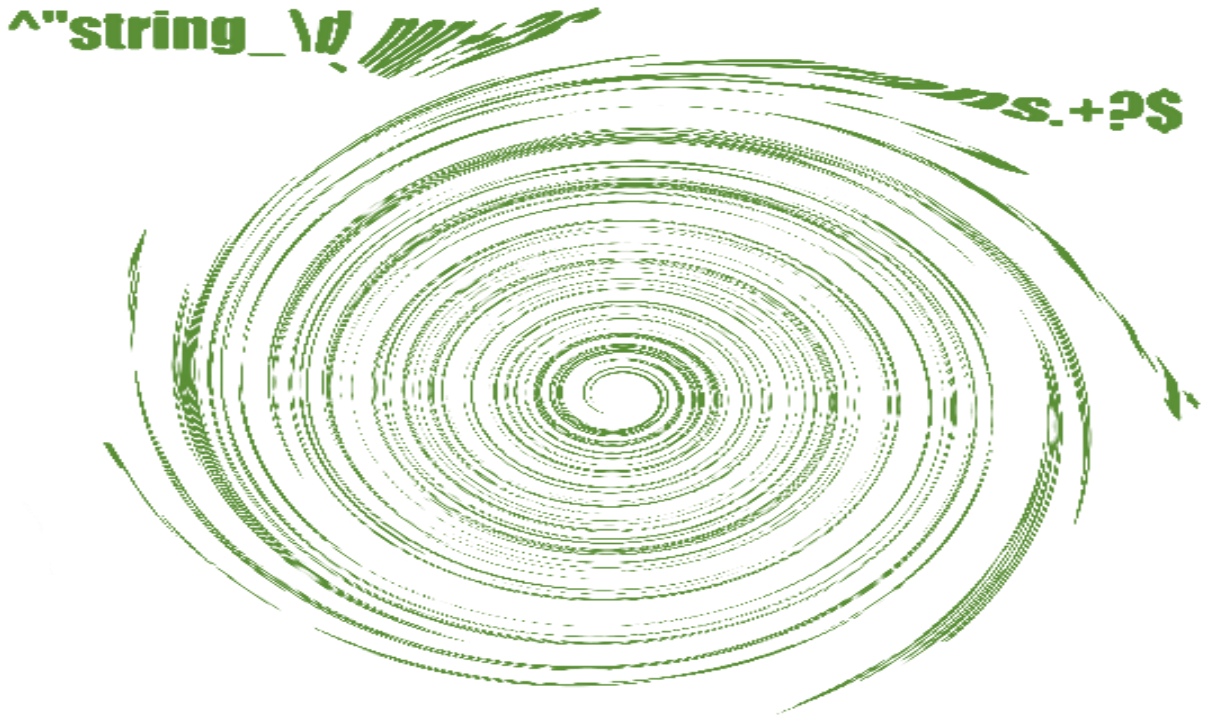

Regular Expressions In Modern CAT Tools

Test case

Rankings

Quick tutorials

By Stepan Korotaev

April 2021

About the author

A linguist by training, I started my career in the translation industry almost two decades ago. My first years were spent working on the quality assurance side—as an editor, and then editorial team lead. Later, I developed interest in translation technology, process optimization and commercial aspects of the business. At the time of writing, I am preparing to enter my new role as CTO at Effectiff, one of the leading Russian LSPs. My previous stints included Logrus and Neotech, two other heavyweights of the Russian translation industry.

Contacts

Email: stepkor@gmail.com

LinkedIn: Stepan Korotaev on LinkedIn

What did the cat say?

The cat, who obviously walks by herself, found home in the headers of this report (or maybe these are different cats, not just one, hard to say). One or many, she is somewhat annoyed by the world she has to live in and has a couple of things to say about it. Let me know what she said, and I will be happy to send you a little gift in acknowledgement of your uncanny expertise in the cat language.

TABLE OF CONTENTS

1. Prerequisites 1-12

2. Introduction 2-13

Caveats 2-15

Terminology note: CAT vs. TMS 2-16

3. Audience and scope 3-17

Target audience 3-17

Eligibility criteria for CAT tools to be included in the report 3-18

Notable absentees 3-19

4. Test case and key concepts 4-20

File formats 4-20

File preparation 4-21

Text file 4-21

The Big Three 4-26

Operations with text 4-27

Test case: step by step 4-29

Key terms 4-31

5. Regular expressions to be used 5-35

6. Evaluation criteria and scores 6-41

Hidden text note 6-44

7. Scoring tables 7-45

Total scores 7-46

File preparation 7-47

Operations with text 7-48

8. Evaluation Matrix 8-49

9. Takeaways 9-50

10. Quick tutorials 10-52

Across Translator Premium Edition v7.0 10-53

File preparation 10-54

Segment filtering 10-60

Search and replace 10-60

Alchemy Catalyst 2021 10-61

Text files 10-61

The Big Three 10-69

Segment filtering 10-70

Search 10-71

Replace 10-72

CafeTran Espresso 10.8.1 10-73

File preparation 10-74

Segment filtering, search and replace 10-81

Deja Vu X3 Professional 10-86

File preparation 10-86

Segment filtering 10-87

Search and replace 10-88

Fluency Now 10-89

File preparation 10-90

Segment filtering 10-91

Search and replace 10-93

MadCap Lingo 11 r2 10-94

Text files 10-94

The Big Three 10-99

Segment filtering 10-99

Search 10-100

Replace 10-100

Matecat, Nucleus 10-101

memoQ 9.5.8 translator pro 10-102
File preparation 10-102
Segment filtering 10-114
Search and replace 10-115
Memsource + Memsource Editor for Desktop 20.21.3 10-117
File preparation 10-118
Segment filtering 10-125
Search and replace 10-126
OmegaT 4.3.2 10-127
File preparation 10-127
Segment filtering 10-128
Search 10-129
Replace 10-130
SDL Trados Studio 2019 Professional 10-132
File preparation 10-133
Segment filtering 10-145
Search and replace 10-148
Smartcat 10-149
Swordfish IV 10-150
Segment filtering 10-151
Search and replace 10-152
translate5 10-153
File preparation 10-153
Segment filtering 10-153
Search and replace 10-154
Translation Workspace XLIFF Editor 10-156
Text files 10-157
The Big Three 10-168
Segment filtering 10-168
Search and replace 10-168

Wordbee 10-169
Text files 10-169
The Big Three 10-175
Segment filtering 10-176
Search and replace 10-177
Wordfast Pro 5 10-178
Text files 10-179
The Big Three 10-185
Segment filtering 10-186
Search and replace 10-187
XTM 10-188
11. memoQ vs. SDL Trados Studio: detailed comparison 11-190
Custom text file filters 11-192
Ease of creation 11-192
Flexibility 11-192
Ease of reuse 11-193
Preview quality 11-194
Custom regex configurations 11-195
Ease of creation 11-195
Flexibility 11-196
Ease of reuse 11-198
Preview quality 11-199
Retagging of the prepared project 11-200
Segment filter 11-202
Functionality 11-202
Usability 11-204
Search and replace 11-206
Functionality 11-206
Usability 11-206
Regex documentation quality 11-207
Winner 11-208

12. Appendix. Sample files 12-209

Text file 12-209

The Big Three 12-209

LIST OF FIGURES

Figure 4-1. Test case: Text file to be translated 4-21

Figure 4-2. Test case: Translatable part of string in text file 4-22

Figure 4-3. Test case: String containing article number 4-23

Figure 4-4. Test case: String containing upper-case UI element 4-25

Figure 4-5. Test case: Word document (DOCX) to be translated 4-26

Figure 4-6. Test case: Step by step 4-29

Figure 7-1. Scoring table (total scores) 7-46

Figure 7-2. Scoring table (breakdown by **File preparation** categories) 7-47

Figure 7-3. Scoring table (breakdown by **Operations with text** categories) 7-48

Figure 8-1. Evaluation Matrix: CAT tools by their regex functionality in **File preparation** and **Operations with text** categories 8-49

Figure 10-1. Across: Tagged SGML section 10-55

Figure 10-2. Across: Text file in Editor view 10-56

Figure 10-3. Across: Regex rule definition window 10-57

Figure 10-4. Across: Regex configuration for DOCX 10-59

Figure 10-5. Catalyst: ezParse rules creation 10-62

Figure 10-6. Catalyst: Regexes added in ezParse window 10-63

Figure 10-7. Catalyst: Text file filter preview 10-64

Figure 10-8. Catalyst: Locks & Keywords start window 10-66

Figure 10-9. Catalyst: Locks & Keywords settings window 10-66

Figure 10-10. Catalyst: Text file in Editor view 10-67

Figure 10-11. Catalyst: Segment filter settings 10-70

Figure 10-12. Catalyst: Search settings 10-71

Figure 10-13. Catalyst: Replace settings 10-72

Figure 10-14. CafeTran: Non-translatable glossary 10-75

Figure 10-15. CafeTran: Adding glossary (step 1: Dashboard) 10-76

Figure 10-16. CafeTran: Adding glossary (step 2: Settings) 10-77

Figure 10-17. CafeTran: New glossary selected on Dashboard 10-77

Figure 10-18. CafeTran: Text file in Editor view 10-78

Figure 10-19. CafeTran: Segment filter settings 10-82

Figure 10-20. CafeTran: Search settings 10-83

Figure 10-21. CafeTran: Replace settings 10-84

Figure 10-22. CafeTran: Undesired consequences of Replace & Edit 10-85

Figure 10-23. Deja Vu: Example of SQL filter settings 10-87

Figure 10-24. Deja Vu: Search and replace settings 10-88
Figure 10-25. Fluency Now: Advanced Settings window (file preparation stage) 10-90
Figure 10-26. Fluency Now: Segment filter settings 10-91
Figure 10-27. Fluency Now: Editor view with segment filter applied 10-92
Figure 10-28. Fluency Now: Search and replace settings 10-93
Figure 10-29. MadCap Lingo: New file filter creation 10-95
Figure 10-30. MadCap Lingo: Text file filter settings 10-96
Figure 10-31. MadCap Lingo: Text file in Editor view 10-97
Figure 10-32. MadCap Lingo: Segment filter settings 10-99
Figure 10-33. MadCap Lingo: Search settings 10-100
Figure 10-34. memoQ: Import with options start screen 10-103
Figure 10-35. memoQ: New file filter creation 10-104
Figure 10-36. memoQ: Text file filter settings—paragraph rules 10-106
Figure 10-37. memoQ: Text file filter preview 10-108
Figure 10-38. memoQ: Regex Tagger settings 10-109
Figure 10-39. memoQ: Regex configuration preview 10-110
Figure 10-40. memoQ: New cascading filter creation 10-111
Figure 10-41. memoQ: Choosing custom cascading filters for text file and Big Three 10-112
Figure 10-42. memoQ: Segment filter settings 10-114
Figure 10-43. memoQ: Editor view with segment filter applied 10-114
Figure 10-44. memoQ: Quick find and replace window 10-115
Figure 10-45. memoQ: Search and replace settings 10-116
Figure 10-46. Memsource: File import settings area 10-118
Figure 10-47. Memsource: Text file filter settings 10-120
Figure 10-48. Memsource: Text file in Editor view 10-121
Figure 10-49. Memsource: Regex configuration for XLSX 10-124
Figure 10-50. Memsource Editor for Desktop: Segment filter settings 10-125
Figure 10-51. Memsource Editor for Desktop: Search and replace settings 10-126
Figure 10-52. OmegaT: Segment filter settings 10-128
Figure 10-53. OmegaT: Search settings 10-129
Figure 10-54. OmegaT: Replace settings 10-130
Figure 10-55. OmegaT: Replace Next functionality 10-131
Figure 10-56. Trados Studio: New file type creation (step 1) 10-134
Figure 10-57. Trados Studio: New file type creation (step 2) 10-135
Figure 10-58. Trados Studio: Document structure node in file types tree 10-136

Figure 10-59. Trados Studio: Text file filter settings (document structure) 10-136
Figure 10-60. Trados Studio: Text file filter settings (inline tags) 10-137
Figure 10-61. Trados Studio: Text file filter preview 10-138
Figure 10-62. Trados Studio: Custom file filter moved up on file types list 10-139
Figure 10-63. Trados Studio: Embedded content sections in file types tree 10-142
Figure 10-64. Trados Studio: Regex configuration for PPTX 10-143
Figure 10-65. Trados Studio: Segment filter settings (default filter) 10-146
Figure 10-66. Trados Studio: Advanced Display Filter location 10-146
Figure 10-67. Trados Studio: Segment filter settings (advanced filter) 10-147
Figure 10-68. Trados Studio: Search and replace settings 10-148
Figure 10-69. Swordfish: Segment filter settings 10-151
Figure 10-70. translate5: Search settings 10-154
Figure 10-71. TWS: Text file filter creation (step 1) 10-158
Figure 10-72. TWS: Text file filter creation (step 2) 10-159
Figure 10-73. TWS: Appending child rule in custom text file filter 10-159
Figure 10-74. TWS: Text file filter settings (delimiters) 10-160
Figure 10-75. TWS: Text file filter settings (block) 10-161
Figure 10-76. TWS: Text file filter settings (final view) 10-162
Figure 10-77. TWS: New text file filter set as default 10-163
Figure 10-78. TWS: Saving new text file filter settings 10-164
Figure 10-79. TWS: Text file in Editor view 10-165
Figure 10-80. TWS: Loading of previously saved .lmx settings file 10-166
Figure 10-81. Wordbee: File filter selection 10-170
Figure 10-82. Wordbee: Adding new rule for text file filter 10-171
Figure 10-83. Wordbee: Text file filter settings 10-171
Figure 10-84. Wordbee: Text file in Editor view 10-172
Figure 10-85. Wordbee: Text file filter preview 10-173
Figure 10-86. Wordbee: Regex configuration settings for Big Three 10-175
Figure 10-87. Wordbee: Segment filter settings 10-177
Figure 10-88. Wordfast: Rules file 10-180
Figure 10-89. Wordfast: New text file filter creation (step 1) 10-181
Figure 10-90. Wordfast: New text file filter creation (step 2) 10-182
Figure 10-91. Text file in Editor view 10-183
Figure 10-92. Wordfast: Text file filter selection 10-184
Figure 10-93. Wordfast: Segment filter settings 10-186

Figure 10-94. Wordfast: Search and replace settings 10-187
Figure 10-95. XTM: Analysis Manager area 10-189
Figure 11-1. memoQ vs. Trados Studio: Comparison criteria 11-191
Figure 11-2. memoQ: Using Regex Tagger in Editor view 11-200
Figure 11-3. memoQ: Segment filter to find source with upper-case words where target does not have such words 11-203
Figure 11-4. Trados Studio: Segment filter to find source with upper-case words where target does not have such words 11-203
Figure 11-5. memoQ vs. Trados Studio: Comparison results 11-208

LIST OF TABLES

Table 4-1. Key terms 4-31

Table 5-1. Regexes to be used in test case 5-35

Table 6-1. Top scores for each evaluated category 6-43

1. Prerequisites

Readers are expected to be familiar with the concepts of computer-assisted translation (*CAT*) tools and regular expressions (*regexes*).

CAT tools are at the heart of today's translation industry. I assume you have at least some basic knowledge of this technology. If not, it would be useful to learn a thing or two about it before continuing with the report. You can start with the Wikipedia page on computer-assisted translation.

