



ReNamer's

User Manual

Denis Kozlov

March 2008

Table of Contents

Table of Contents	2
Introduction	3
Description	3
Basic usage	3
Help notes	3
Quick Start	4
Rules	6
Date-Time Format	8
Pascal Script	10
Standard types available	10
Custom types/variables/constants	10
Registered functions	10
Regular Expressions	16
Simple matches	16
Escape sequences	16
Character classes	17
Predefined Classes	17
Word/Text Boundaries	18
Iterators	18
Alternatives	19
Subexpressions, Backreferences, Substitution	20
RegEx Examples	20
Command Line	22
Parameters	22
Examples	22

Introduction

Description

ReNamer is a very powerful and flexible file renaming tool. It offers all of the standard renaming procedures, including prefixes, suffixes, replacements, case changes, remove contents of brackets, add number sequences, change file extensions, etc. It has an ability to rename folders and network files, process Regular Expressions, and supports Unicode filenames. For advanced users, there is a PascalScript rule, which let users program their very own renaming rule.

The program allows you to combine multiple renaming actions as a rule set, applying each action in logical sequence, which can be saved, loaded and managed within the program. In addition, ReNamer is able to extract large variety of meta tags, such as: ID3v1, ID3v2, EXIF, OLE, AVI, MD5, SHA1, CRC32 and many more.

Basic usage

Step 1: "*Add Files*" or "*Add Folders*" – Add files/folders to the table;

Step 2: "*Rules*" – Setup rules, their parameters and order;

Step 3: "*Preview*" – Generate new names using rules;

Step 4: "*Rename*" – Rename your files.

Help notes

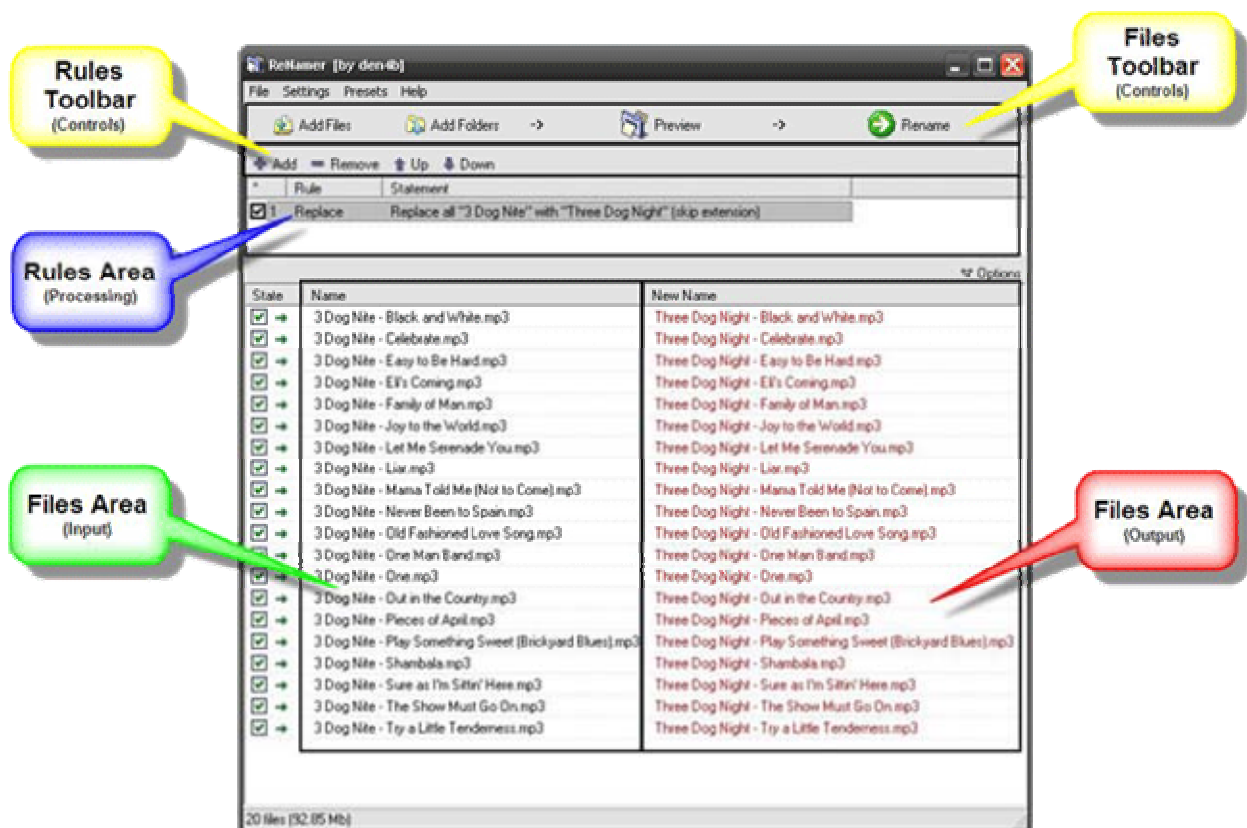
This collection of help documents is yet to be expanded. More help could be found on forums and in the “downloads” section. Unfortunately, I do not have enough free time to create, expand and maintain this document, so if you wish to contribute by creating new or/and expanding already existing sections – you are more than welcomed to do so. Any contributions will be greatly appreciated!

