|  |  |
| --- | --- |
| Regular Expressions |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bulk Rename Utility supports powerful Regular Expression processing. This allows you to enter a flexible Match expression, and a flexible Replacement expression, and the program will generate the appropriate name using these expressions. For example, you can use match and replace strings to swap two words in a filename, or remove numbers, or apply fixed formatting.  A full description of Regular Expressions is beyond the scope of this help file, but a wide range of resources is available on the internet. However, the syntax supported by Bulk Rename Utility is the same as that offered by PERL 5. The precise implementation is via the [PCRE Library](http://www.pcre.org), and full notes on the Perl Regular Expression syntax can be found [here](http://perldoc.perl.org/perlre.html).  We also have a section on our forum dedicated to [Renaming Files using Regular Expressions](http://www.bulkrenameutility.co.uk/forum/).  A quick summary of the syntax is:   |  |  | | --- | --- | | **Character** | **Usage** | | \* | Matches the previous character zero or more times | | + | Matches the previous character one or more times | | ? | Matches the previous character zero or one times | | . | Matches any single character except the newline | | ^ | Matches the start of the input | | $ | Matches the end of the input | | x|y | Matches either first or second character listed | | (pattern) | Matches pattern | | {number} | Matches exactly number times | | {number,} | Matches number, or more, times (note comma) | | {num1, num2} | Matches at least num1 and at most num2 times | | [abc] | Matches any character listed between the [ ] | | [^abc] | Matches all characters except those listed between the [ ] | | [a-e] | Matches any characters in the specified range (a,b,c,d,e) | | [^K-Q] | Matches all characters except in the specified range | | \ | Signifies that the next character is special or a literal. | | \b | Matches only on a word boundary | | \B | Matches only inside a word | | \f | Matches only on a form feed character | | \n | Matches only on a new line | | \r | Matches only on a carriage return | | \s | Matches only on a blank space | | \S | Matches only on nonblank spaces | | \t | Matches only on a tab | | \d | Matches any digit |     Replacements are usually performed on the basis of "components, and these are defined using \ notation, e.g. \1 matches the first element, \2 matches the second  **Example Regular Expression:**  Match:                (Louis Armstrong)(.\[0-9].)([A-Za-z ]\*)  Replace:        \1 \3  **EXAMPLE:**  Assume you have a file called Program Files, and you wish to swap the names around (e.g. Files Program). A Regular Expression which performs this task is :  **^([A-Z][a-z]\*) ([A-Z][a-z]\*)**  Let us break this down into components:  **^** This means start at the beginning of the string  **([A-Z][a-z]\*)** This is a single "group", which we will use later. What this says is that we want any capital letter, followed by any number of lower-case letters. The single capital letter is denoted by the **[A-Z]**, i.e. allow a letter in the range A to Z. The lower-case letters are denoted by **[a-z]** in the same way, followed by an asterisk. This means we can allow any number of letters.  We then allow a single space. If I had wanted multiple spaces I would probably have typed "space asterisk", or possible ( \*) to group.  We then have exactly the same again, i.e. we are denoting two words.  Notice we had two sets of brackets. Everything within each set of brackets is treated as a "grouping", and we refer to these groupings as **\1**, **\2**, **\3** etc.  So, lets say we wanted to swap around the two words in the filename. We would put:  **^([A-Z][a-z]\*) ([A-Z][a-z]\*)**  For the match string, and  **\2 \1**  As the replacement string. Of course, we're free to manipulate the replacements string as we like. For example, it would be quite valid to have:  The \2 which are used to run the \1  For the replacement string. This would result in:  The Files which are used to run the Program.  The above example is very precise. If we wanted to swap the first two words of a name, but keep the remaining text the same, we could put  **^([A-Z][a-z]\*) ([A-Z][a-z]\*)(.\*)**  **\2\1\3**  This says to create three groups: the first group is the first word, the second group is the second word, and the third group is everything that's left. |