2 = <space>

\3 = is

\4 = <space>

\5 = a

\6 = &EM&first&EM&

\7 = test

This one captures each word separately.

The format is

`^(\\D+?\\b)` - for the word at the beginning of the filename

`(\\b\\D+?\\b)` - next word within filename

The question mark makes the Quantifier + Lazy instead of Greedy. So instead of more than one word being consumed it only consumes what is required until it hits `\\b` which is whitespace (the space after the word).

Captures each word - 'This' <space> 'is' <space> 'a'

`(.*)` Then it switches to collecting remainder of line = &EM&first&EM&

`\\s` - follows to the space right before 'test'

`(.*)` - collects the remainder of the line = 'test'

Backtracking

Let's see exactly what this is and how it works.

When a Greedy Quantifier is evaluated, it consumes (captures) all of the characters from the current position on to the end of the string. It does not matter if the match has been satisfied or not. It doesn't matter if there is still more of the RegEx left to process. These characters are then held in the Capture Group that contained the Quantifier.

For example, this Quantifier is commonly used:

(.*)

The dot indicates to match any character. The asterisk is a greedy quantifier that tells it to repeat the dot endlessly until EOL, End of Line, or more to the point in BRU, 'end of filename'.

Example:

String = Test 123.txt

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

Name	New Name
Test 123.txt	Test.txt

The string is made up of 8 characters.

T e s t 1 2 3
 0 1 2 3 4 5 6 7

	Start	Length
Match 1: Test	0	5 ^{CL} _{RF}
Group 1: Test	0	4 ^{CL} _{RF}
Group 2:	4	1

Capture Group 1 (.*): gather up all the characters of the string. Current position is at end of string = Test 123

Capture Group 2 (): **Locate space**. Changes Capture Group 1 value to **Test**. Why? Because the <space> character, already a part of Capture Group 1, is given up to Capture Group 2. Capture Group 1's value changes to reflect all characters up to the <space>. This is the value, 'Test'. The remainder of the string, '123', also a part of Capture Group 1 previously, was not included in Capture Group 2 and is discarded.

This is where it gets interesting. The current position is at the end of the string but the engine has to process Capture Group 2. Processing will now continue from the end of the string to the front. This is called **backtracking**.

Notes:

Additional photos provided by the program Regex Buddy. For more information on this software, refer to Volume II.

Backtracking

It next tests the current character at positions, 7, 6, 5.. until it finds the space at position 4. The current position is at position 4:

```

T e s t   1 2 3
0 1 2 3 4 5 6 7
          ↑

```

Backtracking

The character from position number 4 is stored in Capture Group 2.
= <space>

What happens now is this. The engine must rectify the string stored in the previous Capture Group, because there are too many characters stored.

Capture Group 2 has yet to be processed so the greedy quantifier takes too many characters the first time. The engine is designed to remove whatever characters Capture Group 2 stores from that position (4) to the end of the string from the value stored in Capture Group 1.

Capture Group 1 originally =

```

T e s t   1 2 3
0 1 2 3 4 5 6 7

```

Capture Group 2 = <space> stored from position 4

Capture Group 1 is corrected by taking away all characters from position 4 to end of string.

```

T e s t   1 2 3
0 1 2 3 4 5 6 7

```

Capture Group 1 corrected to Test

```

\1 = Test
\2 = <space>

```

Notes:

1. Backtracking can slow down processing especially if there are a lot of files to rename. If a RegEx is processing slow it may be time to redo it so there is less backtracking involved. For example, you may need to use Possessive Quantifiers and see if you can come up with the same results.
2. Corrections may also be made to a Capture Group if the same character added to one Capture Group is part of the value of another. In that case, each character is simultaneously dropped from the original as it is added to the new.
3. For more detailed information on Backtracking and how it is used, refer to Volume II which provides many examples and full analysis of sample RegEx.

Non-marking Groups

Normally grouping part of the RegEx pattern with the parentheses creates a Capture Group. These Capture Groups, also referred to as ‘marked’ groups, capture data values that can be recalled by using their designated number in the form `\n`. There are often times when a grouping is required without a capturing requirement. This can be used to ‘not include’ certain matched groups and drop them from the result.

syntax: `(?:)`

e.g.,


`(?:ab)+` repeats the "ab" sub-expression match without creating a Capture Group

Example:

String = `Graham James Edward Miller.jpg`

Match: `(.*?)(?: .*) (.*)` which is ... `(.*?)(?:<space>.*)<space>(.*)`


Replace: `\1, \2`

Name ▲	New Name
 Graham James Edward Miller.jpg	Graham, Miller.jpg

This always matches the first item, e.g., ‘Graham’ in Capture Group 1 and the **next to the last item**, e.g., ‘Edward’ in Capture Group 2 no matter what the string, as long as there are spaces between items.

Match: `(.*?)(?: .*)(?:.*) (.*)` which is ... `(.*?)(?:<space>.*)(?:.*)<space> (.*)`

Replace: `\1, \2`

Name ▲	New Name
 Graham James Edward Miller.jpg	Graham, Miller.jpg

This always matches the first item, e.g., ‘Graham’ in Capture Group 1 and the **last item**, e.g., ‘Miller’ in Capture Group 2 no matter what the string, as long as there are spaces between items.

Using Options with RegEx

Case Insensitive

What if we needed a Case Insensitive match?

There is no metacharacter that performs this. But there is something called a **mode modifier or option** that can.

At the beginning of a Regular Expression, you can specify modifiers followed by a close parentheses.

i) case insensitive matching

Example:

```
String = Andrew Gregory Macintyre.jpg
```

```
Match: (andrew|Mac)
```

```
Replace: \1 = Mac
```

Even though “Andrew” was first in the string, it did not match because the pattern is always case sensitive.

However, if you use this:

```
Match: i)(andrew|Mac) with a Capture Group
```

or even this-

```
Match: i)andrew|Mac without a Capture Group
```

It should return ‘andrew’ but it doesn’t. Even though \1 should equal ‘Andrew’ in the first example because the match is made insensitive.

The problem is that BRU does not accept closed parentheses by itself – there must always be an equal number of open and closed parentheses.

For example:

```
Match: (red|white) (king|queen))
```

The expression is **Invalid** because there are only 2 open parentheses and 3 closed parentheses.

There is a way around this. A Work-around has been found and fully tested.

Using Options with RegEx

Case Insensitive cont.

Because BRU cannot have uneven brackets, the modifier cannot be recognized. If you have –

(i)

... then BRU will treat it as a match against ‘i’ instead of as a modifier or option.

Instead use an unmarked group. Why? Because it already has a format that BRU understands. But if the data needed to be captured, this wouldn’t work either:

Example:

String = Andrew Gregory Macintyre.jpg

Match: (?i:gre)

Results in: ‘Gre’ matching but not captured. Just moves the position to the lowercase ‘g’ in Gregory


Name	New Name
 Andrew Gregory Macintyre.jpg	\1.jpg

The solution is to create a Capture Group around the unmarked group:

String = Andrew Gregory Macintyre.jpg


Match: ((?i:gre))

Where: \1 = Gre

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Gre.jpg

Notes:

1. The current position is revealed by adding the Capture Group ‘(...)’ after the initial pattern to capture the current position and two characters after. The value is captured and stored in Capture Group 2. By recalling the value, I can determine the current position of the RegEx engine..

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	gor.jpg

For example:

Match: ((?i:gre))(...)

Where : \2 = gor current position is at ‘g’ after ((?i:gre)) has finished (1st character captured = g)

Using Options with RegEx

Case Insensitive cont.

Using the Case Insensitive Modifier- Turning it On.

As you can see on the previous page, this method does work with BRU. Here is an alternate method:

You can use options in the form (?i) as in (?i)(gre) returns 'Gre' from the string, Andrew Gregory Macintyre.jpg

Example:

String = Graham the aB ab abc abC ABC Miller.jpg

Match: (a(?i)(BC))(.....)
 Replace: Group 1 = \1 Group 2= \2 Group 3 = \3

Where:

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Group 1 = abc Group 2= bc Group 3 = abC AB.jpg

\1 = abc

\2 = bc

\3 = <space>abC<space>AB

You can use the modifier either within a Capture Group (previous page) or outside of a Capture Group (above), but the syntax is slightly different for each. You can even have a mixture within the same RegEx.

Syntax for using the modifier outside of a Capture Group: (?i)

The modifier is placed within parentheses without the colon (this does not Capture anything).

Syntax for using the modifier within a Capture Group: ((?i:<expression>))

The modifier is placed in an Unmarked Group (?:<expression>). The Capturing is performed using an additional set of parentheses around the Unmarked Group ((?:<expression>). Adding the modifier to the Group gives you the proper syntax for this modifier – ((?i:<expression>)).

The rules are as follows:

Within a Capture Group, the modifier only applies to the evaluation of the expression within. Meaning that if you have two Capture Groups, the first with the modifier and the second without, only the first Capture Group is Case Insensitive and the second Capture Group would remain Case Sensitive.

Outside of a Capture Group, the entire expression from that point onward is made Case Insensitive. This will be in effect until either EOL is reached or the modifier is turned off.

Because of the nature of this, I call this work-around of the modifier, Case Insensitivity on Demand.

Using Options with RegEx

Case Insensitive cont.

Using the Modifier – Turning it Off.

Syntax for using the modifier outside of a Capture Group: (?-i)

The modifier is placed within parentheses without the colon (this does not Capture anything).

Syntax for using the modifier within a Capture Group: ((?-i:<expression>))

The modifier is placed in an Unmarked Group (?:<expression>). The Capturing is performed using an additional set of parentheses around the Unmarked Group ((?:<expression>). Adding the modifier to the Group gives you the proper syntax for this modifier – ((?-i:<expression>)).

The rules are as follows:


Within a Capture Group, the modifier to turn it on and off can reside within the same expression (On Demand). Outside of this Capture Group or within another Capture Group, the default of Case Sensitivity would be in effect regardless.

Outside of a Capture Group, the entire expression from that point onward is reverted back to Case Sensitive and will remain so unless another modifier turns Case Insensitivity On once more.

Here are some examples using the string, Andrew Gregory Macintyre.jpg

Outside of Group – Turned on until EOL or turned off by modifier

Match: (?i)(gregory) (macin)
Replace: \1 \2

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Gregory Macin.jpg

Inside of Group – affects current Group only


Match: ((?i:gregory)) (macin)
Replace: \1 \2

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Andrew Gregory Macintyre.jpg

No match because the Case Insensitive is placed within the first Capture Group only, therefore, gregory would match “Gregory” but ‘macin’ would not match “Macin” in the string.

Inside of Groups - Affects each Group only

Match: ((?i:gregory)) ((?i:macin))
Replace: \1 \2

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Gregory Macin.jpg


Using Options with RegEx

Case Insensitive cont.

Here are some additional examples using the string, Andrew Gregory Macintyre.jpg


on Demand, Turned on, on Demand, Turned off

Match: `(?i)(gregory)(?-i) (Macin)`
 Replace: `\1 \2`

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Gregory Macin.jpg

Placed Outside of Groups – Turned on, then off, then on again

Match: `(?i)(andrew) (?-i)(Gregory)(?i) (macin)`
 Replace: `\1 \2 \3`

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Andrew Gregory Macin.jpg


Within Groups – One Group is turned on, one Group, turned off, third group turned on

Match: `((?i:andrew)) ((?-i:Gregory)) ((?i:macin))`
 Replace: `\1 \2 \3`

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Andrew Gregory Macin.jpg

The group that is turned off is not needed because each group is separate, therefore you could have this instead:


Match: `((?i:andrew)) (Gregory) ((?i:macin))`
 Replace: `\1 \2 \3`

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Andrew Gregory Macin.jpg

... and still produce the same result.

Mixed –

Match: `((?i:andrew)) (?-i)(Gregory)(?i) (macin)`
 Replace: `\1 \2 \3`

Name ▲	New Name
 Andrew Gregory Macintyre.jpg	Andrew Gregory Macin.jpg

Using Options with RegEx

Free-Space Option

Another option that BRU supports is Free-Space (**?x**)

Normally the RegEx expression uses a syntax of 'Exact-Spacing'. This means that any whitespace in the expression is deliberate and meant to convey as part of the expression. For example, there may be a <space> in the string that requires a location of a <space> character in the RegEx in order to match.

However, sometimes for clarity it would be nice to freely have whitespace. This is what this option does. Any whitespace in the RegEx is ignored and does not affect the legality of the syntax. If this option is invoked, BRU will still be able to evaluate the expression.


In BRU, whitespace are <space> characters because other whitespace characters, tab, new line and carriage return, are not supported just out of common sense. They would only be useful in a multi-line application and BRU is a single line application. So when referring to any whitespace, I am referring to <space> characters.

Let's use the example taken from the discussion on Greed and Laziness -

String = This is a &EM&first&EM& test.jpg

```
Match:      (.*)(?'test'&)([A-Z]+\2)
Replace:    \2\3
```

The above is the original match RegEx string using exact spacing (except that I have substituted a named Capture Group for Capture Group 2).

Name ▲	New Name
 This is a &EM&first&EM& test.jpg	&EM&.jpg

Now let's try that same expression using the Free-Space option.


```
Match:      (?x)(.*)(?'test'&)([A-Z]+\2)
```

Now with the Free-Space option in place, I can add the whitespace without affecting BRU's evaluation:

```
Match:      (?x) (.*) ('test' &) ( [A-Z] + \2)
Replace:    \2\3
```

Where:

```
\1 = This is a &EM&first
\2 = &
\3 = EM&
```

Name ▲	New Name
 This is a &EM&first&EM& test.jpg	&EM&.jpg

Using Options with RegEx

Free-Space Option cont.

Notes:

1. You can only add whitespace between expressions or sub-expressions in the RegEx but not within the syntax that makes up a component of a sub-expression.

For example you could not have `(? 'test')` because the syntax of the named Capture Group is, `(?'name')`

2. The only whitespace (that I am aware of) that BRU supports are <space> characters.

Using Options with RegEx

Comments

Comments in Free-Space

Although not too practical, BRU supports comments with the Free-Space option. It could be used for documenting purposes.

A comment is indicated by the numeric sign, #. Anything after the numeric is ignored as part of the expression.

So you could have:

```
Match: (?x) (.*) ('test' &) ( [A-Z] + \2) # this is a comment
```

Comments in Exact-Spacing

You are not limited to comments only in Free-Space. You can also add comments in Exact-Space as well. The syntax is a little different.

Syntax:

```
(?# comment )
```

For Example:

```
Match: (.*)(?# This is a comment)('test'&)([A-Z]+\2)(?# This is another comment)
```

Notes:

1. Comments display only within the RegEx and not in New Name. As I said before, not too practical.

If Then Else

A special construct (`?if then|else`) allows you to create conditional Regular Expressions. If the *if* part evaluates to **true**, then the RegEx engine will attempt to match the *then* part. Otherwise, the *else* part is attempted instead. The syntax consists of a pair of parentheses. The opening parenthesis must be followed by a question mark, immediately followed by the *if* part, immediately followed by the *then* part. This part can be optionally followed by a vertical bar and the *else* part along with the closing parenthesis. The *if* part can also be a Lookahead.

Syntax (using a Lookahead):

```
(?(?=RegEx)then|else)
```

Example:

```
String = xghiy
```

```
Match:      ((x)?g?(1)c|h))
```

```
Where :      \1 = xgh      \2 = x
```

Analysis:

Capture Group 2 is nested inside Capture Group 1.

(x)?	optional capturing group	Capture Group 2 = null
g	Search for literal 'g' character. Matches 'g' of 'xghiy' (not captured)	Capture Group 1 = xg Changes Capture Group 2 = x

At this point, because of the match for the 'g', Capture Group 2 changes from a **null** value to holding the value of the 'x' and the aggregated value of Capture Group 1 becomes 'xg'.

(?)(1)c h)	Lookahead used in an If/Then statement.	Capture Group 1 = xgh
------------	---	-----------------------

The Lookahead creates a conditional statement that tests the capturing group for a 'c' OR 'h' character. If Capture Group 1 has a value (successfully matched), *then* Lookahead for the 'c' *else* for the 'h'. A Lookahead is an assertion and does not capture a value, but I placed the Lookahead in a Capture Group, thereby the 'h' value matched is captured.


Notes:

1. Information on Lookaheads will be discussed shortly.

Using multiple Capture Groups to locate Partial Words

String = Graham the aB ab abc abC ABC Miller.jpg

Match: (Gra(ham))
 Replace: Capture Group 1= \1 Capture Group 2 = \2

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Capture Group 1= Graham Capture Group 2 = ham.jpg

Where: \1 = Graham
 \2 = ham

Using this match ...

Match: ((?i)(Gra(ham))(.*)mil)
 Replace: Capture Group 1= \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4

New Name
Capture Group 1= Graham the aB ab abc abC ABC Mil Capture Group 2 = Graham Capture Group 3 = ham Capture Group 4 = the aB ab abc abC ABC .jpg

Where : \1 = Graham the aB ab abc abC ABC Mill
 \2 = Graham
 \3 = ham
 \4 = <space>the aB ab abc abC ABC<space>

.. or this match

Match: ((?i)(gra(Ham))(.*)mIl)
 Replace: Capture Group 1= \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4

New Name
Capture Group 1= Graham the aB ab abc abC ABC Mil Capture Group 2 = Graham Capture Group 3 = ham Capture Group 4 = the aB ab abc abC ABC .jpg

Where: \1 = Graham the aB ab abc abC ABC Mill
 \2 = Graham
 \3 = ham
 \4 = <space>the aB ab abc abC ABC<space>

.. both return the same thing because the pattern is case insensitive by the inclusion of (?i)

Using multiple Capture Groups to locate Partial Words

Analysis: (a bit complex)

Match: `((?i)(gra(Ham))(.*)mIl)`

Nested Groups:

```

          C a p t u r e   G r o u p   1
          ( ( (?i) ( gra ( Ham ) ) ( .* ) mIl ) )
          ( ( gra ( Ham ) ) ( .* ) )
          Capture Group 2           Capture Group 4
          ( gra ( Ham ) ) ( .* )
          Capture Group 3
  
```

Where:

Capture Group 1 consists of nested Capture Groups 2-4 + mIl
 Capture Group 2 consists of nested Capture Group 3
 Capture Group 4 is only nested within Capture Group 1

- | | | |
|----------------------|--|--|
| 1. <code>(?i)</code> | Turn on Case Insensitivity | |
| 2. <code>gra</code> | Match against string, 'gra' | Capture Group 2 = Gra
Capture Group 1 = Gra |
| 3. <code>Ham</code> | Match against string, 'Ham' | Capture Group 3 = ham
Changes Capture Group 2 = Graham
Changes Capture Group 1 = Graham |
| 4. <code>.*</code> | Capture remainder of string | Capture Group 4 = the aB ab abc abC ABC Miller
Changes Capture Group 1 =
Graham the aB ab abc abC ABC Miller |
| 5. <code>mIl</code> | Match against string, 'mIl'
already at <u>EOL</u> so Backtracks | Changes Capture Group 4 =
the aB ab abc abC ABC<space>
Capture Group 1 =
Graham the aB ab abc abC ABC Mil |

In the RegEx, 'mIl' is inclusive to Capture Group 1 and not nested in its own Capture Group as are Groups 2 – 4. This is why Capture Group 4 changes in step 5 to drop off 'Mil', because it is now included in Capture Group 1. Also in step 5, because 'mIl' is part of the Capture Group 1 expression, the value is not changed, it is a continuation of the evaluation.

Capture Group's 1 value it is the aggregated value of Capture Groups 2 – 4 inclusive + the evaluation of mIl.
 Capture Group's 2 value is the aggregated value of Capture Group 3 inclusive + the evaluation of gra.

Colour coded for clarification.

Lookarounds: *Lookahead and Lookbehind*

Lookarounds, are zero-length assertions. They are called zero length because they consume no characters during a match; they only indicate that a match is possible. Other zero length metacharacters are:

^ start of line (Beginning of Line or **BOL**) **\$** end of line (**EOL**) **\b** Word Boundary

But anchors don't match characters, Lookarounds do but don't capture them. That is why they are called "assertions". They do not consume characters in the string, but only assert whether a match is possible or not. Using Lookarounds you can create Regular Expressions that would be impossible otherwise.

Notes:

1. Anchors, match positions, not characters.
2. Word Boundaries match positions between characters based on a character's value to the immediate left and right of the current position.
3. Lookarounds match characters but don't capture them unless a Capture Group is included in the Lookaround.
4. Lookarounds do not consume characters. When a match is found, the position remains unchanged from the position before the match took place. Further evaluation of the RegEx would take place from this position.

The RegEx Engine does not advance after a match of a Lookaround. In a typical match, characters would be consumed and the RegEx Engine would advance to the position of the last character *following* the match.

5. After a Lookaround matches, the RegEx Engine can continue evaluation looking for another match or look ahead or behind for something else.

This will be further explained.

Notes:

1. In Regex Buddy Debug Logs, Lookarounds are not documented except for when they match, then the log will just indicate a successful match by the singular label, 'OK'.

Lookarounds: *Lookahead and Lookbehind*

Lookahead asserts

There are two forms:

positive forward Lookahead asserts

negative forward Lookahead asserts

syntax **(?=n)**

(?!n)

where n is what you want to match without capturing

Positive Lookahead: match something that is *immediately followed* by something else.
(looks to the right)

Negative Lookahead: match something *immediately not followed* by something else.
(looks to the right)

Using Expressions within Lookaheads

You can use any Regular Expression inside a Lookahead. Lookaheads that contain Capturing Groups will capture as normal and backreferences to them will also work normally, even outside the lookahead.

To capture the value of an expression inside of a Lookaround, we do what we did when capturing values in other non-marking groups, by enclosing the non-marking group expression within a second set of parentheses. The first set of parentheses in this example, the outer set, remains as part of the syntax while the inner set serves to Capture any value obtained from the Lookaround before the match is given up and the **null** value results.

Place the RegEx inside the Lookaround within a Capture Group:

(?=(RegEx))

e.g., **(?=(\d{3} dollars))**

100 pesos 100 dollars Plantation 6490

100 pesos 100 dollars Plantation 6490 **100 dollars**

Lookahead (look right) for 3 numeric digits followed by <space> ‘dollars’. Matches the ‘100 dollars’ and this value is captured in Capture Group 1. If it wasn’t for the Capture Group, after the evaluation of the Lookaround, any matches are given up and the value returned is **null**.

This won’t work:

((?=RegEx))

The syntax that makes up the Lookaround is no longer correct. The Capture Group must be the inner set of parentheses not the outer set.

Lookarounds: *Lookahead and Lookbehind*

Lookahead: Positive

String = Graham the aB ab abc abC ABC Miller.jpg


Match: (?=abc)

match only if string contains abc without capturing. Lookahead to find 'abc'

Replace: Capture Group 1= \1


The Match is **true** because of the string 'abc' found in the Input String. If it returns nothing, how do you know it matched?

In BRU, under the New Name column it displays:

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	\1.jpg

This would indicate a problem with the Match expression or the Replace data, but because there are no Capture Groups defined, by turning **red**, one conclusion is that it indicates that a match is found but cannot display any data. The Replace String backreference is evaluated as a string literal, so both the backslash and the numeral '1' appear in New Name. The backslash character is illegal in a Windows filename, and as a result, BRU flags it as **Invalid** in **RED**. When something is flagged as **Invalid**, it means that the rename operation will not take place.

If instead we place the Lookaround expression within a Capture Group, (?=(abc)) now \1 = abc

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Capture Group 1= abc.jpg

Lookarounds: *Lookahead and Lookbehind*

Lookahead: Positive cont.

Examples of not capturing using a Lookahead

String = Graham the aB ab abc abC ABC Miller.jpg


Match: (?=the)(.*)((?=Mil)(.*))
 Replace: Capture Group 1= \1 Second Capture Group = \2
 Where: \1 = <space> the aB ab abc abC ABC <space>
 \2 = Miller

Analysis:

1. (?=the) Lookahead for the string 'the'
(Not Captured)
= true (located after
'Graham <space>')
2. .* Capture string
3. (?=Mil) Lookahead for string, 'Mil'
(Not Captured)
= true. Already at EOL so
it backtracks to match.

New Name
 Capture Group 1= the aB ab abc abC ABC Second Capture Group = Miller.jpg

Capture Group 1 = <space> aB ab abc abC ABC Miller

In Step 3, the backtracking is done testing each and every character moving back then forward and repeats this until it matches against 'Mil' or reaches the beginning of the filename (BOL). This can be clearly seen here in step 33 of the Debug Log: 

Backtracks to match against 'M', then moves forward to match 'Mil'. So although a Lookahead looks forward to the right, backtracking can still take place in which case the movement is more like -

One step back then move forward testing as it goes. Repeat.

Hopscotch perhaps? Not sure, not my game...

4. .* Capture remainder of string, Capture Group 2 = Miller
moving forward again. Changes Capture Group 1 =
<space>
aB ab abc abC ABC
<space>

```

Beginning match attempt at character 7
1 t
2 th
3 the
4 ok
5 the aB ab abc abC ABC Miller
6 the aB ab abc abC ABC Miller
7 the aB ab abc abC ABC Miller backtrack
8 the aB ab abc abC ABC Miller backtrack
9 the aB ab abc abC ABC Mille
10 the aB ab abc abC ABC Mille
11 the aB ab abc abC ABC Mille backtrack
12 the aB ab abc abC ABC Mille backtrack
13 the aB ab abc abC ABC Mill
14 the aB ab abc abC ABC Mill
15 the aB ab abc abC ABC Mill backtrack
16 the aB ab abc abC ABC Mill backtrack
17 the aB ab abc abC ABC Mil
18 the aB ab abc abC ABC Mil
19 the aB ab abc abC ABC Mil backtrack
20 the aB ab abc abC ABC Mil backtrack
21 the aB ab abc abC ABC Mi
22 the aB ab abc abC ABC Mi
23 the aB ab abc abC ABC Mi backtrack
24 the aB ab abc abC ABC Mi backtrack
25 the aB ab abc abC ABC M
26 the aB ab abc abC ABC M
27 the aB ab abc abC ABC M backtrack
28 the aB ab abc abC ABC M backtrack
29 the aB ab abc abC ABC
30 the aB ab abc abC ABC
31 the aB ab abc abC ABC M
32 the aB ab abc abC ABC Mi
33 the aB ab abc abC ABC Mil
34 the aB ab abc abC ABC ok
Match found in 34 steps
    
```

Lookarounds: *Lookahead and Lookbehind*

Lookahead: Positive cont.

String = Graham the aB ab abc abC ABC Miller.jpg

Match: `(?=the)(.*)(?=abC)(.*)`

Where :
\1 = the aB ab abc<space>
\2 = abC ABC Miller

New Name
Capture Group 1 = the aB ab abc Second Capture Group = abC ABC Miller.jpg

The analysis would be similar to the first example so I urge you to study that analysis and if you need additional help, please refer to Volume II.


Notes:

1. In Regex Buddy Debug Logs, Lookarounds are not documented except for when they match, then the log will just indicate a successful match by the singular label, 'OK'.

Lookarounds: *Lookahead and Lookbehind*

Using Capture groups within Lookaheads

String = Graham 456x56 Miller.jpg

Match:	<code>(?=(\d+)\w+\1</code>	Name ▲	New Name
Replace:	Capture Group 1= \1	 Graham 456x56 Miller.jpg	Capture Group 1= 56.jpg
Where:	\1 = 56		

To fully understand what is happening, I am going to change the expression so that each sub-expression is captured, then I am going to show you the difference in the evaluation when a Lookahead is used in the expression and an example where it is not.

This first example uses a Lookahead to look to the right for a numeric digit. If it matches, this value is captured. Typically, a Lookahead does not capture values but because I have placed a Capture Group in the Lookahead, the value found, if any will be held by the `(\d+)`. This does not change the behaviour of the Lookaround. It is still an assert and will not change the current position if a match is found. This is important and the differences will be explained.

Analysis: Match: `(?=(\d+))(\w+)(\1)`

Where:

\1 = 56 \2 = 56x \3 = 56

```
.....
Match 1: 56x56
Group 1: 56
Group 2: 56x
Group 3: 56
```

I have changed the original expression and have created three Capture Groups rather than having one Capture Group.

Capture Group 1 Capture Group 2 Capture Group 3

`(?=(\d+))` `(\w+)` `(\1)`

1. `?=(\d+)`

Lookahead.
Looks to the right to match for a numeric digit.
Made Greedy with the + Quantifier, captures any consecutive numeric digits that follow.
Captures the '5' followed by the '6' until it encounters the 'x'.

Capture Group 1 = 56

How does it come up with the value, '56', I would have thought it would be '456'? Full explanation to follow in the commentary section. The '56' characters captured are in the first occurrence found in the string.

Lookarounds: *Lookahead and Lookbehind*String = `Graham 456x56 Miller.jpg`**Analysis:** Match: `(?=(\d+))(\w+)(\1) cont.`

2. `\w` Word Metacharacter. Capture Group 2 = 5
 Move forward to capture the '5' in the substring, '56x'.

The Lookaround does not change the position, According to step 1, the current position is the '5' of the substring, '56x', in the first occurrence found in the string (the explanation of what happened to the '4' of '456x' will be forthcoming. Be patient.

3. `+` Make it Greedy. Capture Group 2 = 56x
 Continues to match until it encounters the <space> preceding 'Miller'.

Stranger still, I would have thought even following along if the current position is 'x', the value of Capture Group 2 would have been 'x56' not '56x'. Regardless, it should be noted that the '56' captured is in the second occurrence found in the string.

4. `\1` Backreference to the value held in Capture Group 1. Capture Group 3 = 56
 Backtracks to match against the '56' in the substring, 'x56'.

This is where it gets interesting. A Backreference can be included in a RegEx. The reason to use a backreference in this RegEx is to match against the same string characters of Capture Group 1 – in other words, search for repeated substrings of consecutive characters of text.

In this string, you have two occurrences of '56'. Capture Group 1 originally captures the first occurrence and Capture Group 3 captures the second occurrence based on the Backreference to Capture Group 1.

