```
(\d)(?:\u0020|\u0209|\u202F|\u200A){0,1}((m|mm|cm
|km|V|mV|µV|l|ml|°C|Nm|A|mA|bar|s|kV|Hz|kHz|M
Hz|t|kg|g|mg|W|kW|MW|Ah|mAh|N|kN|obr|min|µm
|µS|Pa|MPa|kPa|hPa|mbar|µF|dB)\b)
^\t*'.+?' => '
€(\d+)(,)(\d+)K
(")([a-z0-9])
```

Regular expressions

successes without magic

BDÜ Member of Bundesverband der Dolmetscher und Übersetzer

London, 16.11.2018

Structure

1. Introduction

Naming

- Regex = regular expressions
- In Word also known as “wildcard characters”

Definition

- A regular expression, regex or regexp (sometimes called a rational expression) is, in theoretical computer science and formal language theory, a sequence of characters that define a search pattern.[1]
- It can be used to find similar strings or replace them with other strings
- Widely used in CAT tools[2]

[1] Source: Wikipedia

[2] Mostly invisible to the user – in the parser or in the segmentation rules

2. Regex basics

Basic information

- Different Regex “Languages” (called “flavours”)
  - We're only dealing with .NET here
- Knowledge sources and tools
  - Website http://www.regular-expressions.info/
  - Software RegexBuddy
  - Software Notepad++
  - Blog by Paul Filkin (SDL)

Regex basics

- Each character represents itself
  - a matches “a” in b**a**sics
  - Ab matches “Ab” in **Ab**ba
  - etc.
- Some characters have special functions (and are called “metacharacters”)
  - Dot (.) matches any character
  - +, * are so-called “quantifiers”
  - Brackets () [] {} also have special functions

Simple expressions

- To find a word, we can simply write the word
  - **Test** matches “test”, but also “**test function**” or (under certain conditions) “**test**ing”
  - **T.ick** matches “**Trick**” and “**Track**”
  - **2.1** matches “**201**” or „**221**”, but also “**2,1**” or “**2.1**”

Simple expressions – upper case / lower case

- Regex may or may not be case-sensitive
  - **co** matches “**co**operation” and “cho**co**late” when “case insensitive”
  - **AB** matches “**AB**BA” but not “AbbA”, if “case sensitive”

Simple character classes

- The following special characters are frequently used
  - **\d** represents any digit
  - **\w** stands for a so called “word character”
    - A word character is any character from which words or alphanumeric expressions are formed – thus excluding dots, commas, spaces, etc.
  - **\w** matches “**201**”, “**2.1**”, “**AbbA**” and so on
  - **\d** matches “**201**”, “**2**,**1**”, “A**4**” etc.

Upper case / lower case

- The upper or lower case of metacharacters is of enormous importance
  - **\d** matches a digit
  - **\D** matches everything EXCEPT digits
  - **\w** matches word characters
  - **\W** matches everything EXCEPT word characters
  - **\s** matches so-called “whitespaces” – usually these are common spaces
  - **\S** matches everything EXCEPT spaces

Negation

- The so-called “caret” sign **^** is used for negation
- Must be used together with square brackets
- By negation characters can be excluded from the search
  - **[^\b]** matches everything except “b” – for example “**A**bb**A**” or “**abstraction**“

Functions of brackets – character ranges

- Square brackets [] can be used to enter character ranges
  - [a-c] matches all letters between a and c – “check out”, “play”
  - [1-3] matches digits between 1 and 3 – “201” or “728,1234”
  - [3-4a-i] matches digits 3 or 4 OR letters between a and i OR a combination of these characters – “728,1234”, “testing”, “playing“ or “A4”
    - The order of the matched characters does NOT reflect the order of the entered strings
  - a[d-s] matches “ballroom” or “Market“

Functions of brackets – character groups

- In addition to the above-mentioned function (character ranges), the rectangular brackets are used to define character groups
  - **[arst]** matches every letter from this group
    - “**A**bb**A**”, “**T**e**st** func**t**ion”, “**A**u**str**ali**a**“, “**Tra**ck”
  - The order of the matched characters is arbitrary
    - **[tras]** or **[rast]** will match the same examples