Quick Start




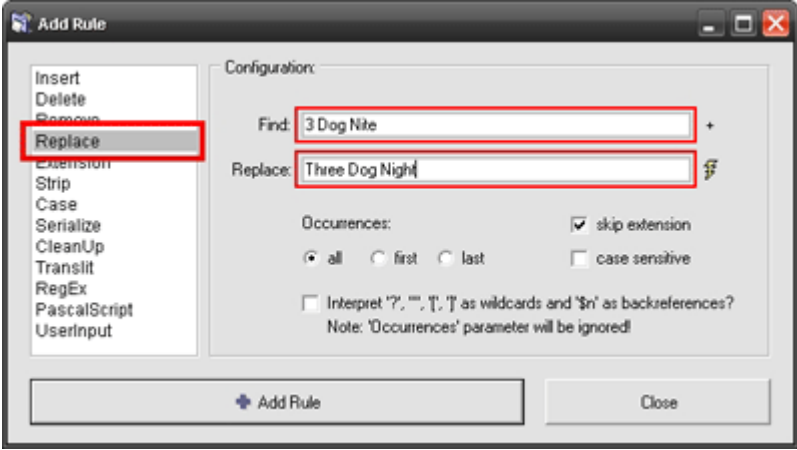


Click on the  **ReNamer** icon to start the program.

The window shown below - is where you will do most of your processing. It consists of two toolbars which control the Files and Rules. The **Rules Area** is used to specify the processing for your name changes, and the **Files Area** is used for displaying the Input & Output file names.

You create the Rules, which are then used to change the file names. You get to choose the Rules you need and configure them to do what you want. Your Rules can be added, edited, deleted, and moved up or down in the processing sequence. There are four basic steps to processing files in ReNamer. These steps are performed using the several buttons that are explained in the table below.



The file names are displayed both "**before**" and "**after**" the name change.

<div>1</div> <div>  Add Files  Add Folders </div>	<p>Select the files that you wish to rename. Alternatively, you can drag & drop or copy & paste your files in to the main window.</p>
<div>2</div> <div>  Add </div>	<p>Create rules which will change the file names. Once you press the button, a separate dialog will come up with a list of available rules and their configuration, as show on the screenshot below:</p> 
<div>3</div> <div>  Preview </div>	<p>Manually execute the rules and display the name changes. This step is not necessary when the "Auto Preview" option is checked (default).</p>
<div>4</div> <div>  Rename </div>	<p>When you are satisfied with the results of all your name changes, you finalize everything by renaming the files, making your file name changes permanent.</p>

P.S. Thanks to Lance Smith for helping creating this section.

Rules

ReNamer has an extensive rules set. These rules can be combined together, in a logical sequence, to perform nearly any thinkable operation with the filename. The table below lists available rules with a brief description of each of the rule.