'456', 'x56', 56x' ??

Now that you are utterly confused, here is the explanation.

Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

Analysis: Match: `(?=(\d+))(\w+)(\1)`

Where:

\1 = 56 \2 = 56x \3 = 56

```
.....
Match 1: 56x56
Group 1: 56
Group 2: 56x
Group 3: 56
```

G r a h a m 4 5 6 x 5 6 Miller
 0 1 2 3 4 5 6 7 8 9

Group 1
 5 6
 8 9

After the failure of the 7th character position, the RegEx Engine moves forward one position and begins the testing at character position 8, the '5' of the first occurrence of '56' in the substring, '456x'.

At character position 8 it finds a match '5' for the `\d` and matches the `+` to '56' before encountering the 'x'. Capture Group 1 = '56'. There is the explanation for how Capture Group 1's value is '56' and not '456'

G r a h a m 4 5 6 x 5 6 Miller
 0 1 2 3 4 5 6 7 8 9 10 11 12

Group 2
 5 6 x 5 6
 8 9 10 11 12

Group 1
 5 6
 8 9

<space> Miller

The next part of the RegEx, the `\w+` captures the '56x56'. Remember that the Lookahead has not changed the position with the match of the '56' for Capture Group 1. The current position is the character before the match, and this would be still at position 6, the `<space>` after 'Graham' so when the `\w` is evaluated, it matches the '5' and the `+` matches to the end of the Word at the `<space>` preceding, 'Miller'. Thereby Capture Group 2 = '56x56' that includes both occurrences of '56' found in the string. Because Capture Group 1's value was the result of a Lookaround, the value in Capture Group 1 is unaffected by the inclusion of the first occurrence of '56' in Capture Group 2. Under normal circumstances, if this had not been the result of a Lookaround, Capture Group 1 would have been forced to give up the '56' match and return a **null** value as a result of a Zero Occurrence Match. This is because the '56' value held previously by Capture Group 1 is now held by Capture Group 2.

The RegEx Engine continues to look for other matches and finding none, hits the EOL.

The next part is the Backreference, `\1`, that holds the key to the value of Capture Group 1. The current value of Capture Group 1 is '56'. The Regex Engine current position is at EOL, so it backtracks to try and match against the first character, '5', as far as it can.

Now this is important. It cannot attempt to match against itself, meaning, Capture Group 1's value is the first occurrence of '56' of the substring, '456'. Therefore, the testing can only go back as far as the 'x' that follows '456'.

Lookarounds: *Lookahead and Lookbehind*

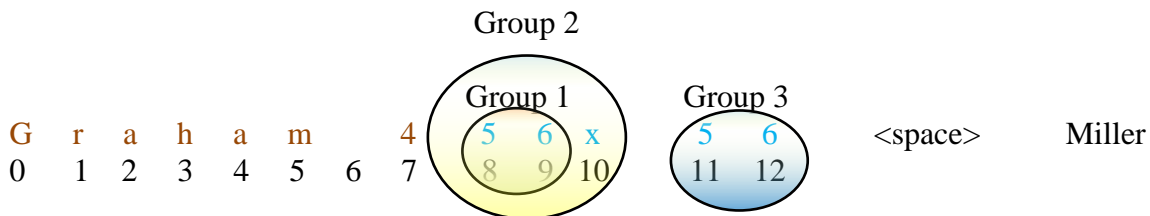
String = Graham 456x56 Miller.jpg

Analysis: Match: `(?=(d+))(\w+)(\1)`

Where:

\1 = 56 \2 = 56x \3 = 56

```
.....
Match 1: 56x56
Group 1: 56
Group 2: 56x
Group 3: 56
```



The characters, '<space> Miller' (backtracking) do not match. However, a match is found at the '5', followed by the '6' in the substring leading up to the 'x' in the second occurrence found in the string. Capture Group 3 = '56'. Capture Group 2 previously held this value as part of its capture and is forced to give up the second occurrence of '56' in its value of '56x56' leaving Capture Group 2 with a value of '56x', that includes only the first occurrence of '56' found in the string. There is your explanation for why Capture Group 2's final value is '56x' and not 'x56'.

Still a bit bewildered?

Here is a visual representation of the successful match at character position 8:

Steps 1 – 3 of the Analysis:



Step 4 of the Analysis:

Capture Group 3 already at EOL backtracks to capture the '56' value.



Furthermore, the Debug log from Regex Buddy on the following page will provide more information.

Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

Analysis: Match: (?=(d+))(\w+)(\1)

Where:

\1 = 56 \2 = 56x \3 = 56

```
.....
Match 1: 56x56
Group 1: 56
Group 2: 56x
Group 3: 56
```

The Debug Log for the Match attempt at character 7:

```
Beginning match attempt at character 7
1 456
2 456
3 ok
4 456x56
5 456x56
6 456x56 backtrack
7 456x56 backtrack
8 456x5
9 456x5
10 456x5 backtrack
11 456x5 backtrack
12 456x
13 456x
14 456x backtrack
15 456x backtrack
16 456
17 456
18 456 backtrack
19 456 backtrack
20 45
21 45
22 45 backtrack
23 45 backtrack
24 4
25 4
26 4 backtrack
27 4 backtrack
28 backtrack
Match attempt failed after 28 steps
```

The Debug Log for the Match attempt at character 8:

```
Beginning match attempt at character 8
1 56
2 56
3 ok
4 56x56
5 56x56
6 56x56 backtrack
7 56x56 backtrack
8 56x5
9 56x5
10 56x5 backtrack
11 56x5 backtrack
12 56x
13 56x
14 56x56
15 56x56
Match found in 15 steps
```

Notes:

1. In Regex Buddy Debug Logs, Lookarounds are not documented except for when they match, then the log will just indicate a successful match by the singular label, 'OK'.


Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

Next, as promised, I will show you a similar RegEx that will not use the Lookahead.

Match: `(\d+)(\w+)(\1)`

Replace: Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3

Name	New Name
 Graham 456x56 Miller	Capture Group 1 = 56 Capture Group 2 = x Capture Group 3 = 56

Analysis:

Where:

\1 = 56 \2 = x \3 = 56

```

.....
Match 1: 56x56
Group 1: 56
Group 2: x
Group 3: 56

```

I have changed the original expression and have created three Capture Groups rather than having one Capture Group;

Capture Group 1 Capture Group 2 Capture Group 3

`(\d+)` `(\w+)` `(\1)`

1. `\d` Match against Numeric Digit. Capture Group 1 = 5
 Matches against the '5' of the first occurrence
 in the substring, '456'.

Why not the '4'? Same explanation as before but I will be discussing it again.

2. `+` Make it Greedy. Capture Group 1 = 56
 Matches against any additional consecutive
 numeric digits that follow.
 Captures the '6' until it encounters the 'x'.

2. `\w` Word Metacharacter. Capture Group 2 = x
 Move forward to capture the 'x' of the second
 occurrence in the substring, 'x56'.

3. `+` Make it Greedy. Capture Group 2 = x56
 Matches against any further subsequent characters
 until a non-Word Character is encountered, the `<space>`
 preceding 'Miller'. Captures the '56' in the second
 occurrence in the substring, 'x56'. Continues to EOL.

4. `\1` Backreference to Capture Group 1. Capture Group 3 = 56
 Backtracks to match against the '56' in the second
 Occurrence in the substring, 'x56'. Capture Group 2
 is Forced to give up the characters '56' from its match. [Changes Capture Group 2 = x](#)

Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

Match: (\d+)(\w+)(\1)

Replace: Capture Group 1= \1 Capture Group 2 = \2 Capture Group 3 = \3

Name	New Name
<input type="checkbox"/> Graham 456x56 Miller	Capture Group 1= 56 Capture Group 2 = x Capture Group 3 = 56

Analysis:

Where:

\1 = 56 \2 = x \3 = 56

Match 1:	56x56
Group 1:	56
Group 2:	x
Group 3:	56

There are two occurrences of '56' in the string:

Graham 456x56 Miller.jpg

G r a h a m 4 5 6 x 5 6 Miller

0 1 2 3 4 5 6 7 8 9 10 11 12

Group 1: (5 6)

After the failure of the 7th character position, the RegEx Engine moves forward one position and begins the testing at character position 8, the '5' of the first occurrence of '56' in the substring, '456x'.

At character position 8 it finds a match '5' for the \d and matches the + to '56' before encountering the 'x'. Capture Group 1 = '56'. This time there is no Lookaround. The current position of the RegEx is at the last position after the match. Current position is the 'x' of the substring, 'x56'.

G r a h a m 4 5 6 x 5 6 Miller

0 1 2 3 4 5 6 7 8 9 10 11 12

Group 1: (5 6)

Group 2: (x 5 6)

The \w+ captures the 'x56' to the end of the Word at the <space> preceding, 'Miller' in Capture Group 2. The RegEx Engine continues to look for other matches and finding none, hits the EOL.

The Backreference, \1, holds the value of Capture Group 1, '56'. The RegEx Engine current position is at EOL, so it backtracks to try and match against the first character, '5', as far as it can. Now this is important. It can't match against itself, meaning, Capture Group 1's value is the first occurrence of '56' of the substring, '456'. Therefore, the testing can only go back as far as the 'x' that follows '456'.

Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

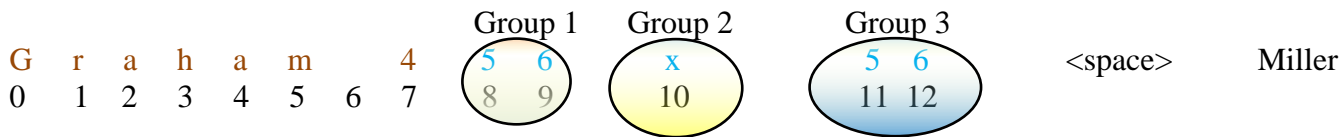
Analysis: Match: `(\d+)(\w+)(\1)`

Where:

\1 = 56 \2 = x \3 = 56

```

.....
Match 1: 56x56
Group 1: 56
Group 2: x
Group 3: 56
    
```



The characters, '<space> Miller' (backtracking) do not match. However, it matches the '5' followed by the '6' in the second occurrence of '56' of the substring, 'x56'. Capture Group 3 = '56'. Capture Group 2 previously held this value as part of its capture and is forced to give up the second occurrence of '56' in its value of 'x56' leaving Capture Group 2 with a value of 'x'.

Here is a visual representation of the successful match at character position 8:

Steps 1 – 3 of the Analysis:



Step 4 of the Analysis:

For Capture Group 3, already at EOL, so it backtracks to capture the '56' value.



Furthermore, the Debug log from Regex Buddy on the following page will provide more information.

Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

Analysis: Match: (\d+)(\w+)(\1)

Where:

\1 = 56 \2 = 56x \3 = 56

```

.....
Match 1: 56x56
Group 1: 56
Group 2: x
Group 3: 56

```

The Debug Log for the Match attempt at character 7:

```

Beginning match attempt at character 7
1 456
2 456
3 456x56
4 456x56
5 456x56backtrack
6 456x56backtrack
7 456x5
8 456x5
9 456x5backtrack
10 456x5backtrack
11 456x
12 456x
13 456xbacktrack
14 456xbacktrack
15 456backtrack
16 45
17 45
18 456x56
19 456x56
20 456x56backtrack
21 456x56backtrack
22 456x5
23 456x5
24 456x5backtrack
25 456x5backtrack
26 456x
27 456x
28 456xbacktrack
29 456xbacktrack
30 456
31 456
32 456backtrack
33 456backtrack
34 45backtrack

```

```

35 4
36 4
37 456x56
38 456x56
39 456x56backtrack
40 456x56backtrack
41 456x5
42 456x5
43 456x5backtrack
44 456x5backtrack
45 456x
46 456x
47 456xbacktrack
48 456xbacktrack
49 456
50 456
51 456backtrack
52 456backtrack
53 45
54 45
55 45backtrack
56 45backtrack
57 4backtrack
58 backtrack
Match attempt failed after 58 steps

```

Lookarounds: *Lookahead and Lookbehind*

String = Graham 456x56 Miller.jpg

Analysis: Match: `(\d+)(\w+)(\1)`

Where:

`\1 = 56` `\2 = 56x` `\3 = 56`

```

.....
Match 1: 56x56
Group 1: 56
Group 2: x
Group 3: 56

```

The Debug Log for the Match attempt at character 8:

```

☐ Beginning match attempt at character 8
1 | 56
2 | 56
3 | 56x56
4 | 56x56
5 | 56x56backtrack
6 | 56x56backtrack
7 | 56x5
8 | 56x5
9 | 56x5backtrack
10| 56x5backtrack
11| 56x
12| 56x
13| 56x56
14| 56x56
   | Match found in 14 steps

```

Lookarounds: *Lookahead and Lookbehind*

Lookahead: Negative

So what about the Negative Lookahead? Not too much different in terms of the examples presented.

A Negative Lookahead matches when something is not located.

String = Graham the aB ab abc abC ABC Miller.jpg

Match: `(?!abc)` matches only if **no** characters in the string are followed by the string 'abc'.
= **false** because 'ab' is followed by 'abc'


String = Graham the aB ab abd abC ABC Miller.jpg

Match: `(?!abc)` matches only if **no** characters in the string are followed by the string 'abc'.
= **true**. There is no substring, 'abc' in the string.

String = Graham the aB ab abc abC ABC Miller.jpg

Match: `(?=the)(.*)((?!Kil)(.))`
Replace: Capture Group 1= \1 Capture Group 2 = \2

Where: \1 = the aB ab abc abC ABC Miller
\2 = "" (**null** or empty)

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Capture Group 1= the aB ab abc abC ABC Miller Capture Group 2 = .jpg

Analysis:

`(?=the)` = **true**. Finds string 'the' after 'Graham <space> '. The string is not captured unless you used `(?=the)`.

`(.*)` Greedy. Gather up the rest of the string = the aB ab abc abC ABC Miller and Capture it in Capture Group 1. Current position = EOL.

Why did it include 'the'? Because a non capturing group, e.g., `(?=the)`, does not consume characters. Consume would advance the position in the string. The current position before the match is <space> after 'Graham', therefore the `.*` starts the capture at 'the'.

`(?!Kil)` Lookahead test for *not finding* string 'Kil' = **true**. Non capturing group. Even if captured, value would be **null**.

`(.*)` = **null**. Already at EOL. Capture Group 2 = **null**.

Lookarounds: *Lookahead and Lookbehind*

Up until now the Lookahead has been basically a simple search string. But what it really can do is to match something that is followed (looks to the right) or not followed (looks to the left) using a Negative Lookahead, by something else.


String = Star Wars Episode USD100 123x12 aNbc.jpg

Match: (d(?!f))

Replace: Capture Group 1= \1

Where: \1 = d First it must find the small letter 'd' of the string 'Episode' to continue the expression = true.

Next, Lookahead test for not finding character 'f' = true. Therefore Regex = Match and the 'd' is captured.

Name ▲	New Name
 Star Wars Episode USD100 123x12 aNbc.jpg	Capture Group 1= d.jpg

Notes:

1. Capture Group 1 captures the 'd' outside of the Lookaround because the entire expression is placed in a Capture Group.
2. If the expression had been, (d(?!e)), although 'd' would still be found, the next part of the expression, testing for not finding the character 'e' = false since, 'e' follows 'd' in the string 'Episode'. Therefore the Regex would fail, and 'd' would not be captured.
3. This Lookahead statement basically means to find 'd', then look to the immediate right for 'f' or in the case of a negative, not 'f'.

Heres' a mixed version:

String = Graham the aB ab abc abC ABC Miller.jpg

Match: (Gra(?!the)).*(abc (?=abC)).*(Mil)

Replace: Capture Group 1= \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4
Capture Group 5 = \5

Where: \1 = Gra
\2 = ham the aB ab<space>
\3 = abc<space>
\4 = abC ABC<space>
\5 = Mil

New Name

Capture Group 1= Gra Capture Group 2 = ham the aB ab Capture Group 3 = abc Capture Group 4 = abC ABC Capture Group 5 = Mil.jpg

Lookarounds: *Lookahead and Lookbehind*String = `Graham the aB ab abc abC ABC Miller.jpg`Match: `(Gra(?!the))(.*)(abc (?=abC))(.*)(Mil)`**Analysis:**

- | | | |
|-------------------------------|---|---|
| 1. <code>(Gra(?!the))</code> | Negative Lookahead Searches to find 'Gra' NOT immediately followed by 'the' (looks to the right) = true because 'ham' follows 'Gra'. | Capture Group 1 = Gra |
| | Current position = 'h' of 'Graham' | |
| 2. <code>.*</code> | Capture the remaining string | Capture Group 2 =
ham the aB ab abc abC ABC Miller |
| 3. <code>(abc (?=abC))</code> | Positive Lookahead searches to find 'abc<space>' immediately followed by 'abC'. (looks to the right) | Capture Group 3 = abc <space>
Changes Capture Group 2 =
ham the aB ab <space> |
| | Already at <u>EOL</u> so it backtracks to match against 'abc <space>' followed by 'abC'.
Current position = 'a' of 'abC' | |
| 4. <code>.*</code> | Capture remainder of string | Capture Group 4 = abC ABC Miller |
| 5. <code>Mil</code> | Match against string, 'Mil' | Capture Group 5 = Mil
Changes Capture Group 4 = abC ABC <space> |

Lookarounds: *Lookahead and Lookbehind*

We interrupt this broadcast for an important message.

Understanding the current position in Lookaheads.

It can get confusing, but if you are to properly analyze the RegEx, it is important to know where the RegEx Engine in regards to the string so you can correctly derive the value from the evaluation of the next sub-expression.

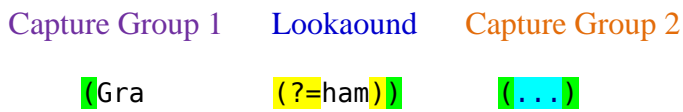
String = `Graham the aB ab abc abC ABC Miller.jpg`

Match: `(Gra(?=ham))(...)`

Here is a positive Lookahead. It searches for 'Gra' followed immediately by 'ham' = `true`. This value is captured in Capture Group 1. The current position as you can see in the Debug output (Regex Buddy program), is the 'h' of 'ham'

Thus, when the next sub-expression, `(...)` takes place, it will start with the 'h' and the value will be 'ham' for Capture Group 2. Remember that the match of a Lookaround is an assert and does not change the position after a match. Therefore, the current position remains at the 'h' of 'ham' prior to the match.

This can be best represented visually as:



This shows the resulting matches:

```
.....
Match 1: Graham
Group 1: Gra
Group 2: ham
```

The match can be illustrated in the string as `Graham` the aB ab abc abC ABC Miller

The Debug output demonstrates how the matches were obtained:

```

Beginning match attempt at character 0
1 G
2 Gr
3 Gra
4 Grah
5 Graha
6 Graham
7 Graok
8 Gra
9 Grah
10 Graha
11 Graham
12 Graham
Match found in 12 steps

```

Lookarounds: *Lookahead and Lookbehind*

Understanding the current position in Lookaheads cont.

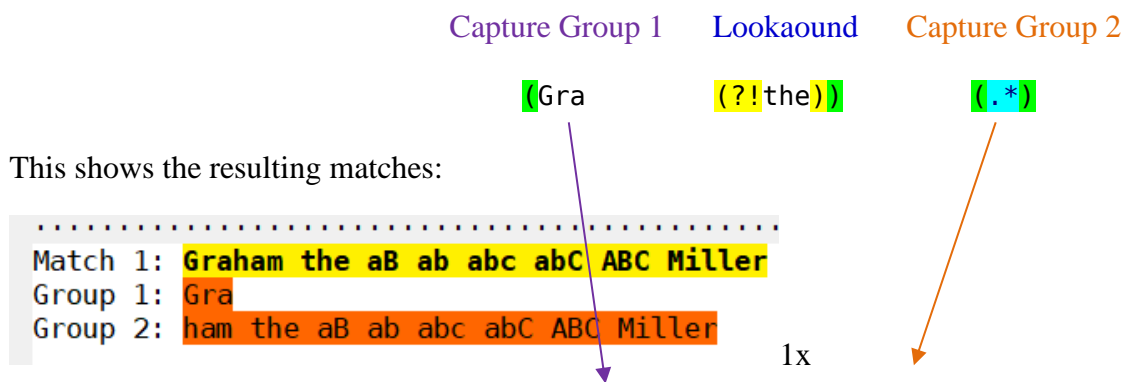
String = Graham the aB ab abc abC ABC Miller.jpg

Match: (Gra(?!the)).*

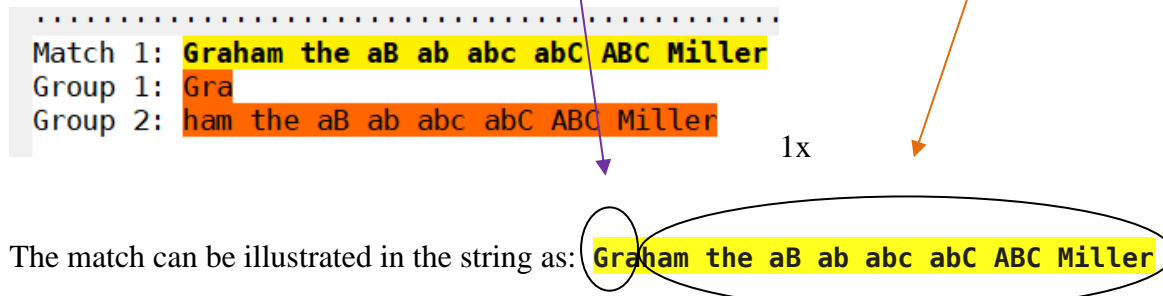
Here is a negative Lookahead. It searches for 'Gra' followed not immediately by 'the' = true (the 'ok' in the Debug Log in Step 5 below is the indicator of where the Lookahead matched). This value is captured in Capture Group 1. The current position as you can see in the Debug output (Regex Buddy program), is the 'h' of 'ham'.

Thus, when the next sub-expression, (.*) takes place, it will start with the 'h' and the value will be 'ham the aB ab abc abC ABC Miller' for Capture Group 2.

This can be best represented visually as:

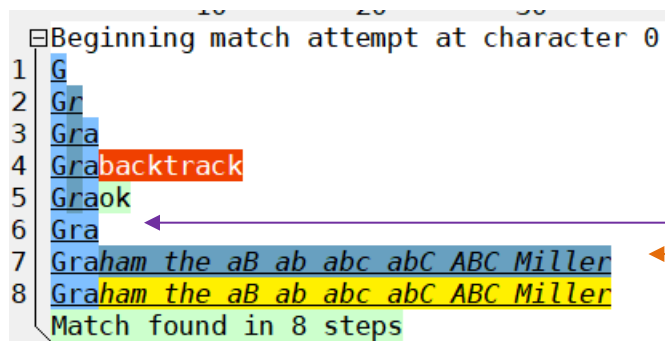


This shows the resulting matches:



The match can be illustrated in the string as: **Graham the aB ab abc abC ABC Miller**

The Debug output demonstrates how the matches were obtained:



In both instances, the Lookaround did not affect the current position.

Typically, a match made using an expression would move the position to the point after the match – the 'm', but the position remains at the point before the Lookahead. Previous to this, a match was made of 'Gra', and so the position remains at the 'h' of 'ham'. With a negative Lookahead there is no difference. The position remains at the 'h' of 'ham' because of the last match of 'Gra'. If there was not a previous match, e.g., BOL, then the value would be **null**.

Lookarounds: *Lookahead and Lookbehind*

Understanding the current position in Lookaheads cont.

This brings us to:

```
(abc(?:=abC))
```

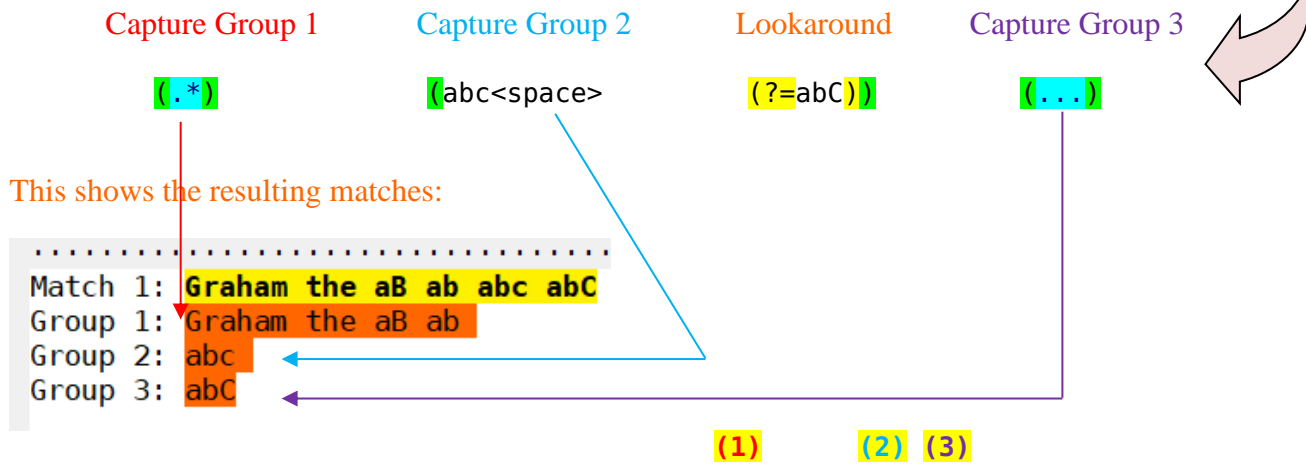
String = Graham the aB ab abc abC ABC Miller.jpg

Match: (Gra(?:!the))(.*)(abc (?:=abC))(.*)(Mil)

In step 3, the current position was at the end of the string, or EOL, therefore it backtracked to match against the 'abc<space>' and looked to the right for 'abC' = true.

The current position remains at the point before the Lookaround after the last match. The last match was 'abc<space>' so the current position is the 'a' of 'abC'.

Let's simplify this a little more because our interest is mainly in the second Lookaround:



The match can be illustrated in the string as: **Graham the aB ab abc abC** ABC Miller

The partial Debug output demonstrates how the matches were obtained:

The diagram shows a debug output of match attempts for the string "Graham the aB ab abc abC ABC Miller". It highlights the backtracking process for the Lookaround (?:=abC) and the final match found in 89 steps. The output is as follows:

Beginning match attempt at character 0	73	Graham the aB ab abc abC ABC Miller	backtrack
1	74	Graham the aB ab abc abC ABC Miller	backtrack
2	75	Graham the aB ab abc abC ABC Miller	backtrack
3	76	Graham the aB ab abc abC ABC Miller	backtrack
4	77	Graham the aB ab abc abC ABC Miller	backtrack
5	78	Graham the aB ab abc abC ABC Miller	backtrack
6	79	Graham the aB ab abc abC ABC Miller	backtrack
7	80	Graham the aB ab abc abC ABC Miller	backtrack
8	81	Graham the aB ab abc abC ABC Miller	backtrack
9	82	Graham the aB ab abc abC ABC Miller	backtrack
10	83	Graham the aB ab abc abC ABC Miller	backtrack
11	84	Graham the aB ab abc abC ABC Miller	backtrack
12	85	Graham the aB ab abc abC ABC Miller	backtrack
	86	Graham the aB ab abc abC ABC Miller	backtrack
	87	Graham the aB ab abc abC ABC Miller	backtrack
	88	Graham the aB ab abc abC ABC Miller	backtrack
	89	Graham the aB ab abc abC ABC Miller	Match found in 89 steps

Annotations include: (1) abc <space>, (2) (?:=abC), and (3) (...). A large arrow points from the Lookaround label to the corresponding row in the debug output.

Lookarounds: *Lookahead and Lookbehind*

Lookbehind asserts

This works the same way as a Lookahead only it looks backs (looks to the left) through the string to find a match.

There are two forms:

positive Lookbehind asserts

negative Lookbehind asserts

syntax `(?<=n)`

`(?<!n)`


where n is what you want to match without capturing

Positive Lookbehind: match something that *is preceded* by something else. (looks to the left)

Negative Lookbehind: match something *not preceded* by something else. (looks to the left)

String = Graham the aB ab abc abC ABC Miller.jpg

Match: `(?<=aB)(.*)(abC)`
 Replace: Capture Group 1= \1 Capture Group 2 = \2
 Where: \1 = <space>ab abc<space>
 \2 = abC

Name ▲	New Name
 Graham the aB ab abc abC ABC Miller.jpg	Capture Group 1= ab abc Capture Group 2 = abC.jpg

Analysis:

1. `(?<=aB)` Lookbehind moves back and forth through the string until it matches 'aB'. Current position is the <space> of '<space>aB'. Position doesn't change from the start of the match and because of the Lookbehind backtracking, this was two characters forward from the match.

It does this by moving one step forward, two steps back to test (because there are two characters to test for – 'a' and 'B'). Because each time the position before the testing of the previous characters was *ahead* of the eventual match of 'aB', the position before the match was the <space> after 'aB' not the <space> after 'the' which it would have been if this had been a Lookahead.

Lookarounds: *Lookahead and Lookbehind*

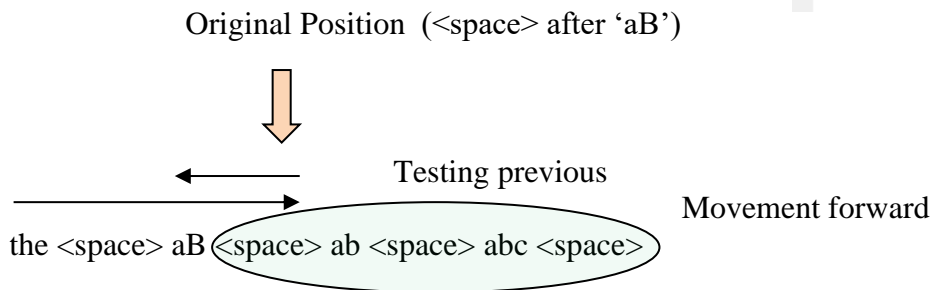
Lookbehind asserts cont.