If you do not know what regexes are, I also encourage you to read up a little on the topic. It will be a time well spent: you will definitely benefit from learning more about regexes, regardless of your typical role in the translation workflow. Any search engine of your choice will give you dozens of quality links, just type *regular expressions* in a search field. I can also recommend a very good article by Riccardo Schiaffino: Regular Expressions: An Introduction for Translators.

If you do not have time for all this, you will still be able to keep up with the report but more technical details might be a bit hard to understand.

2. Introduction

Regular expressions, commonly known by a shorter name *regexes*, are a somewhat niche and yet very important part of the translation and localization process. They are mainly used at the file preparation stage to reduce word count and protect non-translatable content, and later during linguistic (translation/editing/proofreading) steps to quickly filter segments based on complex criteria and perform sophisticated search and replace operations. Skillful application of regexes can increase productivity and help significantly reduce cost, but this potential is often overlooked, especially at smaller translation companies. As a result, regexes are arguably one of the most underestimated and underutilized technologies in the translation industry. Hopefully, this report will contribute to improving this situation.

The report analyzes how different CAT tools' regex capabilities can be used to crack a particular *test case* (see the Test case and key concepts chapter). The test case encompasses two important components of the translation process: technical preparation of files and operations with text at linguistic stages (segment filtering, search and replace).

The report is both a functionality overview and a comparison study making an attempt to rank CAT tools based on their regex prowess. For that, a scoring system was developed (see the Evaluation criteria and scores chapter). The ranking part is summarized in the scoring tables and the Evaluation Matrix.

Additionally, the report can be viewed as a tutorial covering several common techniques in file preparation. The general methodology is outlined in the Test case and key concepts chapter, whereas the Quick tutorials chapter contains sections dedicated to individual tools, with explanation of their scores and the specifics of their regex implementation.

In the last chapter of the report, a deeper, more granular comparison is made between the two market leaders, memoQ and SDL Trados Studio.

Caveats

My experience with different CAT tools may and does vary. Some of them I have been using for many years; others were totally new to me when I started compiling this report. I believe I have invested a reasonable amount of time and effort to research capabilities of the tools I was less familiar with, but I could easily have missed important nuances. Feel free to send me your comments, improvements and disagreements (see my contacts at the beginning of the report). Any feedback is greatly appreciated.

I should also mention that this report was not intended to be a comprehensive review of all regex-related functionality. The research was built around a very specific test case (see the Test case and key concepts chapter). While being, in my opinion, representative of a broad set of popular applications and scenarios, this test case does not cover all possible ways of working with regular expressions in CAT tools. For instance, the report is not concerned with the use of regexes to define segmentation and QA rules, both of which are quite widespread applications.

Finally, due to the wide-ranging nature of the report, not all of the reviewed CAT tools' editions were current even at the time of publication. As a most notable example, the report covers SDL Trados Studio 2019, though the tool's current version is 2021.

Terminology note: CAT vs. TMS

In recent years, CAT tools have been increasingly often called *translation management systems* (TMS). To a degree, it is the reflection of the fact that their functionality has gotten much more complex and comprehensive and may now include workflow management, billing, collaboration environment, etc. Another driver behind it may be the eternal marketing need to always come up with new names for more or less same entities.

For the purposes of this report, I do not draw a distinction between TMS and CAT and use the latter name to designate all systems covered in the report.

3. Audience and scope

Target audience

This report may be of value to project managers, translators, editors, proofreaders, QA checkers, engineers and other translation and localization professionals. It can also be used as reference material in localization courses taught at educational institutions. For that, no special permission is required, but I would be grateful to be notified of such use.

Eligibility criteria for CAT tools to be included in the report

As a starting point, I used an excellent Nimdzi Language Technology Atlas.

To qualify for the report, a CAT tool must have met the following criteria:

- be *Windows-compatible*;
- be a *general-purpose tool* supporting both text files and MS Office formats (pure localization tools were not considered);
- be a *standalone integrated tool* allowing the upload of source files, saving them in one of translation/localization formats (e.g., XLIFF-based) before the translation work starts and, finally, recreating translated documents in their initial formats (add-ins and extensions to Word, OpenOffice, etc. were not considered either);
- have a *graphical user interface*;
- be *current*, not abandoned, with a trial version available;
- be a *desktop or cloud* solution—server-based tools relying on scenarios where most users work with functionally limited client versions were not considered (a half-exception to this restriction are Translation Workspace by Lionbridge and Across Translator Premium Edition—see details in corresponding sections).

Another limitation to bear in mind was that, with all tools, I only researched their standard configuration. No custom plug-ins, add-ins, on-demand templates provided by support specialists, etc. were allowed as a way to achieve a goal. However, if a task could be solved via a manual modification of external settings or properties files, in a number of cases it was deemed fair game.

An *integrated* tool means that all operations must be performed within one application, without sending files from one environment to the other. The exception was made for Memsource where the standalone Editor for Desktop was considered along with the cloud-based project management environment. I hesitated if I should have made this exception but eventually decided to include Editor for Desktop based on following considerations: *a*) it is a Memsource product, not a third-party tool, *b*) it is free, *c*) it integrates seamlessly with the cloud and *d*) it is very widely used by editors and translators working with Memsource projects.

Notable absentees

STAR Transit NXT: Only the Freelance version with limited functionality was available for a free trial so I had to skip this long-standing proprietary tool.

4. Test case and key concepts

File formats

Imagine we have four source files for translation—a plain text file and three documents in popular MS Office formats: a DOCX (a Word document), an XLSX (an Excel spreadsheet) and a PPTX (a PowerPoint presentation). I will be calling the Office documents the *Big Three* for brevity. Why these particular formats? Including plain text and the Big Three gives us a reliable filter to pick out general-purpose CAT tools supporting both localization scenarios (plain text) and more traditional translation workflows (the Big Three). Most small and medium-sized translation companies, regardless of their main specialization, every so often get to tread each of these territories, and very few are prepared—be it technologically, financially, or both—to change their toolset depending on the circumstances. Our selection of formats helps us keep our focus on versatile environments suitable for a wide range of translation/localization tasks.

Our test case encompasses two main stages: *file preparation* (usually a project manager's job) and *operations with text* (more likely to be performed by linguists—translators, editors or proofreaders).

File preparation

Text file

The text file goes as follows:

```
"string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."
"string_2_non_translatable" = "If scared, run."
"string_3_translatable" = "Observe commands on the WATCHYA pane."
"string_4_translatable" = "In case of any disobedience, prepare to press the DESTROY
button."
"string_5_translatable" = "Detach parts V45-36-12 and A0-34-FG6 (for good measure). Also
detach yourself."
"string_6_translatable" = "BEWARE!"
"string_7_non_translatable" = "String 7 should never be translated."
```

Figure 4-1. Test case: Text file to be translated

Though a made-up layout, it closely resembles many formats you may come across in the localization industry, such as software strings. Each string in the file shown above has an *ID section* (or *ID substring*) on the left and a translatable part on the right. An ID substring includes an indication of whether the string should be translated at all. If an ID substring has a *non_translatable* component in it then the string should not be translated. Otherwise, we need to translate the right part of the string (content between straight double quotes).

As an example, two of the strings are shown below, highlighted for clarity:

"string_2_non_translatable" = "If scared, run."

"string_3_translatable" = "Observe commands on the WATCHYA pane."

The only text to be translated is yellow.

Figure 4-2. Test case: Translatable part of string in text file

Our first task is to tell a CAT tool that some of the strings are not to be translated. The best way to achieve this would be to create a separate rule, which, when applied to any file of this type, would automatically block non-translatable strings from being processed. Our current file is small, but imagine we have thousands of much larger files, all with the same layout, coming for translation at different times. Of course, we would not want to set up the same configuration over and over again. There should be an easily applicable rule. Such rules come by different names in different CAT tools. I will call them *file filters* as this seems to be the most common designation. An important thing to note is that, though custom file filters are usually based on existing system configurations (for example, on a predefined processor of plain text files), a newly created filter should be clearly distinct from a default option. We need the flexibility to choose between our custom filter and a default one for any new text file that we might want to translate.

Excluding non-translatable strings from our text file is an obvious first step but we are not done yet.

First, even in translatable strings there is a non-translatable left part (an ID section). We should exclude this text from translation too. Since we assume that an ID section is always present in files of this type, it seems reasonable to include this new rule in our file filter along with the rule to leave out non-translatable strings.

Second, at a closer look at the translatable part, we can see article numbers like *A0-34-FG6*.

```
"string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."
```

Figure 4-3. Test case: String containing article number

Why are they important? Because we know these sequences do not have translatable content within. They must remain the same after translation. Consequently, we would like to *a*) protect them from occasional modification by a translator and *b*) not pay our translator for this text. Point *b*) may come across as a bit petty, but again: imagine we have tens of thousands such numbers. None of them needs translation, but together they amount to a lot of money as their total word count becomes pretty impressive.

A very natural instinct would be to protect this content from translation by putting it in so-called *inline tags*. A very positive side effect of applying this approach would be improved *translation memory leverage*—segments differing only in tags are considered a much higher *fuzzy match* than those with text differences. Going deeper into this is beyond the scope of this report; in case you start feeling a bit lost, feel free to contact me with questions.

Unlike our previous rules, the rule to put article numbers in tags may not be universal. It is quite possible that some files of the same type may have structurally similar elements that *should* be translated. For example, they can contain upper-case words with a numerical part spelled with a hyphen, like *PHASE-4*. So we would only like to apply our rule when it suits us. The rule, therefore, should not be part of our file filter but rather a standalone configuration that can be *added* to a file filter whenever necessary. I will call such rules *regex configurations*.

On the whole, file filters usually broadly define what should be translated (*boundaries* of the translatable text), and regex configurations help *protect* (or *tag*) non-translatable parts within those boundaries. Implementation specifics, though, may vary greatly from tool to tool.

Once our non-translatable sections and article numbers are taken care of, we are left with only one text peculiarity, namely upper-case words like *WATCHYA* denoting UI elements.

```
"string_3_translatable" = "Observe commands on the WATCHYA pane."
```

Figure 4-4. Test case: String containing upper-case UI element

We do not want to protect these words from translation, but we will use them in our search and replace scenario. We will want to replace each of such occurrences with itself followed by the same word in parentheses, so *WATCHYA* should become *WATCHYA (WATCHYA)*. To justify this scenario, let us imagine a customer coming to us later in the process with an additional requirement to put translations for all UI elements in parentheses placed after original terms. For instance, *WATCHYA* should become *WATCHYA (ПОД КОЛПАКОМ)* if we are to translate into Russian—note that the Russian translation in parentheses follows the original term instead of replacing it. We can just communicate this new requirement to our translators and hope for the best, but a better strategy would be to add untranslated versions after all UI elements in advance to create the desired layout and reduce the amount of manual work on translators' part.

The Big Three

The Big Three are identical in terms of their content and very similar to the text file, except they do not have non-translatable strings or ID sections. Here is the Word document as an example (red and blue underlines mark places where Word disapproves of spelling or stylistic choice):

Attach part A0-34-FG6 to evil manifold N on the left.

Observe commands on the WATCHYA pane.

In case of any disobedience, prepare to press the DESTROY button.

Detach parts V45-36-12 and A0-34-FG6 (for good measure). Also detach yourself.

BEWARE!

Figure 4-5. Test case: Word document (DOCX) to be translated

With the Big Three, we do not have to bother with non-translatable strings, but we still would like to protect article numbers. So we do not need a full-fledged file filter here, just a regex configuration to put away article numbers.

Operations with text

At this stage, we are no longer concerned with our source files' type or format. Out CAT tool has already extracted text from whatever we uploaded into it so now we are dealing with *segments* (source—target pairs) rather than source files. If you are a translator, this is where your work starts. However, to simplify things, let us imagine that the text is already translated, and it is now an editor who is working with it. **Accordingly, I will call a view in a CAT tool where translators and editors can work with segments *the Editor view*. I will consistently use this designation throughout the report, even though different CAT tools may have their own names for this view.**

As an editor, you need functionality to quickly *filter* segments so that only those meeting your criteria are shown on the screen. For example, you may want to display only segments containing a particular term (or, vice versa, not containing it). It can be helpful when you need to replace this term with something else or make sure it is translated consistently everywhere in the document. In this and many other cases, the ability to filter segments leads to very substantial gains in productivity and accuracy.

Our first task will be to pick out segments with upper-case UI elements like *WATCHYA*. We will want only these segments on the screen, with all others hidden from view. This is exactly what filtering does for us.

Then, we will test *search and replace* functionality. We will be searching for same upper-case UI elements. The search part in some CAT tools is merged with filtering, meaning that a user can filter segments by running a search operation. It is okay as long as we can achieve our goals. **For the search part, we will need the ability to move between occurrences using the Next/Previous arrows or similar functionality.** This is what separates search from filtering: search is dynamic and lets us navigate from one entry to the other whereas filtering gives us a static list of all occurrences meeting our criteria. **As for the replace part, we will try to perform the operation described earlier: to replace each UI element with itself followed by the same element in parentheses.** As an example, we will want *WATCHYA* to become *WATCHYA (WATCHYA)*.

Test case: step by step

To sum it all up, in each of the reviewed CAT tools, we will try to do the following:

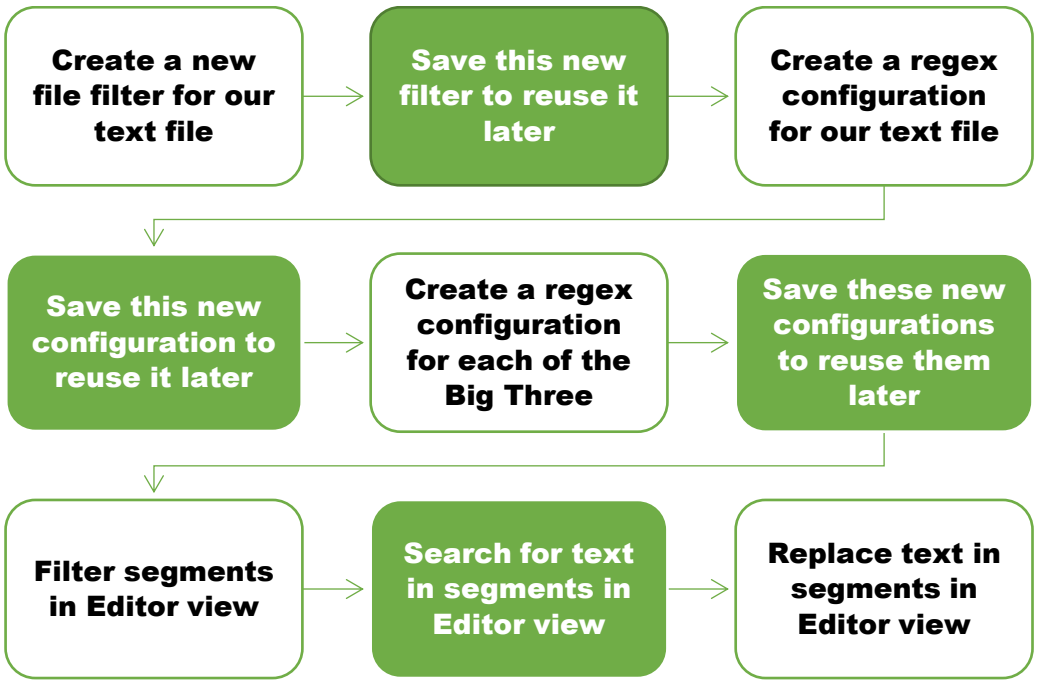

Figure 4-6. Test case: Step by step

Additionally, we will check if a CAT tool allows us to *preview* what the source text is going to look like in the Editor view with our filters and configurations applied. Without a preview, we have no way of knowing that other than going through all the steps of project creation to finally see the result of our rules' application in the Editor view. Since regexes can be seriously tricky, it is very likely that we might not be able to get all the rules right on the first try and would have to adjust them based on what we see in segments. We might end up having to recreate the project many times over before we achieve the results we want. The preview functionality available at the file preparation stage saves a lot of time and effort obviating the need for the tedious process of project recreation. Unfortunately, not too many CAT tools offer this valuable feature, and, if offered, it is often limited in terms of its functionality, usability or compatibility with different file types.