Rules	Description
Insert	Insert specified text into the filename: as prefix, as suffix, at the specified position, before or after specified text. This rule has also an option to insert meta tags into the filename, which are very useful for describing files based on their contents and other properties, for example: modification or creation time of the files; artist, title, album and year of mp3 files; date of the digital photos; image dimensions; title and subject of the documents; version information of the executable files; and many more.
Delete	Delete a portion of the filename usually defined by character positions: from the specified position, from the occurrence of the specified delimiter, until number of characters, until occurrence of the specified delimiter, or till the end. Rule can be inverted to process filename in a right-to-left manner.
Remove	Remove specified text from the filename: first, last or all occurrences. Optionally, wildcards can be used within this rule, to remove masked text fragments.
Replace	Replace rule is very much like Remove rule, with similar options, except instead of removing the text fragments it will replace them with the specified text.
Extension	Change extension of files to the specified extension, or to the extension automatically detected through the internal database of binary signatures.
Strip	Strip characters from the filename. Rule has predefined character sets, like digits, symbols, brackets, but user can also define his/her own character set. Every occurrence of each of the specified characters will be removed from the filename.
Case	Change the case of the filename: capitalize, to lower case, to upper case, invert

	case, and put only first letter capital (like in a sentence). There is also an option to force case for the manually entered fragments, for example: CD, DVD, DJ, etc.
Serialize	Serialize rule uses numeric incremental or random sequences of digits to put filenames into an order.
CleanUp	Cleanup filenames from/for commonly used naming conventions for internet, peer-to-peer networks, and other resources.
Translit	Transliterate Non-English characters from different languages into their English/Latin representation. Useful for preparing files for network storage and transfer. The mappings of characters have to be specified by user. Several examples are included in the rule.
RegEx	Stands for Regular Expressions. This is an expert feature, and might look very confusing for novice users. It is a commonly used concept for complex pattern/expression matching and replacing operations.
PascalScript	Scripting rule, which allows programming-aware users to code their own renaming rule using predefined set of functions. This rule uses Pascal/Delphi programming syntax and conventions. Extremely powerful feature in the right hands.
UserInput	Rule that simply sets the new names of the files to the names entered in a list (one name per line).

Warning: some characters are reserved by the operating system, thus, cannot be used in the name of the file, i.e.: \ / : * ? " < > |

Date-Time Format

Date-Time format is mostly used by the meta tags. You can define almost any thinkable format for all tags which extract a date-time field from the file. Below is a list of variables which you can use.

Variable	Description
d	Displays the day as a number without a leading zero (1-31).
dd	Displays the day as a number with a leading zero (01-31).
ddd	Displays the day as an abbreviation (Sun-Sat).
dddd	Displays the day as a full name (Sunday-Saturday).
e	Displays the year in the current period/era as a number without a leading zero (Japanese, Korean and Taiwanese locales only).
ee	Displays the year in the current period/era as a number with a leading zero (Japanese, Korean and Taiwanese locales only).
g	Displays the period/era as an abbreviation (Japanese and Taiwanese locales only).
gg	Displays the period/era as a full name. (Japanese and Taiwanese locales only).
m	Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier then minute is displayed.
mm	Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier then minute is displayed.
mmm	Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable.
mmmm	Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable.
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).

hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
z	Displays the millisecond without a leading zero (0-999).
zzz	Displays the millisecond with a leading zero (000-999).
am/pm	Uses the 12-hour clock for the preceding h or hh specifier, and displays "am" for any hour before noon, and "pm" for any hour after noon. The am/pm specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
a/p	Uses the 12-hour clock for the preceding h or hh specifier, and displays "a" for any hour before noon, and "p" for any hour after noon. The a/p specifier can use lower, upper, or mixed case, and the result is displayed accordingly.
"xx"	Characters enclosed in single or double quotes are displayed as-is, and do not affect formatting.

For example, if assume that the date is 25-th of October 2007 and the time is 16:59:00, then sample formats and their outputs would be:

- “**dd-mm-yyyy hh.nn.ss**” format will produce “**25-10-2007 16.59.00**”, which is an easily readable format for the date and time.
- “**yyyymmddhhnnss**” format will produce “**20071025165900**”, which is ideal for serializing files because the filename is relatively short, most probably unique, contains only digits, and also makes files automatically sorted in chronological order.