Analysis cont.:

With a Lookbehind

```
(?>aB)(.*) (abC)
```

Illustrated:

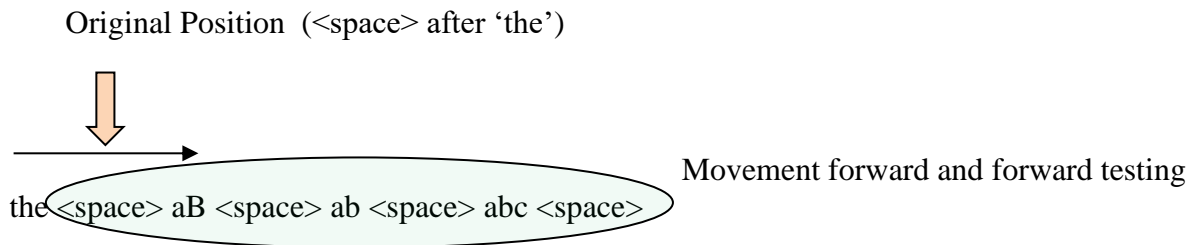


```
.....
Match 1: aB ab abc abC
Group 1: ab abc
Group 2: abC
```

Changing it to a Lookahead...

```
(?=aB)(.*) (abC)
```

Illustrated:



```
.....
Match 1: aB ab abc abC
Group 1: aB ab abc
Group 2: abC
```

- | | | |
|--------|---|--|
| 2. .* | Capture rest of string | Capture Group 1 =
<space> ab abc abC ABC Miller |
| 3. abC | Match against string, 'abC'. Already at <u>EOL</u>
Backtracks. | Capture Group 2 = abC
Changes Capture Group 1 =
<space> ab abc <space> |

Notes:

1. A Lookbehind cannot use expressions. This is because the engine cannot apply a Regular Expression moving backwards. They must be strings of fixed length. They can use single metacharacters but not Quantifiers including '+' or '*'

```
(?<=(\d))(...) - valid      (?<=(\d{3}))(...) - valid      (?<=(\d+))(...) - Invalid
```

Lookarounds: *Lookahead and Lookbehind*

Lookbehind asserts cont.

With this **Lookahead** it can match something that is followed by something else. This is referred to as ‘After the Match’ where an object to be matched comes before the Lookaround seeking the object to be found or not found. Both parts of the expression have to be **true** for the Regex to Match.

For example,

```
(dog(?=bone))
```

Match against ‘dog’ in the string and determine, using the Lookahead, if it is immediately followed by ‘bone’ (look to the right), or in the case of a negative Lookahead, (dog(?!bone)), not immediately followed by ‘bone’.

With this **Lookbehind** it can match something that precedes something else. This is referred to as ‘Before the Match’ where the Lookaround seeks the object to be found or not found prior to an object to be matched. Both parts of the expression have to be **true** for the Regex to Match.

For example,

```
(?<!a)b
```

This will determine that the preceding character before ‘b’ is not an ‘a’ (Looks to the left), and if **true**, will match ‘b’.

Cab would not match because ‘a’ does precede ‘b’.
 Bed would match because ‘a’ does not precede ‘b’
 Debt would match because ‘a’ does not precede ‘b’

Taking the same example and making it a positive Lookbehind:

```
(?<=a)b
```

Cab would match because ‘a’ does precede ‘b’.
 Bed wouldn’t match because ‘a’ does not precede ‘b’
 Debt wouldn’t match because ‘a’ does not precede ‘b’

Notes:

1. You can use a Lookbehind anywhere in the RegEx, not only at the start.
2. A Lookbehind must match a fixed set of characters. You couldn’t have a variable Quantifier used with a Lookbehind, e.g. `\d+` or `\d*` or even `\d?`, but you could use a Range Quantifier, e.g., `\d{3}` that limits the match to a specific number of iterations.
3. ‘Before the Match’ and ‘After the Match’ can be applied to any of the 4 Lookaround types – Lookahead, Positive and Negative and Lookbehind, Positive and Negative.

Lookarounds: *Lookahead and Lookbehind***Lookbehind asserts cont.**

Example:

If you want to find a word not ending with an "s":

Match: `(\b\w+(?!s)\b)`

Examples:

String = Graham the aB ab abc abC ABC Miller.jpg

\1 = Graham

String = Star Wars Episode USD100 123x12 aNbc.jpg

\1 = Star

String = This is a (34) test.jpg

\1 = a

String = Belvedere Plantation 6492.jpg

\1 = BelvedereStar

Analysis:

Using: `This is a (34) test.jpg`

1. `\b` Word Boundary Metacharacter. Establishes the beginning of a Word.
2. `\w` Word Metacharacter. Match against Word Character. Capture Group 1 = T
3. `+` Make it Greedy. This captures all word Characters until it hits a non-Word Character. The non-Word Character encountered is the `<space>` after 'This' Capture Group 1 = This
4. `(?!s)` Move forward through the string, one step forward, one step back (only one character to test for – the 's') Matches against the 's' of 'This'. By matching, it changes the Capture Group 1 consisting of 'This' by the removal of the 's' character because currently it looks for Word Characters that do not have an 's' and 'Thi' is the result. Changes Capture Group 1 = Thi
5. `\b` Word Boundary Metacharacter. Establishes an end of a word. This changes the Lookbehind to search out whole words that do not have an 's' at the end. A word is defined as Word Characters until a non-Word Character is encountered. Changes Capture Group 1 = a

tests	<code>This</code>	=	false
	<code>is</code>	=	false
	<code>a</code>	=	true

Lookarounds: *Lookahead and Lookbehind*

Lookbehind asserts cont.

Here you can see the transition from step 4 to step 5:

Step 4 (`(\b\w+(?!s)`)

```

Beginning match attempt at character 0
1 ok
2 This
3 This
4 This backtrack
5 Thi
6 Th backtrack
7 Thok
8 Thi
Match found in 8 steps

```

Step 5 (`(\b\w+(?!s)\b)`)

Tests 'This'

```

Beginning match attempt at character 0
1 ok
2 This
3 This
4 This backtrack
5 Thi
6 Th backtrack
7 Thok
8 Th backtrack
9 Th
10 T backtrack
11 Tok
12 T backtrack
13 T
14 backtrack
15 ok
16 backtrack
17 backtrack
Match attempt failed after 17 steps

```

Tests 'is'

```

Beginning match attempt at character 5
1 ok
2 is
3 is
4 is backtrack
5 i
6 backtrack
7 ok
8 backtrack
9 backtrack
Match attempt failed after 9 steps

```

Tests 'a'

```

Beginning match attempt at character 8
1 ok
2 a
3 backtrack
4 ok
5 ok
6 a
Match found in 6 steps

```

Lookarounds: *Lookahead and Lookbehind*

Using Lookarounds Before and After the Match

There are four kinds of Lookarounds:

Positive Lookahead (?= ()

Negative Lookahead (?! ()

Positive Lookbehind (?<= ()

Negative Lookbehind (?<! ()

Each of these Lookarounds can be used in two main ways:

Before the expression to be matched

After the expression to be matched

This is an example of a Lookbehind ‘After the Match’.

String = Cotton USD100 JPY100 Plantation 6490.jpg

Match: (.{3})(?<=USD\d{3}))

Where: \1 = 100

Break down:

Capture Group 1

Lookbehind

(.{3} (?<=USD\d{3}))

The three Dot Metacharacters continually test and match against each three characters of the string, but fails to match against the Lookbehind of the string, ‘USD’, until it matches against the ‘100’ in the string. This value is captured in Capture Group 1. After it matches, it evaluates the Lookbehind to look to the immediate left for the ‘USD’ followed by 3 numeric digits. This matches and the RegEx is satisfied. It will continue to search out other matches until EOL or the string is exhausted whichever comes first.

Cotton USD**100** JPY100 Plantation 6490

Lookarounds: *Lookahead and Lookbehind*

Using Lookarounds Before and After the Match cont.

This is an example of a Lookbehind 'After the Match'. cont.

String = Cotton USD100 JPY100 Plantation 6490.jpg

Match: `(.{3})(?<=USD\d{3})`

Where: `\1 = 100`

Here is the Debug Log from RegEx Buddy program:

```

Beginning match attempt at character 0
1 Cot
2 Cotbacktrack
3 backtrack
Match attempt failed after 3 steps
Beginning match attempt at character 1
1 ott
2 ottbacktrack
3 backtrack
Match attempt failed after 3 steps
Beginning match attempt at character 2
1 tto
2 ttobacktrack
3 backtrack
Match attempt failed after 3 steps

```

	Start	Length	
Match 1:	100	10	3 ^{CL} _{RI}
Group 1:	100	10	3

etc until

```

Beginning match attempt at character 10
1 100
2 U
3 US
4 USD
5 USD100
6 ok
7 100
Match found in 7 steps

```

The match of '100' is performed first followed by the evaluation of the Lookaround.

Notes:

1. The added outer group, () around the expression, `.{3}(?<=USD\d{3})` captures the match, '100' of `.{3}` not `\d{3}`. If I used, `((?<=USD\d{3}))`, it will still match but only captures the returned value of **null** because a Lookaround gives up its match values upon completion of the evaluation. If I want to capture the Lookaround value, I have to use, `(?<=(USD\d{3}))`. This captures the Lookaround value before the match is given up and a **null** value results.

Lookarounds: *Lookahead and Lookbehind*

Lookaround Before the Match

String = 100 pesos 100 dollars Plantation 6490.jpg

Match: `((?=\d{3} dollars).\{3})` (*Positive Lookahead*)

Looks ahead for three digits followed by <space> "dollars". Matches "100" in "100 dollars".

Verified this position by adding (...) to the end = '<space>do'
(next character would be '<space>' before 'dollars')

Break down:

Capture Group 1

Lookahead
(`(?=\d{3} dollars)` .`{3}`)

Looks ahead (Looks to the immediate right) for three numeric digits followed by the string, 'dollar'. Matches at the '100 dollars'. Lookaround satisfied. The current position does not change despite the match because this is a Zero Length Assertion. So the current position remains the '1' of '100 dollars'. The Dot Metacharacter matches against the '1' followed by the additional two numeric digits using the Range Quantifier of {3} to capture the '100' in Capture Group 1. The \d{3} is within the non-Capturing Group and that value is given up. It is the .{3} that captures the value '100' and not the \d{3} because there is no Capture Group in the Lookaround.

100 pesos **100** dollars Plantation 6490

Name	New Name
100 pesos 100 dollars Plantation 6490	Capture Group 1= 100

Beginning match attempt at character 10

```

1 100
2 100
3 100 d
4 100 do
5 100 dol
6 100 doll
7 100 dolla
8 100 dollar
9 100 dollars
10 ok
11 100
12 100

```

Match found in 12 steps

	Start	Length	
Match 1:	100	10	3 ^{CL} _{RF}
Group 1:	100	10	3

The Lookaround is performed first followed by the match of '100', thus this Lookaround is 'Before the Match'.

Lookarounds: *Lookahead and Lookbehind*

Lookaround Before the Match cont.

String = 100 pesos 100 dollars USD100 Plantation 6490.jpg

Match: `((?!\d{3} pesos)\d{3})` (*Negative Lookahead*)

Makes sure what follows is not three digits followed by <space> "pesos". Matches "100" in "100 dollars".

Verified this position by adding (...) to the end = '<space>do'
(next character would be '<space>' before 'dollars')

Break down:

Capture Group 1

Negative Lookahead

(`(?!\d{3} pesos)` `\d{3}`)

Looks to the right for three numeric digits NOT followed by the string, 'pesos'. Matches at the '100' of '100 dollars'. The current position does not change despite the match because this is a Zero Length Assertion. So the current position remains at the '1' of '100 dollars'. The `\d{3}` outside of the Lookaround uses a Range Quantifier of {3} to capture the '100' of '100 dollars' in Capture Group 1 (current position, the '1' of '100 dollars' + '00').

100 pesos **100** dollars USD**100** Plantation **6490**

Name	New Name
<input type="checkbox"/> 100 pesos 100 dollars USD100 Plantation 6490	Capture Group 1= 100

.....Start·Length·..		
[-] Match 1: 100	10	3 ^{CL} _{RF}
[-] Group 1: 100	10	3 ^{CL} _{RF}
[-] Match 2: 100	25	3 ^{CL} _{RF}
[-] Group 1: 100	25	3 ^{CL} _{RF}
[-] Match 3: 649	40	3 ^{CL} _{RF}
[-] Group 1: 649	40	3

The Regex Buddy program uses Global, so it matches all three instances of three numeric digits NOT followed by the string, 'pesos'.

[-] Beginning match attempt at character 10	
1	100
2	100
3	100 backtrack
4	ok
5	100
6	100
Match found in 6 steps	

The Lookaround is performed first followed by the match of '100', thus this Lookaround is 'Before the Match'.

Lookarounds: *Lookahead and Lookbehind*

Lookaround Before the Match cont.

String = USD100 JPY100 Plantation 6490.jpg

Match: `((?!USD)\d{3})` *(Negative Lookbehind)*

Makes certain "USD" does not precede three digits. Matches "100" in "JPY100".

Verified this position by adding (...) to the end = '<space> P1'
(next character would be '<space>' before 'Plantation')

Break down:

Capture Group 1

Negative Lookbehind
(`(?!USD)\d{3}`)

Using a back and forth motion, test for three characters that are not 'USD' and if matched, three numeric digits must follow. Match against three numeric digits, look to the left NOT for the string, 'USD'. This will match 'JPY100' and Capture Group 1 will hold the value '100' because the `\d{3}` is outside of the Non-Capturing Group.

Name	New Name
USD100 JPY100 Plantation 6490	Capture Group 1= 100

USD100 JPY**100** Plantation **6490**

```

Beginning match attempt at character 10
1 backtrack
2 ok
3 100
4 100
Match found in 4 steps

```

	Start	Length	
Match 1:	100	10	3 ^{CL} _{RF}
Group 1:	100	10	3 ^{CL} _{RF}
Match 2:	649	25	3 ^{CL} _{RF}
Group 1:	649	25	3

The Regex Buddy program uses Global, so it matches both instances of three numeric digits NOT followed by the string, 'USD'

The Lookaround is evaluated first followed by the match of '100', thus this Lookaround is 'Before the Match'.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match

String = 100 pesos 100 dollars USD100 Plantation 6490.jpg

Match: `(\d{3})(?<=USD\d{3})` (Positive Lookbehind)

Makes certain that "USD" precedes three digits. Matches "100" in "USD100".

Verified this position by adding (...) to the end = 'Pla'
(next character would be 'P' of 'Plantation')

Break down:

Capture Group 1

Lookbehind
(\d{3} (?<=USD\d{3}))

Match against three numeric digits, look to the left for the string, 'USD', followed by three numeric digits.

Name	New Name
100 pesos 100 dollars USD100 Plantation 6490	Capture Group 1= 100

Matches at the '100' of the substring, 'USD100'. The `\d{3}` in the first part of the expression, outside of the Lookaround, captures the '100' in Capture Group 1 and the Lookbehind verifies that 'USD' precedes it.

100 pesos 100 dollars USD**100** Plantation 6490

The match of '100' is evaluated first followed by the evaluation of the Lookbehind, thus, this Lookaround is 'After the Match'.

This may seem redundant, but first it matches against the three numeric digits in the string and then verifies that the substring, 'USD', precedes it. In order for the Lookbehind to verify the string, it must also include the three numeric digits that have already been matched. This is the syntax of the Lookbehind `(?<=USD\d{3})`.

This would not match:

`(\d{3})(?<=USD)`

The current position after the match of '100' captured in Capture Group 1 using the `\d{3}` Range Quantifier, is the `<space>` preceding 'Plantation'. To the immediate left of this is the substring 'USD100' not 'USD'. Therefore, `(?<=USD)` will not match. That is why for the RegEx to succeed it has to be, `(\d{3})(?<=USD\d{3})`.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match cont.

String = 100 pesos 100 dollars USD100 Plantation 6490.jpg

Match: `(\d{3}(?<=USD\d{3}))` (Positive Lookbehind)

Here is an excerpt from Regex101.com that shows how the Lookbehind is evaluated. The '100' has just been matched in `\d{3}` at the beginning of the expression previous to step 56.

<p>MATCH STEP 56</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>	<p>MATCH STEP 57</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>
<p>MATCH STEP 58</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>	<p>MATCH STEP 59</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>
<p>MATCH STEP 60</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>	<p>MATCH STEP 61</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>
<p>MATCH STEP 62</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>	<p>MATCH STEP 63</p> <pre>/ (\d{3}(?<=USD\d{3})) 100*pesos*100*dollars*USD100*Plantation*6490</pre>

What it doesn't show is that it is moving back and forth testing from the '100' from between step 56 to step 57:

'U' = 'D' = False	tests back one position
'U' = 'S' = False	tests back one position
'U' = 'U' = True	tests forward one position beginning step 57
'S' = 'S' = True	tests forward one position
'D' = 'D' = True	Tests for <code>\d{3}</code> at step 60
'1' = <code>\d</code> = True	moves forward one position
'0' = <code>\d</code> = True	moves forward one position
'0' = <code>\d</code> = True	Match. Lookbehind satisfied.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match cont.

String = 100 pesos 100 dollars USD100 Plantation 6490.jpg

Match: `(\d{3}(?<=USD\d{3}))` (Positive Lookbehind)

The current position, the <space> after the substring, 'USD100' (preceding 'Plantation'), doesn't change after the evaluation of the Lookaround. Remember that it is a Zero Length Assertion. Any movement or any values captured are only temporary until after the evaluation. This can be further verified by examining the Debug Log from the Regex Buddy program:

```

Beginning match attempt at character 25
1 100
2 U
3 US
4 USD
5 USD100
6 ok
7 100
Match found in 7 steps

```

	Start	Length	
Match 1:	100	25	3 ^{CL} _{RF}
Group 1:	100	25	3

This is not to say that backtracking is not occurring. The Lookbehind typically uses a back and forth movement that may not be obvious, but can be clearly seen in Failed attempts as evidenced by an attempted match at character position 10 that begins the substring, pesos 100':

```

Beginning match attempt at character 10
1 100
2 backtrack
3 backtrack
4 backtrack
Match attempt failed after 4 steps

```

The match of '100' is evaluated first followed by the evaluation of the Lookbehind, thus, this Lookaround is 'After the Match'.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match

String = 100 pesos 100 dollars Plantation 6490.jpg

Match: `(\d{3}(?= dollars))` *(Positive Lookahead)*

Makes certain <space> "dollars" follows three numeric digits. Matches "100" in "100 dollars".

Break down:

Capture Group 1

(Lookahead
(\d{3}(?= dollars)))

Match 3 numeric digits. Lookahead looks to the immediate right for the substring, <space> 'dollars'. Matches at the '100' following the <space> after 'pesos'. Because the `\d{3}` is outside of the non-Capturing Group, the value of '100' is captured in Capture Group 1.

Name	New Name
<input type="checkbox"/> 100 pesos 100 dollars Plantation 6490	Capture Group 1= 100

100 pesos **100** dollars Plantation 6490

Beginning match attempt at character 10

```

1 100
2 100
3 100 d
4 100 do
5 100 dol
6 100 doll
7 100 dolla
8 100 dollar
9 100 dollars
10 100ok
11 100
Match found in 11 steps

```

	Start	Length	
Match 1:	10	3	^{CL} _{RF}
Group 1:	10	3	

The match of '100' is performed first followed by the evaluation of the Lookaround, thus, this Lookaround is 'After the Match'.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match cont.

String = 100 pesos 100 dollars Plantation 6490.jpg

Match: `(\d{3}(?! dollars))` (*Negative Lookahead*)

Makes certain `<space>` "dollars" doesn't follow three numeric digits. Matches "100" in "100 pesos".

Break down:

Capture Group 1

Negative Lookahead

(`(\d{3}(?! <space> dollars))`)

Match 3 numeric digits. Lookahead looks to the immediate right NOT for the substring, `<space>` 'dollars'. Matches at the '100' at the start of the string. The `\d{3}` uses a Range Quantifier to capture the '100' in Capture Group 1.

Name	New Name
100 pesos 100 dollars Plantation 6490	Capture Group 1= 100

100 pesos 100 dollars Plantation **6490**

Beginning match attempt at character 0

- 100
- 100
- 100 backtrack
- 100ok
- 100

Match found in 5 steps

	Start	Length	
Match 1: 100	0	3	^C _R ^L _F
Group 1: 100	0	3	^C _R ^L _F
Match 2: 649	33	3	^C _R ^L _F
Group 1: 649	33	3	

The Regex Buddy program uses Global, so it matches both instances of three numeric digits NOT followed by the string, `<space>` 'dollars'

The match of '100' is performed first followed by the evaluation of the Lookaround, thus, this Lookaround is 'After the Match'.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match cont.

String = 100 pesos 100 dollars USD100 Plantation 6490.jpg

Match: `(.3){?<=USD\d{3}}` (Positive Lookbehind)

Makes certain 'USD' precedes three numeric digits. Matches "100" in "USD100".

Break down:

Capture Group 1

(
 Lookbehind
 (.3){?<=USD\d{3}}
)

The three Dot Metacharacters continually test and match against each three characters of the string, but fails to match against the Lookbehind of the string, 'USD', until it matches against the '100' of the substring, 'USD100'. This value is captured in Capture Group 1. After it matches, it evaluates the Lookbehind that looks to the immediate left for the substring 'USD' followed by 3 numeric digits. This matches and the RegEx is satisfied. It will continue to search out other matches until EOL or the string is exhausted whichever comes first.

Name	New Name
100 pesos 100 dollars USD100 Plantation 6490	Capture Group 1= 100

100 pesos 100 dollars USD**100** Plantation 6490

Beginning match attempt at character 25

```

1 100
2 U
3 US
4 USD
5 USD100
6 ok
7 100

```

Match found in 7 steps

	Start	Length	
Match 1:	100	25	3 ^{CL} _{RF}
Group 1:	100	25	3

The match of '100' is performed first followed by the evaluation of the Lookaround, thus, this Lookaround is 'After the Match'.

Notes:

1. Lookarounds like Word Boundaries and non-Word Boundaries and other non-Capturing Groups are not documented well by Regex Buddy. In the above, only the match is identified by 'ok' in the Debug Log.

Lookarounds: *Lookahead and Lookbehind*

Lookaround After the Match cont.

String = USD100 JPY100 Plantation 6490.jpg

Match: `(\d{3}(?!USD\d{3}))` (*Negative Lookbehind*)

Makes certain that 3 numeric digits are NOT preceded by "USD". Matches "100" in "JPY100".

Break down:

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

(`\d{3}` Negative Lookbehind `(?!USD)\d{3}`)

Match against three numeric digits. Look to the immediate left NOT for the substring, 'USD'. This will match 'JPY100' and Capture Group 1 will hold the value '100'. The `\d{3}` uses a Range Quantifier to capture the '100' in Capture Group 1 in the first part of the expression, and the Lookbehind verifies that 'USD' doesn't precede it.

Name	New Name
USD100 JPY100 Plantation 6490	Capture Group 1= 100

USD**100** JPY100 Plantation 6490

Beginning match attempt at character 3

1	100
2	U
3	US
4	USD
5	USD100
6	ok
7	100

Match found in 7 steps

	Start	Length	...
Match 1:	100	3	3 ^C _R ^L
Group 1:	100	3	3

The match of '100' is performed first followed by the evaluation of the Lookaround, thus, this Lookaround is 'After the Match'.

Summary Notes:

1. An outer group () was used to capture the value of `\d{3}` to Capture Group 1, e.g. `((?!USD)\d{3})`.
2. If you want proof that the above works, add (...) as a separate Capture Group e.g. `((?!USD)\d{3})(...)`. This will return the first few characters of the string after the match providing the current position using a Replace data of `\1\2`. For example, in the last match 100 following JPY, adding (...) yields 'Plan' in Capture Group 2.
3. If using Lookbehinds in BRU, alternatives must be a fixed length. This does not apply to Lookaheads.
4. A Lookaround will not consume characters. The position will remain at the point *of* the capture whereas normally, the position will move to the point *after* the capture.

Using Alternates with Capture Groups

Quick Review:

The | (pipe) is a metacharacter that is an OR operator. It works in it's simplest form as

Cat | Mouse

Look in the string and match either against 'Cat' or 'Mouse', whichever comes first.

Using ONE Capture Group with two alternates

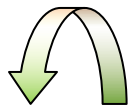
gr(a|e)y can match gray or grey because the expression states that 'gr' followed by either 'a' or 'e' then 'y'. The characters 'a' and 'e' are in an isolated Capture Group that groups them together in an OR statement. The parentheses also show an order of evaluation. What it really does is this:

test for characters 'gr', if **true** then...

does 'a' follow 'gr'? if **true** then Capture Group = 'a'; if **false** then Capture Group = **undefined**
 does 'e' follow 'gr'? if **true** then Capture Group = 'e'; if **false** then Capture Group = **undefined**

if Capture Group 1 **true** then test for 'y'; if **true** then expression is a match

This testing of the alternates can be envisioned as:



gr (a | e) y

Using ONE Capture Group and three alternates

Example:

String = cat dog mice.jpg


Match: (cat|dog|mice)

Where: \1 = cat

Using Alternates with Capture Groups

Using TWO Capture Groups and three alternates

Example #1 isolate cat in its own Capture Group. Finds 'cat' first.


Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = cat

String = cat dog mice.jpg

Match: ((cat)|dog|mice)
 Replace: Capture Group 1 = \1 Capture Group 2 = \2
 Where: \1 = cat
 \2 = cat

Example #2 isolate dog in its own Capture Group


This is the same as example #1 only isolating 'dog' instead of 'cat'. Finds 'cat' before 'dog'

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = \2.jpg

String = cat dog mice.jpg

Match: ((dog)|cat|mice)
 Replace: Capture Group 1 = \1 Capture Group 2 = \2
 Where: \1 = cat
 \2 = undefined

Example #3 change first element to dog in string. Isolate 'dog' in RegEx. Finds 'dog' first.

Name ▲	New Name
 dog cat mice.jpg	Capture Group 1 = dog Capture Group 2 = dog.jpg

String = dog cat mice.jpg

Match: ((dog)|cat|mice)
 Replace: Capture Group 1 = \1 Capture Group 2 = \2
 Where: \1 = dog
 \2 = dog

Notes:

1. An alternate works on a first come first served basis meaning that BRU will match against the first **true** alternate and discard the rest even if there are more than one true matches in the string. In the above string, all three values, 'dog', 'cat' and 'mice' are present but because 'dog' comes before the other two values, only the first alternate is **true** and the other two alternates are **false** and not even tested.
2. Why does Capture Group 1 and 2 hold the 'same' value? This will be discussed momentarily under 'Nested' Capture Groups.

Using Alternates with Capture Groups

Using THREE Capture Groups and three alternates

Example #1 One element isolated from the other two that are grouped together


String = cat dog mice.jpg

Match: ((dog)|(cat|mice))

Where: \1 = cat

\2 = "" (**null**) This displays as .jpg under the 'New Name' column of BRU

\3 = cat

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = Capture Group 3 = cat

Here is what the RegEx looks like:

```

      C a p t u r e   G r o u p   1
    (
      Capture Group 2   |   Capture Group 3
      (dog)             |   (cat|mice)
    )
  
```

The second alternate, 'cat', is **true** because 'cat' is matched first. Why? Because 'cat' appears in the string before the other two values.

1. First alternate dog is tested against the string value 'cat' and fails
2. This falls to the next alternate, cat, which matches = **true**.
3. Capture Group 1 and Capture Group 3 hold the same value – this is because Capture Group 3 is 'nested' within Capture Group 1. This is discussed next.

Using Alternates with Capture Groups

Using THREE Capture Groups and three alternates

Nested Capture Groups

This is a good time to bring up the subject of ‘Nested’ Capture Groups.


Here is the RegEx again:

```

      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   |   C a p t u r e   G r o u p   3
      ( d o g )                       |   ( c a t | m i c e )
    )
  
```

Capture Group 1 is the aggregated values from all of the Capture Groups within (inclusive). There are two Capture Groups within Capture Group 1, but because they represent alternates, there is only one value.

If I just had this:

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = Capture Group 2 = cat


Match: (dog)|(cat|mice)

Then Capture Group 1 tests the alternate ‘dog’ and fails = **null**. Although ‘dog’ is part of the alternate ‘dog | cat | mice’, Capture Group 1 is **true** only if the outcome of the alternate was the first match, ‘dog’, which it is not. Therefore Capture Group 1 captures a **null** value. This is what some refer to as a Zero Occurrence Match. This typically occurs when the RegEx engine evaluates an expression within a Capture Group but returns no value. It is empty or **null**. Another scenario is when an existing Capture Group has been forced to give up it’s match or has given back all of its matched characters. For more information on this, please refer to Volume II. A Zero Occurrence Match is also a Zero Length Match.

Capture Group 2 = **true** = ‘cat’

There is no Capture Group 3 because Capture Group 2 is made up of an alternate, ‘dog |cat |mice’

But instead, I have this:

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = Capture Group 3 = cat

Match: ((dog)|(cat|mice))

Let’s simplify this. The RegEx tests for three alternates. With each test, pass or fail, that value is held by any Capture Group that represents that alternate. If an alternate is tested and fails, then a **null** or empty value is held. If, on the other hand, an alternate matches, the value of that alternate is held and subsequent testing of alternates are not performed. For any Capture Group that represents an untested alternate, that Capture Group will have an **undefined** value or no value.

Using Alternates with Capture Groups

Using THREE Capture Groups and three alternates

Nested Capture Groups

Undefined is not the same as a **null** value because there is no value. Never tested.

In BRU, an undefined value displays as 'Invalid' in red:

Match: ((cat)|(dog))
 Replace: \3

 cat dog mice.jpg \3.jpg

It also displays the Backreference to Capture Group 3 indicating that the problem with the Replacement String resides within Capture Group 3. This could also indicate a syntax error in the RegEx expression, but in this case, it refers to an **undefined** Capture Group because the Capture Group holds no value, not even **null**. By expressing Capture Group 3 in the Replacement String, BRU says 'No Can Do'. The file cannot be renamed as long as the 'red' **Invalid** indicator remains. BRU flags it as **Invalid** because of the backslash that appears as a literal in New Name.