Functions of brackets – determining the № of occurrences

- Curly braces **{}** are used to specify the number of character occurrences
  - **\d{3}** matches “**201**” or “09.07.**2016**”
  - **\w{3}** matches both “**AbbA**” and “**realization**“ (and other contiguous groups of three letters or digits)
  - **\d{2}.\d{4}** matches “09.**07.2016**” or “7**28,1234**”

Functions of brackets – № of occurrences from – to

- To find a certain number of a character or of a defined string between a start and an end value, curly braces **{}** are also used together with the comma (**,**)
  - **{1,5}** matches the given character between 1 and 5 times
    - **b{1,5}** matches “A**bb**A”, “A**BB**A” and “a**b**sence”
  - **{2,}** matches 2 and more (at least 2) occurrences
    - **b{2,}** therefore only matches “A**bb**A”, “A**BB**A”, but not “absence”
  - **{0,2}** matches up to 2 occurrences
    - **\d{0,2}** always matches groups of up to two digits, “**20**1” or “2**01**” and of course any single digit in these strings

Functions of brackets – capturing group

- Like square brackets, round brackets () are used to define character groups
- However, the order of the characters entered is important here
  - Looking at “Spitfire” as example and using “tips” or “spit”
    - **(tips)** won’t find **ANYTHING**
    - **(spit)** will however only match “**Spit**fire”
- A character group enclosed in round brackets can also be used for “backward references”

Searching for metacharacters

- If a metacharacter such as dot (**.**) or parenthesis (e.g. **[**) is searched, the inverted slash (****, called “backslash”) must be used to cancel its meta functionality
  - To match the dot (**.**), **\.** must be entered
    - This matches “20**.**25” or “09**.**07**.**2016”
  - To match either **(** or **[**, you must “escape” them and insert these in a grouping parenthesis
    - **[\(\[]** matches **(]** in “*metacharacters are, for example, ., **(**), [**]**, {}*”

Summary of the basics

- Each character represents itself
- Character classes
  - **\d** digit
  - **\w** word character
  - **\s** space
  - **\W** non-word characters
- Parenthesis
  - **()** group with backward reference
  - **[]** character group
  - **{}** determining the number of occurrences
- Quantifiers
  - **+** one or more occurrences
  - ***** zero or more occurrences
- Negation
  - **^** negation (must be applied in **[ ]**)
- Searching for metacharacters
  - \ a metacharacter must be "escaped", this means placing a backslash in front of a metacharacter

3. Use cases

Searching for a date

- Our task is to find dates in the format **dd.mm.yyyy** in the example text
  - Date consists only of digits and dots
    - The required expressions are \d and \.
    - The matched expression shall consist of two digits, followed by a period, followed by two digits, followed by a period and four digits
    - We now try to design the expression together in RegexBuddy

Searching for a date

- Solution
  - **\d{2}\.\d{2}\.\d{4}**
- or
  - **\d\d\.\d\d\.\d\d\d\d**
- but
  - The date **6.3.1938** was NOT found

Searching for a date

- Modified solution
  - **\d{1.2}\.\d{1.2}\.\d{2.4}**
    - Matches **6.3.1938**, but also **14.09.18**

Searching for numbers

- Now the task is to find numbers in the same text
  - Numbers also consist of numbers and dots (for larger numbers) and, if necessary, a comma and other digits, but they have a different structure as date
  - Expressions to be used
    - **\d**, **\b**, **\.** and **comma** for itself

Searching for numbers

- Solution
  - **\d+\.\d{3}\b**
    - **\b** means “word boundary” – thus excludes another word character behind it
    - Word boundary is an important part of Regex

Greedy or lazy?