Pascal Script

This rule uses Delphi/Pascal programming syntax and conventions. Changes to the **FileName** variable will be treated as changes to the New Name of the File. The **FilePath** constant holds the original path to the file, and provided for the direct file access. Main code must be within the "**begin**" and "**end.**" keywords. User defined procedures, functions, variables, constants and types are supported, as well as importing of external functions from DLLs.

Do not override registered variables, types and functions listed below. Some of the functions able to alter your file system, so use those with caution! All manipulations with the **FileName** variable should be done using Unicode functions, i.e. **WideString** type should be used instead of an ordinary **String** type.

Standard types available

Byte, ShortInt, Char, Word, SmallInt, Cardinal, LongInt, Integer, String, Real, Double, Single, Extended, Boolean, Array, Record, Enumerations, Variant.

Custom types/variables/constants

- **var** FileName: WideString;
 - Name of the currently processing file, including extension (e.g. "file.txt");
- **const** FilePath: WideString;
 - Full path of the currently processing file (e.g. "c:\temp\file.txt");
- **type** TDateTime = Double;
 - Used for manipulating date and time values;
- **type** TStringsArray = array of WideString;
 - Used for storing and manipulating lists/arrays of strings.

Registered functions

*** Basic String Handling Routines ***

```
procedure Insert(Source: String; var S: String; Index: Integer);  
procedure Delete(var S: String; Index, Count: Integer);  
function Copy(S: String; Index, Count: Integer): String;  
function Pos(Substr: String; S: String): Integer;
```

*** Length Managing Routines ***

```
procedure SetLength(var S: Array; NewLength: Integer);  
procedure SetLength(var S: String; NewLength: Integer);  
procedure SetLength(var S: WideString; NewLength: Integer);  
function Length(const S: Array): Integer;  
function Length(const S: String): Integer;  
function Length(const S: WideString): Integer;
```

*** Unicode String Handling Routines ***

```
procedure WideInsert(const Substr: WideString; var Dest: WideString; Index: Integer);  
procedure WideDelete(var S: WideString; Index, Count: Integer);  
procedure WideSetLength(var S: WideString; NewLength: Integer);  
function WideLength(const S: WideString): Integer;  
function WideCopy(const S: WideString; Index, Count: Integer): WideString;  
function WidePos(const SubStr, S: WideString): Integer;  
function WidePosEx(const SubStr, S: WideString; Offset: Cardinal): Integer;  
function WideUpperCase(const S: WideString): WideString;  
function WideLowerCase(const S: WideString): WideString;  
function WideCompareStr(const S1, S2: WideString): Integer;  
function WideCompareText(const S1, S2: WideString): Integer;  
function WideSameText(const S1, S2: WideString): Boolean;  
function WideTextPos(const SubStr, S: WideString): Integer;  
function WideTrim(const S: WideString): WideString;  
function WideReplaceStr(const S, OldPattern, NewPattern: WideString): WideString;  
function WideReplaceText(const S, OldPattern, NewPattern: WideString): WideString;  
function WideSplitString(const Input, Delimiter: WideString): TStringArray;  
function WideCaseCapitalize(const S: WideString): WideString;  
function WideCaseInvert(const S: WideString): WideString;
```

*** Meta Tags Extraction ***

```
function CalculateMetaTag(const FilePath: WideString; const MetaTagName String):  
String;
```

*** Regular Expressions ***

```
function ReplaceRegEx(const Input, Find, Replace: WideString; const CaseSensitive,  
UseSubstitution: Boolean): WideString;  
function MatchesRegEx(const Input, Find: WideString; const CaseSensitive: Boolean):  
TStringsArray;  
function SubMatchesRegEx(const Input, Find: WideString; const CaseSensitive: Boolean):  
TStringsArray;
```