As opposed to a **null** value based on a Zero Occurrence Match:

Match: (dog)|(cat|mice)
 Replace: \1

 cat dog mice.jpg .jpg

This shows New Name in green, meaning that there is no problem with Capture Group 1 represented by Backreference \1, except that there is no visible value. It displays only the file's extension. The value is **null** or empty but there is a value. This is not to be confused with a <space> value, also not visible, but would display as pushing the filename represented by New Name, one position to the right, e.g., <space>.jpg vs .jpg

Bottom line, no matter how complicated the Alternate groups may appear, they really boil down to:

This Or This Or This Or
 n | n1 | n2 | etc.

The rest are just Capture Groups that may hold a value for any alternate tested or may not hold any value for any alternate not tested. The order in which the alternates are tested and any resulting values held by a Capture Group is based on the order in which they are evaluated in the RegEx. The expression that is within a Capture Group that is evaluated first will always be designated as Capture Group 1, for example.

In addition, any Nested Capture Groups will pass their aggregated values to the Capture Group in which they inhabit. For example, if Group 1 is inclusive of Group 2 and Group 3, and Group 2 has a value 'B' and Group 3 has a value 'C' and Group 1 has an expression outside of Groups 2 and 3 that evaluates to a value of 'A', then Group 1 has an aggregated value of 'ABC'.

Using Alternates with Capture Groups

Using THREE Capture Groups and three alternates


Nested Capture Groups

So let's go back to our example:

String = `cat dog mice.jpg`

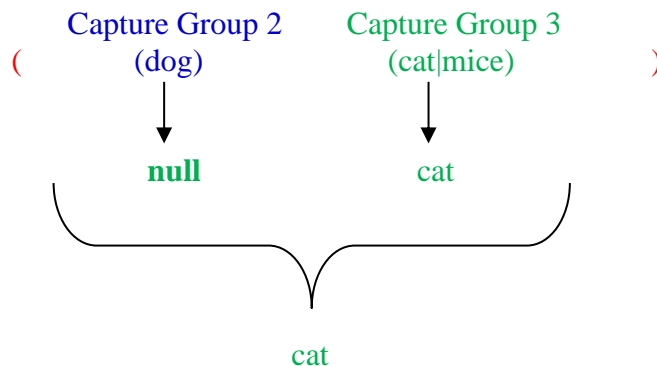
Match: `((dog)|(cat|mice))`

Replace: `Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3`

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = Capture Group 3 = cat.jpg

Breaks down:

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



Tests alternatives ↓ `cat dog mice.jpg`

`dog | cat | mice`

`dog` fails against first value `'cat'` in string

`cat` matches against first value `'cat'` in string = `true`

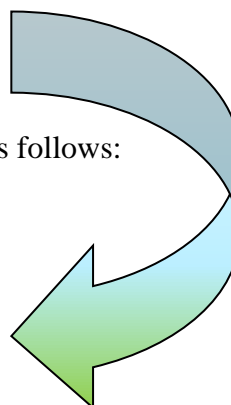
`mice` not tested but this alternate is part of Capture Group 3

So after the alternates have been tested, the Captured values are as follows:

Capture Group 1 = (aggregated value `'null'` + `'cat'` =) `'cat'`

Capture Group 2 = (`dog` tested and fails so..) `null` value

Capture Group 3 = (`cat` or `mice` tested and `'cat'` matches =) `'cat'`



Using Alternates with Capture Groups

Using THREE Capture Groups and three alternates

Nested Capture Groups

Notes:

1. Capture Group 1 currently holds the value 'cat' because any value, e.g., 'cat' will replace a **null** value.
2. Because the alternate, cat, matched the string and satisfied the RegEx expression, the other parts of the string made up of 'dog' and 'mice' are not tested against the alternates.
3. If the alternate, mice, had been isolated in its own Capture Group, that Capture Group would be **Invalid** since mice was never tested. However, because mice was part of Capture Group 3 which included the alternate, cat, Capture Group 3 holds the matched value, 'cat' and is not designated '**Invalid**'.
4. Do not get confused when I am referring to an alternate, cat, part of the RegEx expression, and the value 'cat' held by a Capture Group after a successful match.
5. A nested Capture Group takes its value inclusive of all values of Capture Groups held within.
6. 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 1 designation while the Inner Groups are assigned the next designation, 2 and 3 and so on. Nesting can therefore have an impact on the order in which the alternates are tested.
7. The Regex performs the evaluations from left to right and assigns values to any Capture Groups based on those evaluations. With nested Capture Groups, however, evaluations are performed from outer to inner, meaning that the Capture Group that represents the outer group is designated a lower Capture Group number than the Capture Groups within. The Capture Groups within are designated Capture Group numbers in the same manner from left to right as they appear in the RegEx and any nested Capture Groups within are handled similarly.

Using Alternates with Capture Groups

Using THREE Capture Groups and three alternates

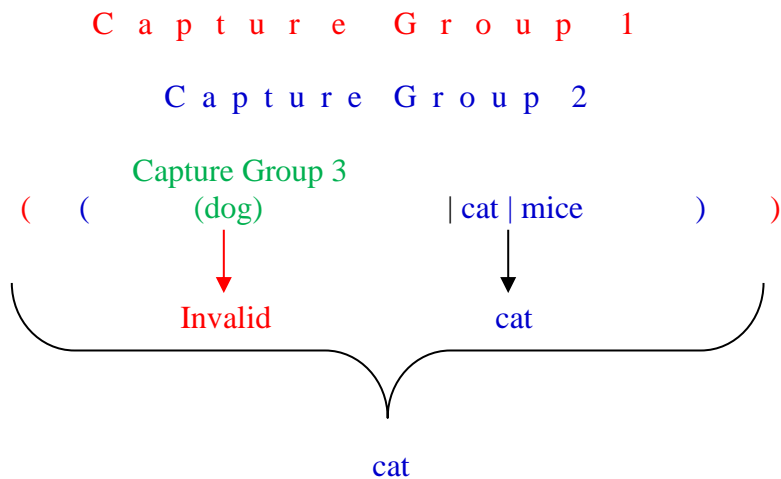
Nested Capture Groups

Example #2 Group all three elements together but isolate dog within this grouping

String = cat dog mice.jpg

Match: (((dog)|cat|mice))
 Replace: Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3
 Where: \1 = cat
 \2 = cat
 \3 = **Invalid**

This is what the RegEx looks like:



Tests alternatives ↓
 dog | cat | mice

cat dog mice.jpg

In Capture Group 1, cat matches against first value 'cat' in string = **true** (aggregated value of Capture Groups 2 and 3)

In Capture Group 2, cat matches against first value 'cat' in string = **true** = 'cat'

In Capture Group 3, dog alternate not tested because the RegEx was satisfied = **Invalid**

Name ▲	New Name
📁 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = cat Capture Group 3 = \3.jpg

BRU displays New Name as **Invalid**. Remove Capture Group 3 from Replacement String:

Name ▲	New Name
📁 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = cat.jpg

Using Alternates with Capture Groups

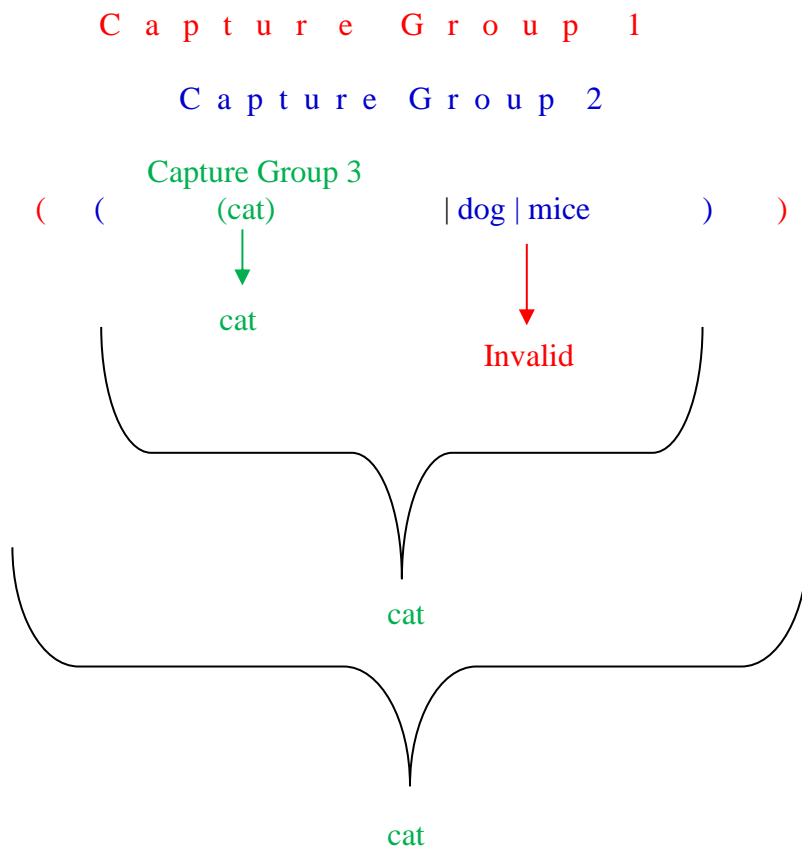
Using THREE Capture Groups and three alternates

Nested Capture Groups

If I had –

String = cat dog mice.jpg

Match:	<code>((cat) dog mice)</code>
Replace:	Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3
Where:	\1 = cat
	\2 = cat
	\3 = cat



Name ▲	New Name
🐱 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = cat Capture Group 3 = cat.jpg

Although the alternates, ‘dog’ and ‘mice’ are never tested because the RegEx has been satisfied, and would be **Invalid**, they reside within Capture Group 2 which takes its value, ‘cat’, from Capture Group 3, also within Capture Group 2. Therefore Capture Group 2’s value is ‘cat’. Capture Group 1’s value is ‘cat’ representing the aggregated values of all Capture Groups inclusive.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

Nested Capture Groups

Example #1 group three elements together but keep them isolated within the grouping

String = cat dog mice.jpg

Match:	((cat) (dog) (mice))		
Replace:	Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4		
Where:	\1 = cat		
	\2 = cat	cat comes before dog	
	\3 = undefined		
	\4 = undefined		

Here is what it looks like:

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

Capture Group 2 Capture Group 3 Capture Group 4

((cat) | (dog) | (mice))

Name ▲	New Name
cat dog mice	Capture Group 1 = cat Capture Group 2 = cat Capture Group 3 = \3 Capture Group 4 = \4

Tests for 'cat' = true, Capture Group 2 = 'cat', Capture Group 1 = 'cat' (aggregated value of Capture Groups 2 – 4)

The second alternate, 'dog' is never tested because the first alternate matched. Capture Group 3 = undefined.

The same for the alternate, 'mice' which is never tested. Capture Group 4 = undefined.

Notes:

1. **Undefined** refers to a Capture Group that never existed as is the explanation here. By now you should be familiar with how the **Invalid** flag attached to New Name is identified and why. If you need a refresher, there is one coming up on the next page.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

Nested Capture Groups

String = `cat dog mice.jpg`

Example #2 Mixing up the alternates

`((dog)|(cat)|(mice))`

Match:	<code>((dog) (cat) (mice))</code>			
Replace:	Capture Group 1 = \1	Capture Group 2 = \2	Capture Group 3 = \3	Capture Group 4 = \4
Where:	\1 = cat	\2 = null	\3 = cat	cat comes before dog
	\4 = undefined			

Here is what it looks like:

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

Capture Group 2 Capture Group 3 Capture Group 4

((dog) | (cat) | (mice))

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = Capture Group 3 = cat Capture Group 4 = \4.jpg

Capture Group 1 = 'cat' (aggregated value of Capture Groups 2-4)

Capture Group 2 is tested but **fails** = **null**

Capture Group 3 tests for 'cat' = **true** = 'cat'

Capture Group 4 never tested = **Invalid**

Notes:

- Just a reminder that the **Invalid** flag is primarily because Capture Group 4 doesn't exist because it was never tested. Therefore the Replacement String's reference to Capture Group 4 is evaluated as a literal string. The backslash is an illegal windows character in a filename and therefore New Name is flagged as **Invalid**. It does however provide a clue as to what is causing the problem with the RegEx, which obviously has to do with or at least start with Capture Group 4.
- The **Invalid** flag indicates that the Rename Operation cannot be performed as long as the New Name remains flagged. **Invalid** entries appear in **Red**. Legitimate entries appear in **Green** in New Name.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

Nested Capture Groups

String = cat dog mice.jpg

Example #3 Mixing up the alternates

((mice)|(cat)|(dog))

Match:	((mice) (cat) (dog))		
Replace:	Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4		
Where:	\1 = cat		
	\2 = null		
	\3 = cat	cat comes before dog	
	\4 = undefined		

Here is what it looks like:

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

Capture Group 2 Capture Group 3 Capture Group 4

((mice) | (cat) | (dog))

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = cat Capture Group 2 = Capture Group 3 = cat Capture Group 4 = \4.jpg

As far as the outcome and the analysis, nothing changes from that of Example #2 because the 'cat' alternate remains in the second position held by Capture Group 3. But it doesn't matter if 'cat' is in the first, second or third position. The outcome and the analysis may change as far as the positioning in the renaming, but 'cat' will always match. No matter how I change the order of the alternates, it will always match against 'cat' because 'cat' appears in the string before either 'dog' or 'mice'. But this is not always the case as you will see.

For now, though, what if I took 'cat' out of the equation ...

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

Nested Capture Groups

String = cat dog mice.jpg

Example #4 Match against dog

If I change 'cat' to 'rabbit', then 'dog' will be the first match.

((mice)|(rabbit)|(dog))


Match:	((mice) (rabbit) (dog))			
Replace:	Capture Group 1 = \1	Capture Group 2 = \2	Capture Group 3 = \3	Capture Group 4 = \4
Where:	\1 = dog	\2 = null	\3 = null	\4 = dog
				dog comes before mice

Here is what it looks like:

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

Capture Group 2 Capture Group 3 Capture Group 4

((mice) | (rabbit) | (dog))

Name ▲	New Name
 cat dog mice.jpg	Capture Group 1 = dog Capture Group 2 = Capture Group 3 = Capture Group 4 = dog.jpg

Capture Group 1 = 'dog' (aggregated value of Capture Groups 2-4)

Capture Group 2 is tested but **fails** = **null**

Capture Group 3 is tested but **fails** = **null**

Capture Group 4 tests for 'dog' = **true** = 'dog'

1. The first alternate, mice, is tested against the first string value, 'cat' and **fails**.
2. The second alternate, rabbit, is tested against the first string value, 'cat' and **fails**.
3. The third alternate, dog, is tested against the first string value, 'cat' and **fails**.
4. The first alternate, 'cat', is tested against the second string value, 'dog' and **fails**.
5. The second alternate, rabbit, is tested against the second string value, 'dog' and **fails**.
6. The third alternate, dog, is tested against the second string value, 'dog' and matches.
RegEx satisfied, Capture Group 4 = 'dog'

Notes:

1. The RegEx engine is actually testing each and every character that makes up the string value against the alternates.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

Now that you understand about Alternates, Capture Groups and Nested Capture Groups, let's continue.

String = Star Wars Episode USD100 123x12 aNbc.jpg

What all of the following have in common is that they capture the value of 'Wars'. The match will always be 'Wars' because that is the first occurrence before either 'Episode' or 'Nbc'. Therefore, the only value that can be captured is 'Wars'. Here is a simple OR statement using three Capture Groups:

Evaluation order:

e.g. $\overset{1}{(Wars)}|\overset{2}{(Episode)}|\overset{3}{(Nbc)}$

where: 1 = Capture Group 1 2 = Capture Group 2 3 = Capture Group 3

Match: $(\mathbf{Wars})|(Episode)|(Nbc)$

Where: \1 = Wars

New Name

Capture Group 1 = Wars Capture Group 2 = \2 Capture Group 3 = \3.jpg

Analysis: 'Wars' is in Capture Group 1 position. Since the OR has been satisfied, Capture Groups 2-3 are **undefined** (never tested).

Match: $(Episode)|(\mathbf{Wars})|(Nbc)$

Where: \1 = **null**

\2 = Wars

New Name

Capture Group 1 = Capture Group 2 = Wars Capture Group 3 = \3.jpg

Analysis: 'Wars' is in Capture Group 2 position, therefore Capture Group 1 has a **null** value. Capture Group 3 is **undefined** (never tested).

Match: $(Nbc)|(\mathbf{Wars})|(Episode)$

Where: \1 = **null**

\2 = Wars

New Name

Capture Group 1 = Capture Group 2 = Wars Capture Group 3 = \3.jpg

Analysis: 'Wars' is in Capture Group 2 position, therefore Capture Group 1 has a **null** value. Capture Group 3 is **undefined** (never tested).

Match: $(Nbc)|(Episode)|(\mathbf{Wars})$

Where: \1 = **null**

\2 = **null**

\3 = Wars

New Name

Capture Group 1 = Capture Group 2 = Capture Group 3 = Wars.jpg

Analysis: 'Wars' is in Capture Group 3 position, therefore Capture Group 1 and Capture Group 2 return **null**.

Using Alternates with Capture Groups

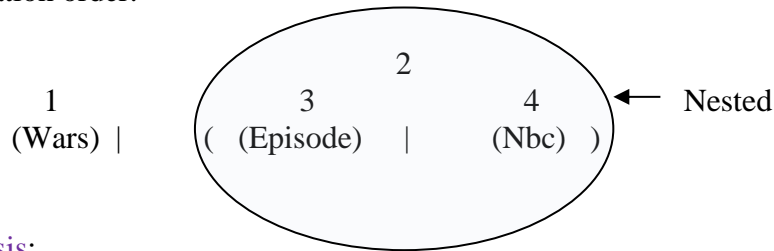
Using FOUR Capture Groups and three alternates

Many of these evaluations are just following a repetitious pattern. But things will start to change when I add a nested Capture Group. This next couple of examples have the alternate, Wars, in the first and second position. There won't be any difference in the outcome. Wars will still match.

String = Star Wars Episode USD100 123x12 aNbc.jpg

Match: (Wars)|((Episode)|(Nbc))
 Replace: Capture Group 1 = \1 Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4
 Where: \1 = Wars
 \2 - 4 = **not defined**

Evaluation order:



Analysis:

1 = Capture Group 1:

Tests alternate, Wars, against the string to see if it is the first occurrence. If **true**, Capture Group 1 captures the value and the OR has been satisfied. Capture Groups 2-4 will be left **undefined**. If **false**, then it continues to test using Capture Group 2 and Capture Group 1 will hold a **null** value..

2 = Capture Group 2:

If either alternate of Capture Group 3 or 4 is the first occurrence, then Capture Group 2 holds the aggregated value.

3 & 4 = Capture Group 3 and Capture Group 4

If either of these alternates are **true**, then the Capture Group that matches will capture that value. The value is passed along to Capture Group 2. Additionally...

- If Capture Group 3 is **true** then Capture Group 4 will be left **undefined** (never tested).
- If Capture Group 4 is **true** then Capture Group 3 will have a **null** value (tested but **false**).

Name	New Name
Star Wars Episode USD100 123x12 aNbc.jpg	Capture Group 1 = Wars Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4.jpg

Capture Group 1 matches against 'Wars' in the string as the first occurrence, satisfying the OR expression. Capture Groups 2-4 are never tested and are therefore **undefined**.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

String = Star Wars Episode USD100 123x12 aNbc.jpg

Wars in Second Position

Match: (Nbc)|((Wars)|(Episode))

Where: \1 = **null**
 \2 = Wars
 \3 = Wars
 \4 = **undefined**

New Name

Capture Group 1 = Capture Group 2 = Wars Capture Group 3 = Wars Capture Group 4 = \4.jpg

Evaluation order:

1	3	2	4
(Nbc)	(Wars)		(Episode))

Nbc in the first group is tested but because it appears at the end of the string, this fails the test. The second alternate is tested and matches in Capture Group 3. In the string, 'Wars', comes before either 'Nbc' or 'Episode', and because Capture Group 3 is nested in Capture Group 2, Capture Group 2 is the aggregated value –

Or in other words, Capture Group 2 takes its value from both Capture Group 3 and Capture Group 4. Capture Group 4 is left **undefined** because it is never tested since the RegEx was satisfied by Capture Group 3's match. Capture Group 2's value is left as comprised of only the match from Capture Group 3. Capture Group 2 therefore holds the value of 'Wars'. Capture Group 1 returns a **null** value because it was tested but failed to match.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

String = Star Wars Episode USD100 123x12 aNbc.jpg

This next one is different.

I have changed around the RegEx to have Episode as the first alternate under Capture Group 1.

Wars in 3rd Position, Episode in First Position

Match: (Episode)|((Nbc)|(Wars))

Where: \1 = **null**
 \2 = Wars
 \3 = **null**
 \4 = Wars

New Name

Capture Group 1 = Capture Group 2 = Wars Capture Group 3 = Capture Group 4 = Wars

Evaluation order:

1		3		2		4
(Episode)		(Nbc)		(Wars)		(Wars)

Analysis:

Wars is the first occurrence before either Episode or Nbc. Capture Group 4 captures the value, 'Wars', and Capture Group 3 is the aggregate value of both Capture Group 3 and Capture Group 4. Capture Group 3 is tested and returns a **null** value. Thus, Capture Group 2 is comprised of only the value from Capture Group 4. Capture Group 2 therefore holds the value of 'Wars'. Capture Group 1 also was tested and returned a **null** value.

Notes:

- The analysis is a simplified account. What actually happens is that each character of the string is tested against the first character of the alternate, 'E' of 'Episode'. If that fails, it tests the first character of the second and third alternate, falling back to the first alternate again for the next character in the string until a match can be found.

When a match is found, the value of **null** or **undefined** is dependent on where in the RegEx the matched alternate occurs. If, for example, the match occurs at the first alternate, the second and third alternates are not tested and the values remain **undefined**. If the match occurs on the second alternate, then the first alternate returns a **null** value and the third alternate is never tested and remains **undefined**. If the match occurs on the third alternate, then both the first and second alternate return a **null** value.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

String = Star Wars Episode USD100 123x12 aNbc.jpg

Wars in 2nd Position, Episode in First Position

Match: (Episode)|((Wars)|(Nbc))

Where:	\1 = null	<input type="text" value="New Name"/>
	\2 = Wars	Capture Group 1 = Capture Group 2 = Wars Capture Group 3 = Wars Capture Group 4 = \4
	\3 = Wars	
	\4 = undefined	

Evaluation order:

1		2	
(Episode)		((Wars)	(Nbc))

Analysis:

Once more Wars is the first occurrence. In the second position this leaves Capture Group 1 with a **null** value and Capture Group 4 remains **undefined**.

Notes:

1. No difference in the outcome. Wars will always match as long as it is the first occurrence in the string regardless of the positioning in the RegEx.

Using Alternates with Capture Groups

Using FOUR Capture Groups and three alternates

String = Star Wars Episode USD100 123x12 aNbc.jpg

Wars in 3rd Position, Episode in First Position

Here's a tricky one. See if you can follow it.

Match: (Episode) | ((Nbc) | (Wars))

Where: \1 = Episode
 \2 - 4 = **undefined**

New Name
Capture Group 1 = Episode Capture Group 2 = \2 Capture Group 3 = \3 Capture Group 4 = \4

Evaluation order:

1	3	2	4
(Episode)	(Nbc)	(Wars))

Analysis:

Whoa, what happened? Episode matched over Wars?? Hmm... This requires a closer look.

Look carefully at the Match String. Do you see? It has <spaces>.

It is actually:

(Episode) <space> |<space> ((Nbc)<space> |<space> (Wars))

This means that the first alternate has to match against 'Episode' followed by a <space>. The second alternate has to match against 'Nbc' preceded by and followed by a <space> and the third alternate has to match against 'Wars' preceded by a <space>.

Okay. Got it. So why did Episode match and Wars didn't? Simple. In the string, 'Nbc' is preceded by the lowercase letter, 'a' and not a <space>, therefore, Wars is not ever tested because 'Nbc' can never match. Remember it is first come first served. So if 'Nbc' cannot match anywhere, it will fall back to the first alternate. Episode matches in Capture Group 1 while Capture Groups 2 through Capture Group 4 remain **undefined**.

Conclusion:

This section hopefully provided an understanding of how RegEx evaluates a match using alternatives. I am no expert in RegEx, but like most of you, I try to understand **how** it works and *not* just **that** it works.

Personal examples

Here's a few I use for cataloging purposes.

1. Fix Version Numbers

Match file names that require a dot for version numbers

String = ASP.NET 3 5 2008.pdf

Match: `(.*)\s((([0-9])\s([0-9]))(.*))`

Replace: `\1.\2\5`

Note the dot after \1

Observations:

1. The whole idea of Grouping is to isolate various elements of the string called 'parsing'.
2. Each grouping or Capture Group is identified by \1, \2 etc. This is called a backreference.
3. The string is read all the way through using .*
4. The pattern is set by the expression and sub-expressions both outside and within the groups.
5. The above searches for the space character between two numeric digits.
6. If matched, the Replace String puts together the New Name string with an added dot character.

Result:

Name	New Name
 ASP.NET 3 5 2008.pdf	ASP.NET 3.5 2008.pdf

Where:


\1 = ASP.NET 3 \2 = 5 2 \3 = 5 \4 = 2 \5 = 008.pdf

Unfortunately, the Result is not universal. I really need to work on a better RegEx for this one.

String = Here is another string 7 0 no date this time.pdf

Replace: `\1 \3.\4\5` (requires a different Replace String)

Result:

Name	New Name
 Here is another string 7 0 no date this time.pdf	Here is another string 7.0 no date this time.pdf

Personal examples

1. Fix Version Numbers cont.

New and Improved:

Name	New Name
This is a 3 4 test.jpg	This is a 3.4 test.jpg
This is version 3 5 2010 test	This is version 3.5 2010 test
ASP.NET 3 5 2008.pdf	ASP.NET 3.5 2008.pdf
Here is another string 7 0 no date this time	Here is another string 7.0 no date this time

This one is more universal and should match more strings without having to customize the Replacement String as the previous RegEx required.

Match: `(.*?)(\b\d+?\b)(s+?)(\b\d+?\b)(s+?)(\w+)(.*)`

Replace: `|1|.4|5|6|7`

Defines Word Boundaries that are inclusive of one or more numeric digits that are separated by one or more <spaces>, and ends the match search on a Word (Capture Group 4) to differentiate, for example, a year from a version identification. This was just one of the problems with the previous RegEx.

Examples: Regex Buddy photos based on a simpler version for clarity - `(\b\d+?\b)s+?(\b\d+?\b)s+?(\w+)`

Here is another string **7 0 no** date this time

```
.....
Match 1: 7 0 no
Group 1: 7
Group 2: 0
Group 3: no
```

```
Beginning match attempt at character 23
1 ok
2 7
3 7ok
4 7
5 7
6 7 ok
7 7 0
8 7 0ok
9 7 0
10 7 0
11 7 0 no
12 7 0 no
Match found in 12 steps
```

This is version **3 5 2010** test

```
.....
Match 1: 3 5 2010
Group 1: 3
Group 2: 5
Group 3: 2010
```

```
Beginning match attempt at character 16
1 ok
2 3
3 3ok
4 3
5 3
6 3 ok
7 3 5
8 3 5ok
9 3 5
10 3 5
11 3 5 2010
12 3 5 2010
Match found in 12 steps
```

The other RegEx could not handle this string:

`(.*)\s((\d+)(\s)(\d+))(.*)`

```
.....
Match 1: This is version 3 5 2010 test
Group 1: This is version 3
Group 2: 5 2
Group 3: 5
Group 4: 2
Group 5: 010 test
```

It miscalculated the <space> between '5' and '2010' as a version number

Personal examples

1. Fix Version Numbers cont.

Even better. Results same, but less complicated and expounds on the original.

Sample strings:

Here is another string 7 0 no date this time.pdf

ASP.NET 3 5 2008.pdf






8 0 is the version

This is version 3 5 2010 test

700 8 is next

Match: `(.+)?((([0-9])\s)([0-9][^0-9]))(.*)`

Replace: `\1\3.\5\6`

Name ▲	New Name
 Here is another string 7 0 no date this time.p...	Here is another string 7.0 no date this time.pdf
 ASP.NET 3 5 2008.pdf	ASP.NET 3.5 2008.pdf
 8 0 is the version	8.0 is the version
 This is version 3 5 2010 test	This is version 3.5 2010 test
 700 8 is next	700.8 is next

Compared with the original version:

Current: `(.+)? (([0-9]) (\s) ([0-9] [^0-9])) (.*)`

Original: `(.*) \s (([0-9]) \s ([0-9])) (.*)`

Analysis:

- `(.+)?` Matches from the beginning using `.+?` rather than at the end using `.*`. Also by making it Lazy, it can take into account those strings that begin with a numeric digit.
- `(([0-9])\s)([0-9][^0-9])` Next it searches out a numeric digit followed by a `<space>` followed by a second numeric digit. The negated class consisting of a numeric digit ensures that any other numeric digits that follow will be ignored as the evaluation moves forward through the string. Also could have used `[^\w]` in place of `[^0-9]`.
- `(.*)` Any text that remains in the string is captured.