- Regular expressions with indefinite quantifiers (**+**, *****) are greedy
  - This will ensure matching as much characters as possible
    - **\w*** matches everything that consists of word characters – for example “**AbbA**” or “**201**”
    - In “20.01” both digit blocks will be matched
  - That makes the search imprecise

Greedy or lazy…

- To edit documents in CAT, the tags must be “masked”
  - For this purpose regular expressions are used
- Our task now is to find tags in our text
  - Expressions to be used
    - < and > for start and end of the tag
    - . for any character
    - Quantifiers
    - Grouping characters (brackets)
    - others…

Greedy or lazy…

- First attempt
  - < start of the tag
  - . any character
  - + at least one or more occurrences
    - or
  - * 0 or more occurrences
  - > end of the tag

Greedy

- Solution
  - **<.+>**
- Result
  - Almost all the text is highlighted because the expression is “greedy”
  - This means searching beyond the “<” sign, until after the “>” sign no further occurrence of “>” can be detected
- Unsuitable expression, because too much would have been masked

Lazy

- To make the expression “lazy”, the search must stop at the FIRST occurrence of “**>**”
- For this purpose, “**?**” is used

Lazy

- Solution
  - **<.+?>**
  - To be read as:
    - Search for any character following the “<” sign, until the first occurrence of the “>” sign is found

Lazy, search IN tags

- In the tag **<img src="selfhtml.png" alt="Selfhtml">** the text of the attribute “alt” shall remain translatable
  - Expressions to be used
    - **<** and **>** for start and end of the tag
    - **.** any character
    - Quantifiers
    - Grouping characters (brackets)
    - others…

Lazy, search IN tags

- Solution
  - **<img.+?alt="** is to be used for the first part of the tag
  - **">** represents the end of the tag

4. Search and Replace

Backward references

- When searching and replacing, it is often important to be able to reuse what has been found
- This is the purpose of the so-called backward references
  - The expressions to be searched for must be grouped using round brackets **()** ("capturing group")
  - When replacing, the nth group can be referenced with \n[1] and inserted again

[1] In SDL Trados Studio, the dollar sign **$** is used for the backward reference in the replacement function instead of the backslash \!

Search and Replace using Regex

- Our task now is to correct misspelled numbers and measurement units
- Expressions to be used
  - **\d** for digit
  - **Space** for itself
  - **Measurement units** for themselves
  - Groupings
  - Backward references
  - Other characters

Search and Replace using Regex

- Solution
  - Find numbers and measurement units without spaces with **(\d)(m|cm|mm|g|kg|°C|V|A)**
    - The expressions in **()** form the “capturing groups” and can be backreferenced
  - Replace with **\1/°\2**
    - **\1** inserts the first “capturing group”, **\2** the second one etc.
    - **°** stands for a non-breaking space (called also hard space or protected space)

Search and Replace using Regex

- This expression can be used to search for misspelled measurement units
  - Search for:
    ```
    (\d)(?:\u0020|\u0209|\u202F|\u200A){0,1}((m|mm|cm|km|V|mV|µV|l|ml|°C|Nm|A|mA|bar|s|kV|Hz|kHz|MHz|t|kg|g|mg|W|kW|MW|Ah|mAh|N|kN|obr|min|µm|µm|µS|Pa|MPa|kPa|hPa|mbar|µF|dB|gal)\b)
    ```
  - Replace with:
    ```
    \1°\2
    ```

Search and Replace using Regex

- Explanation
  - **(\d)**
    - any digit, is the capturing group number 1
  - **(?:\u0020|\u0209|\u202F|\u200A){0,1}**
    - 0 or 1 occurrences of any space character, but not the non-breaking space!
    - ?: causes this group to be a non-capturing group (to make the replacement easier)
  - **((m|mm|cm|km|V|mV|µV|l|ml|°C|Nm|A|mA|bar|s|kV|Hz|kHz|MHz|t|kg|g|mg|W|kW|MW|Ah|mAh|N|kN|obr|min|µm|µm|µS|Pa|MPa|kPa|hPa|mbar|µF|dB|gal)\b)**
    - Measurement units, separated by **|** (pipe), where the parentheses around the measurement units are used to search for them exactly as written
    - **\b** represents word end and the **outer parenthesis pair** forms the 2nd capturing group

Search and Replace using Regex

- The next task is to prepare special texts (such as specific XLIFF files) for translation
- To do this, certain text must be copied and pasted elsewhere
  - The text can contain letters, numbers, dots, commas and other characters!