Key terms

Below is a glossary of key terms used throughout the report.

Table 4-1. Key terms

Article numbers	Groups of upper-case Latin letters and digits connected with hyphens. Example: *A0-34-FG6*
Custom (filter, configuration, etc.)	Modified by user of a CAT tool.
Editor view	View in a CAT tool displaying segments and containing key translation functionality. Translators and editors mostly work with this view.
File filter	Set of rules defining the way text for translation is extracted from files of a certain type. E.g., a file filter for *.txt* defines the text extraction rules for files with the *.txt* extension.
File preparation	Procedure to choose and/or modify file filters. Usually, is carried out by a project manager or localization engineer.
ID section (*or* ID substring)	Portion of text between straight double quotes on the left side of each string contained in the text file referenced in the report. ID section contains information on whether a string should be translated.

Inline tags	Tags appearing within segments next to text (as opposed to tags taking up the whole of a segment). Entities that are typically put in inline tags may go by the names of *placeables*, *non-translatables*, etc.
Linguistic capabilities/side	CAT tool's functions supporting operations with text.
Managerial capabilities/side	CAT tool's functions used in file preparation.
Operations with text	Segment filtering and search and replace collectively.
Preview	Ability to see how regex rules will affect text extracted for translation directly from the window where rules are defined.
Project	All source files, translation memories, glossaries, settings, etc. that are chosen, defined or created by a project manager to carry out a translation job. The simplest project possible would consist of just one source file. Most modern CAT tools are project-oriented, which means projects are created automatically whenever a new file or batch of files are translated.
Regex configuration	Set of regex rules applying additional conditions to text extracted for translation based on a file filter.

Reuse	Ability to save custom file filters and/or regex configurations to reuse them later with other projects.
Search and replace	Capabilities to search for text and replace it in the Editor view.
Segment	Portion of text extracted for translation paired with the translation of this text. Segments' boundaries are defined by a CAT tool's segmentation rules (which are beyond the scope of this report). In most cases, one segment contains one source sentence and one target sentence (i.e., the translation of the source sentence). In a less strict sense, the term *segment* may also be used in this report to denote only one sentence (source or target) instead of a source—target pair.
Segment filter(ing)	Settings in the Editor view allowing to hide segments not meeting filtering criteria. In this report, only filters supporting text criteria (= allowing a user to type a text query and then filtering based on this query) are considered.
Source	In a general sense: any initial format, language or state. E.g., a *source* language is a language we translate from.
Source file	File to be translated.

Tags	Mechanism in CAT tools allowing to mark portions of text or *protect* text from translation. *Paired* tags serve as boundaries between which a certain rule should be applied (e.g., the start and end of the *italic* font). More often than not, paired tags do not replace text—a sentence reads the same even if all such tags are ignored. In contrast, *single* tags usually represent a portion of text, and their omission would lead to loss of meaning. In most of this report's use cases, it is single tags that would be applied by CAT tools to implement custom regex rules aimed at protecting portions of text from translation. The term *text protection* stems from the fact that, once the text is converted into a tag, it cannot be modified by a translator (and, thus, becomes *protected*). See also inline tags.
Target	In a general sense: any final format, language or state. E.g., a *target* language is a language we translate into.
The Big Three	The DOCX (Microsoft Word), PPTX (Microsoft PowerPoint) and XLSX (Microsoft Excel) formats collectively.

5. Regular expressions to be used

Regexes come in different flavors. Most CAT tools use Java- or .NET-based implementations. They are quite similar, but sometimes a slight modification was needed to adjust our default rules to a particular CAT tool's requirements. So do not be surprised if regexes on some screenshots will look a bit different from the default versions below (which use the .NET syntax).

Table 5-1. Regexes to be used in test case

Regex	Comment
**^"string_\d_non.+?$** Meaning: Whole non-translatable line, from the start of line (*^*) to the end of line (*$*). Example (yellow denotes content to be captured by the regex): "string_2_non_translatable" = "If scared, run."	To consider cases where the number before the *_non* component is greater than 9, the expression should be changed to *^"string_\d+_non.+?$* Note the *+* character after *\d* meaning that there can be any number of digits in a row.

Regex	Comment
**^"string_\d_trans.+?$** Meaning: Whole translatable line, from the start of line (^) to the end of line ($). Example: "string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."	To consider cases where the number before the *_trans* component is greater than 9, the expression should be changed to *^"string_\d+_trans.+?$*
(?<= = ").+?(?=") Meaning: Content between the sequence = ″on the left and a straight quote (″) on the right. Example: "string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."	This regex captures the translatable part of a string (content on the right between straight double quotes). The regex uses so-called *lookarounds* to check the text before and after the part to be captured. The sequence of an equal sign, single space and straight double quote is expected before it, and a straight double quote is expected after it. Text is captured if both conditions are met.

Regex	Comment
^"string_\d_trans.+?" = " Meaning: Content from the start of line to the sequence = " (including this sequence). Example: "string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."	This regex captures an ID section of a translatable string and the sequence of a single space, equal sign, another single space and straight double quote that follows it.
"$ Meaning: Straight double quote before the end of line. Example: "string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."	

Regex	Comment
([A-Z0-9]+-[A-Z0-9]+)(-[A-Z0-9]+)* Meaning: ✓ Sequence of upper-case Latin letters or digits *followed by* ✓ a hyphen *followed by* ✓ another sequence of upper-case Latin letters or digits *optionally followed by* ✓ any number (including 0) of sequences, each consisting of: o a hyphen *followed by* o a sequence of upper-case Latin letters or digits	This regex captures article numbers. Article numbers must contain at least one hyphen between alphanumeric groups consisting of upper-case Latin letters and digits. E.g., *12-A3* is a legitimate article number, but *ASD4* is not (no hyphen). The number of alphanumeric groups is not limited, but each of them, except for the very first, should be preceded by a hyphen. *0-9* is equivalent to *\d* and means any digit. Usually, there is no difference between the two notations, but some tools may only recognize one of them and reject the other.

Regex	Comment
\b[A-Z]{2,}\b Meaning: Any word consisting of upper-case Latin letters only and having at least two such letters.	\b denotes a word boundary. E.g., \b*ham*\b captures *ham* but does not capture anything in *sham* (the first word boundary in the latter is at *s*, not *h*). In contrast, *ham*\b would be found in *sham* as this time only the second word boundary should match (at *m*). Word boundaries may not be supported in some CAT tools.

Regex	Comment
(\b[A-Z]{2,}\b) $1 ($1) Meaning: First line (*find operation*): find any word consisting of upper-case Latin letters only and having at least two such letters. Second line (*replace operation*): replace the found word with itself followed by itself in parentheses. Example: Observe commands on the WATCHYA pane. *(after replacement)* Observe commands on the WATCHYA (WATCHYA) pane.	*$1* is a so-called *group backreference* capturing the text matched by the regex inside the first (and, in our case, only) pair of parentheses on the search line. This text can then be reused in a replacement string. Instead of *$*, a backslash (\) can often be used to reference a group.

6. Evaluation criteria and scores

Based on the test case description, I evaluated CAT tools by presence and quality of functionality to:

- define and reuse custom text file filters;
- preview text files with a custom file filter applied;
- define and reuse custom regex configurations for text files and the Big Three;
- preview files with custom regex configurations applied;
- filter segments based on regexes in the Editor view;
- search for text fragments within segments based on regexes in the Editor view;
- perform replace operations based on regexes in the Editor view.

Scores in each category range from 0 (cannot be done) to 10 (perfect implementation). The exception is the score for the ability to save custom regex configurations, which can only be as high as 5. The reason is that it is a less important feature than the ability to save custom file filters. Unlike file filters, configurations may change quite often, and in many cases it is more convenient to just have a list of all key regexes at hand and apply a fitting subset of them manually to a new translation batch instead of relying on an existing configuration.

Of course, scores are subjective though I did my best to leave my personal preferences out of the evaluation. As a general rule, if a task could be solved in a satisfactory way, I assigned it 10 (or 5 for saving regex configurations) and only reduced the score if the process seemed too complicated, unintuitive or outright nerdy.

In case of merged functionality, as when there is no distinction between file filters and regex configurations, I awarded a full score in one category and half that in the other. For example, if a tool allows to create a custom file filter with additional rules for inline tags (built into the filter) but does not allow to set up *separate* regex configurations to be *combined* with this filter, the maximum score is 10 for the file filter and 5 for the regex configuration. The reuse category score for a regex configuration in such cases is also halved (maximum value: 2.5).

The perfect score is 140. It can be achieved by scoring top marks in all categories.

Table 6-1. Top scores for each evaluated category

Custom file filter for text files—creation	10
Custom file filter for text files—preview	10
Custom file filter for text files—saving and reusing	10
Custom regex configuration for text files—creation	10
Custom regex configuration for text files—preview	10
Custom regex configuration for text files—saving and reusing	5
Custom regex configuration for DOCX files—creation	10
Custom regex configuration for DOCX files—saving and reusing	5
Custom regex configuration for XLSX files—creation	10
Custom regex configuration for XLSX files—saving and reusing	5
Custom regex configuration for PPTX files—creation	10
Custom regex configuration for PPTX files—saving and reusing	5
Custom regex configurations for the Big Three—preview	10
Segment filtering in the Editor view	10
Search in the Editor view	10
Replace in the Editor view	10

There is only one preview category for the Big Three as usually preview is available for either all of the Big Three formats or none. It is not uncommon, however, for preview functionality to be supported for text files but not for MS Office formats.

Hidden text note

A special note concerning Word documents (DOCX): when processing such documents, most CAT tools allow to control the extraction of what is called *hidden text*. By default, hidden text is not extracted for translation and is put in inline tags. Word itself is equipped with limited yet usable regex functionality (so-called *Word wildcards*). It means that we can hide content in Word using wildcards, and that would be more or less equivalent to applying a custom regex configuration in a CAT tool. Certainly, it is a kludgy way of doing things, but it is better than nothing. CAT tools that do not have DOCX-related regex functionality but protect hidden text during content extraction were awarded 2 points in the *Custom regex configuration for DOCX files—creation* category.

7. Scoring tables

Anticlimactic as it is, below are the final standings. **The participating tools were ranked based on their capabilities relevant to the test case used in the report.** The tables below do not reflect all possible regex-related applications of the included CAT tools. See the Caveats section to learn more about restrictions of the test case.

The cloud solutions are in green font. To learn more about each of the tools boasting at least some rudimentary regex functionality, go to a corresponding section in the Quick tutorials chapter.

Total scores

Tool	Scores		
	File preparation	Operations with text	Total
memoQ 9.5.8 translator pro	106	30	136
SDL Trados Studio 2019 Professional	94.25	30	124.25
Alchemy Catalyst 2021	96.5	27	123.5
Wordbee	87.5	8	95.5
CafeTran Espresso 10.8.1	60	20	80
Across Translator Premium Edition v7.0	80	0	80
Wordfast Pro 5	29.5	30	59.5
Memsource + Memsource Editor for Desktop	28	24	52
MadCap Lingo 11 r2	17	20	37
OmegaT 4.3.2	2	27	29
Translation Workspace XLIFF Editor	23.75	5	28.75
Fluency Now	0	25	25
Deja Vu X3 Professional	2	22	24
Swordfish IV	0	15	15
translate5	2	6	8
XTM	2	0	2
Nucleus	2	0	2
Smartcat	2	0	2
Matecat	2	0	2
Crowdin	0	0	0

Figure 7-1. Scoring table (total scores)

File preparation

Tool	File preparation													Total
	Text files						Big Three						Preview	
							Word		PowerPoint		Excel			
	File filter	Preview	Reuse	Regex config	Preview	Reuse	Regex config	Reuse	Regex config	Reuse	Regex config	Reuse		
memoQ 9.5.8 translator pro	10	10	10	10	8	5	10	5	10	5	10	5	8	106
Alchemy Catalyst 2021	10	10	10	10	7	5	10	5	10	5	5	2.5	7	96.5
SDL Trados Studio 2019 Professional	10	10	7	5	10	1.75	10	3.5	10	3.5	10	3.5	10	94.25
Wordbee	10	5	10	5	5	2.5	10	5	10	5	10	5	5	87.5
Across Translator Premium Edition v7.0	5	5	5	5	5	5	10	5	10	5	10	5	5	80
CafeTran Espresso 10.8.1	6	0	10	6	0	5	6	5	6	5	6	5	0	60
Wordfast Pro 5	10	0	10	5	0	2.5	2	0	0	0	0	0	0	29.5
Memsource + Editor for Desktop 20.21.3	8	0	0	4	0	0	8	0	0	0	8	0	0	28
Translation Workspace XLIFF Editor	10	0	7	5	0	1.75	0	0	0	0	0	0	0	23.75
MadCap Lingo 11 r2	5	0	10	0	0	0	2	0	0	0	0	0	0	17
OmegaT 4.3.2	0	0	0	0	0	0	2	0	0	0	0	0	0	2
Deja Vu X3 Professional	0	0	0	0	0	0	2	0	0	0	0	0	0	2
translate5	0	0	0	0	0	0	2	0	0	0	0	0	0	2
XTM	0	0	0	0	0	0	2	0	0	0	0	0	0	2
Nucleus	0	0	0	0	0	0	2	0	0	0	0	0	0	2
Smartcat	0	0	0	0	0	0	2	0	0	0	0	0	0	2
Matecat	0	0	0	0	0	0	2	0	0	0	0	0	0	2
Fluency Now	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Swordfish IV	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Crowdin	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7-2. Scoring table (breakdown by File preparation categories)

Operations with text

Tool	Operations with text			Total
	Segment filtering	Search	Replace	
memoQ 9.5.8 translator pro	10	10	10	30
SDL Trados Studio 2019 Professional	10	10	10	30
Wordfast Pro 5	10	10	10	30
Alchemy Catalyst 2021	10	7	10	27
OmegaT 4.3.2	10	7	10	27
Fluency Now	5	10	10	25
Memsource + Editor for Desktop 20.21.3	8	8	8	24
Deja Vu X3 Professional	2	10	10	22
CafeTran Espresso 10.8.1	10	5	5	20
MadCap Lingo 11 r2	10	10	0	20
Swordfish IV	10	0	5	15
Wordbee	8	0	0	8
translate5	0	6	0	6
Translation Workspace XLIFF Editor	0	0	5	5
Across Translator Premium Edition v7.0	0	0	0	0
XTM	0	0	0	0
Nucleus	0	0	0	0
Smartcat	0	0	0	0
Matecat	0	0	0	0
Crowdin	0	0	0	0

Figure 7-3. Scoring table (breakdown by Operations with text categories)

8. Evaluation Matrix

To add another dimension to the rankings, the reviewed tools were placed in four quadrants: *Fledglings*, *Manager's Helpers*, *Editor's Friends* and *Regular Beasts*. Tools farther to the right are better at file preparation tasks. Tools higher up are better at operations with text such as segment filtering and search and replace.