Personal examples

1. Fix Version Numbers cont.

Sample strings:

Here is another string 7 0 no date this time.pdf ASP.NET 3 5 2008.pdf 8 0 is the version
This is version 3 5 2010 test 700 8 is next

Match: (.+)?((([0-9])(\s)([0-9][^0-9]))(.*?))\$
Replace: \1\3.\5\6

Name ▲	New Name
 Here is another string 7 0 no date this time.p...	Here is another string 7.0 no date this time.pdf
 ASP.NET 3 5 2008.pdf	ASP.NET 3.5 2008.pdf
 8 0 is the version	8.0 is the version
 This is version 3 5 2010 test	This is version 3.5 2010 test
 700 8 is next	700.8 is next








But, although it captures more of the sample strings, it can't match against –

this is version 8 0

this will, using \d:

or this, using classes:

Match: (.+)?(\d)(\s)(\d)(?! \d)(.*) Match: (.+)?([0-9])(\s)([0-9])(?! [0-9])(.*)
Replace: \1\2.\4\5 Replace: \1\2.\4\5

Name ▲	New Name
 this is version 8 0	this is version 8.0
 This.is version v7 0	This.is version v7.0
 700 8 is next	700.8 is next
 8 0 is the version	8.0 is the version
 ASP.NET 3 5 2008.pdf	ASP.NET 3.5 2008.pdf
 This is version 3 5 2010 test	This is version 3.5 2010 test
 Here is another string 7 0 no date this time.p...	Here is another string 7.0 no date this time.pdf

Notes:

1. No difference in processing using \d or classes in this example. Both require 88 steps with backtracking. Interesting to note that the 'New and Improved' version that was a bit more complicated did not require backtracking and had much fewer steps, although it did not match against the samples that ended with numeric digits.
2. Have to enable 'Rename File Extensions As Being part of File Name' of the File/Folder Extensions sub-menu from the Renaming Options Menu to accommodate the dot character in the string, 'This.is version 7 0'.
3. The (?!\d) is a Lookaround that Looks Ahead (to the right of the current position) to see that the next character after the match of the numeric digit following the <space> is not another numeric digit. This means that it would not match against a sample string, 'This is test 3 00'. You would need to change the Match String to, (.+)?(\d)(\s)(\d)(.*) with a Replace String of \1\2.\4\5 specific to that pattern.

Personal examples

2. Fix Author's Initials

Match file names that require a dot after an Author's initial

String = J Doe.pdf

Match: `(.*)([A-Z])([A-Z])(.*)`

Replace: `\2. \4\5` (Note the dot and space after \2)

Analysis:

1. The above searches for the space between two capitalized letters
2. If matched, the Replacements put together the string with an added dot character
3. Will not match filenames where the author's initial is lowercase (and would not be practical to do so without resulting in adding a dot between each lowercase word of a filename). A trade-off.

Result:

Name	New Name
 J Doe.pdf	J. Doe.pdf

Where:

\1 = **null** \2 = J \3 = (space) \4 = D \5 = oe.pdf

Notes:

1. The initial 'J' has been isolated in Group 2 so it needs to have the dot and a space added - \2<dot> <space>

Personal examples

3. Remove Excess Spaces after an Author's Initials

Match file names that have two spaces after an Initial (only) and remove the additional space.

Match: `(.*)((?<=.)\.)((.*))(\s)(\s.*)(.*)`

Replace: `\1\2\6`

Analysis:

1. Gather up the entire string.
2. This uses a Lookbehind for the dot (signifying an initial).
3. Gather up the dot.
4. Search for two spaces after the dot.
5. Gather up the space.
6. If matched, the Replacements put together the string.




Example:

String = This is the front – J. Doe; More stuff here to End.pdf

2 <spaces>



In this example, notice that only the file name with two spaces after an initial is affected. The other file name that also has two spaces is unaffected.

Name	New Name
 This is the front – J. Doe; More stuff here to End.pdf	This is the front – J. Doe; More stuff here to End.pdf
 This file has two spaces.pdf	This file has two spaces.pdf
 This file is fine.pdf	This file is fine.pdf

Where:

\1 = This is the front – J.pdf

\2 = . (dot)

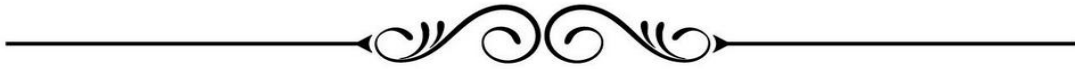
\3 = (null)

\4 = (null)

\5 = (space)

\6 = Doe; More stuff here to End.pdf

\7 = (null)



JavaScript Bulk Rename Utility Constants and Variables

JavaScript BRU Constants and Variables

The following special constants and variables are available in Bulk Rename Utility JavaScript. Variables can be modified, while constants have a fixed value and can not be changed.

Constant / Variable	Type	Explanation
name	Constant	This constant contains the name of the object (i.e. file or folder name) being processed, after all Bulk Rename Utility renaming options 1 to 13 and all specials have been applied. If the Bulk Rename Utility renaming options 1 to 13 and specials did not modify the name, then <i>name</i> will be the same as <i>origName</i> (see below).
newName	Variable	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 <i>newName</i> , then the javascript code has no effect on the name of an object.
origName	Constant	This constant contains the original name of the object before all Bulk Rename Utility renaming options 1 to 13 and all specials.
ext	Constant	This constant contains the extension of the object (i.e. file or folder name) being processed, after all Bulk Rename Utility renaming options 1 to 13 and all specials have been applied. If the Bulk Rename Utility renaming options 1 to 13 and specials did not modify the extension, then <i>ext</i> will be the same as <i>origExt</i> (see below).
newExt	Variable	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 <i>newExt</i> , then the javascript code has no effect on the extension of an object.
origExt	Constant	This constant contains the original extension of the object before all Bulk Rename Utility renaming options 1 to 13 and all specials.
location	Constant	This constant contains the location of the object as specified in <i>Location (13)</i>
newLocation	Variable	This variable contains a modified <i>location (13)</i> for the object. If you do not modify the variable <i>newLocation</i> , then the javascript code has no effect on the location (13) of an object.
subDir	Constant	This constant contains the object sub directory (it matches the <i>subdir</i> column in the file list). It is only applicable if subfolders have been included in Selection (12).
counter	Constant	A counter that is incremented during the renaming operation, starting from 1. To retrieve the Bulk Rename Utility 'auto number' as used in section Numbering (10), use the <i>object</i> function below.

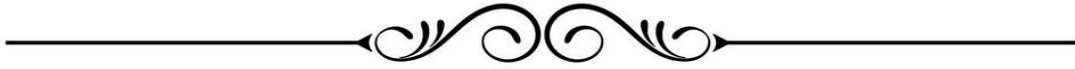
JavaScript BRU Utility Functions

The following special functions are available in Bulk Rename Utility JavaScript:

Function	Explanation	Details / Examples
object("value")	This function returns specific values for the object being processed.	<p>The following <i>values</i> are supported.</p> <p>object("folder") : returns the folder for the current object being processed.</p> <p>object("autonumber") : returns the autonumber for the current object being processed, taken from Numbering (10).</p> <p>object("autodate") : returns the autodate for the current object being processed, taken from Auto Date (8)</p> <p>object("isdir") : returns true if the current object being processed is a directory.</p> <p>object("size") : returns the file size for the current object being processed in bytes.</p> <p>object("modified") : returns the modified timestamp for the current object being processed. (Javascript Date)</p> <p>object("created") : returns the created timestamp for the current object being processed. (Javascript Date)</p> <p>object("accessed") : returns the accessed timestamp for the current object being processed. (Javascript Date)</p>
exif("value")	This function returns EXIF values for the object being processed. See details.	<p>The following values are supported.</p> <p>exif("%d") : Date/Time Taken (Javascript date)</p> <p>exif("taken") : Date/Time Taken, same as %d (Javascript date)</p> <p>exif("%a") : Aperture</p> <p>exif("%c") : Comments</p> <p>exif("%e") : Exposure</p> <p>exif("%f") : Focal Length</p> <p>exif("%xb") : Exposure Bias</p> <p>exif("%ma") : Camera Make</p> <p>exif("%mo") : Camera Model</p>
fileProperty("name") fileProperty("label")	<p>This function returns the Windows File Property value for the object being processed, as a <i>text string</i>, by name or by label. See details.</p> <p>This function also returns the EXIF Properties of an image.</p> <p>If the property is not found, the function will return <i>null</i>.</p>	<p>fileProperty("Height")</p> <p>fileProperty("Dimensions")</p> <p>fileProperty("Total bitrate")</p> <p>fileProperty("System.Media.Duration")</p> <p>fileProperty("Channels")</p> <p>fileProperty("exif:XResolution")</p> <p>fileProperty("exif:ISOSpeedRatings")</p> <p>fileProperty("hash:sha256")</p> <p>fileProperty("hash:md5")</p> <p>etc.</p> <p>You can list all file property labels or names that can be used in Bulk Rename Utility from the context menu of a file in the main file list (right-click) or in Windows Explorer, right-click on a file and select "Properties". Each file type might have different properties available.</p>
filePropertyDate("name") filePropertyDate("label")	<p>This function returns the Windows File Property value for the object being processed, as a <i>date object</i>, by name or by label. See details.</p> <p>This function also returns the EXIF Properties of an image.</p> <p>If the property is not found, the function will return <i>null</i>.</p>	<p>filePropertyDate("System.Document.DateCreated")</p> <p>filePropertyDate("exif:DateTimeOriginal")</p> <p>You can list all file property labels or names that can be used in Bulk Rename Utility from the context menu of a file in the main file list (right-click) or in Windows Explorer, right-click on a file and select "Properties". Each file type might have different properties available.</p>

JavaScript BRU Utility Functions

filePropertyNum ("name") filePropertyNum ("label")	This function returns the Windows File Property value for the object being processed, as a <i>number</i> , by name or by label. See details. This function also returns the EXIF Properties of an image. If the property is not found, the function will return <i>null</i> .	newName = filePropertyNum("Width") + name; newName = filePropertyNum("exif:Orientation") + "-" + name; You can list all file property labels or names that can be used in Bulk Rename Utility from the context menu of a file in the main file list (right-click) or in Windows Explorer, right-click on a file and select "Properties". Each file type might have different properties available.
clipboard()	This function returns the text that is currently in the Windows Clipboard.	newName = clipboard() + " - video";
alert("text")	This function shows a popup message.	alert("object new name is: " + newName);
regEx("text", "match", "replace")	This function processes a regular expression in the same way as it is done in section RegEx (1). Using this function is possible to process multiple regular expressions.	This code will switch a file name around "-". <u>Please note that the character \ must be doubled \\ in Javascript strings.</u> newName = regEx(name, "(.*)-(.*)", "\\2\\1");
setEnv ("varname", "value")	Sets the value of an environment variable	setenv('BRUNAME', name);
getEnv ("varname")	Gets the value of an environment variable	var value = getenv('USERNAME');
include ("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.	include('js/sugar.js'); include('file.js'); include('../afile.js');
require ("filename")	Same as include but if the file is not found, javascript execution is stopped and an error is reported.	require('js/date.js'); require('myfile.js');



EXIF Metadata Reference

EXIF Metadata Reference

This is a complete listing of the 2.3 Standard EXIF Tags. BRU uses the 2.2 standard currently so some tags may not apply.

This is provided for the EXIF Tag capability of both the ‘[Section #7: Add](#)’ and the ‘[Section #14: JavaScript](#)’. You will probably never see half of these, but here they are in case you do. I have already formatted each tag with ‘EXIF:’ so all you need to do is to copy and paste the desired tag into BRU. They are not in alphabetical order but in the order in which they are likely to appear, as per the Exiv2 v0.27.2.

EXIF:ProcessingSoftware

The name and version of the software used to post-process the picture.

EXIF:NewSubfileType

A general indication of the kind of data contained in this subfile.

EXIF:SubfileType

A general indication of the kind of data contained in this subfile. This field is deprecated. The [NewSubfileType](#) field should be used instead.

EXIF:ImageWidth

The number of columns of image data, equal to the number of pixels per row.

EXIF:ImageLength

The number of rows of image data.

EXIF:BitsPerSample

The number of bits per image component. In this standard each component of the image is 8 bits, so the value for this tag is 8.

EXIF:Compression

The compression scheme used for the image data. When a primary image is JPEG compressed, this designation is not necessary and is omitted. When thumbnails use JPEG compression, this tag value is set to 6.

EXIF:PhotometricInterpretation

The pixel composition.

EXIF:Thresholding

For black and white TIFF files that represent shades of gray, the technique used to convert from gray to black and white pixels.

EXIF:CellWidth

The width of the dithering or halftoning matrix used to create a dithered or halftoned bilevel file.

EXIF:CellLength

The length of the dithering or halftoning matrix used to create a dithered or halftoned bilevel file.

EXIF:FillOrder

The logical order of bits within a byte.

EXIF Metadata Reference

EXIF:DocumentName

The name of the document from which this image was scanned.

EXIF:ImageDescription

A character string giving the title of the image.

EXIF:Make

The manufacturer of the equipment that generated the image.

EXIF:Model

The model name or model number of the equipment that generated the image.

EXIF:StripOffsets

For each strip, the offset of that strip.

EXIF:Orientation

The image orientation viewed in terms of rows and columns.

EXIF:SamplesPerPixel

The number of components per pixel. Since this standard applies to RGB and YCbCr images, the value set for this tag is 3.

EXIF:RowsPerStrip

The number of rows per strip. This is the number of rows in the image of one strip when an image is divided into strips.

EXIF:StripByteCounts

The total number of bytes in each strip. With JPEG compressed data this designation is not needed and is omitted.

EXIF:XResolution

The number of pixels per **<ResolutionUnit>** in the **<ImageWidth>** direction. When the image resolution is unknown, 72 [dpi] is designated.

EXIF:YResolution

The number of pixels per **<ResolutionUnit>** in the **<ImageLength>** direction.

EXIF:PlanarConfiguration

Indicates whether pixel components are recorded in a chunky or planar format.

EXIF:GrayResponseUnit

The precision of the information contained in the **GrayResponseCurve**.

EXIF:GrayResponseCurve

For grayscale data, the optical density of each possible pixel value.

EXIF:T4Options

T:4-encoding options.

EXIF Metadata Reference

EXIF:T6Options

T:6-encoding options.

EXIF:ResolutionUnit

The unit for measuring `<XResolution>` and `<YResolution>`. If the image resolution is unknown, 2 (inches) is designated.

EXIF:PageNumber

The page number of the page from which this image was scanned.

EXIF:TransferFunction

A transfer function for the image, described in tabular style.

EXIF:Software

This tag records the name and version of the software or firmware of the camera or image input device used to generate the image.

EXIF:DateTime

The date and time of image creation. In EXIF standard, it is the date and time the file was changed.

EXIF:Artist

This tag records the name of the camera owner, photographer or image creator.

EXIF:HostComputer

This tag records information about the host computer used to generate the image.

EXIF:Predictor

A predictor is a mathematical operator that is applied to the image data before an encoding scheme is applied.

EXIF:WhitePoint

The chromaticity of the white point of the image.

EXIF:PrimaryChromaticities

The chromaticity of the three primary colours of the image.

EXIF:ColorMap

A colour map for palette colour images. This field defines a Red-Green-Blue colour map (often called a lookup table) for palette-colour images.

EXIF:HalftoneHints

The purpose of the HalftoneHints field is to convey to the halftone function the range of gray levels within a colorimetrically-specified image that should retain tonal detail.

EXIF:TileWidth

The tile width in pixels. This is the number of columns in each tile.

EXIF:TileLength

The tile length (height) in pixels. This is the number of rows in each tile.

EXIF Metadata Reference

EXIF:TileOffsets

For each tile, the offset of that tile, as compressed and stored on disk. The offset is specified with respect to the beginning of the TIFF file.

EXIF:TileCounts

For each tile, the number of (compressed) bytes in that tile.

EXIF:SubIFDs

Defined by Adobe Corporation to enable TIFF Trees within a TIFF file.

EXIF:InkSet

The set of inks used in a separated (PhotometricInterpretation=5) image.

EXIF:InkNames

The name of each ink used in a separated (PhotometricInterpretation=5) image.

EXIF:NumberOfInks

The number of inks. Usually equal to **SamplesPerPixel**, unless there are extra samples.

EXIF:DotRange

The component values that correspond to a 0% dot and 100% dot.

EXIF:TargetPrinter

A description of the printing environment for which this separation is intended.

EXIF:ExtraSamples

Specifies that each pixel has *m* extra components whose interpretation is defined by one of the values listed in **SampleFormat**, **SMinSampleValue** and **SMaxSampleValue**.

EXIF:SampleFormat

This field specifies how to interpret each data sample in a pixel.

EXIF:SMinSampleValue

This field specifies the minimum sample value.

EXIF:SMaxSampleValue

This field specifies the maximum sample value.

EXIF:TransferRange

Expands the range of the **TransferFunction**

EXIF:ClipPath

A TIFF ClipPath is intended to mirror the essentials of PostScript's path creation functionality.

EXIF:XClipPathUnits

The number of units that span the width of the image, in terms of integer ClipPath coordinates.

EXIF Metadata Reference

EXIF:YClipPathUnits

The number of units that span the height of the image, in terms of integer ClipPath coordinates.

EXIF:Indexed

Indexed images are images where the 'pixels' do not represent colour values, but rather an index (usually 8-bit) into a separate colour table, the ColorMap.

EXIF:JPEGTables

This optional tag may be used to encode the JPEG quantization and Huffman tables for subsequent use by the JPEG decompression process.

EXIF:OPIProxy

OPIProxy gives information concerning whether this image is a low-resolution proxy of a high-resolution image (Adobe OPI).

EXIF:JPEGProc

This field indicates the process used to produce the compressed data

EXIF:JPEGInterchangeFormat

The offset to the start (SOI) of JPEG compressed thumbnail data.

EXIF:JPEGInterchangeFormatLength

The number of s of JPEG compressed thumbnail data.

EXIF:JPEGRestartInterval

This Field indicates the length of the restart interval used in the compressed image data.

EXIF:JPEGLosslessPredictors

This Field points to a list of lossless predictor-selection values, one per component.

EXIF:JPEGPointTransforms

This Field points to a list of point transform values, one per component.

EXIF:JPEGQTables

This Field points to a list of offsets to the quantization tables, one per component.

EXIF:JPEGDCTables

This Field points to a list of offsets to the DC Huffman tables or the lossless Huffman tables, one per component.

EXIF:JPEGACTables

This Field points to a list of offsets to the Huffman AC tables, one per component.

EXIF:YCbCrCoefficients

The matrix coefficients for transformation from RGB to YCbCr image data.

EXIF:YCbCrSubSampling

The sampling ratio of chrominance components in relation to the luminance component.

EXIF Metadata Reference

EXIF:YCbCrPositioning

The position of chrominance components in relation to the luminance component. This field is designated only for JPEG compressed data or uncompressed YCbCr data.

EXIF:ReferenceBlackWhite

The reference black point value and reference white point value.

EXIF:XMLPacket

XMP Metadata.

EXIF:Rating

Rating tag used by Windows.

EXIF:RatingPercent

Rating tag used by Windows, value in percent.

EXIF:ImageID

ImageID is the full pathname of the original, high-resolution image, or any other identifying string that uniquely identifies the original image (Adobe OPI).

EXIF:CFARepetPatternDim

Contains two values representing the minimum rows and columns to define the repeating patterns of the colour filter array.

EXIF:CFAPattern

Indicates the colour filter array (CFA) geometric pattern of the image sensor when a one-chip colour area sensor is used. It does not apply to all sensing methods.

EXIF:BatteryLevel

Contains a value of the battery level as a fraction or string.

EXIF:Copyright

Copyright information. In this standard the tag is used to indicate both the photographer and editor copyrights.

EXIF:ExposureTime

Exposure time, given in seconds.

EXIF:FNumber

The F number.

EXIF:IPTCNAA

Contains an IPTC/NAA record.

EXIF:ImageResources

Contains information embedded by the Adobe Photoshop application.

EXIF:EXIFTag

A pointer to the EXIF IFD.

EXIF Metadata Reference

EXIF:InterColorProfile

Contains an InterColor Consortium (ICC) format colour space characterization/profile.

EXIF:ExposureProgram

The class of the program used by the camera to set exposure when the picture is taken.

EXIF:SpectralSensitivity

Indicates the spectral sensitivity of each channel of the camera used.

EXIF:GPSTag

A pointer to the GPS Info IFD.

EXIF:ISOSpeedRatings

Indicates the ISO Speed and ISO Latitude of the camera or input device as specified in ISO 12232.

EXIF:OECF

Indicates the Opto-Electric Conversion Function (OECF) specified in ISO 14524.

EXIF:Interlace

Indicates the field number of multiframe images.

EXIF:TimeZoneOffset

This optional tag encodes the time zone of the camera clock (relative to Greenwich Mean Time) used to create the **DataTimeOriginal** tag-value when the picture was taken. It may also contain the time zone offset of the clock used to create the **DateTime** tag-value when the image was modified.

EXIF:SelfTimerMode

Number of seconds image capture was delayed from button press.

EXIF:DateTimeOriginal

The date and time when the original image data was generated.

EXIF:CompressedBitsPerPixel

Specific to compressed data; states the compressed bits per pixel.

EXIF:ShutterSpeedValue

Shutter speed.

EXIF:ApertureValue

The lens aperture.

EXIF:BrightnessValue

The value of brightness.

EXIF:ExposureBiasValue

The exposure bias.

EXIF Metadata Reference

EXIF:MaxApertureValue

The smallest F number of the lens.

EXIF:SubjectDistance

The distance to the subject, given in meters.

EXIF:MeteringMode

The metering mode.

EXIF:LightSource

The kind of light source.

EXIF:Flash

Indicates the status of flash when the image was shot.

EXIF:FocalLength

The actual focal length of the lens, in mm.

EXIF:FlashEnergy

Amount of flash energy (BCPS).

EXIF:SpatialFrequencyResponse

SFR of the camera.

EXIF:Noise

Noise measurement values.

EXIF:FocalPlaneXResolution

Number of pixels per **FocalPlaneResolutionUnit** in **ImageWidth** direction for main image.

EXIF:FocalPlaneYResolution

Number of pixels per **FocalPlaneResolutionUnit** in **ImageLength** direction for main image.

EXIF:FocalPlaneResolutionUnit

Unit of measurement for **FocalPlaneXResolution** and **FocalPlaneYResolution**.

EXIF:ImageNumber

Number assigned to an image, e:g:, in a chained image burst.

EXIF:SecurityClassification

Security classification assigned to the image.

EXIF:ImageHistory

Record of what has been done to the image.

EXIF:SubjectLocation

Indicates the location and area of the main subject in the overall scene.

EXIF Metadata Reference

EXIF:ExposureIndex

Encodes the camera exposure index setting when image was captured.

EXIF:TIFFEPStandardID

Contains four ASCII characters representing the TIFF/EP standard version of a TIFF/EP file, e.g. '1', '0', '0', '0'

EXIF:SensingMethod

Type of image sensor.

EXIF:XPTitle

Title tag used by Windows, encoded in UCS2.

EXIF:XPComment

Comment tag used by Windows, encoded in UCS2.

EXIF:XPAuthor

Author tag used by Windows, encoded in UCS2.

EXIF:XPKeywords

Keywords tag used by Windows, encoded in UCS2.

EXIF:XPSubject

Subject tag used by Windows, encoded in UCS2.

EXIF:PrintImageMatching

Print Image Matching, description needed.

EXIF:DNGVersion

This tag encodes the DNG four-tier version number. For files compliant with version 1:1:0:0 of the DNG specification, this tag should contain the s.1, 1, 0, 0.

EXIF:DNGBackwardVersion

This tag specifies the oldest version of the Digital Negative specification for which a file is compatible.

EXIF:UniqueCameraModel

Defines a unique, non-localized name for the camera model that created the image in the raw file.

EXIF:LocalizedCameraModel

Similar to the **UniqueCameraModel** field, except the name can be localized for different markets to match the localization of the camera name.

EXIF:CFAPlaneColor

Provides a mapping between the values in the **CFAPattern** tag and the plane numbers in **LinearRaw** space. This is a required tag for non-RGB CFA images.

EXIF:CFALayout

Describes the spatial layout of the CFA.

EXIF Metadata Reference

EXIF:LinearizationTable

Describes a lookup table that maps stored values into linear values.

EXIF:BlackLevelRepeatDim

Specifies repeat pattern size for the **BlackLevel** tag.

EXIF:BlackLevel

Specifies the zero light (a:k:a.thermal black or black current) encoding level, as a repeating pattern.

EXIF:BlackLevelDeltaH

If the zero light encoding level is a function of the image column, **BlackLevelDeltaH** specifies the difference between the zero light encoding level for each column and the baseline zero light encoding level.

EXIF:BlackLevelDeltaV

If the zero light encoding level is a function of the image row, this tag specifies the difference between the zero light encoding level for each row and the baseline zero light encoding level.

EXIF:WhiteLevel

This tag specifies the fully saturated encoding level for the raw sample values. Saturation is caused either by the sensor itself becoming highly non-linear in response, or by the camera's analog to digital converter clipping.

EXIF:DefaultScale

DefaultScale is required for cameras with non-square pixels. It specifies the default scale factors for each direction to convert the image to square pixels.

EXIF:DefaultCropOrigin

Raw images often store extra pixels around the edges of the final image. These extra pixels help prevent interpolation artifacts near the edges of the final image. **DefaultCropOrigin** specifies the origin of the final image area, in raw image coordinates (i:e., before the **DefaultScale** has been applied), relative to the top-left corner of the **ActiveArea** rectangle.

EXIF:DefaultCropSize

Raw images often store extra pixels around the edges of the final image. These extra pixels help prevent interpolation artifacts near the edges of the final image. **DefaultCropSize** specifies the size of the final image area, in raw image coordinates (i:e., before the **DefaultScale** has been applied).

EXIF:ColorMatrix1

ColorMatrix1 defines a transformation matrix that converts XYZ values to reference camera native colour space values, under the first calibration illuminant. The matrix values are stored in row scan order. The **ColorMatrix1** tag is required for all non-monochrome DNG files.

EXIF:ColorMatrix2

ColorMatrix2 defines a transformation matrix that converts XYZ values to reference camera native colour space values, under the second calibration illuminant. The matrix values are stored in row scan order.

EXIF Metadata Reference

EXIF:CameraCalibration1

CameraCalibration1 defines a calibration matrix that transforms reference camera native space values to individual camera native space values under the first calibration illuminant. The matrix is stored in row scan order. This matrix is stored separately from the matrix specified by the **ColorMatrix1** tag to allow raw converters to swap in replacement colour matrices based on **UniqueCameraModel** tag, while still taking advantage of any per-individual camera calibration performed by the camera manufacturer.

EXIF:CameraCalibration2

CameraCalibration2 defines a calibration matrix that transforms reference camera native space values to individual camera native space values under the second calibration illuminant. The matrix is stored in row scan order. This matrix is stored separately from the matrix specified by the **ColorMatrix2** tag to allow raw converters to swap in replacement colour matrices based on **UniqueCameraModel** tag, while still taking advantage of any per-individual camera calibration performed by the camera manufacturer.

EXIF:ReductionMatrix1

ReductionMatrix1 defines a dimensionality reduction matrix for use as the first stage in converting colour camera native space values to XYZ values, under the first calibration illuminant. The matrix is stored in row scan order.

EXIF:ReductionMatrix2

ReductionMatrix2 defines a dimensionality reduction matrix for use as the first stage in converting colour camera native space values to XYZ values, under the second calibration illuminant. The matrix is stored in row scan order.

EXIF:AnalogBalance

Normally the stored raw values are not white balanced, since any digital white balancing will reduce the dynamic range of the final image if the user decides to later adjust the white balance; however, if camera hardware is capable of white balancing the colour channels before the signal is digitized, it can improve the dynamic range of the final image. AnalogBalance defines the gain, either analog (recommended) or digital (not recommended) that has been applied the stored raw values.

EXIF:AsShotNeutral

Specifies the selected white balance at time of capture, encoded as the coordinates of a perfectly neutral colour in linear reference space values.

EXIF:AsShotWhiteXY

Specifies the selected white balance at time of capture, encoded as x-y chromaticity coordinates.

EXIF:BaselineExposure

Camera models vary in the trade-off they make between highlight headroom and shadow noise. Some leave a significant amount of highlight headroom during a normal exposure. This allows significant negative exposure compensation to be applied during raw conversion, but also means normal exposures will contain more shadow noise. Other models leave less headroom during normal exposures. This allows for less negative exposure compensation, but results in lower shadow noise for normal exposures. Because of these differences, a raw converter needs to vary the zero point of its exposure compensation control from model to model. BaselineExposure specifies by how much (in EV units) to move the zero point. Positive values result in brighter default results, while negative values result in darker default results.

EXIF Metadata Reference

EXIF:BaselineNoise

Specifies the relative noise level of the camera model at a baseline ISO value of 100, compared to a reference camera model. Since noise levels tend to vary approximately with the square root of the ISO value, a raw converter can use this value, combined with the current ISO, to estimate the relative noise level of the current image.

EXIF:BaselineSharpness

Specifies the relative amount of sharpening required for this camera model, compared to a reference camera model. Camera models vary in the strengths of their anti-aliasing filters. Cameras with weak or no filters require less sharpening than cameras with strong anti-aliasing filters.

EXIF:BayerGreenSplit

Only applies to CFA images using a Bayer pattern filter array. This tag specifies, in arbitrary units, how closely the values of the green pixels in the blue/green rows track the values of the green pixels in the red/green rows. A value of zero means the two kinds of green pixels track closely, while a non-zero value means they sometimes diverge. The useful range for this tag is from 0 (no divergence) to about 5000 (quite large divergence).

EXIF:LinearResponseLimit

Some sensors have an unpredictable non-linearity in their response as they near the upper limit of their encoding range. This non-linearity results in colour shifts in the highlight areas of the resulting image unless the raw converter compensates for this effect. LinearResponseLimit specifies the fraction of the encoding range above which the response may become significantly non-linear.

EXIF:CameraSerialNumber

CameraSerialNumber contains the serial number of the camera or camera body that captured the image.

EXIF:LensInfo

Contains information about the lens that captured the image.

EXIF:ChromaBlurRadius

ChromaBlurRadius provides a hint to the DNG reader about how much chroma blur should be applied to the image.

EXIF:AntiAliasStrength

Provides a hint to the DNG reader about how strong the camera's anti-alias filter is. A value of 0:0 means no anti-alias filter (i.e., the camera is prone to aliasing artifacts with some subjects), while a value of 1:0 means a strong anti-alias filter (i.e., the camera almost never has aliasing artifacts).

EXIF:ShadowScale

This tag is used by Adobe Camera Raw to control the sensitivity of its 'Shadows' slider.