*** Unicode Character Handling Routines ***

```
function IsWideCharUpper(WC: WideChar): Boolean;  
function IsWideCharLower(WC: WideChar): Boolean;  
function IsWideCharDigit(WC: WideChar): Boolean;  
function IsWideCharSpace(WC: WideChar): Boolean;  
function IsWideCharPunct(WC: WideChar): Boolean;  
function IsWideCharCntrl(WC: WideChar): Boolean;  
function IsWideCharBlank(WC: WideChar): Boolean;  
function IsWideCharXDigit(WC: WideChar): Boolean;  
function IsWideCharAlpha(WC: WideChar): Boolean;  
function IsWideCharAlphaNumeric(WC: WideChar): Boolean;  
function WideCharUpper(const WC: WideChar): WideChar;  
function WideCharLower(const WC: WideChar): WideChar;
```

*** Unicode Conversion Routines ***

```
function WideToAnsi(const WS: WideString): String;  
function AnsiToWide(const S: String): WideString;  
function UTF8Encode(const WS: WideString): String;  
function UTF8Decode(const S: String): WideString;
```

*** Basic Conversion Routines ***

```
function IntToStr(Value: Integer): String;  
function StrToInt(const S: String): Integer;  
function StrToIntDef(const S: String; const Default: Integer): Integer;  
function DateToStr(D: TDateTime): String;
```

function **StrToDate**(const S: String): TDateTime;

function **Chr**(X: Byte): Char;

function **Ord**(X: Char): Byte;

*** Date And Time Routines ***

function **Date**: TDateTime;

function **Time**: TDateTime;

function **Now**: TDateTime;

function **EncodeDate**(Year, Month, Day: Word): TDateTime;

function **EncodeTime**(Hour, Min, Sec, MSec: Word): TDateTime;

function **TryEncodeDate**(Year, Month, Day: Word; var Date: TDateTime): Boolean;

function **TryEncodeTime**(Hour, Min, Sec, MSec: Word; var Time: TDateTime): Boolean;

procedure **DecodeDate**(const DateTime: TDateTime; var Year, Month, Day: Word);

procedure **DecodeTime**(const DateTime: TDateTime; var Hour, Min, Sec, MSec: Word);

function **DayOfWeek**(const DateTime: TDateTime): Word;

function **DateTimeToUnix**(D: TDateTime): Int64;

function **UnixToDateTime**(U: Int64): TDateTime;

function **FormatDateTime**(const fmt: String; D: TDateTime): String;

*** File Management Routines ***

function **WideFileSize**(const FileName: WideString): Int64;

function **WideFileExists**(const FileName: WideString): Boolean;

function **WideDirectoryExists**(const Directory: WideString): Boolean;

function **WideForceDirectories**(Dir: WideString): Boolean;

function **WideCreateDir**(const Dir: WideString): Boolean;

function **WideDeleteFile**(const FileName: WideString): Boolean;

function **WideRenameFile**(const OldName, NewName: WideString): Boolean;

function **WideFileSearch**(const Name, DirList: WideString): WideString;

function **WideGetCurrentDir**: WideString;

function **WideSetCurrentDir**(const Dir: WideString): Boolean;

procedure **WideScanDirForFiles**(Dir: WideString; var Files: TStringsArray;

const Recursive, IncludeHidden, IncludeSystem: Boolean; const Mask: WideString);

procedure **WideScanDirForFolders**(Dir: WideString; var Folders: TStringsArray;

const Recursive, IncludeHidden, IncludeSystem: Boolean);

*** File Name Utilities ***

```

function WideExtractFilePath(const FileName: WideString): WideString;
function WideExtractFileDir(const FileName: WideString): WideString;
function WideExtractFileDrive(const FileName: WideString): WideString;
function WideExtractFileName(const FileName: WideString): WideString;
function WideExtractBaseName(const FileName: WideString): WideString;
function WideExtractFileExt(const FileName: WideString): WideString;
function WideChangeFileExt(const FileName, Extension: WideString): WideString;
function WideStripExtension(const FileName: WideString): WideString;
function WideExpandFileName(const FileName: WideString): WideString;
function WideExtractRelativePath(const BaseName, DestName: WideString): WideString;
function WideExtractShortPathName(const FileName: WideString): WideString;
function WideIncludeTrailingPathDelimiter(const S: WideString): WideString;
function WideExcludeTrailingPathDelimiter(const S: WideString): WideString;
function WideSameFileName(const S1, S2: WideString): Boolean;
function WideGetEnvironmentVar(const VarName: WideString): WideString;