Again, cloud solutions are in green.

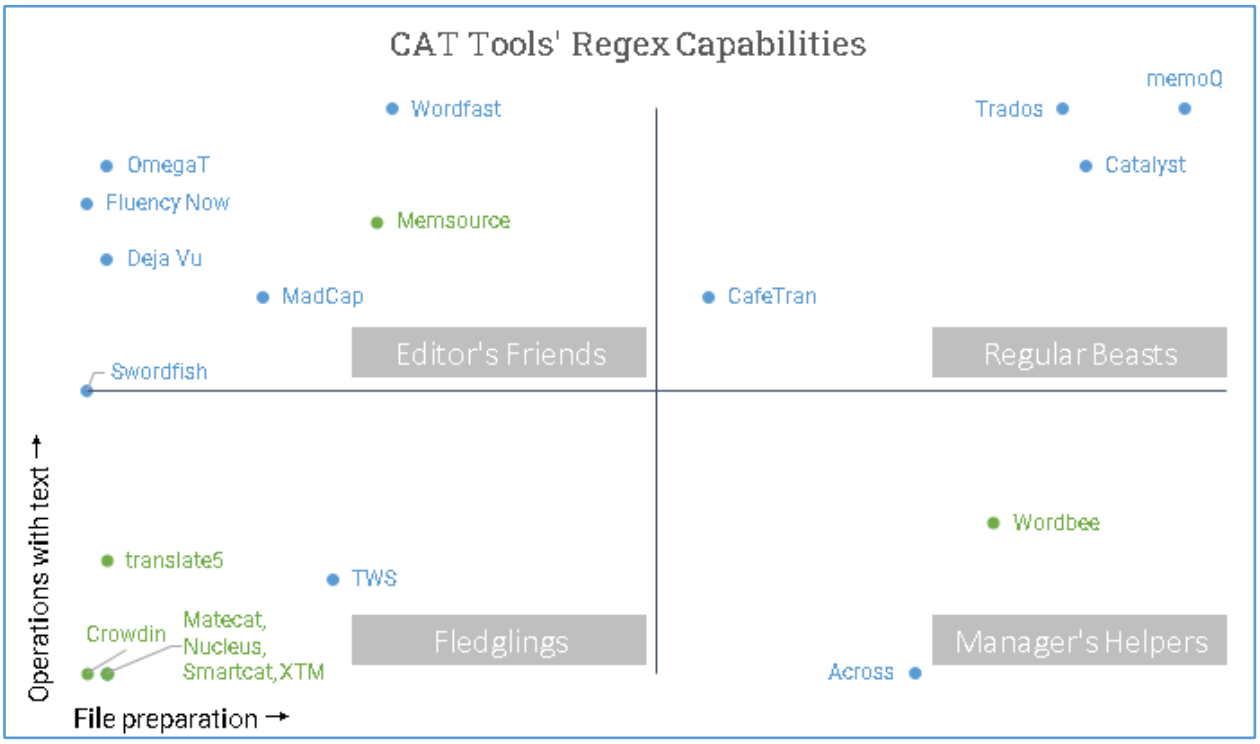

Figure 8-1. Evaluation Matrix: CAT tools by their regex functionality in File preparation and Operations with text categories

9. Takeaways

In the Battle of Regexes, three CAT tools stand tall, head and shoulders above everybody else: memoQ, SDL Trados Studio and Alchemy Catalyst. All of them are desktop tools that have been around for quite a while now. No surprise here: it does take time to build regex muscles as this functionality rarely finds its way to the top of the priority list during a new tool development. Older, more mature CAT tools have a clear advantage over newer entrants in the market when it comes to regex power. It does not mean, though, that older is automatically better: many other veteran tools have fallen far behind the leaders.

On the whole, cloud solutions are no match for their desktop counterparts. The two exceptions are Wordbee, equipped with pretty robust file preparation functionality, and Memsource that feels quite at home among other Editor's Friends thanks to its Editor for Desktop (which is... well, desktop, not cloud). Other cloud participants possess hardly more than rudimentary regex capabilities.

The general tendency is that linguistic capabilities (for operations with text) are easier to find than managerial functionality (for file preparation). Of the 20 reviewed tools, 11 are located above or at (Swordfish) the average line for operations with text, while only 6 are to the right of the average line for file preparation.

As the Evaluation Matrix shows, only 4 tools are advanced enough to be put in the upper right corner. Of them, CafeTran is probably a somewhat stretched choice as its functionality, though broad, is not always reliable or manager-friendly. Catalyst can also be called into question—its regex capabilities are indeed impressive, but it is rather a localization tool, with quite a twisted approach to dealing with Excel files.

All this makes one wonder if the CAT tools market is as saturated and mature as it may seem... Certainly, regex capabilities are just a relatively narrow subset of overall CAT functionality, yet it is an important subset that an experienced user is very likely to expect. Despite that, CAT tools providing the right combination of regex-enabling options on both managerial and linguistic sides can easily be counted on one hand.

10. Quick tutorials

Sections in this chapter describe the specifics of how our test case goals can be achieved in different CAT tools. Note that the structure of subsections in each section may vary depending on the tool's architecture and the scope of its regex capabilities. Also bear in mind that the tutorials only cover functionality directly related to the creation and application of regex rules. Other necessary actions (like the creation of a new project, navigation between different views within a tool, etc.) are rarely described in detail or shown on screenshots. To find more information, please refer to the documentation on the respective tools.

Tutorials are given in alphabetical order.

Across Translator Premium Edition v7.0

Quadrant: Manager's Helpers

Overall score: 80

Across is a well-known CAT tool, especially popular among enterprise customers. Conceptually, Across has always been a server-based environment, and server is beyond this report's scope (which is defined as desktop/cloud solutions). Across, however, does have a relatively function-rich client version, Across Translator Premium Edition. It is still linked to the Across server (user's credentials are checked during launch and to access resources), but its project creation capabilities are more or less self-contained, so I decided to include it in the report.

Across' regex muscles are all concentrated on the file preparation side where it is clearly above average. In contrast, no regexes at all are supported in segment filtering and search and replace functions.

File preparation

Across Translator Premium Edition offers very clean and intuitive functionality for the creation of file filters. The only drawback is lack of support for plain text, which is rare among mature CAT tools.

Text files

Surprisingly, Across does not provide native support for text files. Instead, such files are processed using the Word filter that is also used to work with the old *.doc* format. Not only that, but for Across to be able to open a text file, a user has to have a 32-bit edition of Office installed on their computer.

After a series of experiments, I managed to bypass this restriction by resaving our sample text file with the *.sgml* extension and then creating a custom SGML filter. Of course, it cannot be considered a viable solution, even if it made do in our case. I decided to halve the score for text-file-related categories to reflect this lack of native support.

Custom file filter creation

Score: 5

Except for the quirk with changing the extension to *.sgml*, the procedure is quite smooth. A new file filter can be configured during the creation of a new project or in advance via Tools > System Settings. In our unorthodox case, we have to deal with the SGML section.

The easiest way is to define non-translatable content on the Placeables tab. First, we hide whole non-translatable strings, then ID sections of translatable strings, then end-of-line straight quotes, and finally article numbers. In Across, a *$* character means the end of file, and for the end of line a *\r* sequence must be used. To take into account both cases, I went with *(\r|$)*.

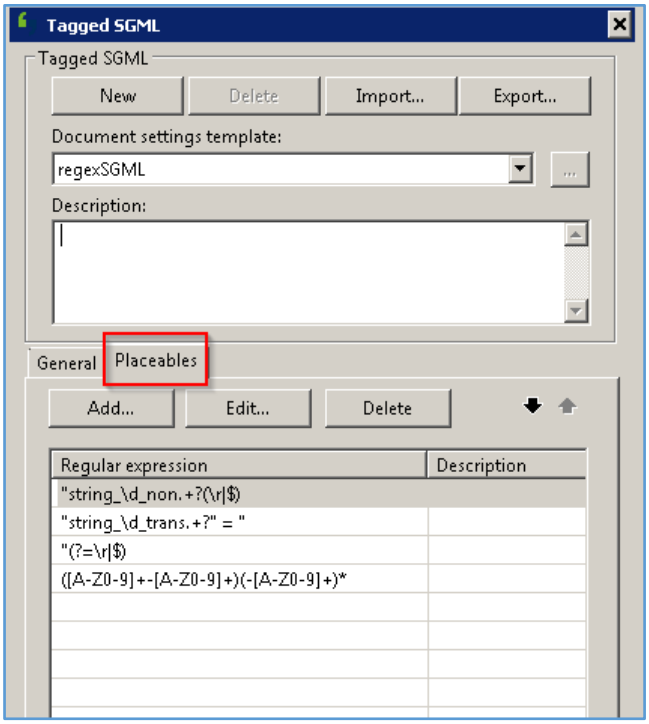

Figure 10-1. Across: Tagged SGML section

And here is how our TXT-turned-SGML file is displayed in the Editor view. Note a non-translatable string, which was put inside tags but still shown on the screen.

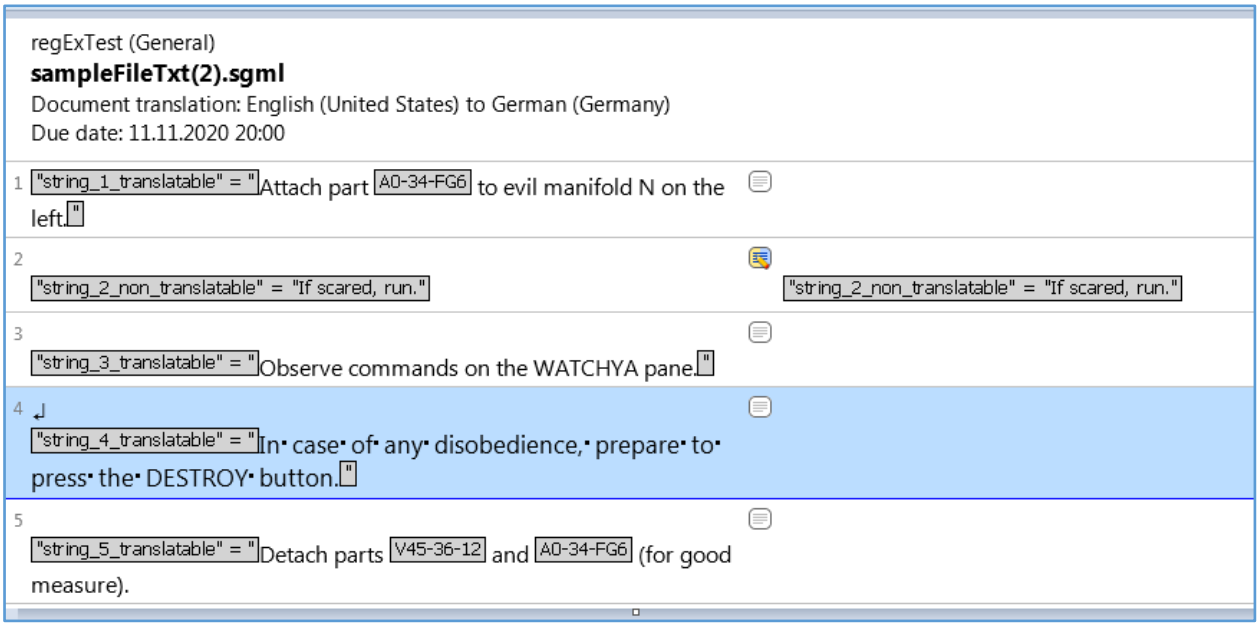

Figure 10-2. Across: Text file in Editor view

Custom file filter preview

Score: 5

A preview is visually good, but it has two limitations: *1)* you can only see the result of one current rule being applied and *2)* it is only available for a pasted sample of a source file. You cannot load a file in its entirety to preview it. With larger files, it can make a difference.

A preview becomes available when we add another rule:

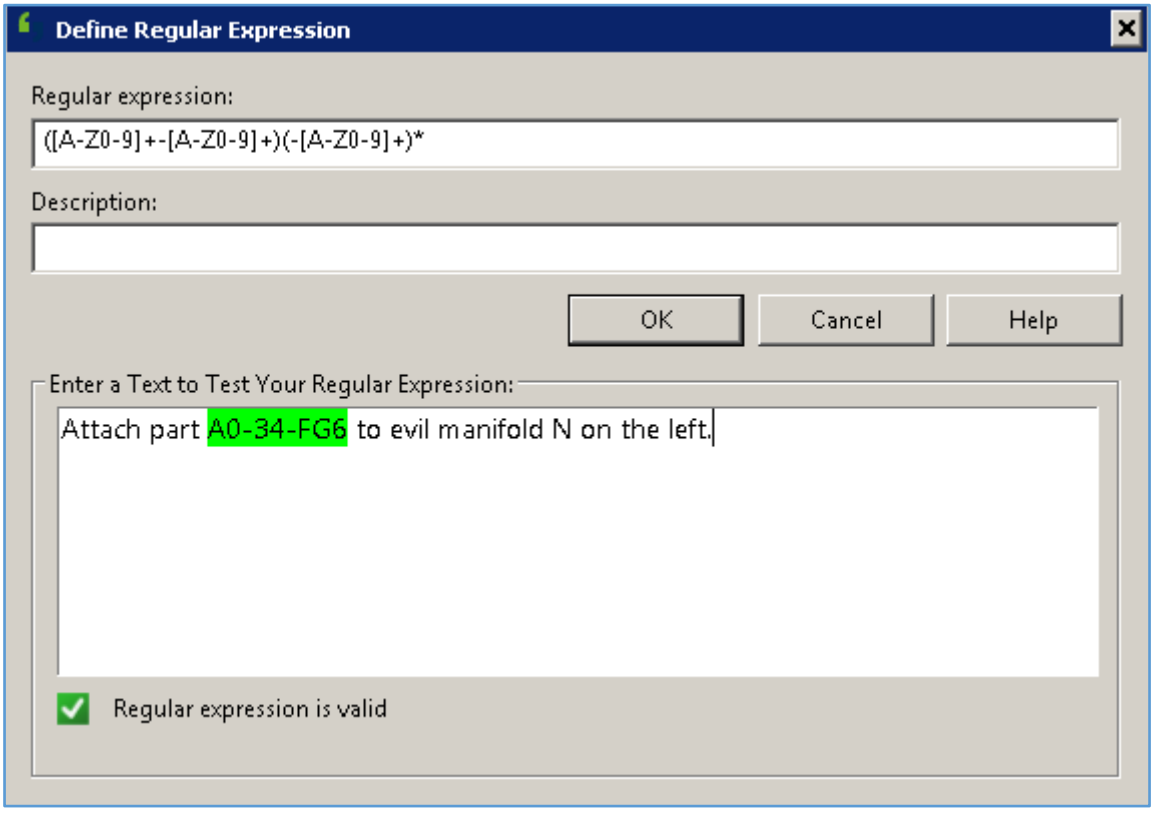

Figure 10-3. Across: Regex rule definition window

Custom file filter reuse

Score: 5

Any new filter is totally reusable, but the score is halved due to lack of native support for text files (see above).

Custom regex configuration creation

Score: 5

Our methodology would require to halve the score in this category as general file filters and more specific regexes for inline tags cannot be separated in Across (see chapters on Translation Workspace, Wordbee and Wordfast for similar cases). However, the score for the text file filter has already been lowered due to lack of native support for this format. I decided not to penalize Across twice and left the score at 5.