EXIF:DNGPrivateData

Provides a way for camera manufacturers to store private data in the DNG file for use by their own raw converters, and to have that data preserved by programs that edit DNG files.

EXIF Metadata Reference

EXIF:MakerNoteSafety

MakerNoteSafety lets the DNG reader know whether the **EXIF MakerNote** tag is safe to preserve along with the rest of the EXIF data. File browsers and other image management software processing an image with a preserved **MakerNote** should be aware that any thumbnail image embedded in the **MakerNote** may be stale, and may not reflect the current state of the full size image.

EXIF:CalibrationIlluminant1

The illuminant used for first set of colour calibration tags (**ColorMatrix1**, **CameraCalibration1**, **ReductionMatrix1**).

EXIF:CalibrationIlluminant2

The illuminant used for an optional second set of colour calibration tags (**ColorMatrix2**, **CameraCalibration2**, **ReductionMatrix2**).

EXIF:BestQualityScale

For some cameras, the best possible image quality is not achieved by preserving the total pixel count during conversion. For example, Fujifilm SuperCCD images have maximum detail when their total pixel count is doubled. This tag specifies the amount by which the values of the **DefaultScale** tag need to be multiplied to achieve the best quality image size.

EXIF:RawDataUniqueID

This tag contains a 16- unique identifier for the raw image data in the DNG file. DNG readers can use this tag to recognize a particular raw image, even if the file's name or the Metadata contained in the file has been changed.

EXIF:OriginalRawFileName

If the DNG file was converted from a non-DNG raw file, then this tag contains the file name of that original raw file.

EXIF:OriginalRawFileData

If the DNG file was converted from a non-DNG raw file, then this tag contains the compressed contents of that original raw file.

EXIF:ActiveArea

This rectangle defines the active (non-masked) pixels of the sensor. The order of the rectangle coordinates is top, left, bottom, right.

EXIF:MaskedAreas

This tag contains a list of non-overlapping rectangle coordinates of fully masked pixels, which can be optionally used by DNG readers to measure the black encoding level. The order of each rectangle's coordinates is top, left, bottom, right.

EXIF:AsShotICCProfile

This tag contains an ICC profile that, in conjunction with the **AsShotPreProfileMatrix** tag, provides the camera manufacturer with a way to specify a default colour rendering from camera colour space coordinates (linear reference values) into the ICC profile connection space.

EXIF:AsShotPreProfileMatrix

This tag is used in conjunction with the **AsShotICCProfile** tag. It specifies a matrix that should be applied to the camera colour space coordinates before processing the values through the ICC profile specified in the **AsShotICCProfile** tag.

EXIF Metadata Reference

EXIF:CurrentICCPProfile

This tag is used in conjunction with the **CurrentPreProfileMatrix** tag. The **CurrentICCPProfile** and **CurrentPreProfileMatrix** tags have the same purpose and usage as the **AsShotICCPProfile** and **AsShotPreProfileMatrix** tag pair, except they are for use by raw file editors rather than camera manufacturers.

EXIF:CurrentPreProfileMatrix

This tag is used in conjunction with the **CurrentICCPProfile** tag. The **CurrentICCPProfile** and **CurrentPreProfileMatrix** tags have the same purpose and usage as the **AsShotICCPProfile** and **AsShotPreProfileMatrix** tag pair, except they are for use by raw file editors rather than camera manufacturers.

EXIF:ColorimetricReference

The DNG colour model documents a transform between camera colours and CIE XYZ values. This tag describes the colorimetric reference for the CIE XYZ values.

EXIF:CameraCalibrationSignature

A UTF-8 encoded string associated with the **CameraCalibration1** and **CameraCalibration2** tags.

EXIF:ProfileCalibrationSignature

A UTF-8 encoded string associated with the camera profile tags.

EXIF:AsShotProfileName

A UTF-8 encoded string containing the name of the "as shot" camera profile, if any.

EXIF:NoiseReductionApplied

This tag indicates how much noise reduction has been applied to the raw data on a scale of 0:0 to 1:0. A 0:0 value indicates that no noise reduction has been applied. A 1:0 value indicates that the "ideal" amount of noise reduction has been applied. A value of 0/0 indicates that this parameter is unknown.

EXIF:ProfileName

A UTF-8 encoded string containing the name of the camera profile.

EXIF:ProfileHueSatMapDims

This tag specifies the number of input samples in each dimension of the hue/saturation/value mapping tables. The data for these tables are stored in **ProfileHueSatMapData1** and **ProfileHueSatMapData2** tags.

EXIF:ProfileHueSatMapData1

This tag contains the data for the first hue/saturation/value mapping table. Each entry of the table contains three 32-bit IEEE floating-point values. The first entry is hue shift in degrees; the second entry is saturation scale factor; and the third entry is a value scale factor.

EXIF:ProfileHueSatMapData2

This tag contains the data for the second hue/saturation/value mapping table. Each entry of the table contains three 32-bit IEEE floating-point values. The first entry is hue shift in degrees; the second entry is a saturation scale factor; and the third entry is a value scale factor.

EXIF Metadata Reference

EXIF:ProfileToneCurve

This tag contains a default tone curve that can be applied while processing the image as a starting point for user adjustments.

EXIF:ProfileEmbedPolicy

This tag contains information about the usage rules for the associated camera profile.

EXIF:ProfileCopyright

A UTF-8 encoded string containing the copyright information for the camera profile.

EXIF:ForwardMatrix1

This tag defines a matrix that maps white balanced camera colours to XYZ D50 colours.

EXIF:ForwardMatrix2

This tag defines a matrix that maps white balanced camera colours to XYZ D50 colours.

EXIF:PreviewApplicationName

A UTF-8 encoded string containing the name of the application that created the preview stored in the IFD.

EXIF:PreviewApplicationVersion

A UTF-8 encoded string containing the version number of the application that created the preview stored in the IFD.

EXIF:PreviewSettingsName

A UTF-8 encoded string containing the name of the conversion settings (for example, snapshot name) used for the preview stored in the IFD.

EXIF:PreviewSettingsDigest

A unique ID of the conversion settings (for example, MD5 digest) used to render the preview stored in the IFD.

EXIF:PreviewColorSpace

This tag specifies the colour space in which the rendered preview in this IFD is stored. The default value for this tag is sRGB for colour previews and Gray Gamma 2:2 for monochrome previews.

EXIF:PreviewDateTime

This tag is a string containing the name of the date/time at which the preview stored in the IFD was rendered. The date/time is encoded using ISO 8601 format.

EXIF:RawImageDigest

This tag is an MD5 digest of the raw image data.

EXIF:OriginalRawFileDigest

This tag is an MD5 digest of the data stored in the [OriginalRawFileData](#) tag.

EXIF:SubTileBlockSize

Normally, the pixels within a tile are stored in simple row-scan order. This tag specifies that the pixels within a tile should be grouped first into rectangular blocks of the specified size. These blocks are stored in row-scan order. Within each block, the pixels are stored in row-scan order.

EXIF Metadata Reference

EXIF:RowInterleaveFactor

This tag specifies that rows of the image are stored in interleaved order. The value of the tag specifies the number of interleaved fields.

EXIF:ProfileLookTableDims

This tag specifies the number of input samples in each dimension of a default "look" table. The data for this table is stored in the **ProfileLookTableData** tag.

EXIF:ProfileLookTableData

This tag contains a default "look" table that can be applied while processing the image as a starting point for user adjustment.

EXIF:OpcodeList1

Specifies the list of opcodes that should be applied to the raw image, as read directly from the file.

EXIF:OpcodeList2

Specifies the list of opcodes that should be applied to the raw image, just after it has been mapped to linear reference values.

EXIF:OpcodeList3

Specifies the list of opcodes that should be applied to the raw image, just after it has been demosaiced.

EXIF:NoiseProfile

NoiseProfile describes the amount of noise in a raw image. Specifically, this tag models the amount of signal-dependent photon (shot) noise and signal-independent sensor readout noise, two common sources of noise in raw images.

EXIF:TimeCodes

The optional TimeCodes tag shall contain an ordered array of time codes. All time codes shall be 8 bytes and in binary format. The tag may contain from 1 to 10 time codes. When the tag contains more than one time code, the first one shall be the default time code.

EXIF:FrameRate

The optional FrameRate tag shall specify the video frame rate in number of image frames per second, expressed as a signed rational number. The numerator shall be non-negative and the denominator shall be positive.

EXIF:TStop

The optional TStop tag shall specify the T-stop of the actual lens, expressed as an unsigned rational number. T-stop is also known as T-number or the photometric aperture of the lens. (F-number is the geometric aperture of the lens). When the exact value is known, the T-stop shall be specified using a single number. Alternately, two numbers shall be used to indicate a T-stop range, in which case the first number shall be the minimum T-stop and the second number shall be the maximum T-stop.

EXIF:ReelName

The optional ReelName tag shall specify a name for a sequence of images, where each image in the sequence has a unique image identifier (including but not limited to file name, frame number, date time, time code).

EXIF Metadata Reference

EXIF:CameraLabel

The optional CameraLabel tag shall specify a text label for how the camera is used or assigned in this clip.

EXIF:Photo:ExposureTime

Exposure time, given in seconds (sec).

EXIF:Photo:FNumber

The F number.

EXIF:Photo:ExposureProgram

The class of the program used by the camera to set exposure when the picture is taken.

EXIF:Photo:SpectralSensitivity

Indicates the spectral sensitivity of each channel of the camera used.

EXIF:Photo:ISOSpeedRatings

Indicates the ISO Speed and ISO Latitude of the camera or input device as specified in ISO 12232.

EXIF:Photo:OECF

Indicates the Opto-Electronic Conversion Function (OECF) specified in ISO 14524. <OECF> is the relationship between the camera optical input and the image values.

EXIF:Photo:SensitivityType

The SensitivityType tag indicates which one of the parameters of ISO12232 is the **PhotographicSensitivity** tag.

EXIF:Photo:StandardOutputSensitivity

This tag indicates the standard output sensitivity value of a camera or input device defined in ISO 12232.

EXIF:Photo:RecommendedExposureIndex

This tag indicates the recommended exposure index value of a camera or input device defined in ISO 12232.

EXIF:Photo:ISOSpeed

This tag indicates the ISO speed value of a camera or input device that is defined in ISO 12232.

EXIF:Photo:ISOSpeedLatitudeyyy

This tag indicates the ISO speed latitude yyy value of a camera or input device that is defined in ISO 12232

EXIF:Photo:ISOSpeedLatitudezzz

This tag indicates the ISO speed latitude zzz value of a camera or input device that is defined in ISO 12232.

EXIF:Photo:EXIFVersion

The version of this standard supported. Nonexistence of this field is taken to mean nonconformance to the standard.

EXIF:Photo:DateTimeOriginal

The date and time when the original image data was generated. For a digital still camera the date and time the picture was taken are recorded.

EXIF Metadata Reference

EXIF:Photo:DateTimeDigitized

The date and time when the image was stored as digital data.

EXIF:Photo:ComponentsConfiguration

Information specific to compressed data.

EXIF:Photo:CompressedBitsPerPixel

Information specific to compressed data. The compression mode used for a compressed image is indicated in unit bits per pixel.

EXIF:Photo:ShutterSpeedValue

Shutter speed. The unit is the APEX (Additive System of Photographic Exposure) setting.

EXIF:Photo:ApertureValue

The lens aperture. The unit is the APEX value.

EXIF:Photo:BrightnessValue

The value of brightness. The unit is the APEX value.

EXIF:Photo:ExposureBiasValue

The exposure bias. The units is the APEX value

EXIF:Photo:MaxApertureValue

The smallest F number of the lens. The unit is the APEX value.

EXIF:Photo:SubjectDistance

The distance to the subject, given in meters.

EXIF:Photo:MeteringMode

The metering mode.

EXIF:Photo:LightSource

The kind of light source.

EXIF:Photo:Flash

This tag is recorded when an image is taken using a strobe light (flash).

EXIF:Photo:FocalLength

The actual focal length of the lens, in mm. Conversion is not made to the focal length of a 35 mm film camera.

EXIF:Photo:SubjectArea

This tag indicates the location and area of the main subject in the overall scene.

EXIF:Photo:MakerNote

A tag for manufacturers of EXIF writers to record any desired information. The contents are up to the manufacturer.

EXIF Metadata Reference

EXIF:Photo:UserComment

A tag for EXIF users to write keywords or comments on the image besides those in **<ImageDescription>**, and without the character code limitations of the **<ImageDescription>** tag.

EXIF:Photo:SubSecTime

A tag used to record fractions of seconds for the **<DateTime>** tag.

EXIF:Photo:SubSecTimeOriginal

A tag used to record fractions of seconds for the **<DateTimeOriginal>** tag.

EXIF:Photo:SubSecTimeDigitized

A tag used to record fractions of seconds for the **<DateTimeDigitized>** tag.

EXIF:Photo:FlashpixVersion

The FlashPix format version supported by a FPXR file.

EXIF:Photo:ColorSpace

The colour space information tag is always recorded as the colour space specifier. Normally sRGB is used to define the colour space based on the PC monitor conditions and environment. If a colour space other than sRGB is used, Uncalibrated is set.

EXIF:Photo:PixelXDimension

Information specific to compressed data. When a compressed file is recorded, the valid width of the meaningful image must be recorded in this tag, whether or not there is padding data or a restart marker.

EXIF:Photo:PixelYDimension

Information specific to compressed data. When a compressed file is recorded, the valid height of the meaningful image must be recorded in this tag, whether or not there is padding data or a restart marker.

EXIF:Photo:RelatedSoundFile

This tag is used to record the name of an audio file related to the image data. The only relational information recorded here is the EXIF audio file name and extension (a string consisting of 8 characters + '.' + 3 characters). The path is not recorded.

EXIF:Photo:InteroperabilityTag

Interoperability IFD is composed of tags which stores the information to ensure the Interoperability and pointed by the following tag located in EXIF IFD.

EXIF:Photo:FlashEnergy

Indicates the strobe energy at the time the image is captured, as measured in Beam Candle Power Seconds (BCPS).

EXIF:Photo:SpatialFrequencyResponse

This tag records the camera or input device spatial frequency table and SFR values in the direction of image width, image height, and diagonal direction, as specified in ISO 12233.

EXIF:Photo:FocalPlaneXResolution

Indicates the number of pixels in the image width (X) direction per **<FocalPlaneResolutionUnit>** on the camera focal plane.

EXIF Metadata Reference

EXIF:Photo:FocalPlaneYResolution

Indicates the number of pixels in the image height (V) direction per <FocalPlaneResolutionUnit> on the camera focal plane.

EXIF:Photo:FocalPlaneResolutionUnit

Indicates the unit for measuring <FocalPlaneXResolution> and <FocalPlaneYResolution>. This value is the same as the <ResolutionUnit>.

EXIF:Photo:SubjectLocation

Indicates the location of the main subject in the scene. The value of this tag represents the pixel at the center of the main subject relative to the left edge, prior to rotation processing as per the <Rotation> tag. The first value indicates the X column number and second indicates the Y row number.

EXIF:Photo:ExposureIndex

Indicates the exposure index selected on the camera or input device at the time the image is captured.

EXIF:Photo:SensingMethod

Indicates the image sensor type on the camera or input device.

EXIF:Photo:FileSource

Indicates the image source.

EXIF:Photo:SceneType

Indicates the type of scene.

EXIF:Photo:CFAPattern

Indicates the colour filter array (CFA) geometric pattern of the image sensor when a one-chip colour area sensor is used. It does not apply to all sensing methods.

EXIF:Photo:CustomRendered

This tag indicates the use of special processing on image data, such as rendering geared to output.

EXIF:Photo:ExposureMode

This tag indicates the exposure mode set when the image was shot. In auto-bracketing mode, the camera shoots a series of frames of the same scene at different exposure settings.

EXIF:Photo:WhiteBalance

This tag indicates the white balance mode set when the image was shot.

EXIF:Photo:DigitalZoomRatio

This tag indicates the digital zoom ratio when the image was shot. If the numerator of the recorded value is 0, this indicates that digital zoom was not used.

EXIF:Photo:FocalLengthIn35mmFilm

This tag indicates the equivalent focal length assuming a 35mm film camera, in mm. A value of 0 means the focal length is unknown. Note that this tag differs from the <FocalLength> tag.

EXIF Metadata Reference

EXIF:Photo:SceneCaptureType

This tag indicates the type of scene that was shot. It can also be used to record the mode in which the image was shot. Note that this differs from the <SceneType> tag.

EXIF:Photo:GainControl

This tag indicates the degree of overall image gain adjustment.

EXIF:Photo:Contrast

This tag indicates the direction of contrast processing applied by the camera when the image was shot.

EXIF:Photo:Saturation

This tag indicates the direction of saturation processing applied by the camera when the image was shot.

EXIF:Photo:Sharpness

This tag indicates the direction of sharpness processing applied by the camera when the image was shot.

EXIF:Photo:DeviceSettingDescription

This tag indicates information on the picture-taking conditions of a particular camera model.

EXIF:Photo:SubjectDistanceRange

This tag indicates the distance to the subject.

EXIF:Photo:ImageUniqueID

This tag indicates an identifier assigned uniquely to each image. It is recorded as a string equivalent to hexadecimal notation and 128-bit fixed length.

EXIF:Photo:CameraOwnerName

This tag records the owner of a camera used in photography as an ASCII string.

EXIF:Photo:BodySerialNumber

This tag records the serial number of the body of the camera that was used in photography as an ASCII string.

EXIF:Photo:LensSpecification

This tag notes minimum focal length, maximum focal length, minimum F number in the minimum focal length, and minimum F number in the maximum focal length, which are specification information for the lens that was used in photography. When the minimum F number is unknown, the notation is 0/0

EXIF:Photo:LensMake

This tag records the lens manufacturer as an ASCII string.

EXIF:Photo:LensModel

This tag records the lens's model name and model number as an ASCII string.

EXIF:Photo:LensSerialNumber

This tag records the serial number of the interchangeable lens that was used in photography as an ASCII string.

EXIF:Iop:InteroperabilityIndex

Indicates the identification of the Interoperability rule

EXIF Metadata Reference

EXIF:Jop:InteroperabilityVersion

Interoperability version.

EXIF:Jop:RelatedImageFileFormat

File format of image file.

EXIF:Jop:RelatedImageWidth

Image width.

EXIF:Jop:RelatedImageLength

Image height.

EXIF:GPSInfo:GPSVersionID

Indicates the version of **<GPSInfoIFD>**.

EXIF:GPSInfo:GPSLatitudeRef

Indicates whether the latitude is north or south latitude. The ASCII value 'N' indicates north latitude, and 'S' is south latitude.

EXIF:GPSInfo:GPSLatitude

Indicates the latitude. The latitude is expressed as three RATIONAL values giving the degrees, minutes, and seconds, respectively. When degrees, minutes and seconds are expressed, the format is dd/1,mm/1,ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format is dd/1,mmmm/100,0/1.

EXIF:GPSInfo:GPSLongitudeRef

Indicates whether the longitude is east or west longitude. ASCII 'E' indicates east longitude, and 'W' is west longitude.

EXIF:GPSInfo:GPSLongitude

Indicates the Longitude. The Longitude is expressed as three RATIONAL values giving the degrees, minutes, and seconds, respectively. When degrees, minutes and seconds are expressed, the format is ddd/1,mm/1,ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format is ddd/1,mmmm/100,0/1.

EXIF:GPSInfo:GPSAltitudeRef

Indicates the altitude used as the reference altitude. If the reference is sea level and the altitude is above sea level, 0 is given. If the altitude is below sea level, a value of 1 is given and the altitude is indicated as an absolute value in the **GPSAltitude** tag. The reference unit is meters. Note that this tag is BYTE type, unlike other reference tags.

EXIF:GPSInfo:GPSAltitude

Indicates the altitude based on the reference in **GPSAltitudeRef.Altitude** is expressed as one RATIONAL value. The reference unit is meters.

EXIF:GPSInfo:GPSTimeStamp

Indicates the time as UTC (Coordinated Universal Time). **<TimeStamp>** is expressed as three RATIONAL values giving the hour, minute, and second (atomic clock).

EXIF Metadata Reference

EXIF:GPSInfo:GPSSatellites

Indicates the GPS satellites used for measurements. This tag can be used to describe the number of satellites, their ID number, angle of elevation, azimuth, SNR and other information in ASCII notation. The format is not specified. If the GPS receiver is incapable of taking measurements, value of the tag is set to **NULL**.

EXIF:GPSInfo:GPSStatus

Indicates the status of the GPS receiver when the image is recorded. "A" means measurement is in progress, and "V" means the measurement is Interoperability.

EXIF:GPSInfo:GPSMeasureMode

Indicates the GPS measurement mode. "2" means two-dimensional measurement and "3" means three-dimensional measurement is in progress.

EXIF:GPSInfo:GPSDOP

Indicates the GPS DOP (data degree of precision). An HDOP value is written during two-dimensional measurement, and PDOP during three-dimensional measurement.

EXIF:GPSInfo:GPSSpeedRef

Indicates the unit used to express the GPS receiver speed of movement. "K" "M" and "N" represents kilometers per hour, miles per hour, and knots.

EXIF:GPSInfo:GPSSpeed

Indicates the speed of GPS receiver movement.

EXIF:GPSInfo:GPSTrackRef

Indicates the reference for giving the direction of GPS receiver movement. "T" denotes true direction and "M" is magnetic direction.

EXIF:GPSInfo:GPSTrack

Indicates the direction of GPS receiver movement.

EXIF:GPSInfo:GPSImgDirectionRef

Indicates the reference for giving the direction of the image when it is captured. "T" denotes true direction and "M" is magnetic direction.

EXIF:GPSInfo:GPSImgDirection

Indicates the direction of the image when it was captured.

EXIF:GPSInfo:GPSMapDatum

Indicates the geodetic survey data used by the GPS receiver. If the survey data is restricted to Japan, the value of this tag is "TOKYO" or "WGS-84".

EXIF:GPSInfo:GPSDestLatitudeRef

Indicates whether the latitude of the destination point is north or south latitude. The ASCII value "N" indicates north latitude, and "S" is south latitude.

EXIF Metadata Reference

EXIF:GPSInfo:GPSDestLatitude

Indicates the latitude of the destination point. The latitude is expressed as three RATIONAL values giving the degrees, minutes, and seconds, respectively. If latitude is expressed as degrees, minutes and seconds, a typical format would be dd/1,mm/1,ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format would be dd/1,mmmm/100,0/1.

EXIF:GPSInfo:GPSDestLongitudeRef

Indicates whether the longitude of the destination point is east or west longitude. ASCII "E" indicates east longitude, and "W" is west longitude.

EXIF:GPSInfo:GPSDestLongitude

Indicates the longitude of the destination point. The longitude is expressed as three RELATIVE values giving the degrees, minutes, and seconds, respectively. If longitude is expressed as degrees, minutes and seconds, a typical format would be ddd/1,mm/1,ss/1. When degrees and minutes are used and, for example, fractions of minutes are given up to two decimal places, the format would be ddd/1,mmmm/100,0/1.

EXIF:GPSInfo:GPSDestBearingRef

Indicates the reference used for giving the bearing to the destination point. ASCII "T" denotes true direction and "M" is magnetic direction.

EXIF:GPSInfo:GPSDestBearing

Indicates the bearing to the destination point.

EXIF:GPSInfo:GPSDestDistanceRef

Indicates the unit used to express the distance to the destination point. ASCII "K", "M" and "N" represent kilometers, miles and knots.

EXIF:GPSInfo:GPSDestDistance

Indicates the distance to the destination point.

EXIF:GPSInfo:GPSProcessingMethod

A character string recording the name of the method used for location finding. The first byte indicates the character code used, and this is followed by the name of the method.

EXIF:GPSInfo:GPSAreaInformation

A character string recording the name of the GPS area. The first byte indicates the character code used, and this is followed by the name of the GPS area.

EXIF:GPSInfo:GPSDateStamp

A character string recording date and time information relative to UTC (Coordinated Universal Time).The format is "YYYY:MM:DD:".

EXIF:GPSInfo:GPSDifferential

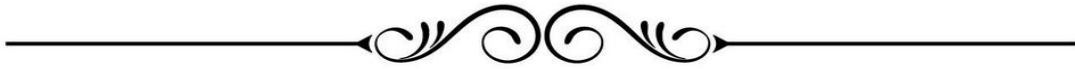
Indicates whether differential correction is applied to the GPS receiver.

EXIF Metadata Reference

Note:

DNG Image is a Adobe Digital Negative Raw Image file.

Source Material this section: exiv2.org



ASCII Character Codes

ASCII Character Chart (Version 1)

I have provided two different versions

Here are the ASCII tables for reference values 0- 127:

ASCII Character Codes Chart 1

Visual Studio 2005 | [Other Versions](#) | 37 out of 55 rated this helpful - [Rate this topic](#)

Ctrl	Dec	Hex	Char	Code	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
^@	0	00		NUL	32	20	!	64	40	@	96	60	'
^A	1	01		SOH	33	21	!	65	41	À	97	61	a
^B	2	02		STX	34	22	!"	66	42	Á	98	62	b
^C	3	03		ETX	35	23	!"#\$	67	43	Â	99	63	c
^D	4	04		EOT	36	24	!"#\$%	68	44	Ã	100	64	d
^E	5	05		ENQ	37	25	!"#\$%&	69	45	Ä	101	65	e
^F	6	06		ACK	38	26	!"#\$%&'	70	46	Å	102	66	f
^G	7	07		BEL	39	27	!"#\$%&'(71	47	Ä	103	67	g
^H	8	08		BS	40	28	!"#\$%&'()	72	48	Å	104	68	h
^I	9	09		HT	41	29	!"#\$%&'() *	73	49	Ä	105	69	i
^J	10	0A		LF	42	2A	!"#\$%&'() * +	74	4A	Ä	106	6A	j
^K	11	0B		VT	43	2B	!"#\$%&'() * + ,	75	4B	Ä	107	6B	k
^L	12	0C		FF	44	2C	!"#\$%&'() * + , -	76	4C	Ä	108	6C	l
^M	13	0D		CR	45	2D	!"#\$%&'() * + , - .	77	4D	Ä	109	6D	m
^N	14	0E		SO	46	2E	!"#\$%&'() * + , - . /	78	4E	Ä	110	6E	n
^O	15	0F		SI	47	2F	!"#\$%&'() * + , - . / 0	79	4F	Ä	111	6F	o
^P	16	10		DLE	48	30	!"#\$%&'() * + , - . / 0 1	80	50	Ä	112	70	p
^Q	17	11		DC1	49	31	!"#\$%&'() * + , - . / 0 1 2	81	51	Ä	113	71	q
^R	18	12		DC2	50	32	!"#\$%&'() * + , - . / 0 1 2 3	82	52	Ä	114	72	r
^S	19	13		DC3	51	33	!"#\$%&'() * + , - . / 0 1 2 3 4	83	53	Ä	115	73	s
^T	20	14		DC4	52	34	!"#\$%&'() * + , - . / 0 1 2 3 4 5	84	54	Ä	116	74	t
^U	21	15		NAK	53	35	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6	85	55	Ä	117	75	u
^V	22	16		SYN	54	36	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7	86	56	Ä	118	76	v
^W	23	17		ETB	55	37	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8	87	57	Ä	119	77	w
^X	24	18		CAN	56	38	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9	88	58	Ä	120	78	x
^Y	25	19		EM	57	39	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 :	89	59	Ä	121	79	y
^Z	26	1A		SUB	58	3A	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 : ;	90	5A	Ä	122	7A	z
^[27	1B		ESC	59	3B	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; :	91	5B	Ä	123	7B	{
^\	28	1C		FS	60	3C	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; : <	92	5C	Ä	124	7C	
^]	29	1D		GS	61	3D	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; : < =	93	5D	Ä	125	7D	}
^^	30	1E	▲	RS	62	3E	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; : < = >	94	5E	Ä	126	7E	~
^-	31	1F	▼	US	63	3F	!"#\$%&'() * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; : < = > ?	95	5F	Ä	127	7F	ÿ

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Notes:

1. Chart 1 is referred to as standard ASCII.
2. With the exception of a few characters, most can be entered directly using a standard keyboard. For the others use ALT + <decimal equivalent> from the numeric keypad to display the character.

For example, to enter ÿ (Dec 127)

Alt + 127

3. Some characters can not be displayed. They have no visual identity. i.e. values 0-29.

ASCII Character Chart (Version 1)

Here are the ASCII tables for reference values 128 - 255:

ASCII Character Codes Chart 2

Visual Studio 2005 | [Other Versions](#) ▾ | 19 out of 26 rated this helpful - [Rate this topic](#)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	à	192	C0	Ł	224	E0	α
129	81	È	161	A1	á	193	C1	ł	225	E1	Β
130	82	É	162	A2	â	194	C2	Ŧ	226	E2	Γ
131	83	Ê	163	A3	ã	195	C3	†	227	E3	Π
132	84	Ë	164	A4	ä	196	C4	—	228	E4	Σ
133	85	Ì	165	A5	å	197	C5	‡	229	E5	ϐ
134	86	Í	166	A6	æ	198	C6	ƒ	230	E6	ϒ
135	87	Î	167	A7	ø	199	C7	ƒ	231	E7	Υ
136	88	Ï	168	A8	ù	200	C8	ƒ	232	E8	Ϙ
137	89	Ñ	169	A9	ú	201	C9	ƒ	233	E9	ϙ
138	8A	Ò	170	AA	û	202	CA	ƒ	234	EA	Ϛ
139	8B	Ó	171	AB	¼	203	CB	ƒ	235	EB	δ
140	8C	Ô	172	AC	½	204	CC	ƒ	236	EC	Ϙ
141	8D	Õ	173	AD	¾	205	CD	=	237	ED	ϙ
142	8E	Ö	174	AE	«	206	CE	ƒ	238	EE	ϛ
143	8F	×	175	AF	»	207	CF	ƒ	239	EF	Ϝ
144	90	À	176	B0	⋮	208	D0	ƒ	240	F0	≡
145	91	Á	177	B1	⋮	209	D1	ƒ	241	F1	±
146	92	Â	178	B2	⋮	210	D2	ƒ	242	F2	∞
147	93	Ã	179	B3	⋮	211	D3	ƒ	243	F3	∞
148	94	Ä	180	B4	⋮	212	D4	ƒ	244	F4	∞
149	95	Å	181	B5	⋮	213	D5	ƒ	245	F5	∞
150	96	Æ	182	B6	⋮	214	D6	ƒ	246	F6	∞
151	97	Ç	183	B7	⋮	215	D7	ƒ	247	F7	∞
152	98	Ɔ	184	B8	⋮	216	D8	ƒ	248	F8	∞
153	99	Ɔ	185	B9	⋮	217	D9	ƒ	249	F9	∞
154	9A	Ɔ	186	BA	⋮	218	DA	ƒ	250	FA	∞
155	9B	Ɔ	187	BB	⋮	219	DB	■	251	FB	∞
156	9C	Ɔ	188	BC	⋮	220	DC	■	252	FC	∞
157	9D	Ɔ	189	BD	⋮	221	DD	■	253	FD	∞
158	9E	Ɔ	190	BE	⋮	222	DE	■	254	FE	∞
159	9F	Ɔ	191	BF	⋮	223	DF	■	255	FF	∞

Notes:

1. Chart 2 is called extended or higher ASCII.
2. To enter these values you use the Numeric keypad and enter ALT + <decimal equivalent>.