```

*** File Read/Write Routines ***

```

function FileReadFragment(const FileName: WideString; Start, Length: Integer): String;
function FileReadLine(const FileName: WideString; LineNum: Integer): String;
function FileCountLines(const FileName: WideString): Integer;
function FileReadContent(const FileName: WideString): String;
procedure FileWriteContent(const FileName: WideString; const Content: String);
procedure FileAppendContent(const FileName: WideString; const Content: String);

```

*** File Properties Routines ***

```

function FileTimeModified(const FileName: WideString): TDateTime;
function FileTimeCreated(const FileName: WideString): TDateTime;
function SetFileTimeCreated(const FileName: WideString; const DateTime: TDateTime): Boolean;
function SetFileTimeModified(const FileName: WideString; const DateTime: TDateTime): Boolean;

```

*** Process Execution Routines ***

```

function ExecuteProgram(const Command: String; WaitForProgram: Boolean): Cardinal;
function ExecConsoleApp(const CommandLine: String; out Output: String): Cardinal;

```

*** Interactive Dialogs ***

```
procedure ShowMessage(const Msg: String);  
procedure WideShowMessage(const Msg: WideString);  
function DialogYesNo(const Msg: String): Boolean;  
function WideDialogYesNo(const Msg: WideString): Boolean;  
function InputBox(const ACaption, APrompt, ADefault: String): String;  
function InputQuery(const ACaption, APrompt: String; var Value: String): Boolean;  
function WideInputBox(const ACaption, APrompt, ADefault: WideString): WideString;  
function WideInputQuery(const ACaption, APrompt: WideString; var Value: WideString):  
Boolean;
```

*** Other Routines ***

```
procedure Randomize;  
procedure Sleep(Milliseconds: Cardinal);  
procedure DivMod(Dividend: Integer; Divisor: Word; var Result, Remainder: Word);  
procedure SetClipboardText(const S: WideString);  
function GetClipboardText: WideString;  
function RandomRange(const AFrom, ATo: Integer): Integer;  
function GetTickCount: Cardinal;  
function SizeOf(X): Integer;
```

For more information:

- www.delphibasics.co.uk
 - Good Delphi related site which has some useful and easy to understand tutorials and reference guide.
- www.remobjects.com
 - RemObjects is a software development company that offers this great PascalScript component, which allows run-time scripting.

Regular Expressions

Regular Expressions are a widely-used method of specifying patterns of text to search for. Special metacharacters allow you to specify, for instance, that a particular string you are looking for occurs at the beginning or end of a line, or contains n recurrences of a certain character. Regular expressions look ugly for novice users, but they are very powerful tools.

Simple matches

Any single character matches itself, unless it is a metacharacter with a special meaning described below. You can cause characters that normally function as metacharacters or escape sequences to be interpreted literally by "escaping" them by preceding them with a backslash "\", for instance: metacharacter "^" match beginning of string, but "\^" match character "^", "\\" match "\" and so on.

foobar	matches string "foobar"
\^FooBarPtr	matches "\^FooBarPtr"

Escape sequences

Characters may be specified using a escape sequences syntax much like that used in C and Perl: "\n" matches a new line, "\t" a tab, etc. More generally, "\xnn", where "nn" is a string of hexadecimal digits, matches the character whose ASCII value is nn. If You need wide (Unicode) character code, You can use "\x{nnnn}", where "nnnn" - one or more hexadecimal digits.