Custom regex configuration preview

Score: 5

See the previous section for the explanation of the score.

Custom regex configuration reuse

Score: 5

See above for the explanation of the score.

The Big Three

Total score: 50

The Big Three are handled in the same fashion as text files, with the exception that we are not forced to do anything weird with file extensions. As an example, here is how a custom Word filter may look like (the truncated regex at the bottom is our standard expression for article numbers):

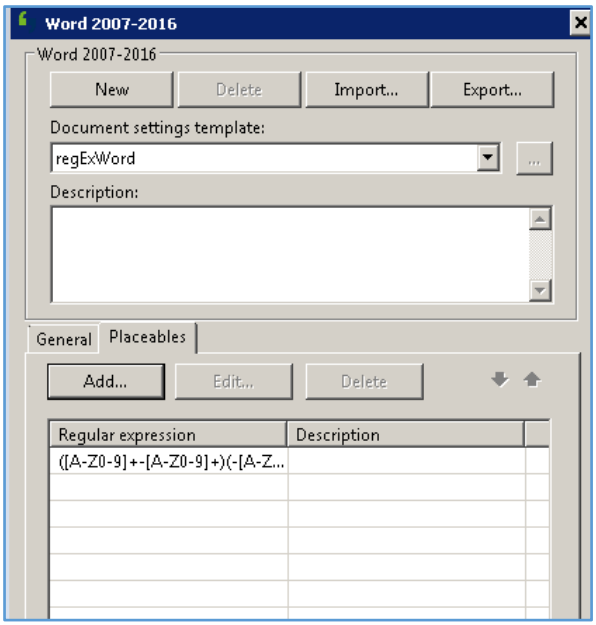

Figure 10-4. Across: Regex configuration for DOCX

The score calculation goes as follows:

Custom regex configuration creation for the Big Three: 10 + 10 + 10 = 30

Custom regex configuration reuse for the Big Three: 5 + 5 + 5 = 15

Custom regex configuration preview for the Big Three: 5

Segment filtering

Score: 0

Across only supports wildcards like *and ?but no regexes.

Search and replace

Total score: 0

Again, only wildcards are supported.

Alchemy Catalyst 2021

Quadrant: Regular Beasts

Overall score: 123.5

Along with memoQ and SDL Trados, Catalyst is one of the most formidable forces in the regex land, wielding the *Locks & Keywords* mechanism as its superior weapon. I was not sure, though, whether to include Catalyst in the report as it is widely perceived as localization software. Still, Catalyst does support the Big Three so it could not be disqualified on technical grounds. Maybe even more importantly, Catalyst's regex functionality is so powerful that it would have been a shame to leave this tool out.

Text files

Catalyst is very well equipped to deal with text files and offers many ways to parse content and create filters. To start our work, we first need to create a *TTK project* and insert our source files there.

Custom file filter creation

Score: 10

File filters for text files can be created using so-called *ezParse rules* (available through File > Settings):

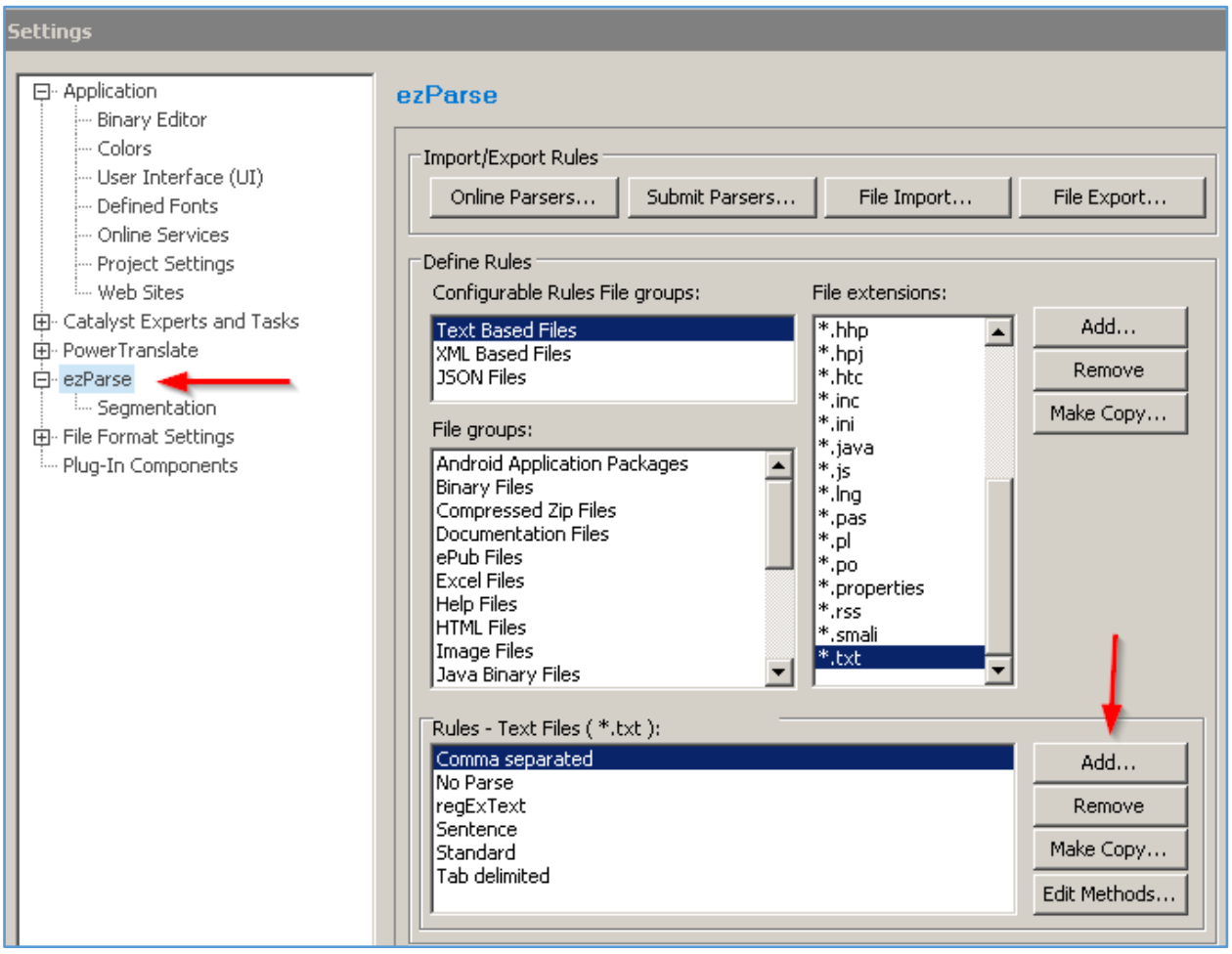

Figure 10-5. Catalyst: ezParse rules creation

The rules define start and end tags around localizable content as well as segments to be excluded from translation (i.e., non-translatable strings). To exclude segments, we should write rules defining so-called *context strings*. Context strings are listed in a separate area, below translatable strings.

In our case, a filter could look as follows:

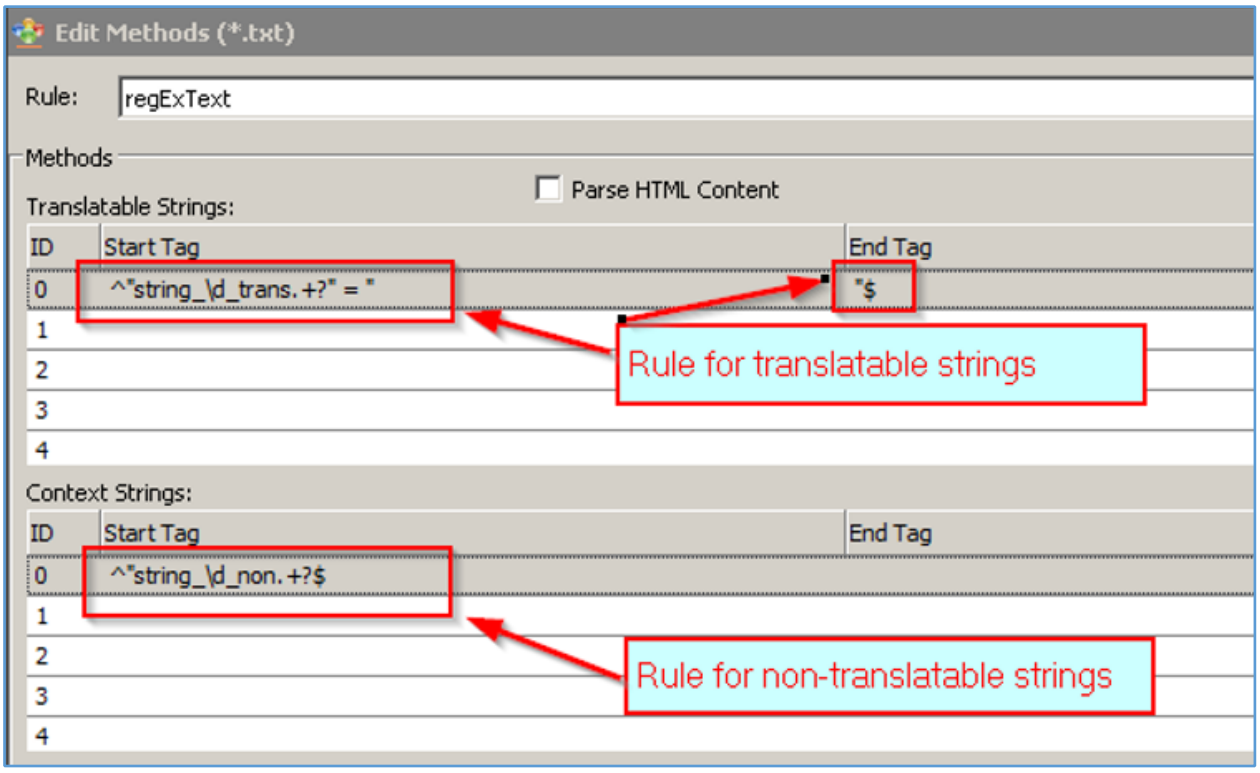

Figure 10-6. Catalyst: Regexes added in ezParse window

Custom file filter preview

Score: 10

A very nice preview is available right below our rules. Green highlight is used to indicate translatable content:

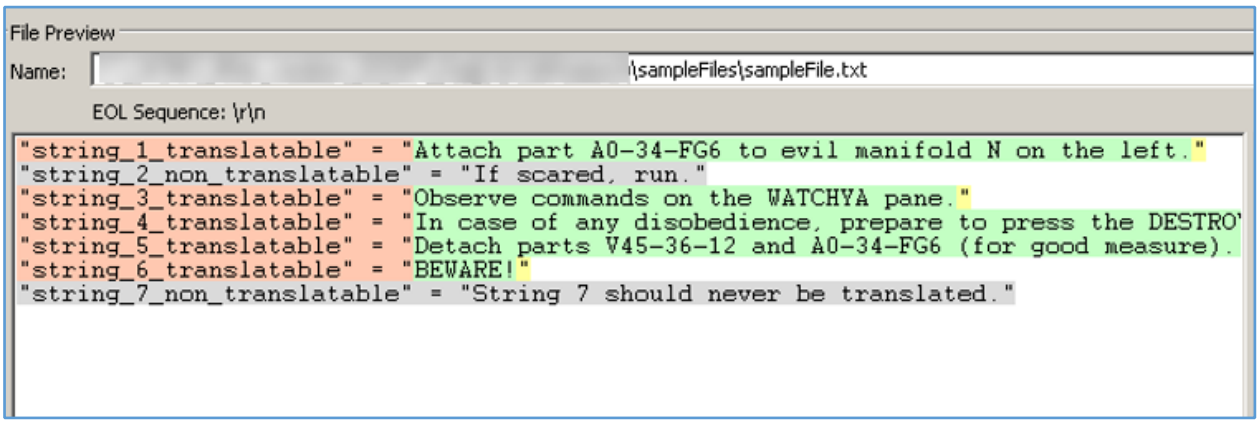

Figure 10-7. Catalyst: Text file filter preview

Custom file filter reuse

Score: 10

Once created, a new filter can be used with any other project.

Custom regex configuration creation

Score: 10

Great as they are, ezParse rules are complemented in Catalyst by an even greater technology: Locks & Keywords. It allows us to define protected content, which, in Catalyst's parlance, is called *Keywords*. Locks & Keywords are universal and can be used with any source file format. Moreover, they can even be applied *after* the project is created and displayed in the Editor view! This capability is really impressive and is only matched by memoQ's Regex Tagger functionality.

To create a new rule, we need to go to Experts > Locks & Keywords on the upper menu and then click the Edit button:

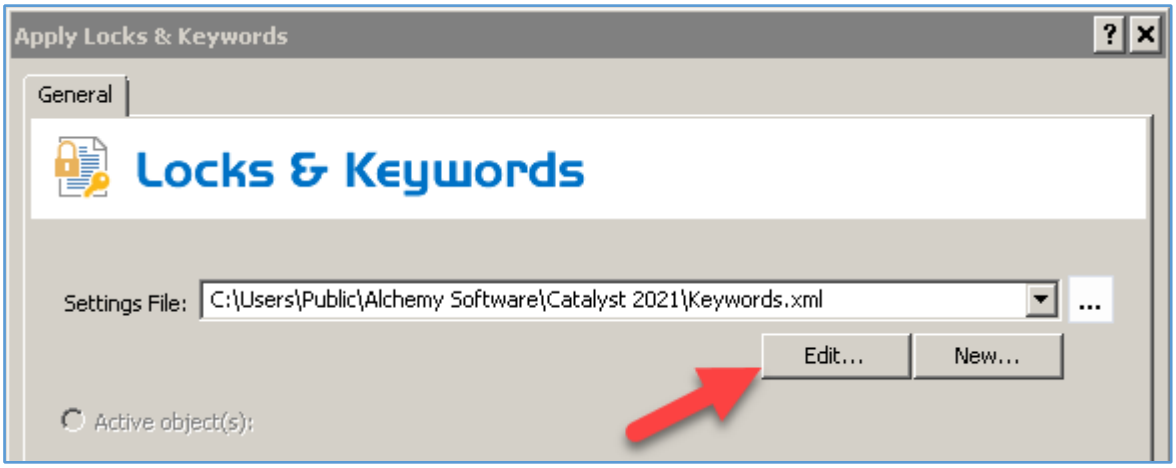

Figure 10-8. Catalyst: Locks & Keywords start window

Then we can add a new rule using the + icon. We choose the KEYWORD label and paste our rule for article numbers in the next column:

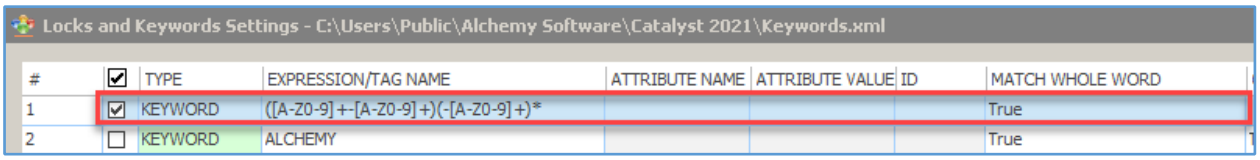

Figure 10-9. Catalyst: Locks & Keywords settings window

As a final step, we should save our new configuration.