For example, to enter the character 'Ç',

Alt + 128

3. The value for 255 can not be displayed. It has no visual identity.

ASCII Character Chart (Version 2)

(larger version)

Page 1

Character Name	Char	Code	Decimal	Binary	Hex
Null	NUL	Ctrl @	0	00000000	00
Start of Heading	SOH	Ctrl A	1	00000001	01
Start of Text	STX	Ctrl B	2	00000010	02
End of Text	ETX	Ctrl C	3	00000011	03
End of Transmit	EOT	Ctrl D	4	00000100	04
Enquiry	ENQ	Ctrl E	5	00000101	05
Acknowledge	ACK	Ctrl F	6	00000110	06
Bell	BEL	Ctrl G	7	00000111	07
Back Space	BS	Ctrl H	8	00001000	08
Horizontal Tab	TAB	Ctrl I	9	00001001	09
Line Feed	LF	Ctrl J	10	00001010	0A
Vertical Tab	VT	Ctrl K	11	00001011	0B
Form Feed	FF	Ctrl L	12	00001100	0C
Carriage Return	CR	Ctrl M	13	00001101	0D
Shift Out	SO	Ctrl N	14	00001110	0E

ASCII Character Chart (Version 2)

(larger version)


Page 2

Character Name	Char	Code	Decimal	Binary	Hex
Shift In	SI	Ctrl O	15	00001111	0F
Data Line Escape	DLE	Ctrl P	16	00010000	10
Device Control 1	DC1	Ctrl Q	17	00010001	11
Device Control 2	DC2	Ctrl R	18	00010010	12
Device Control 3	DC3	Ctrl S	19	00010011	13
Device Control 4	DC4	Ctrl T	20	00010100	14
Negative Acknowledge	NAK	Ctrl U	21	00010101	15
Synchronous Idle	SYN	Ctrl V	22	00010110	16
Cancel	CAN	Ctrl X	24	00011000	18
End of Medium	EM	Ctrl Y	25	00011001	19
Substitute	SUB	Ctrl Z	26	00011010	1A
Escape	ESC	Ctrl [27	00011011	1B
File Separator	FS	Ctrl \	28	00011100	1C
Group Separator	GS	Ctrl]	29	00011101	1D
Record Separator	RS	Ctrl ^	30	00011110	1E
Unit Separator	US	Ctrl _	31	00011111	1F

ASCII Character Chart (Version 2)

(larger version)

Page 3

Character Name	Char	Code	Decimal	Binary	Hex
Space			32	00100000	20
Exclamation Point	!	Shift 1	33	00100001	21
Double Quote	"	Shift `	34	00100010	22
Pound/Number Sign	#	Shift 3	35	00100011	23
Dollar Sign	\$	Shift 4	36	00100100	24
Percent Sign	%	Shift 5	37	00100101	25
Ampersand	&	Shift 7	38	00100110	26
Single Quote	'	'	39	00100111	27
Left Parenthesis	(Shift 9	40	00101000	28
Right Parenthesis)	Shift 0	41	00101001	29
Asterisk	*	Shift 8	42	00101010	2A
Plus Sign	 +	Shift =	43	00101011	2B
Comma	,	,	44	00101100	2C
Hyphen / Minus Sign	-	-	45	00101101	2D

ASCII Character Chart (Version 2)

(larger version)

Page 4

Character Name	Char	Code	Decimal	Binary	Hex
Period	.	.	46	00101110	2E
Forward Slash	/	/	47	00101111	2F
Zero Digit	0	0	48	00110000	30
One Digit	1	1	49	00110001	31
Two Digit	2	2	50	00110010	32
Three Digit	3	3	51	00110011	33
Four Digit	4	4	52	00110100	34
Five Digit	5	5	53	00110101	35
Six Digit	6	6	54	00110110	36
Seven Digit	7	7	55	00110111	37
Eight Digit	8	8	56	00111000	38
Nine Digit	9	9	57	00111001	39
Colon	:	Shift ;	58	00111010	3A

ASCII Character Chart (Version 2)

(larger version)

Page 5

Character Name	Char	Code	Decimal	Binary	Hex
Semicolon	;	;	59	00111011	3B
Less-Than Sign	<	Shift ,	60	00111100	3C
Equals Sign	=	=	61	00111101	3D
Greater-Than Sign	>	Shift .	62	00111110	3E
Question Mark	?	Shift /	63	00111111	3F
At Sign	@	Shift 2	64	01000000	40
Capital A	A	Shift A	65	01000001	41
Capital B	B	Shift B	66	01000010	42
Capital C	C	Shift C	67	01000011	43
Capital D	D	Shift D	68	01000100	44
Capital E	E	Shift E	69	01000101	45
Capital F	F	Shift F	70	01000110	46
Capital G	G	Shift G	71	01000111	47
Capital H	H	Shift H	72	01001000	48
Capital I	I	Shift I	73	01001001	49

ASCII Character Chart (Version 2)

(larger version)

Page 6

Character Name	Char	Code	Decimal	Binary	Hex
Capital J	J	Shift J	74	01001010	4A
Capital K	K	Shift K	75	01001011	4B
Capital L	L	Shift L	76	01001100	4C
Capital M	M	Shift M	77	01001101	4D
Capital N	N	Shift N	78	01001110	4E
Capital O	O	Shift O	79	01001111	4F
Capital P	P	Shift P	80	01010000	50
Capital Q	Q	Shift Q	81	01010001	51
Capital R	R	Shift R	82	01010010	52
Capital S	S	Shift S	83	01010011	53
Capital T	T	Shift T	84	01010100	54
Capital U	U	Shift U	85	01010101	55
Capital V	V	Shift V	86	01010110	56

ASCII Character Chart (Version 2)

(larger version)

Page 7

Character Name	Char	Code	Decimal	Binary	Hex
Capital W	W	Shift W	87	01010111	57
Capital X	X	Shift X	88	01011000	58
Capital Y	Y	Shift Y	89	01011001	59
Capital Z	Z	Shift Z	90	01011010	5A
Left Bracket	[[91	01011011	5B
Backward Slash	\	\	92	01011100	5C
Right Bracket]]	93	01011101	5D
Caret	^	Shift 6	94	01011110	5E
Underscore	_	Shift -	95	01011111	5F
Back Quote	`	`	96	01100000	60
Lower-case A	a	A	97	01100001	61
Lower-case B	b	B	98	01100010	62

ASCII Character Chart (Version 2)

(larger version)

Page 8

Character Name	Char	Code	Decimal	Binary	Hex
Lower-case C	c	C	99	01100011	63
Lower-case D	d	D	100	01100100	64
Lower-case E	e	E	101	01100101	65
Lower-case F	f	F	102	01100110	66
Lower-case G	g	G	103	01100111	67
Lower-case H	h	H	104	01101000	68
Lower-case I	I	I	105	01101001	69
Lower-case J	j	J	106	01101010	6A
Lower-case K	k	K	107	01101011	6B
Lower-case L	l	L	108	01101100	6C
Lower-case M	m	M	109	01101101	6D
Lower-case N	n	N	110	01101110	6E
Lower-case O	o	O	111	01101111	6F

ASCII Character Chart (Version 2)

(larger version)

Page 9

Character Name	Char	Code	Decimal	Binary	Hex
Lower-case P	p	P	112	01110000	70
Lower-case Q	q	Q	113	01110001	71
Lower-case R	r	R	114	01110010	72
Lower-case S	s	S	115	01110011	73
Lower-case T	t	T	116	01110100	74
Lower-case U	u	U	117	01110101	75
Lower-case V	v	V	118	01110110	76
Lower-case W	w	W	119	01110111	77
Lower-case X	x	X	120	01111000	78
Lower-case Y	y	Y	121	01111001	79
Lower-case Z	z	Z	122	01111010	7A
Left Brace	{	Shift [123	01111011	7B
Vertical Bar		Shift \	124	01111100	7C
Right Brace	}	Shift]	125	01111101	7D
Tilde	~	Shift `	126	01111110	7E
Delta	Δ		127	01111111	7F

Extended ASCII Character Chart

ALT+0128	€
ALT+0129	•
ALT+0130	,
ALT+0131	f
ALT+0132	„
ALT+0133	...
ALT+0134	†
ALT+0135	‡
ALT+0136	^
ALT+0137	‰
ALT+0138	Š
ALT+0139	<
ALT+0140	Œ
ALT+0141	•
ALT+0142	Ž
ALT+0143	•
ALT+0144	•
ALT+0145	‘
ALT+0146	'
ALT+0147	"
ALT+0148	"
ALT+0149	•
ALT+0150	—
ALT+0151	—
ALT+0152	~
ALT+0153	™
ALT+0154	š
ALT+0155	›
ALT+0156	œ
ALT+0157	•
ALT+0158	ž
ALT+0159	ÿ

ALT+0160	
ALT+0161	ı
ALT+0162	ç
ALT+0163	£
ALT+0164	¤
ALT+0165	¥
ALT+0166	ı
ALT+0167	§
ALT+0168	¨
ALT+0169	©
ALT+0170	ª
ALT+0171	«
ALT+0172	¬
ALT+0173	
ALT+0174	®
ALT+0175	¯
ALT+0176	°
ALT+0177	±
ALT+0178	²
ALT+0179	³
ALT+0180	´
ALT+0181	µ
ALT+0182	¶
ALT+0183	·
ALT+0184	,
ALT+0185	¹
ALT+0186	º
ALT+0187	»
ALT+0188	¼
ALT+0189	½
ALT+0190	¾
ALT+0191	¿

ALT+0192	À
ALT+0193	Á
ALT+0194	Â
ALT+0195	Ã
ALT+0196	Ä
ALT+0197	Å
ALT+0198	Æ
ALT+0199	Ç
ALT+0200	È
ALT+0201	É
ALT+0202	Ê
ALT+0203	Ë
ALT+0204	Ì
ALT+0205	Í
ALT+0206	Î
ALT+0207	Ï
ALT+0208	Ð
ALT+0209	Ñ
ALT+0210	Ò
ALT+0211	Ó
ALT+0212	Ô
ALT+0213	Õ
ALT+0214	Ö
ALT+0215	×
ALT+0216	Ø
ALT+0217	Ù
ALT+0218	Ú
ALT+0219	Û
ALT+0220	Ü
ALT+0221	Ý
ALT+0222	Þ
ALT+0223	ß

ALT+0224	à
ALT+0225	á
ALT+0226	â
ALT+0227	ã
ALT+0228	ä
ALT+0229	å
ALT+0230	æ
ALT+0231	ç
ALT+0232	è
ALT+0233	é
ALT+0234	ê
ALT+0235	ë
ALT+0236	ì
ALT+0237	í
ALT+0238	î
ALT+0239	ï
ALT+0240	ð
ALT+0241	ñ
ALT+0242	ò
ALT+0243	ó
ALT+0244	ô
ALT+0245	õ
ALT+0246	ö
ALT+0247	÷
ALT+0248	ø
ALT+0249	ù
ALT+0250	ú
ALT+0251	û
ALT+0252	ü
ALT+0253	ý
ALT+0254	þ
ALT+0255	ÿ

PC Extended ASCII Character Chart

ALT+0		ALT+36	\$	ALT+72	H	ALT+108	l
ALT+1	☺	ALT+37	%	ALT+73	I	ALT+109	m
ALT+2	☹	ALT+38	&	ALT+74	J	ALT+110	n
ALT+3	♥	ALT+39	'	ALT+75	K	ALT+111	o
ALT+4	♦	ALT+40	(ALT+76	L	ALT+112	p
ALT+5	♣	ALT+41)	ALT+77	M	ALT+113	q
ALT+6	♠	ALT+42	*	ALT+78	N	ALT+114	r
ALT+7		ALT+43	+	ALT+79	O	ALT+115	s
ALT+8		ALT+44	,	ALT+80	P	ALT+116	t
ALT+9		ALT+45	-	ALT+81	Q	ALT+117	u
ALT+10		ALT+46	.	ALT+82	R	ALT+118	v
ALT+11	♂	ALT+47	/	ALT+83	S	ALT+119	w
ALT+12	♀	ALT+48	0	ALT+84	T	ALT+120	x
T+13		ALT+49	1	ALT+85	U	ALT+121	y
ALT+14	♪	ALT+50	2	ALT+86	V	ALT+122	z
ALT+15	☼	ALT+51	3	ALT+87	W	ALT+123	{
ALT+16	▶	ALT+52	4	ALT+88	X	ALT+124	
ALT+17	◀	ALT+53	5	ALT+89	Y	ALT+125	}
ALT+18	↕	ALT+54	6	ALT+90	Z	ALT+126	~
ALT+19	!!	ALT+55	7	ALT+91	[ALT+127	△
ALT+20	¶	ALT+56	8	ALT+92	\		
ALT+21	§	ALT+57	9	ALT+93]		
ALT+22	—	ALT+58	:	ALT+94	^		
ALT+23	↕	ALT+59	;	ALT+95	_		
ALT+24	↑	ALT+60	<	ALT+96	`		
ALT+25	↓	ALT+61	=	ALT+97	a		
ALT+26	→	ALT+62	>	ALT+98	b		
ALT+27	←	ALT+63	?	ALT+99	c		
ALT+28	L	ALT+64	@	ALT+100	d		
ALT+29	↔	ALT+65	A	ALT+101	e		
ALT+30	▲	ALT+66	B	ALT+102	f		
ALT+31	▼	ALT+67	C	ALT+103	g		
ALT+32		ALT+68	D	ALT+104	h		
ALT+33	!	ALT+69	E	ALT+105	i		
ALT+34	"	ALT+70	F	ALT+106	j		
ALT+35	#	ALT+71	G	ALT+107	k		

PC Extended ASCII Character Chart

Page 2

ALT+128	Ç
ALT+129	ü
ALT+130	é
ALT+131	â
ALT+132	ä
ALT+133	à
ALT+134	å
ALT+135	ç
ALT+136	ê
ALT+137	ë
ALT+138	è
ALT+139	ï
ALT+140	î
ALT+141	ì
ALT+142	Ä
ALT+143	Å
ALT+144	É
ALT+145	æ
ALT+146	Æ
ALT+147	ô
ALT+148	ö
ALT+149	ò
ALT+150	û
ALT+151	ù
ALT+152	ÿ
ALT+153	Ö
ALT+154	Ü
ALT+155	ç
ALT+156	£
ALT+157	¥
ALT+158	Pts
ALT+159	f

ALT+160	á
ALT+161	í
ALT+162	ó
ALT+163	ú
ALT+164	ñ
ALT+165	Ñ
ALT+166	^a
ALT+167	°
ALT+168	ı
ALT+169	—
ALT+170	¬
ALT+171	½
ALT+172	¼
ALT+173	ı
ALT+174	«
ALT+175	»
ALT+176	⋮
ALT+177	⋮
ALT+178	⋮
ALT+179	
ALT+180	†
ALT+181	‡
ALT+182	‡
ALT+183	¶
ALT+184	‡
ALT+185	‡
ALT+186	
ALT+187	¶
ALT+188	¶
ALT+189	¶
ALT+190	¶
ALT+191	¶

ALT+192	Ł
ALT+193	Ł
ALT+194	ŧ
ALT+195	†
ALT+196	—
ALT+197	†
ALT+198	‡
ALT+199	‡
ALT+200	Ł
ALT+201	ŧ
ALT+202	Ł
ALT+203	ŧ
ALT+204	‡
ALT+205	=
ALT+206	‡
ALT+207	Ł
ALT+208	Ł
ALT+209	ŧ
ALT+210	π
ALT+211	Ł
ALT+212	Ł
ALT+213	ŧ
ALT+214	π
ALT+215	‡
ALT+216	‡
ALT+217	ŧ
ALT+218	ŧ
ALT+219	■
ALT+220	■
ALT+221	■
ALT+222	■
ALT+223	■

ALT+224	α
ALT+225	β
ALT+226	Γ
ALT+227	π
ALT+228	Σ
ALT+229	σ
ALT+230	μ
ALT+231	τ
ALT+232	Φ
ALT+233	Θ
ALT+234	Ω
ALT+235	δ
ALT+236	∞
ALT+237	φ
ALT+238	ε
ALT+239	∩
ALT+240	≡
ALT+241	±
ALT+242	≥
ALT+243	≤
ALT+244	∫
ALT+245	∫
ALT+246	÷
ALT+247	≈
ALT+248	°
ALT+249	·
ALT+250	·
ALT+251	√
ALT+252	ⁿ
ALT+253	²
ALT+254	■
ALT+255	

[This Page Intentionally Blank]

Last Word

What started as a simple instructional text to my File Manager, to expand upon a program I found had a better search and replace feature, became a written guide for my personal use and then a supplemental manual to BRU's manual, to a total rewrite of the manual itself, and finally, this book.

From a user's point of view, not as an expert, I detailed what it was I experienced or discovered, and documented it to help myself and others and share what I have learned along the way.

*And, in doing so,
I hope you have enjoyed this journey together.*

Timothy R. Mongeon

[This Page Intentionally Blank]

Index by Topic

A

Alternate

Alternate, 13, 58, 434–436, 438, 549, 593, 642–660
Alternative, 259, 301, 363, 375, 423, 433, 529, 549, 569, 641, 647, 649, 660
Vertical bar, 423, 433–436, 599

Always on Top

Always on Top, 11, 332

Appendix

Appendix, 6, 13, 47, 50–51, 65, 67, 74, 106, 131, 181, 188, 258, 295, 364, 396, 400, 417, 546

ASCII

Apostrophe, 515
ASCII, 13, 27, 59, 103, 125, 127, 131, 162, 379, 391, 400, 429, 515, 680, 692–695, 697–711
ASCII Character Chart (Version 1), 13, 698–699
ASCII Character Chart (Version 2), 13, 700–708
ASCII Character Codes, 13, 697

Extended ASCII Character Chart, 13, 709–711

PC Extended ASCII Character Chart, 13, 710–711

AutoNumber

Alpha Base, 223, 231, 241
AutoNumber, 11, 23, 232–236, 368–369
Base, 211, 223, 229, 231, 241–243, 573
Changeover, 241–243
Case Conversion, 8, 92–93, 228–229, 231
Numbering, 9, 23, 28, 101, 222–246, 306, 324, 351, 353–354, 368–369, 371
Resetting an (Auto) Numbering back to start value, 237
Break, 210, 237, 240, 368, 395, 426, 429, 437, 440, 489, 491, 494, 630, 632–635, 638–641, 647
Using Break and Folder Options Together, 240
Retain Autonumber, 11, 368–369
Roman Numerals Section Removed and Placed under 'Type' Section, 9, 228
Using The Special Character ' : ' Colon, 234

B

Before I begin

Before I begin, 3, 8, 509

Beginning of Line

Beginning of Line, 424, 436, 454, 462, 505, 534, 602

Braces

Angular Brackets, 49, 58, 103, 106, 158, 161, 361–362, 567
Braces, 398, 424, 576
Curly Braces, 424
Parentheses, 48, 51, 64, 68, 160–161, 361–362, 425, 430, 436, 438–448, 590–591, 593–594, 599, 603, 642
Closed Parenthesis, 74, 591
Open Parenthesis, 56, 74, 591
Parenthesis, 56, 72–81, 425, 599
Square Brackets, 49, 425, 427, 429, 442

C

Capture Group

Capture Group, 8, 13, 48–49, 51–60, 68, 86, 89, 91–93, 106, 421, 424–425, 430–432, 434–436, 438–441, 443–444, 446–451, 455–458, 475, 477–478, 486–487, 490, 492, 494–495, 501–503, 506, 512–519, 521–524, 530, 533–539, 541–542, 544–545, 547–548, 551–554, 556–565, 567, 569–572, 574–575, 577–586, 588–594, 596–597, 599–605, 607–611, 613–616, 619–626, 628, 630, 632–635, 638–662
Grouping, 425, 438, 515, 590, 649, 651, 661
Named Capture Group, 8, 49, 53–60,

596–597

Mixing Named and Unnamed Capture Groups, 56

Using Named Backreferences with Capture Groups, 8, 54

Nested Capture Group, 59, 440, 601, 645–656

Nested, 59, 440–441, 495, 599, 601, 643–657

Null, 21, 42, 49, 51, 55, 110, 387, 393, 453, 510, 517, 523–524, 533–534, 546, 572, 599, 603, 609–610, 619, 623, 631, 644–648, 652–659, 665–666, 694

Numbered Capture Group, 53–54, 56–57, 60

Undefined, 49, 642–643, 645–646, 651–653, 655–660

Using Alternates with Capture Groups, 13, 642–660

Using multiple Capture Groups to locate Partial Words, 13, 600–601

Change File Attributes

Change File Attributes, 12, 272, 338, 383

Character Translations

Character Translation Table, 391
Character Translations, 12, 27, 272, 391, 395

Clipboard

Clipboard, 9–10, 12, 111, 164, 281–282, 289–290, 313, 410–414
Clipboard Copy, 12, 410–413
<clip>, 111, 164
Copy all Available Column Data for Highlighted Files using the Clipboard Copy, 413
Extended File Details, 410–411
New Filename, 68, 312, 347, 354, 410–411, 445
Open Containing Folder, 12, 412
Original Filename with New Filename, 410
Pathname, and Paste to Location, 410

Colours

Active Criteria, 11, 42, 348

Changing the Colours, 11, 349
Colours, 11, 347, 349, 674, 685–686
Highlight Active Criteria, 11, 348
New Name - Invalid, 347
New Name - Ok, 347

Column

Autofit All Columns (Ctrl + Alt + +), 342
Column, 9, 11–12, 18, 20, 37, 47–49, 161, 178, 180, 184, 189, 205, 223, 254, 256–257, 261, 281, 310, 312–313, 328–329, 334–342, 353–354, 356–357, 360–365, 378, 385, 387, 391, 410, 413–414, 421, 604, 644, 672–674, 677, 681, 691
Column Headers, 37, 223, 354
Custom Column, 11, 356–357, 360–362
 Set Content of Custom Column, 356–357
 Show File Sizes as, 11, 333, 336
 Show Gridlines, 11, 332
 Show Icons, 11, 13, 333, 419
Header Name, 37
Select Columns, 11, 334–341

Consideration

Command Line options, 4
Consideration, 4–5, 8, 115, 175, 180, 183, 188, 192, 199, 210, 214, 386, 421, 494
INI File, 5
Portability, 5
Recursive Scan, 5, 13, 253, 259, 419

Content Pane

Auto-Select All Items After Listing a Folder, 10, 308
Clear All Items from Current List, 10, 308
Clear All non-Selected Items from Current List (Ctrl + 0, 309
Content Pane, 5, 15–18, 20–22, 30, 36–37, 45, 47–48, 103, 175, 189, 223, 245, 251, 254, 259, 280–281, 283, 293–294, 299, 302–303, 308–309, 311, 325, 327, 332, 334, 341, 343, 345, 350–351, 356, 358–359, 365, 375, 378, 383–385, 401, 403, 410–412, 415, 419
File List, 5, 11, 15, 17, 22, 30, 34, 37, 47, 103, 165, 223, 254, 258–259, 302–303,

305, 316, 340, 343–346, 354, 363, 401, 406
Refresh Files (F5), 10, 302
Remove From List (Delete key), 305
Reposition, 10, 37, 223, 280, 303–305, 351, 353–354, 566
 Move Bottom Selected Item (Ctrl + Alt + PgDn), 304
 Move Down Selected Item (Ctrl + Alt + Down Arrow), 304
 Move Top Selected Item (Ctrl + Alt + PgUp), 304
 Move Up Selected Item (Ctrl + Alt + Up Arrow), 304
 Swap Two Selected Items (Ctrl + Alt + S), 305
Show Only Items Affected by Renaming Criteria (Ctrl + 9), 307

Copy/Move to Location

Copy/Move to Location, 10, 18, 262–270, 297, 329, 339, 341, 375–376
Moving Files from one Directory to Another Location with Files content, 10, 268
Only Files can be Copied or Moved, not the Directories with their File Contents, 265

Criteria

Criteria, 8, 10–11, 18–19, 21, 23, 25–26, 29–32, 34–36, 38–42, 47, 64, 100, 103, 151, 224, 246, 255, 264, 267, 272–273, 276–277, 303, 307–308, 317–320, 325, 327, 329, 343, 348, 354, 368–369, 373, 377–379, 402, 406, 408, 416
Order of Evaluation, 26, 36, 100, 320, 371, 402, 440, 642
Order of Expression Evaluation, 8, 26–31
Reset All Renaming Criteria (Ctrl + T), 10, 327
Revert All Criteria to Last Saved (Ctrl + E), 10, 327

Custom Date

Custom Date, 27, 319
Custom Format, 173, 175, 177

D

Delimiter

Delimiter, 61, 66, 68, 82–85, 95–97, 104, 113, 125, 147, 233, 236, 252, 391, 427–428, 448–449, 554
Separator Character, 172, 175, 217, 233
Understanding the Importance of Delimiters in Simple, 82

Drag and Drop

Drag and Drop from Explorer, 8, 15–16, 259
Windows Explorer, 12, 15–17, 23, 39, 149, 151, 159, 223, 268, 286, 288, 296, 299, 301–302, 338, 354, 383–384, 409–410, 412, 415–416

E

End of Line

End of Line, 50, 66, 70, 424, 436, 454, 463, 505, 530, 588, 602

F

Favourite

Favourite, 3–4, 8, 10, 21, 23, 35, 39–45, 276–277, 327, 369, 395, 416
Favourite file, 4, 39, 41, 45, 276–277, 327, 395, 416
New (Ctrl + N), 10, 276
Open (Ctrl + O), 10, 276

Recent, 10, 40, 179, 277, 387
Save (Ctrl + S), 10, 276
Save As, 10, 39, 44, 276
Save On Exit, 8, 10, 39–40, 44, 276–277, 327, 416
Store Pathname, 8, 10, 39, 45, 277
Understanding Favourites, 8, 39–45, 276–277

Filters

Boolean Operator, 252
Exclamation point, 252, 508
BRU's Sub Dir Column vs Full path, 9, 256
Long File Paths, 257, 295
Long Path Support, 257
Max_Path, 257, 295
Sub Dir., 256–257, 334
WinAPI, 257
Condition, 12, 258–259, 404, 406, 690, 692
File Name Minimum Length/ Maximum Length, 255
Filespec, 251–252
Filters, 5, 9, 13, 18, 221, 238, 240, 250–261, 302, 334, 338, 341, 383, 404, 406, 419, 683
Mask, 30, 251, 258–259
Path Minimum Length/ Maximum Length, 255
Recursion, 5, 9, 238, 253–254, 256–257, 268–269, 316, 341, 416
Set Subfolder Level to Control Recursive Scanning, 9, 260
Wildcard, 23, 133, 251

Font

Font, 6, 11, 350, 411
Use Larger Font, 11, 350
Use Smaller Font, 11, 350

I

Introduction to the 2nd Edition of Volume I

Introduction to the 2nd Edition of Volume I, 6, 8
TGRMN, 3, 65, 67–68, 86–87, 118, 207, 214, 246, 255, 257, 311, 363, 403, 422
v3.4 New Additions, 8–13, 51–93, 104–111, 187–215, 283–301, 309, 315–317, 356–366, 413–414, 419, 422
v3.42 New Additions, 9, 218–221, 260–261, 390
v3.43 New Additions, 8–9, 94–98, 114–121, 227–231
Volume II, 6, 38, 47, 50, 55, 62, 65, 67, 74, 87, 91, 98, 105–106, 188, 258, 273, 295, 312, 364, 398, 400, 422, 461, 515, 540, 546, 588–589, 606, 645

Item Date

Anomalies, 205, 210–211, 214

J

JavaScript

About Conditional Renaming, 12, 401
 Conditional Renaming, 12, 401, 404, 406
 JavaScript Condition, 258–259, 406
Code Entry Form, 5, 38, 42, 259, 273, 327, 396, 398–400, 406
 Avoid Common Errors: <spaces>, 398
 Avoid Common Errors: ASCII Mistakes, 400
 Avoid Common Errors: CASE SENSITIVITY, 399
 Handling Syntax Errors, 12, 398
 Understanding the JavaScript Code Entry Form Colour Coding, 398
 Using the Test facility, 12, 397
JavaScript, 5, 12–13, 18, 23, 28–29, 38,

41–42, 64–65, 82, 87, 89, 91, 148, 159, 165, 217, 258–259, 272–273, 318, 325–327, 377–378, 396–406, 408, 412, 419, 667–670, 672

JavaScript (Extension) Libraries, 405
 Date.js, 5, 12, 396, 405, 408
 Sugar.js, 5, 12, 396, 405
JavaScript Bulk Rename Utility Constants and Variables, 13, 667
 JavaScript BRU Constants and Variables, 13, 668
 JavaScript BRU Utility Functions, 13, 669–670
JavaScript Renaming (Ctrl + F7), 396
Use JavaScript in Renaming, 272