\xnn	char with hex code nn
\x{nnnn}	two bytes char with hex code nnnn (unicode)
\t	tab (HT/TAB), same as \x09
\n	new line (NL), same as \x0a

\r	carriage return (CR), same as \x0d
\f	form feed (FF), same as \x0c
foo\x20bar	matches "foo bar" (note space in the middle)
\tfoobar	matches "foobar" predefined by tab

Character classes

You can specify a character class, by enclosing a list of characters in [], which will match any one character from the list. If the first character after the "[" is "^", the class matches any character not in the list. Within a list, the "-" character is used to specify a range, so that a-z represents all characters between "a" and "z", inclusive. If you want "-" itself to be a member of a class, put it at the start or end of the list, or escape it with a backslash. If you want "]" you may place it at the start of list or escape it with a backslash.

[-az]	matches "a", "z" and "-"
[a\ -z]	matches "a", "z" and "-"
[a-z]	matches all twenty six small characters from "a" to "z"
[\n-\x0D]	matches any of #10,#11,#12,#13
[^0-9]	matches any none digit character
[\d-t]	matches any digit, '-' or 't'
[]-a]	matches any char from ']'..'a'
foob[aeiou]r	finds strings "foobar", "foober" etc. but not "foobbr", "foobcr" etc.
foob[^aeiou]r	find strings "foobbr", "foobcr" etc. but not "foobar", "foober" etc.

Predefined Classes

\w	an alphanumeric character (including "_")
\W	a non-alphanumeric
\d	a numeric character

\D	a non-numeric
\s	any space (same as [\t\n\r\f])
\S	a non space
.	any character in line (the symbol is just a dot)

Word/Text Boundaries

A word boundary (**\b**) is a spot between two characters that has a **\w** on one side of it and a **\W** on the other side of it (in either order), counting the imaginary characters off the beginning and end of the string as matching a **\W**.

\b	word boundary
\B	non-(word boundary)
\A	start of text
\Z	end of text

Iterators

Any item of a regular expression may be followed by another type of metacharacters - iterators. Using this metacharacters You can specify number of occurrences of previous character, metacharacter or subexpression. So, digits in curly brackets of the form **{n,m}**, specify the minimum number of times to match the item **n** and the maximum **m**. The form **{n}** is equivalent to **{n,n}** and matches exactly **n** times. The form **{n,}** matches **n** or more times. A little explanation about "greediness". "Greedy" takes as many as possible, "non-greedy" takes as few as possible. For example, **'b+'** and **'b*'** applied to string **'abbbbc'** return **'bbbb'**, **'b+?'** returns **'b'**, **'b*?'** returns empty string, **'b{2,3}?'** returns **'bb'**, **'b{2,3}'** returns **'bbb'**.

*	zero or more ("greedy"), similar to {0,}
+	one or more ("greedy"), similar to {1,}
?	zero or one ("greedy"), similar to {0,1}

{n}	exactly n times ("greedy")
{n,}	at least n times ("greedy")
{n,m}	at least n but not more than m times ("greedy")
*?	zero or more ("non-greedy"), similar to {0,}?
+?	one or more ("non-greedy"), similar to {1,}?
??	zero or one ("non-greedy"), similar to {0,1}?
{n}?	exactly n times ("non-greedy")
{n,}?	at least n times ("non-greedy")
{n,m}?	at least n but not more than m times ("non-greedy")
foob.*r	matches strings like 'foobar', 'foobalkjdfk9r' and 'foobr'
foob.+r	matches strings like 'foobar', 'foobalkjdfk9r' but not 'foobr'
foob.?r	matches strings like 'foobar', 'foobbr' and 'foobr' but not 'foobalkj9r'
fooba{2}r	matches the string 'foobaar'
fooba{2,}r	matches strings like 'foobaar', 'foobaaar', 'foobaaaar' etc.
fooba{2,3}r	matches strings like 'foobaar', or 'foobaaar' but not 'foobaaaar'

Alternatives

You can specify a series of alternatives for a pattern using "|" to separate them, so that fee|fie|foe will match any of "fee", "fie", or "foe" in the target string (as would f(e|i|o)e). The first alternative includes everything from the last pattern delimiter ("(", "[", or the beginning of the pattern) up to the first "|", and the last alternative contains everything from the last "|" to the next pattern delimiter. For this reason, it's common practice to include alternatives in parentheses, to minimize confusion about where they start and end. Alternatives are tried from left to right, so the first alternative found for which the entire expression matches, is the one that is chosen. This means that alternatives are not necessarily greedy. For example: when matching foo|foot against "barefoot", only the "foo" part will match, as that is the first alternative tried, and it successfully matches the target string. (This might not seem important, but it is important when you are capturing matched text using parentheses.) Also remember

that "|" is interpreted as a literal within square brackets, so if You write [fee|fie|foe] You're really only matching [feio].