And here is our text file in the Editor view, with the new keyword applied:

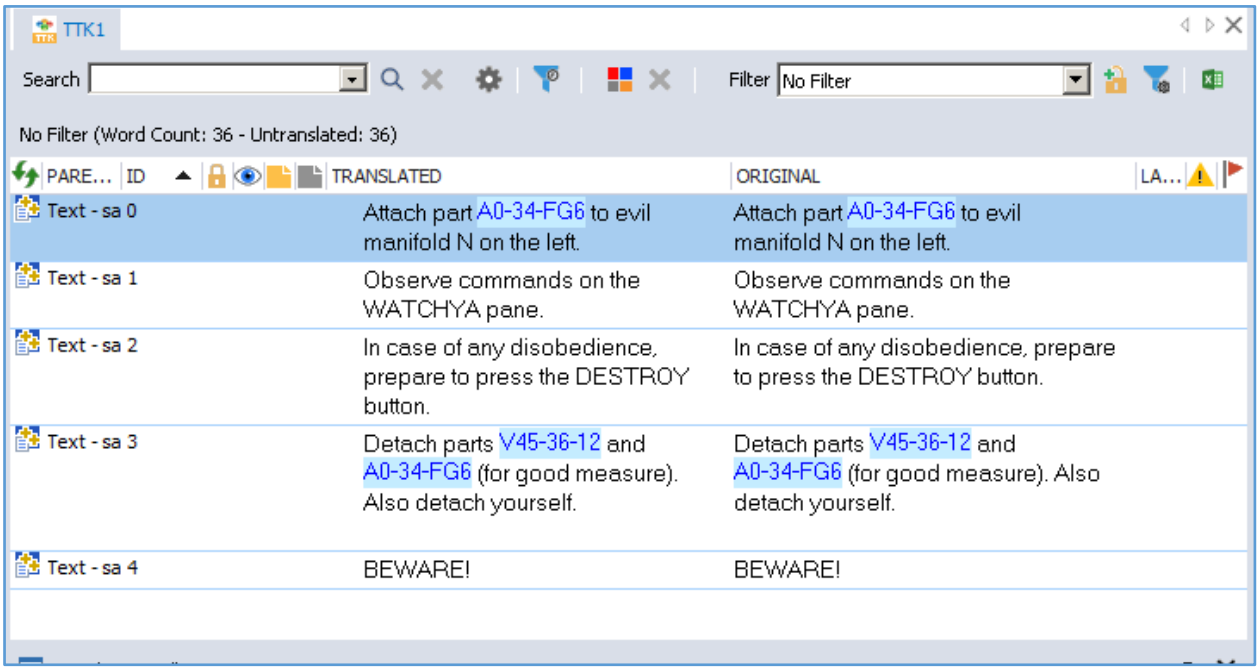

Figure 10-10. Catalyst: Text file in Editor view

Custom regex configuration preview

Score: 7

A preview in its pure form is not available, but it is, to a degree, made up for by Catalyst's almost unique ability to retag segments on the fly. At any moment, we can go back to the Locks & Keywords window, change the rule and see how these changes affect our project, without having to recreate it. A solid score had to be awarded for that, if not a full score of 10.

Custom regex configuration reuse

Score: 5

All Keywords are reusable across all projects and file types. Regex configurations are not stored in separate files. Instead, they are presented as a list including all rules you have ever created. To combine rules into a new configuration, we only need to select/deselect checkboxes next to respective rules. This is a very flexible and efficient approach that merits a full score.

The Big Three

Total score: 44.5

The Big Three can be handled using the very same Locks & Keywords window. In fact, once we have set up our rules for a text file, we do not need to do anything else. Accordingly, all scores remain the same. The only exception is XLSX. Once loaded, this file format can be processed just like PPTX and DOCX; the problem is that to load it, you will have to jump through considerable hoops. In Catalyst, Excel files are treated as databases, so you will need to establish a connection to a data source and create a *.ddf* file. It is a very nerdy and tedious process, which only serves to show that Catalyst, with all its regex prowess, is far from being a mainstream general-purpose CAT tool. I felt I had to reflect this fact in the score so I lowered it for XLSX to 5 (and 2.5 in the reuse category).

The explanation of the total score for the Big Three:

Custom regex configuration creation for the Big Three: 10 + 10 + 5 (XLSX) = 25

Custom regex configuration reuse for the Big Three: 5 + 5 + 2.5 (XLSX) = 12.5

Custom regex configuration preview for the Big Three: 7

Segment filtering

Score: 10

Catalyst has a solid filtering system. Settings for text-based filters can be found under the cog icon (QuickFind Options).

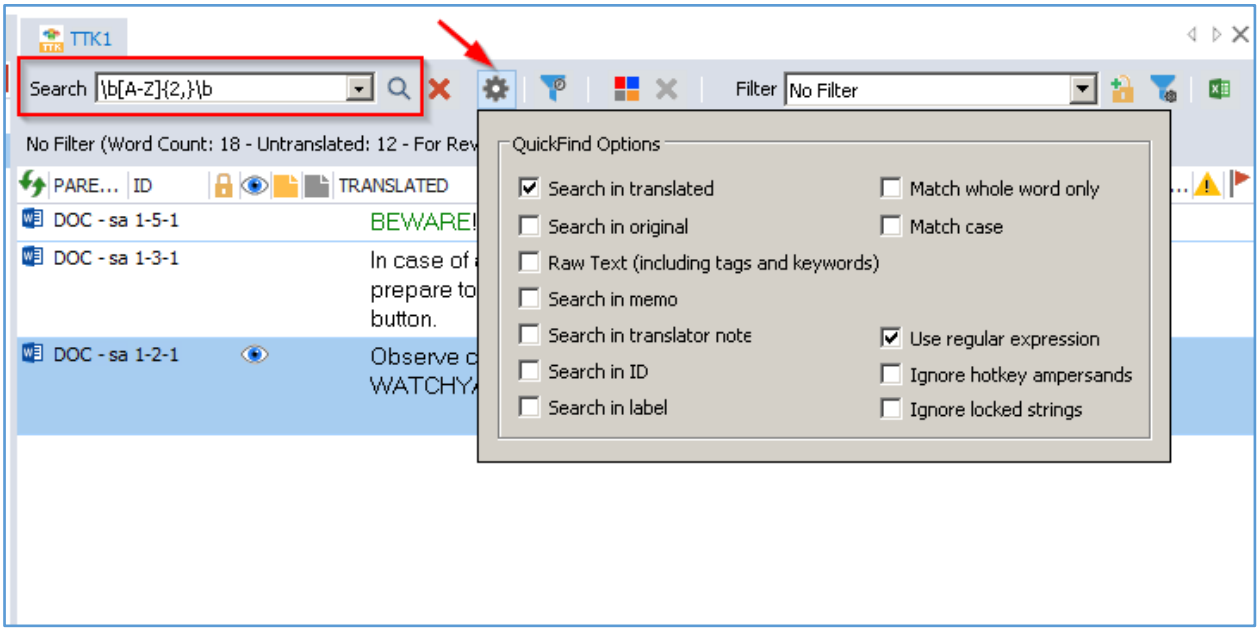

Figure 10-11. Catalyst: Segment filter settings

Search

Score: 7

The search implementation in Catalyst is slightly below par compared to the tool's many other features. The main deficiency is the absence of navigation between occurrences. Partly compensating for it, Catalyst provides a neat list of all found entries in a separate window.

The search and replace window is invoked through the standard *Ctrl-f* key combination.

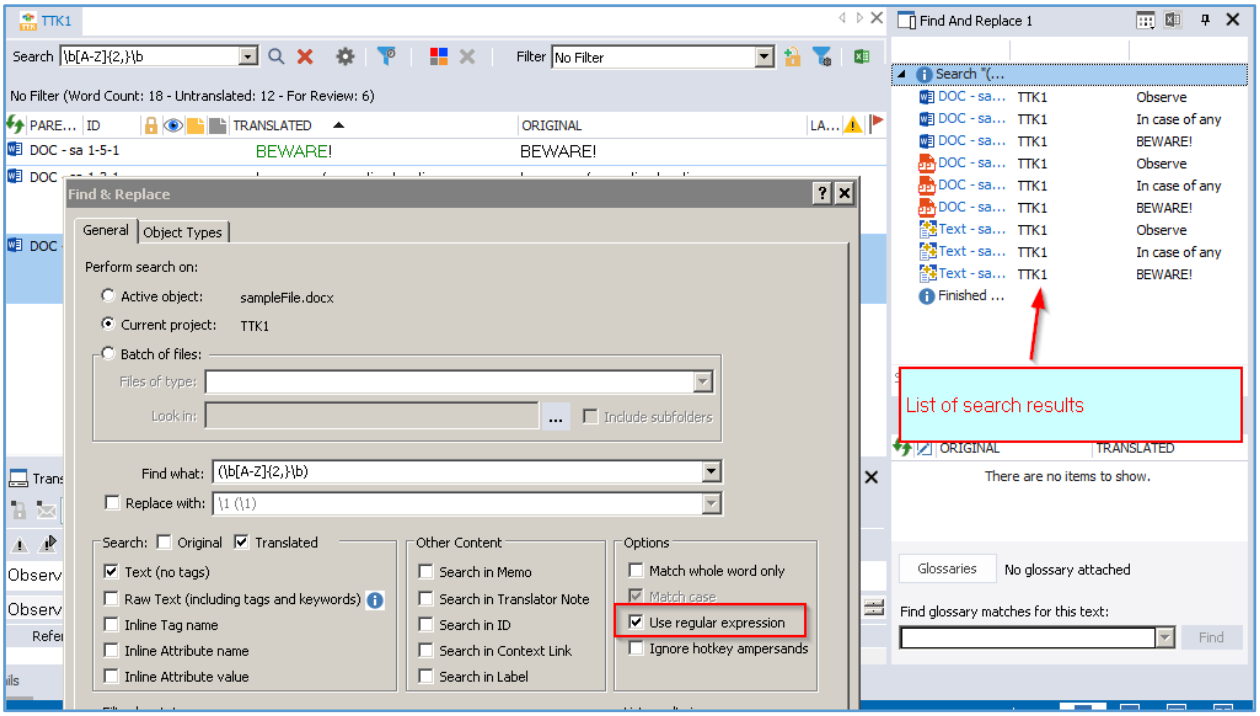

Figure 10-12. Catalyst: Search settings

Replace

Score: 10

To replace, we need to select the checkbox in the same Find & Replace window:

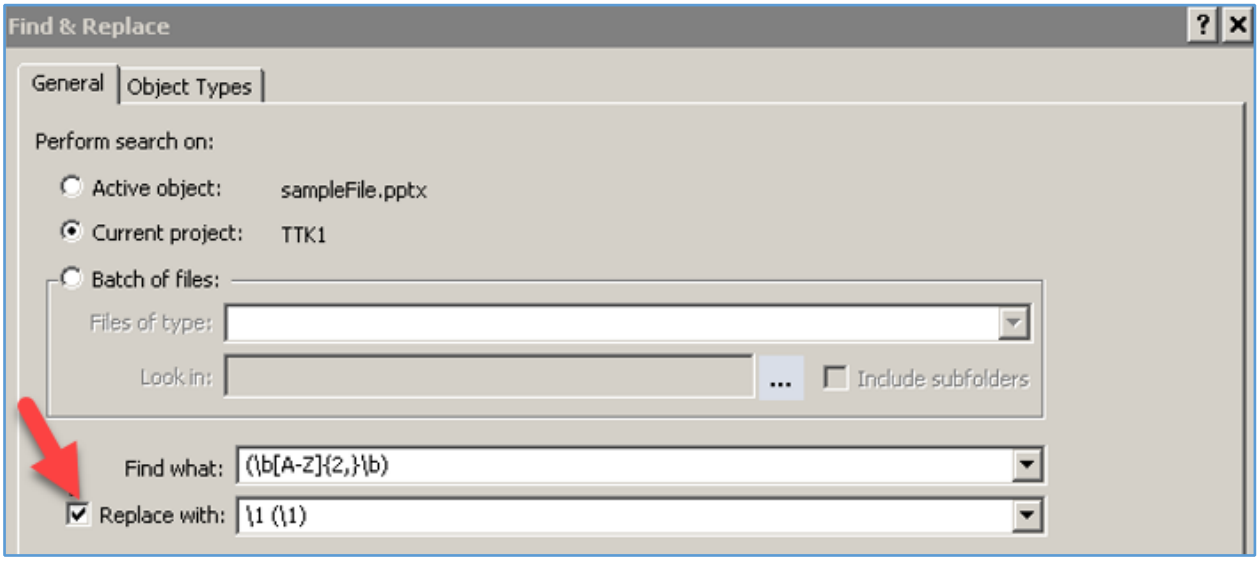

Figure 10-13. Catalyst: Replace settings

Note that Catalyst uses a backslash (\) to reference groups in a replacement expression.

Unlike search, replace operations are performed in an interactive manner, going from one occurrence to the next, so a full score is well deserved here.

CafeTran Espresso 10.8.1

Quadrant: Regular Beasts

Overall score: 80

Easily the most unconventional CAT tool on the market, CafeTran Espresso goes against the grain in many ways, from its very unorthodox user interface to the handling of regular expressions. Developed and maintained largely by one person, Igor Kmitowski (which is extraordinary in and of itself, considering CafeTran's longevity and rich functionality), this atmospheric tool gave life to a very active ecosystem and even developed a kind of cult following among freelance translators.

Translators have always been CafeTran's target audience, which might explain why the tool's translation-related functionality is so much richer than its project management features. However, for the purposes of our test case, CafeTran's file preparation capabilities proved to be surprisingly good. It has to be admitted, though, the process is not for the faint of heart and requires patience.

File preparation

Total score: 60

The only way to protect content from translation in our scenario seems to be via so-called *non-translatable glossaries*. Glossaries usually come to mind in the context of terminology work, but CafeTran has a specific use for them where glossary entries contain rules defining non-translatable content. This mechanism works regardless of a source file format, which is a rarity among CAT tools. Unfortunately, the implementation is not very reliable, so you can only make it work through trial and error.

How-to

Glossary creation

First, we need to create a glossary. It is a plain text file so we can use any text editor for that. Every line should start with a pipe (|) to indicate this is a regex and end with a caret (^) to denote non-translatable content. Some information on this very special syntax can be found here: Using Hidden Tags in CafeTran. I should say the notation is somewhat confusing as both the caret and the pipe are widely used in regexes with a totally different meaning (the caret usually means the start of line, and the pipe is the logical OR operator). However, this deviation from familiar conventions only requires some adjustment and does not jeopardize our progress. What is worse, the exact outcome of any given regex is not always predictable.

For example, it seems that the order matters in character classes: e.g., [\dA-Z] is not the same as [A-Z\d]. This behavior is very unusual. I had a lot of trouble with article numbers and could only manage to accommodate them with a pretty bizarre expression (see the last line on a screenshot below).

The final glossary looks as follows (CRLFs are end-of-line marks in Notepad++, my text editor; they are not part of the regexes):

```
|^"string_\d+_translatable"·\=·"^CRLF
|"$^CRLF
|^"string_\d+_non.+$^CRLF
|([A-Z]*\d*[A-Z]*\-)+([A-Z]*\d*[A-Z]*\-?)^CRLF
```

Figure 10-14. CafeTran: Non-translatable glossary

The glossary includes both rules for a text file (the first three lines) and a universal rule for article numbers, which is also applicable to the Big Three.

Adding glossary on Dashboard

Second, we need to add our glossary on the Dashboard and specify its purpose (a glossary of non-translatables). To do that, we browse to the prepared text file (see the previous section on glossary creation) and then select one crucial checkbox in settings (see step 2 below).