Jump to Path (Ctrl + J)

Jump to Path (Ctrl + J), 10, 283–301
Network/UNC Paths are Supported, 10, 283
 Network, 6, 10, 254, 283–285, 288–289, 291–292, 295, 298–301, 359
 UNC, 10, 283–284, 290–291, 293–295, 297–299, 301, 315
 UNC Path, 10, 283, 290–291, 294–295, 299, 301, 315
 Universal Naming Convention, 283
Share, 183, 199, 211, 283–289, 291–292, 296–297, 299, 713
 Create a Share, 286
 Mapped Drive, 295, 333
 Net Share, 292
 Share Name, 283, 291
 Sharing, 284–285, 300, 359
Troubleshooting UNC under Jump to Path, 10, 299
 Function Discovery Resource Publication (fdPHost), 301
 Net View, 300
 Network and Sharing Center, 300
 Network Discovery, 299–300

L

Last Word

Last Word, 13, 436, 453, 460, 474, 571, 713

Literal

Backslash, 51–52, 105, 217, 310, 374, 421, 423, 426, 428, 447–449, 524, 604, 646, 652

Escape, 51, 61, 105–106, 423, 426, 428, 437, 551

Literal, 13, 49, 52, 66, 68, 74, 77–78, 84, 105–106, 362, 391, 421, 423, 426, 428, 447–448, 468, 470–471, 473–474, 476–478, 482, 505–506, 510, 515–519, 521, 524, 536, 541–545, 547, 551, 554, 556–557, 559–561, 563–565, 567, 569, 572, 574–575, 577, 579, 599, 604, 646, 652

The Escape (backslash), 426

371, 402, 440–442, 480, 497, 521, 533, 539, 555, 563, 569, 580, 586, 593, 596, 601–602, 607, 622, 631, 635, 637–642, 648, 655–660, 663

Backtrack, 50, 53–54, 106, 435, 439, 456, 486, 506, 512–514, 518–521, 536–539, 547, 550, 552, 555–557, 560–567, 569, 573, 577–580, 582, 601, 605, 608–611, 613–616, 621, 626

Consumed, 50, 587, 602

Parsing, 50, 62, 85, 661

Permutations, 555, 559, 569, 573

Recalculate, 555, 563

Reevaluate, 569, 578, 585–586

Expression, 8, 13, 18, 21, 23, 26–31, 47–98, 224, 258, 265, 329, 360, 391, 420–426, 429, 436–438, 440–445, 448, 450–452, 456, 477, 483–484, 486, 490, 492–493, 496, 510, 512, 514–519, 522–523, 529, 533–534, 537–538, 546, 548, 555, 559, 563, 565–569, 571, 573–574, 576–577, 582–584, 590–591, 593–594, 596–599, 601–604, 607, 609, 613–614, 620, 622–623, 626–627, 630–631, 635–636, 641–642, 645–646, 648, 656, 661

Input String, 47, 50, 438, 442, 452–453, 566–569, 604

Match string, 47, 51, 53–55, 67–70, 72, 75–78, 80, 82–84, 86, 95–96, 106, 511, 513, 535, 632, 635, 638, 660, 664

Pattern, 3, 23, 47–48, 50, 82, 103, 425, 431–432, 435, 438, 443–445, 448–453, 455–457, 508, 539–540, 555, 566–570, 573, 577, 580, 582, 587, 590–592, 600, 656, 661, 664, 677, 681, 683, 691

Sub-expression, 48, 50, 53, 55, 425, 450–451, 456, 490, 492–493, 496, 512, 514, 517–519, 523, 533–534, 538, 555, 566–567, 569, 573, 577, 582, 590, 597, 607, 622–623, 661

Menus

Actions Menu, 10, 20–21, 29, 34, 37, 40, 43, 45, 180, 223, 279–330, 333, 339–340, 351, 353–354, 373, 378, 408,

M

Match string

Analysis, 6, 53–55, 65, 94–98, 330, 366, 396, 401, 432, 435, 439, 443–444, 455–458, 475, 477–478, 480–481, 486, 488, 490, 492, 495, 509, 513–521, 534, 536, 538–539, 542, 545, 547, 551–552, 556–557, 559–561, 563–567, 569, 573, 575, 577, 589, 599, 601, 605–619, 621, 625–626, 628, 653, 655–656, 658–660, 663, 665–666

Component, 509, 565, 570, 573, 597, 672–673, 675–677

Evaluation, 6, 8, 21, 26–31, 36, 50, 54, 56, 64, 100, 111, 246, 318, 320, 325, 329,

416
 Context Menu, 12, 159, 165, 180, 182, 184, 200, 205, 305, 375, 378, 409–416
 Display Options Menu, 11, 13, 22, 36, 155, 223, 254, 280, 331–366, 419
 File Menu, 10, 21, 39–41, 43, 45, 275–277, 327, 369
 Menus, 10, 45, 177, 273–274
 Renaming Options Menu, 11, 13, 23, 37, 148, 150, 153, 167, 178, 180, 184, 189, 217, 223, 263, 280, 337, 339, 351, 354, 365, 367–381, 387, 408, 419, 421, 550, 664
 Special Menu, 12, 174, 176–177, 259, 272–273, 325, 338, 382–408

Metadata

EXIF, 9, 11–13, 23, 148, 150–153, 159, 165–169, 171, 175, 178–182, 184–185, 187–193, 196–198, 258, 337, 339, 356–358, 361, 363–364, 366, 377–378, 386–387, 390, 406, 408, 412, 419, 671–696
 Exchangeable image file format, 181, 189
 EXIF Metadata Reference, 13, 671–696
 EXIF Properties as Dates and Numbers, 9, 168
 EXIFTool, 151, 181, 189, 366
 Extract-EXIF data (Photos), 153
 CaptureDate, 181
 CreateDate, 181, 187, 189, 364
 EXIF DateTime, 179, 181
 EXIF DateTimeDigitized, 179, 181
 EXIF DateTimeOriginal, 178–181, 190–192, 197–198, 364, 386–387
 Extract EXIF Data (Photos), 11, 150, 178, 180, 337, 378, 387
 Show List of EXIF Info (.JPG Files), 12, 412
 System.DateAccessed, 182, 360
 System.DateCreated, 182, 360
 System.DateModified, 182, 212, 360
 Using the EXIF property 'Taken (Original)', 9, 178
 File Metadata Stored Within the File Itself, 184

File Header, 182
 ID3/ EXIF Data / File Properties, 148, 150
 Extract ID3 Data (MP3), 11, 148, 339, 377
 Metadata, 6, 13, 23, 64, 149, 151–153, 159, 168, 178, 180–193, 196–200, 203–207, 209–211, 213–215, 337, 339, 356–358, 363–364, 366, 377–378, 387, 389, 412, 419, 671–696
 Metadata when Copying and Moving Folders in a Directory Sub-Structure, 186
 Metadata when Moving or Copying Files and Folders, 185
 Understanding EXIF and Windows Properties as they apply to BRU, 181
 Windows Properties, 148, 151, 153, 159–161, 176, 181–182, 184–185, 187, 190, 192–193, 196–198, 200, 205, 212, 215, 336, 359–360, 363, 378, 404, 408, 412
 Date Accessed, 23, 151, 182–185, 194–196, 199–200, 202, 205, 207, 215, 272, 336, 360, 385–386
 Date Created, 23, 151, 176, 182–186, 188–190, 192–205, 207, 209, 211–215, 272, 336, 360, 363–364, 385–387, 389
 Date Modified, 23, 151, 171, 182–186, 190, 192–205, 207, 209, 211–215, 272, 336, 360, 363–364, 385–386
 Extract Windows File Properties, 11, 153, 184, 189, 365, 378
 File / Folder Extensions, 379
 Master File Table, 183, 336
 \$MFT, 183
 MFT, 183, 186, 336
 Resident Attributes, 183, 207
 Show List of File Properties, 12, 159, 182, 184, 199–200, 378, 412
 Standard Attribute, 183, 200, 207
 The Windows File System Metadata, 183
 exFAT, 182
 FAT32, 182–183, 186
 File System, 180, 182–186, 189, 199–200, 205, 207, 209, 211, 215,

- 295, 336, 359
- NTFS, 183, 186
- Understanding Metadata, 183, 215
- XDM, 184–185
 - Extensible Device Metadata, 184

Modifier

- Added Ability to use \E \L \I \U \u Modifiers in the Replace Field of the 'Simple' RegEx, 8, 94
- Case Conversion, 8, 92–93, 228–229, 231
- Case Insensitive /i, 8, 91
 - Case Insensitive, 8, 13, 91, 258, 591–595, 600
- Comments, 13, 150, 152, 378, 598, 690
 - Comments in Exact-Spacing, 598
 - Comments in Free-Space, 598
- Exact-Spacing, 596, 598
- Free-Space Option, 13, 596–598
 - Free-Space, 13, 596–598
- Global Switch, 8, 86–90, 105, 422, 445, 447–448, 511, 518, 575
- Modifier, 8–9, 87, 91, 94, 97, 108–109, 422, 591–594

N

Navigation Pane

- Directory, 3–4, 9–10, 16, 19, 23, 39, 45, 47, 183, 185–186, 217–220, 238–240, 252–257, 259–261, 263–270, 277, 280–283, 286, 290–291, 293–296, 298–299, 310–312, 315–316, 327, 334–335, 352, 373–375, 379, 383, 386, 405–406, 410, 412, 415–416, 419
- Directory Structure, 3, 186, 256, 266, 268, 270, 295, 298, 315–316
- Folders, 3, 5, 11, 15–18, 23, 26, 185–186, 221, 251, 253–255, 258, 260, 292, 302–303, 308, 329, 333, 338, 355, 373, 375, 381, 383, 406, 416, 567

- Navigation Pane, 16–17, 19, 22, 45, 238, 251, 254, 282–283, 288, 290, 292, 294, 296, 299, 302, 308, 311, 327, 333, 340, 374–375, 410–411
- Refresh Tree (Ctrl + F5), 10, 302
- Show/Hide Tree (F11), 10, 302
- subdirectory, 5, 16–17, 186, 218–220, 238–239, 253–254, 257, 259, 266, 270, 315, 334, 375, 419
- Subfolder, 5, 9, 13, 23, 221, 238, 253–254, 257, 259–260, 419
- Tree, 3, 10, 17, 22, 254, 288, 299, 302, 345, 374, 675
- Tree Hierarchy, 254, 288, 299, 302

non-Marking Groups

- If Then Else, 13, 599
 - Conditional Statement, 599
- Lookarounds: Lookahead and Lookbehind, 13, 602–641
 - Lookahead, 13, 545–546, 599, 602–641
 - Lookahead asserts, 603
 - Lookahead: Negative, 619
 - Lookahead: Positive, 604–606
 - Negative Lookahead, 546, 603, 619–621, 623, 627, 630, 633, 639
 - Positive Lookahead, 545, 621–622, 630, 632, 638
 - Understanding the Current Position in Lookaheads, 622–624
 - Using Capture Groups within Lookaheads, 607
 - Using Expressions within Lookaheads, 603
- Lookarounds, 13, 546, 602–641
- Lookbehind, 13, 602–641, 666
 - Lookbehind asserts, 625–629
 - Negative Lookbehind, 625, 630, 634, 641
 - Positive Lookbehind, 625, 627, 630, 635–637, 640
- non-Marking Groups, 13, 590, 603
- Using Lookarounds Before and After the Match, 630–631
 - Lookaround After the Match, 635–641
 - After the Match, 474, 503, 533, 551,

614–615, 623, 627, 630–631,
635–641, 664
Lookaround Before the Match, 603,
632–634
 Before the Match, 521, 572, 602–603,
 609–610, 619, 625, 627, 631–634
Zero Length Assertion, 497, 511, 632–633,
637
Zero Length Match, 510–511, 513, 534,
572, 645
Zero Occurrence Match, 55, 511, 513, 517,
523–524, 532–534, 546, 572, 609–610,
645–646

P

Program Notes

Program Notes, 8, 32–38, 280, 351, 354

Q

Quantifiers

Greedy, 54–55, 67–70, 72–74, 76, 78,
80–82, 85–86, 89, 94–97, 439, 442–443,
447–448, 451, 455, 457, 480–481, 493,
496, 510–514, 516–519, 521–522,
529–530, 533–539, 541, 552, 555–557,
559–561, 566–569, 572, 575, 577, 579,
582, 587–589, 607–608, 613, 619, 628
Asterisk, 133, 164, 251, 424, 588
Plus Sign, 423–424, 567
Zero or More Times, 424, 510, 516–517,
533, 555, 572
Lazy, 13, 55, 67–71, 73, 76, 78, 80, 82–83,
85–86, 94, 97, 511, 522, 529–533,

537–539, 563, 566–587, 663
non-Greedy, 522, 530, 533, 566, 569
Optional, 38, 65, 174, 424, 438, 441,
511, 522–523, 525–534, 536–539,
547, 562–566, 569, 572–573, 599,
676, 678, 684, 687–688
Question Mark, 424, 438, 522, 530, 537,
562, 566, 569, 587, 599
Reluctant, 522, 566
Zero or One Time, 424, 522, 530, 562
Posix Character Class (Bracket Expression),
429
Possessive Quantifiers, 555, 559, 562, 589
 Possessive, 555–556, 559–566, 589
Quantifiers, 50, 59, 424, 509–566, 569, 589,
626
Range Quantifier, 424, 541–545, 547–548,
551–553, 564, 627, 632–633, 635, 639,
641
 Iteration, 62, 424, 442–444, 448,
 450–451, 490, 492, 569, 627
 Range delimiter, 428

R

Regular Expressions

Boost, 13, 58, 65, 98, 419, 422, 576
BRU Supports PCRE v2 with Boost, 13, 422
ECMAScript, 65, 396
PCRE, 6, 8, 13, 50–52, 56–60, 65, 87–88,
98, 419, 422, 430, 433–434, 442, 445,
447, 533, 568, 576
PCRE (v1), 58
PCRE Engine, 50, 56–57, 65, 568, 576
PCRE v1, 6, 51–52, 56, 58–59, 88, 98, 422,
430, 433–434, 442, 445, 447
PCRE v2, 6, 13, 51–52, 56, 58, 60, 65,
87–88, 98, 419, 422, 434, 445, 447, 533
PCRE2 (v2), 58
Perl, 50, 65, 98, 422

Personal examples, 13, 661–666
 Fix Author's Initials, 665
 Fix Version Numbers, 661–664
 Remove Excess Spaces after an Author's Initials, 666

Regex Buddy, 88, 91, 224, 430, 455, 476, 487, 498–500, 505, 520, 553, 567, 573, 584, 588, 602, 605–606, 611–612, 616, 622–623, 631, 633–634, 637, 639–640, 662

Regex101.com, 444, 482, 515, 569, 573, 636

Regular Expressions, 8, 13, 23, 47–98, 224, 258, 420–426, 522, 546, 576, 599, 602

Regular Expressions (RegEx) Manual, 13, 420–426

Size limitations in PCRE and PCRE2, 8, 59
 code units, 59
 Unicode, 5, 59, 103, 311, 380
 UTF-8, 59, 685–686

Specifying Multiple Regular Expressions
 Using The (?X) Separator, 8, 61

v1, 6, 8, 23, 50–52, 56, 58–59, 64–66, 86–89, 91, 98, 148, 358, 377, 422, 430, 433–434, 442, 445, 447–448, 493, 526, 539, 548, 574, 576–577

v2, 6, 8, 13, 23, 50–52, 56, 58–61, 63–65, 86–89, 91, 93, 98, 153, 187, 377, 419, 422, 434, 445, 447–448, 518, 533, 574–576

v2 vs v1 & Simple, 8, 65

Rename (Ctrl + R)

Allow Overwrite / Delete Existing Files During Renaming If Needed, 11, 263, 376
 Log Renaming Activity to File (Ctrl + L), 380
 Rename File Extensions as being Part of File Name, 11, 379, 550, 664
 Rename Folder Extensions as being Part of Folder Name, 11, 379
 Show Confirmation Message After Renaming, 11, 381
 Show Warning Message Before Renaming, 11, 373, 381
 Allow Using '\' in Renaming Criteria for Creation of New Folders, 11, 373

Bulk Rename Here, 12, 415–416
 Create Undo Batch File, 10, 330
 Debug New Name, 10, 29, 318, 325, 330, 380
 Import Rename-Pairs (Rename from a Text File), 10, 310, 315–317
 Clear Imported-Pairs, 311, 314
 Directory List Method, 312
 Full Path Support, 10, 315
 Import Rename-Pairs, 10, 43, 310–313, 315–317
 Spreadsheet Program Method, 312
 View Imported Rename Pairs, 311, 314

Prevent Duplicates, 11, 217, 263, 372, 376–377

Preview (Ctrl + P), 10, 328
 Rename (Ctrl + R), 10, 329
 Rename in Reverse Order, 11, 37, 223, 280, 351, 354, 370
 Rename Object Manually (F2), 10, 302
 Undo Rename (Ctrl + Z), 10, 329
 Undo, 10, 20, 33, 254, 329–330, 339, 373, 375, 377, 380–381

Replace String

Alternate Syntax, 58
 Backreference, 8, 48–49, 51–57, 59–60, 92, 94–97, 448, 569, 571–572, 574, 577, 582, 603–604, 608–610, 613–615, 646, 661
 Illegal, 234, 347, 421, 448, 524, 537, 567, 604, 652
 Invalid, 11, 52, 106, 168, 347, 421, 447–448, 523–524, 591, 604, 626, 646, 648–652
 Named Backreferences, 8, 51, 53–55, 60
 Numbered Backreferences, 8, 51, 54, 56, 60
 Numbered Backreferences using the \$ Substitution Syntax, 51
 Replace String, 49, 52–57, 68, 78, 86, 90, 94–98, 106, 108, 422, 430, 433, 439, 445, 447–449, 510, 517, 521, 523–524, 554, 574, 576, 604, 661, 664
 Substituted Backreferences, 60
 substitution Syntax, 51, 53

S

Select

- Consecutive Selection, 32, 244, 354
 - Shift + Down arrow, 20, 32, 103
 - Shift + Up Arrow, 32
- Deselect All (Ctrl + D), 10, 280
 - Ctrl + D, 10, 20, 280
- Deselecting a File with a non-Consecutive Selection Also Deselects the Previous File that held Focus, 244
- Invert Selection (Ctrl + I), 10, 280
- non-Consecutive Selection, 32, 244, 354
 - Ctrl + Left Mouse Click, 32, 103, 280, 354
 - Ctrl + Mouse Click, 20
 - Individually Select with Keyboard and Mouse, 280
- Select, 10–11, 15–17, 19–20, 26, 32, 34, 45, 58, 72, 103, 137–138, 141, 159, 161, 165–166, 171–172, 179–180, 182, 189, 200, 217, 224, 245–246, 252, 256, 264, 267–269, 280–282, 284–285, 288–290, 292, 305, 307–308, 310–311, 313, 317, 330, 332, 334–342, 347, 349, 353, 356, 365, 378, 384, 388, 401, 403, 405, 415, 421, 495, 549
- Select All (Ctrl + A), 10, 103, 280
 - Ctrl + A, 10, 20, 103, 244, 280
- Select From Clipboard, 10, 281–282

Show Picture Viewer (Ctrl + W)

- Show Picture Viewer (Ctrl + W), 11, 334

Simple

- longest Match, 67–68, 70, 78, 85–86
 - Long, 5, 52, 67–68, 85, 87, 94, 161, 167, 199, 211, 254, 257, 295, 330, 364, 375, 391, 419, 451, 497, 543, 590, 646, 652, 659
- Shortest Match, 67–68, 70–71, 78, 85–86
 - Short, 67–68, 85, 94, 185
- Simple, 8, 23, 39, 50, 58, 61, 64–65, 67–68,

70, 72, 78, 82, 85–86, 94, 98, 147, 183, 208, 297, 334, 373, 386, 401, 404, 440, 442, 449, 460, 493–494, 496, 520, 620, 655, 660, 686, 713

- Tag, 9, 23, 65–68, 70–80, 82–86, 94–97, 111, 116–120, 148–154, 156–158, 161, 165–168, 178, 181, 184, 191–192, 319, 337, 356–358, 360, 363, 377–378, 408, 412, 567, 672–674, 676–678, 680–694

Sort

- Absolute Sorting, 351–352
- Ascending Sort, 353, 355
- Descending Sort, 355
- Displayed sequence, 36, 223, 280, 351, 354, 370
- Group Affected, 11, 354
- Logical Sorting, 11, 351–352
- Order of Processing, 36, 351
- Order of the Displayed Sequence, 36, 223, 280, 351, 354, 370
- Random Order, 306
- Sort, 10–11, 37, 211, 223, 280, 305–306, 351–355, 385, 414
- Sort Files and Folders Together, 11, 355
- Sorting, 11, 36–37, 223, 280, 306, 351–353, 375

Special Characters

- Anchor, 424, 436, 452–454, 461, 496–497, 530–532, 543, 545, 549, 602
 - Caret, 424–425, 427, 436
 - Dollar Sign, 424, 436
- Class, 47, 54–55, 89, 423–425, 427–429, 431–432, 442, 449, 451, 455–456, 460–467, 474, 483, 488, 490, 492–493, 497, 505, 522, 530, 545–546, 559–562, 569, 572, 575, 577, 582, 663–664, 678, 688
 - Character Class, 425, 427–429, 431–432, 442, 449, 451, 455–456
 - Character Class (or just Class), 427–428
 - Negating Character Class, 425
- Dot Metacharacter, 428, 430, 442–443, 451, 512, 519, 521, 533, 537, 551–552, 556–557, 567, 569, 572, 577, 579, 630,

632, 640
Metacharacter, 13, 51, 92, 106, 423–565,
567, 569, 572, 577, 579, 591, 602, 608,
613, 626, 628, 630, 632, 640, 642
non-Numeric Digit, 425, 451
non-Printable Characters, 426
non-Whitespace character, 426, 432
non-Word Boundary, 423, 497–508, 640
non-Word Characters, 456, 460, 465, 467,
477, 497, 499–501, 505–506, 508
punctuation, 49, 423, 427, 429, 456,
460, 465, 467, 477, 496–497,
505–506, 508, 515
Numeric Digits, 241–242, 425, 427–429,
449, 451, 455, 488, 490, 492–493,
548–549, 554, 603, 607, 613, 630,
632–635, 638–641, 661–664
Special Characters, 13, 47, 106, 423–426,
437
Whitespace, 423, 426–427, 429, 431–432,
455–456, 460, 464, 466, 477, 497, 505,
587, 596–597
Word Boundaries, 423, 460–461, 477,
481–482, 488, 493, 495–496, 498–500,
602, 640, 662
Word Characters, 423, 429, 455–456,
460–461, 465, 467, 477, 480–481, 488,
497–501, 505–506, 508, 628

Speeding up the Program

Speeding up the Program, 5, 13, 418–419

Substitute Tags

File Size, 9, 11, 155, 188, 333, 336, 413–414
Hash Value Tags, 156
Cyclic Redundancy Check (CRC), 156
Checksum, 156, 158
CRC-32, 156
Keccak, 156–157, 360
Hash, 9, 156–158, 356–357, 360–361,
363
Secure Hash Algorithm (SHA), 157
MD5, 158, 360–361, 686
MD5 (Message-Digest), 158
SHA-1, 157
SHA-2, 157

SHA-3, 157
Property Formatting Markers, 162, 168
File Properties as Dates and Numbers, 9,
162
Using EXIF Tags, 9, 165, 167, 184,
337, 378, 408, 412
Using Windows Clipboard Data, 9, 164
Removed Tag <removed>, 154
<removed>, 154
Substitute Tags, 165, 377–378
Substitution Tags available for JPEG image
files, 150
Substitution Tags available for MP3 files, 148
Using Substitution Tags, 9, 148, 377, 408

System.ItemDate

Epoch, 188, 364
EXIF:Photo.DateTimeDigitized, 187
GIGO, 11, 366
HEIC, 188, 364
Item Date, 9, 11–12, 23, 187–190, 192–215,
341, 363–366, 386, 390, 413–414
Nirsoft, 205–207, 214
non-Document or Other Filetype Breaks
Microsoft's Rules, 210
System.DateImported, 211–212, 360
Windows Properties' Date Modified is
Earlier than the Date Created value,
215
RAW, 156, 179–180, 188, 364, 680–687, 696
System.Photo.DateTaken, 163, 187,
191–193, 364
The Tests Used, 194
Content Created, 199–200, 210–214, 360
Date Last Saved, 190, 192, 199–204,
210–214, 360–361
Document Files, 192, 204, 207
Document Filetype, 192–193, 198–199,
203–204, 210–211, 214, 363
Document Metadata, 199, 210–211,
213–214
How I Tested Document Metadata, 199,
211
System.Document.DateCreated, 199,
211–212, 360
System.Document.DateSaved, 192–193,

199–200, 211–212, 360–361
XMP:DateTimeOriginal, 187

T

Technical Notes

Technical Notes, 3

The 14 Sections Used to Specify the Criteria

Section # 10 - Numbering, 9, 222
Section #10: Numbering, 28, 101, 227, 229, 232, 306, 351, 368–369
Section # 11 - Extension, 9, 247
Section #11: Extension, 28, 64, 379
Section # 12 - Filters, 9, 250
Section #12: Filters, 5, 13, 221, 238, 240, 257, 260, 302, 334, 338, 341, 383, 404, 406, 419
Section # 13 - Copy/Move to Location, 10, 262
Section #13: Copy/Move to Location, 297, 375–376
Section # 14 - Special, 10, 271
Section #14: Special, 27–31, 41, 176–177, 321, 325, 336, 382, 401–402, 408
Section # 1: Regular Expressions, 47–98
Section #1: RegEx, 6, 26, 47, 65, 100, 105–106, 232, 258, 320, 419
Section # 2 - (File) Name, 99
Section #2: Name, 100, 320
Section # 3 - Replace, 9, 102
Section #3: Replace, 26, 41, 105–106, 125, 263, 297, 320, 375
Multiple Replacements, 9, 104
Pipe, 104–106, 310, 312, 423, 642
Position Modifier, 9, 108–109
Section # 4 - Case, 9, 112
Section #4: Case, 27, 29, 223, 227–229,

325, 401–402
<clear>, 116–118
<ic>, 119
<rnlo>, 120
<rnup>, 120
New Changes in the Except(ion) Field, 116
New York Times Title Case, 9, 114
Title Enhanced, 114, 116–121
Section # 5 - Remove, 9, 122
Section #5: Remove, 27, 154, 393
Removed Tag <removed>, 154
Section # 6 - Move/Copy Parts, 9, 136
Section # 6: Move/Copy Parts, 137–143
Section # 7 - Add, 9, 144, 363
Section #7: Add, 27, 100, 148, 159–160, 167, 184, 337, 375, 377–378, 387, 408, 672
Section # 8 - Auto Date, 9, 170
Section #8: Auto Date, 6, 27, 181–182, 341, 377–378, 387, 389
Section # 9 - Append Folder Name, 9, 216
Section #9: Append Folder Name, 27, 218, 257, 375
The 14 Sections Used to Specify the Criteria, 8, 25

The Program Screen

Data Entry Field, 47–48, 53, 132, 147, 155, 159, 161, 166, 173, 225, 248, 258–259, 303, 361, 378, 388, 406, 425, 439
Expand File List (Ctrl + F9), 11, 343–344
Filter Refresh, 15, 252
Maximize File List (F9), 11, 343, 345–346
Status Bar, 20, 39, 41–43, 254, 277, 294
The Program Screen, 8, 17–20
Up/Down Indicators, 145, 174, 225
User Interface, 8, 21–22, 272, 329, 343, 345
Expansion Button, 22
Preview, 10, 20–21, 26, 34, 149, 151, 297–298, 310, 328–329, 374, 399, 401, 430, 686
Rename Action, 21, 227, 248, 254, 310, 330, 368, 421
Reset, 10–11, 21, 35, 41–42, 237–240, 276, 327, 341, 350, 366, 368, 401

Revert, 10, 21, 40, 327, 416
Zoom Feature, 21

Timestamp

12 Hour format, 174
24 Hour Military Time, 172, 174
Adding a New Date & Timestamp, 9, 171, 176
Change File Timestamps, 12, 151, 176, 272, 385, 390
 Status Not Set, 272
Change File Timestamps Function Allows Item Date, 390
Military Time Conversion, 177
Offset, 174–175, 389, 673, 675–676, 678
Timestamp, 9, 12, 18, 21, 23, 31, 151, 168, 171, 174–177, 179, 181–182, 184–189, 194, 196, 198–203, 208–211, 213, 272–273, 325, 328–329, 336, 366, 375, 380, 385–387, 389–390, 401, 693
Understanding Delta, 12, 389

Type of Date Data

Accessed (Current), 171, 182
Accessed (New), 176
Creation (New), 176
Creation Date (Current), 171
Current, 5, 10, 21, 37, 39, 42, 89, 100–101, 111, 139–140, 159–161, 167, 171, 174–176, 182, 185–186, 194–195, 198, 200–201, 203, 211, 218–220, 223, 227–228, 248, 253, 256–257, 264, 266, 270, 276–277, 280, 292–294, 305–309, 316, 327, 330, 334, 336–337, 342, 351, 354, 364, 368, 373, 386, 389, 396, 412, 416, 419, 423, 439, 442–445, 449, 456, 461, 472, 474, 483, 486, 519, 534, 536–539, 545, 547, 552, 556–557, 560–561, 563–565, 567, 569–570, 572, 577, 579, 582, 588–589, 592, 594, 602, 607–610, 614–615, 619, 621–625, 632–633, 635, 637, 641, 663–664, 681, 683–684
Modified (New), 176
Modified Date (Current), 171
Taken (Digitized), 179, 181, 387

Taken (Modified), 179–181, 387
Taken (Original), 9, 151, 171, 175, 178–181, 188, 190, 192–193, 196–197, 205, 337, 341, 363–364, 366, 378, 387, 390, 413–414
Taken (Recent), 179, 387
Type of Date Data, 171