foo(bar foo)	matchs strings 'foobar' or 'foofoo'
---------------------	-------------------------------------

Subexpressions, Backreferences, Substitution

The bracketing construct (...) may also be used to define r.e. subexpressions. Subexpressions are numbered based on the left to right order of their opening parenthesis. First subexpression has number '1' (whole r.e. match has number '0'). Metacharacters \1 through \9 are interpreted as backreferences, and match previously matched subexpression within the expression. Matched subexpressions can also be accessed in the replace operation using metacharacters \$1 through \$9.

(foobar){8,10}	matches strings which contain 8, 9 or 10 instances of the 'foobar'
foob([0-9][a+])r	matches 'foob0r', 'foob1r', 'foobar', 'foobaar', 'foobaar' etc.
(.)\1+	matches 'aaaa' and 'cc'
(+)\1+	matches 'aaaa', 'cc', 'abab', '123123'

RegEx Examples

Expression	Replace	Description
(.*) (.*)	\$2, \$1	Switch two words around and put a comma after the resulting first word. Example: if input string is "John Smith", then output will be "Smith, John".
\b(d{2})-(d{2})-(d{4})\b	\$3-\$2-\$1	Find date-alike sequences in a format 25-10-2007, and invert them into 2007-10-25. Note: "\d" represents any digit in range of 0-9, so the sequence like 99-99-9999 will also match the expression.
\[.*?\]		Remove the contents of the [...] (square brackets), and the brackets too.

For more information:

- www.regular-expressions.info
 - Excellent site devoted to regular expressions. Nicely structured and with many easy-to-understand examples.
- www.regexpstudio.com
 - Freeware regular expressions library for Delphi. One can find more detailed information regarding this particular RegEx engine.

Command Line

Program supports several command line parameters.

Parameters

Parameter	Description
<files>	Paths to files and folders which will be automatically added to the program. Program's default settings will be used for adding folders. Masked paths can also be used, e.g. "C:\Pictures*.jpg"
/preset <preset> <files>	Load preset specified by a preset name or a full path. Optionally, paths to files/folders can be appended to the end on this command, and they will be automatically added to the program.
/rename <preset> <files>	Load preset specified by a preset name or a full path and proceed with Preview and Rename actions. Upon successful Preview and Rename operations, program will close automatically. Otherwise, graphical user interface will become visible and an appropriate error message will be displayed. Paths to files/folders can be appended to the end on this command, and they will be automatically added to the program.
/enqueue <files>	Add following files/folders to already running instance of the program. If no running instance is found - launch a new one.
/list <files>	Load a list of files/folders from the following list files.
/uninstall	Remove all manually turned on associations with the program, e.g. presets association. For advanced users only!

Examples

- **"ReNamer.exe" /enqueue "C:\Folder" "C:\Pictures*.jpg"**
 - This command will add to already running instance of the program contents of folder "C:\Folder" (depending on the default settings) and all *.JPG files from folder "C:\Pictures".

- **"ReNamer.exe" /preset "MyRules" "C:\Folder"**
 - This command will launch a new instance of the program, will load the preset with the name "MyRules", and will add contents of folder "C:\Folder" (depending on the default settings).
- **"ReNamer.exe" /rename "MyRules" "C:\Folder"**
 - This command will launch a new instance of the program, will load the preset with the name "MyRules", will add contents of folder "C:\Folder" (depending on the default settings), and will execute Preview and Rename operations (program will close upon successful completion of all operations).
- **"ReNamer.exe" /list "List1.txt" "List2.txt"**
 - Where "List1.txt" and "List2.txt" are lists of files (one per line), with absolute or relative paths (relative to the list file). The contained paths will be loaded into ReNamer.