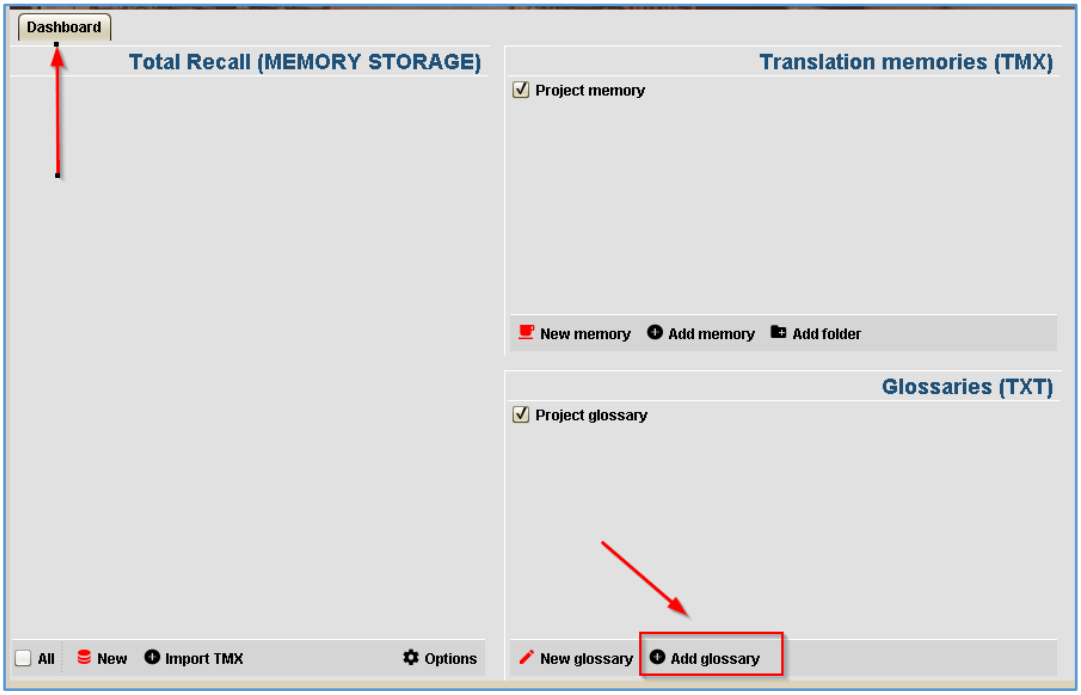

Figure 10-15. CafeTran: Adding glossary (step 1: Dashboard)

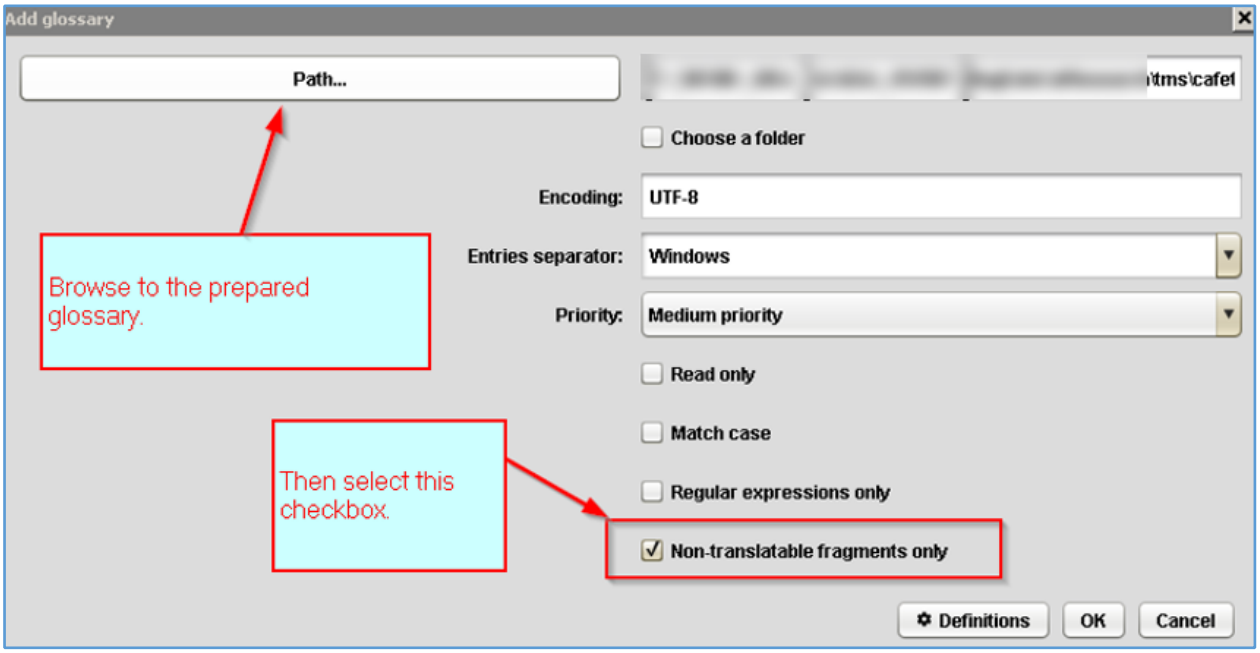

Figure 10-16. CafeTran: Adding glossary (step 2: Settings)

As a final step, we need to make sure the checkbox next to a new glossary on the Dashboard is selected. In our case, the glossary was named *hidden_text*.

Figure 10-17. CafeTran: New glossary selected on Dashboard

New project creation and Editor view

Third, we create a new project. This is a standard step that can be initiated from the Dashboard. The rules from our glossary are applied to strings extracted for translation. Non-translatables are not marked in a general view of all units, but they appear as numerical placeholders when an individual segment is clicked. Below, an article number is seen as is (*A0-34-FG6*) in the first unit of the general view, whereas in the active unit similar numbers are represented by the placeholders *1* and *2*.

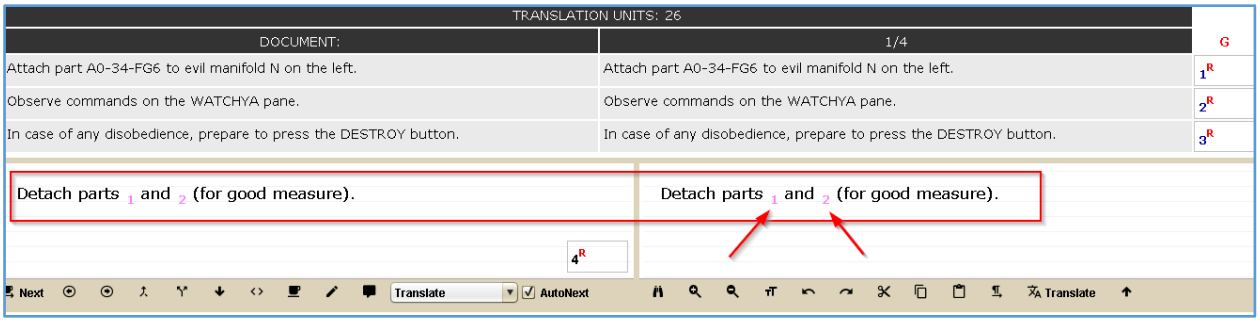

Figure 10-18. CafeTran: Text file in Editor view

File filters and regex configurations

The non-translatable glossary that we created worked as a combined file filter (for the text file) and regex configuration (for both the text file and the Big Three). However, we can easily separate these functions and create two glossaries: one with a file filter for text files and the other with a rule for article numbers. Together, they will work just as fine as one combined glossary. We can use as many glossaries as we like. Though file filters and regex configurations are in this case structurally identical, we can have different folders for them and create various combinations depending on our project requirements. This ability to use universal regex rules for different file formats is a very rare feature, only present in a couple of other CAT tools (memoQ and Catalyst).

But it is not all roses.

First, as we have just seen, the glossary creation process is quite cumbersome.

Second, putting non-translatables in tags does not influence the word count in CafeTran. Our DOCX, for example, is 39 words regardless of whether rules to protect non-translatables are applied. It is bad news for any project manager out there trying to prevent unnecessary spending.

Scores explained

So how do we rate CafeTran's file preparation ability? Despite my warm disposition toward the tool, I could not turn a blind eye to the unpredictability of the regex handling in non-translatable glossaries and lack of correlation between non-translatables and the word count. On the other hand, the reuse flexibility is there, though file filters and regex configurations cannot be separated at the structural level. Based on these considerations, I awarded CafeTran 6 points in each of the filter/configuration creation categories and top scores in each of the reuse categories (for the total score of 60):

Custom file filter creation for text files: 6

Custom file filter reuse for text files: 10

Custom regex configuration creation for text files: 6

Custom regex configuration reuse for text files: 5

Custom regex configuration creation for the Big Three: 6 + 6 + 6 = 18

Custom regex configuration reuse for the Big Three: 5 + 5 + 5 = 15

Segment filtering, search and replace

CafeTrans has text-based segment filtering and search and replace functionality (invoked via *Ctrl-f* or Edit > Find on the upper menu) merged in one window. In my view, it is a bit confusing, and a cleaner design would be to separate filters from search and replace functions. I would not have penalized the tool for this alone as this report takes a lenient approach as long as things can be done, one way or the other. However, the search and replace functionality in CafeTran has other shortcomings, apart from just being integrated with segment filtering.

Segment filtering

Score: 10

Segment filtering works as expected, provided the Segments filter checkbox is selected and you are determined enough not to give up when you came to filter and all you can see is the Find button. Note the Match Case checkbox—otherwise, the search is case-insensitive:

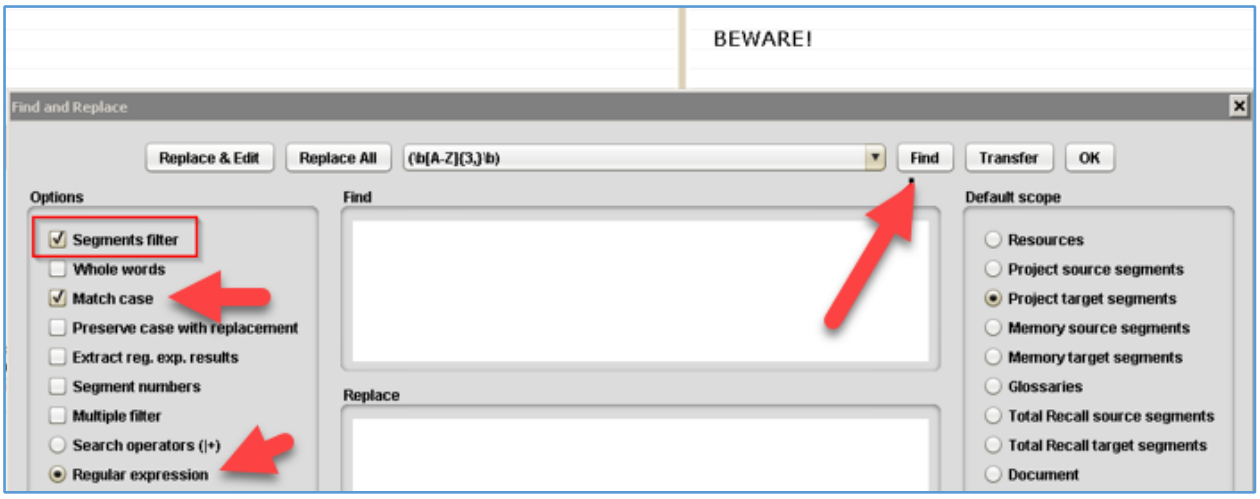

Figure 10-19. CafeTran: Segment filter settings

Search

Score: 5

When we run a search, all found occurrences are displayed. We can also quickly see the exact text that corresponds to our regex query (if the checkbox Extract reg. exp. results is selected). It can be convenient. However, a user cannot navigate between occurrences so the search effectively works as just another version of a segment filter. This led to the reduction in score.

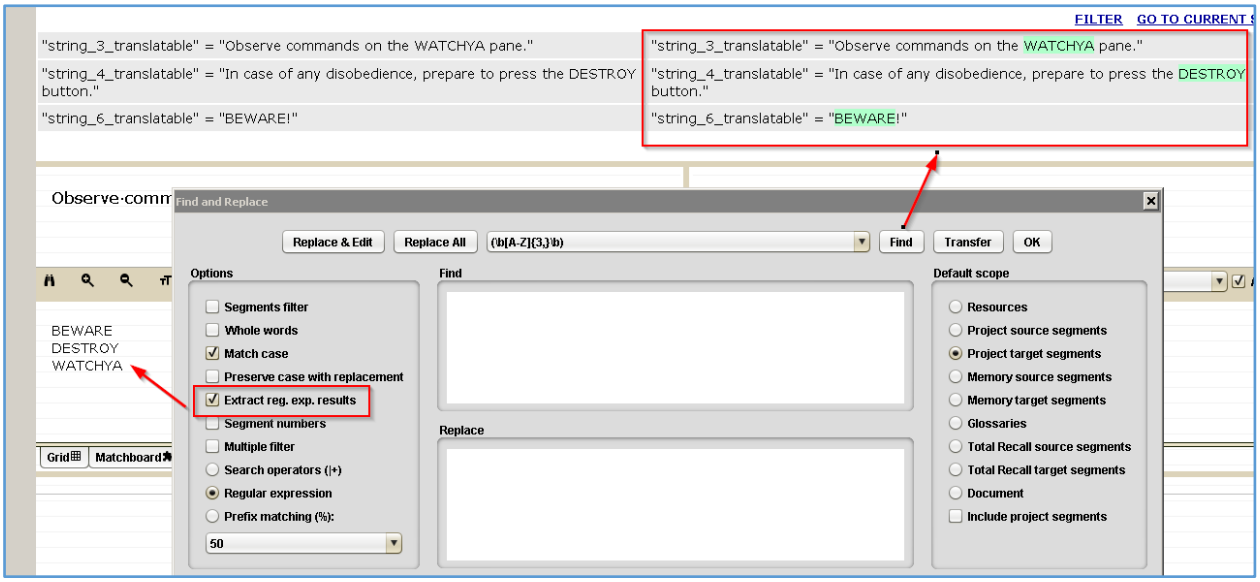

Figure 10-20. CafeTran: Search settings

Replace

Score: 5

Replace only works in batch mode (Replace All) and cannot be undone.