Search and Replace using Regex

- In the example document there is only text present between the tags **<english>…</english>**
- The translation must however be entered between a new tag pair – **<target>…</target>**, where “target” corresponds to the language of the translation
  - The tags **<english>…</english>** with the text in between **must be kept**!
- The task is now to copy the text between the tags and “duplicate” it surrounded by appropriate tags for the target language

Search and Replace using Regex

- To be searched
  - **(<english>)(.+?)(</english>)**
    - The use of **?** causes the text to be found only between the opening and closing tag instead of between the first opening and the last closing tag, as this expression is "lazy"
- To be replaced
  - **\1\2\3\r\n\t\t<polish>\2</polish>**
  - To be read as
    - **\1\2\3** copies the <english> tags and the text in between
    - **\r\n** represents a new line, while **\t** represents a tabulator
    - **<polish>\2</polish>** returns the text (the second "capturing group") surrounded by the desired tags

5. Wildcard characters in Word

Wildcard characters in Word

- Very similar to Regex
- The main differences are the metacharacters
  - ***** in Word stands for any number of arbitrary characters and therefore has no counterpart in Regex
  - **?** represents a single character in Word
  - Word can also replace formatting

Wildcard characters in Word

- Searching in Word is more complicated
  - To find **€1,931K**, the expression should be like this
    - **€[0-9],[0-9]{3}K**

Wildcard characters in Word

- Replace for same text **€1,931K**
  - Search for: **€([0-9]),([0-9]{3})K**
  - Replace with: **\1.\2^sTsd. €**
    - **^s** represents a non-breaking space

Wildcard characters in Word

- Search for formatting
  - Leave the search field (“**Find what:**”) empty
  - Click “**More**” in the bottom of the “**Replace**” dialog in the “**Search and Replace**” box
  - Select the desired formatting from “**Format**” in the bottom left corner

Wildcard characters in Word

- Find and replace formatting
- Task: only certain text marked in colour (here red) should remain translatable
  - Leave the search field empty (“**Find what:**”), but the cursor shall be placed in it
  - Chose “**Font**” from the “**Format**” and select the font colour of the text to be replaced (here: “**Automatic**”)
  - Leave the replace field (“**Replace with:**”) also empty, but the cursor shall be now placed in it
  - Select “**Font**” from the format tab again and then mark the option “**Hidden**”
  - Replace all occurrences

Wildcard characters in Word

- Find and replace formatting
- Task: only certain highlighted text (here yellow) should remain translatable
  - Leave the search field empty, but the cursor must be placed in it
  - In the format tab select “**Highlight**”
  - Select “**Highlight**” again, this changes the search to “**Not Highlight**”
  - Leave the replace field also empty, but the cursor must be now placed in it
  - Select “**Font**” from the “**Format**” again and then mark the option “**Hidden**”
  - Replace all occurrences

6. Your questions

7. Regex reference

Regex reference

.	=	any character
\d	=	digit
\D	=	anything BUT digit
\w	=	word character
\W	=	anything BUT word character
\s	=	so called whitespace and line breaks and the like
\S	=	NO “Whitespace” – corresponds to [^\s]
\t	=	tabulator
\u1234	=	Unicode character with the code 1234
[a-z]	=	a single character from the range a-z
[abz]	=	one (two or all) of the characters a, b, z
[^a]	=	any character, but not “a”
\n	=	line feed (LF)
\r	=	carriage return (CR)
+	=	at least one or more occurrences
*	=	zero or more occurrences
?	=	the quantifier will be “lazy”
{n}	=	exactly *n* occurrences
{n,}	=	at least *n* occurrences
{n,m}	=	at least *n* and maximum *m* occurrences
{0,n}	=	no more than *n* occurrences
(abc)	=	the expression in brackets must be found exactly as typed
(abc)*	=	the expression in brackets must be found exactly as typed 0 or more times
(abc)+	=	the expression in brackets must be found exactly as typed 1 or more times
.+?a	=	search for any character until “a” (the first character behind “?” has been found (so called “lazy” search)
^	=	start of line (entered without brackets)
$	=	end of line or string end
\	=	is used to override the meta functionality
\\\\	=	matches \
\b	=	start or end of word
\r\n	=	line break in Windows
|	=	separator
?:	=	makes a group to a “non-capturing group”

Many thanks for your attention!

TransDocu®

BDÜ Member