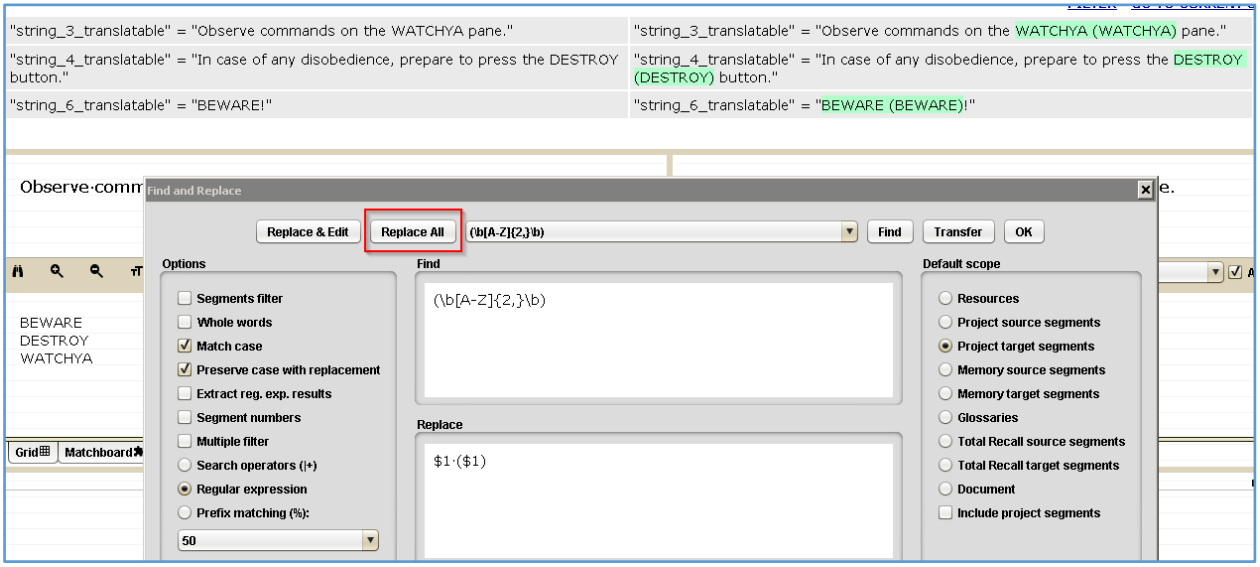

Figure 10-21. CafeTran: Replace settings

There is also the Replace & Edit button that supposedly allows a user to move between occurrences, but its behavior was, in my experience, unpredictable at best. For example, when I iterated through occurrences with the left and right arrows, the replacement strings piled on with every full cycle through all matching segments: first *WATCHYA* became *WATCHYA (WATCHYA)*, then *WATCHYA (WATCHYA) (WATCHYA (WATCHYA))*, etc.

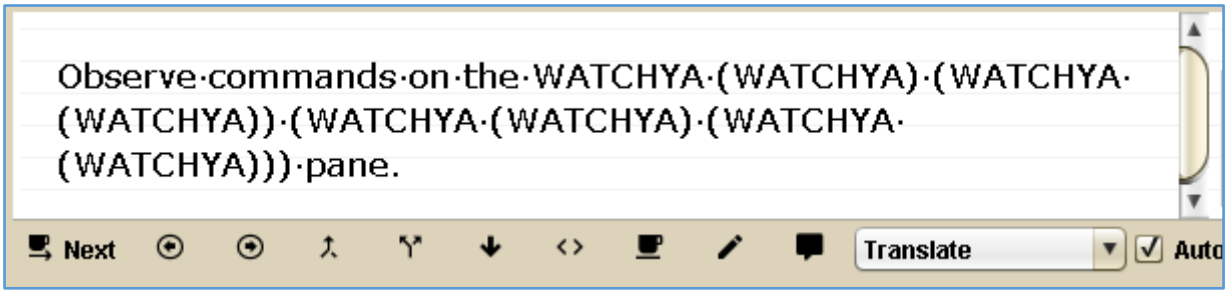

Figure 10-22. CafeTran: Undesired consequences of Replace & Edit

I managed to achieve better results when I tried non-regex replacement operations, but with regexes, it did not seem to work right.

Deja Vu X3 Professional

Quadrant: Editor's Friends

Overall score: 24

Deja Vu is a legendary tool in the industry, with a very long and rich history going all the way back to the 1990s. In the last decade or so, its glory seem to have somewhat faded, but it still remains a viable and well-respected option in the CAT tool market. However, in terms of its regex capabilities, Deja Vu has a lot of work ahead to catch up with the leaders.

File preparation

Total score: 2

In Deja Vu, we can control only DOCX, and only through hidden text. It merits 2 points, as in all similar cases.

Segment filtering

Score: 2

Filters support SQL statements as the only way to build advanced text-based queries. SQL does not allow to use complex regexes and limits us to wildcards like # (which means any digit). I awarded 2 points for scenarios where it might be enough. Queries like the one below are possible:

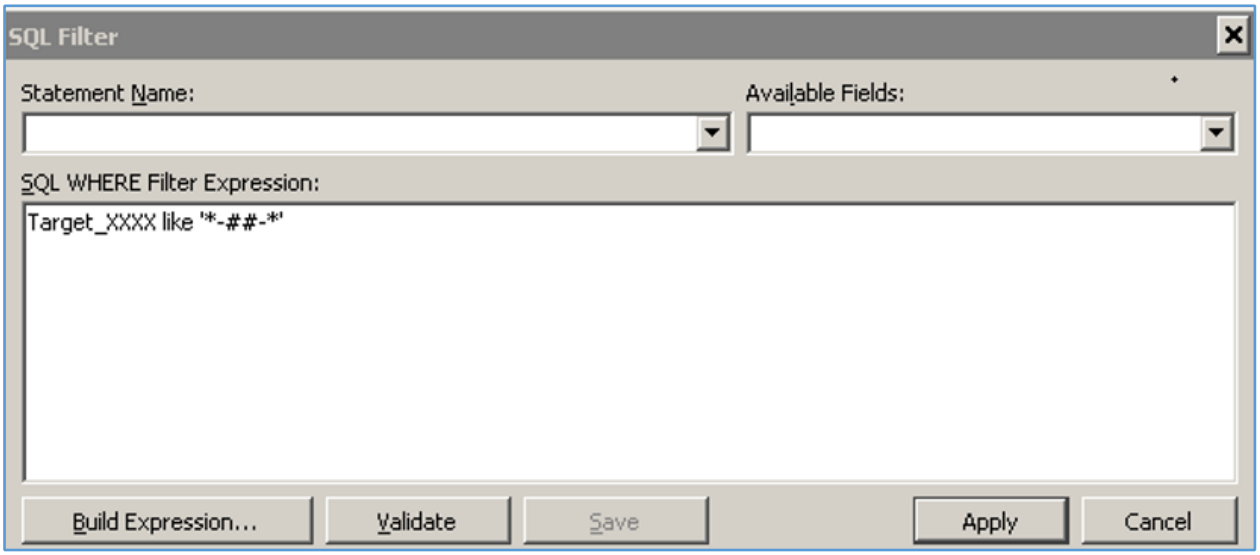

Figure 10-23. Deja Vu: Example of SQL filter settings

Here we are filtering our target sentences with the condition *includes two digits in a row with hyphens on both sides*.

This functionality is not enough for our purposes, however.

Search and replace

Total score: 20

The search and replace functionality works in a predictable manner and deserves a full score. Note the Match case checkbox as the regex search is case-insensitive by default:

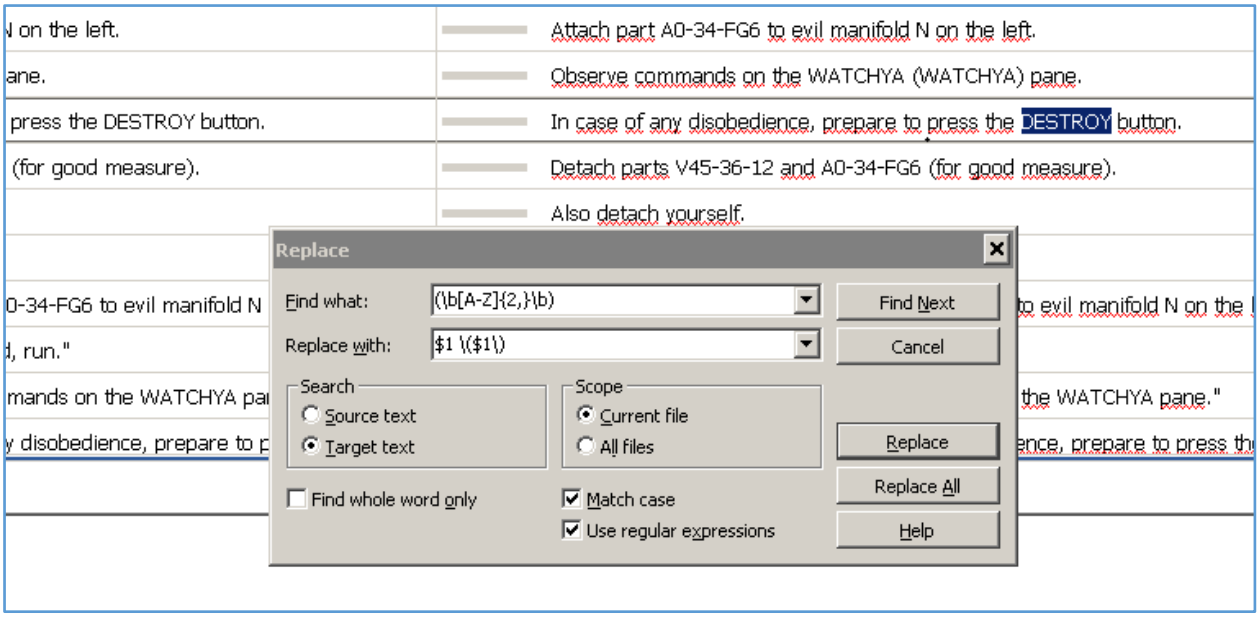

Figure 10-24. Deja Vu: Search and replace settings

Fluency Now

Quadrant: Editor's Friends

Overall score: 25

While being a significantly smaller CAT than the likes of MemoQ and Trados, Fluency Now offers reasonably rich functionality and has its fans in the industry. In terms of the tool's regex features, next to nothing is going on the file preparation side but segment filtering and especially search and replace capabilities keep Fluency Now from sinking to the bottom of our scoring table.

File preparation

Total score: 0

At the project creation stage, Fluency Now offers Advanced Settings that, at first glance, give some hope that a good thing or two could be done here.

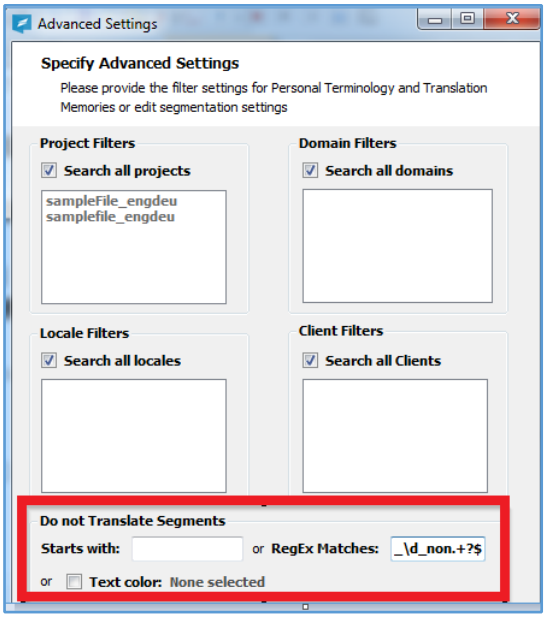

Figure 10-25. Fluency Now: Advanced Settings window (file preparation stage)

Unfortunately, it just does not seem to work. Even non-regex attempts in the Starts with field did not lead to any changes in how text was extracted.

As for our usual last resort, hidden text in DOCX, Fluency Now does put it between tags but does not protect this text from changes. So even 2 points could not be awarded here.

Segment filtering

Score: 5

Fluency Now has a very peculiar way of filtering segments. Filters, available through Edit > Segment filter on the upper menu, support regexes, but their application does not hide any segments. Instead, it influences the behavior of arrows that let a user navigate between segments.

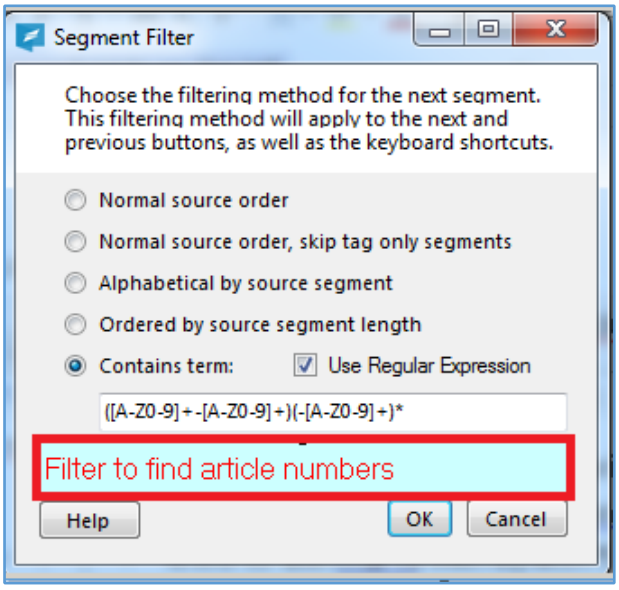

Figure 10-26. Fluency Now: Segment filter settings

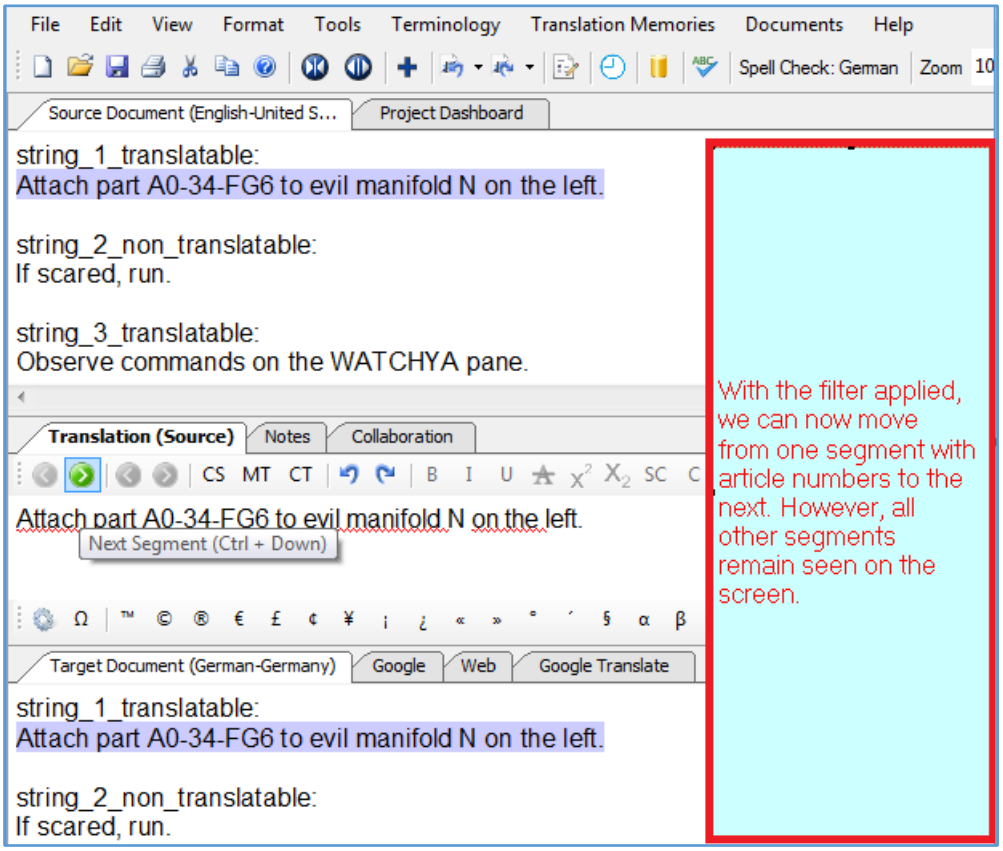

Figure 10-27. Fluency Now: Editor view with segment filter applied

This filter implementation does not seem very intuitive or helpful, so I could not award it more than 5 points.

Search and replace

Total score: 20

Search and replace behaves and merits a full score of 10 for each.

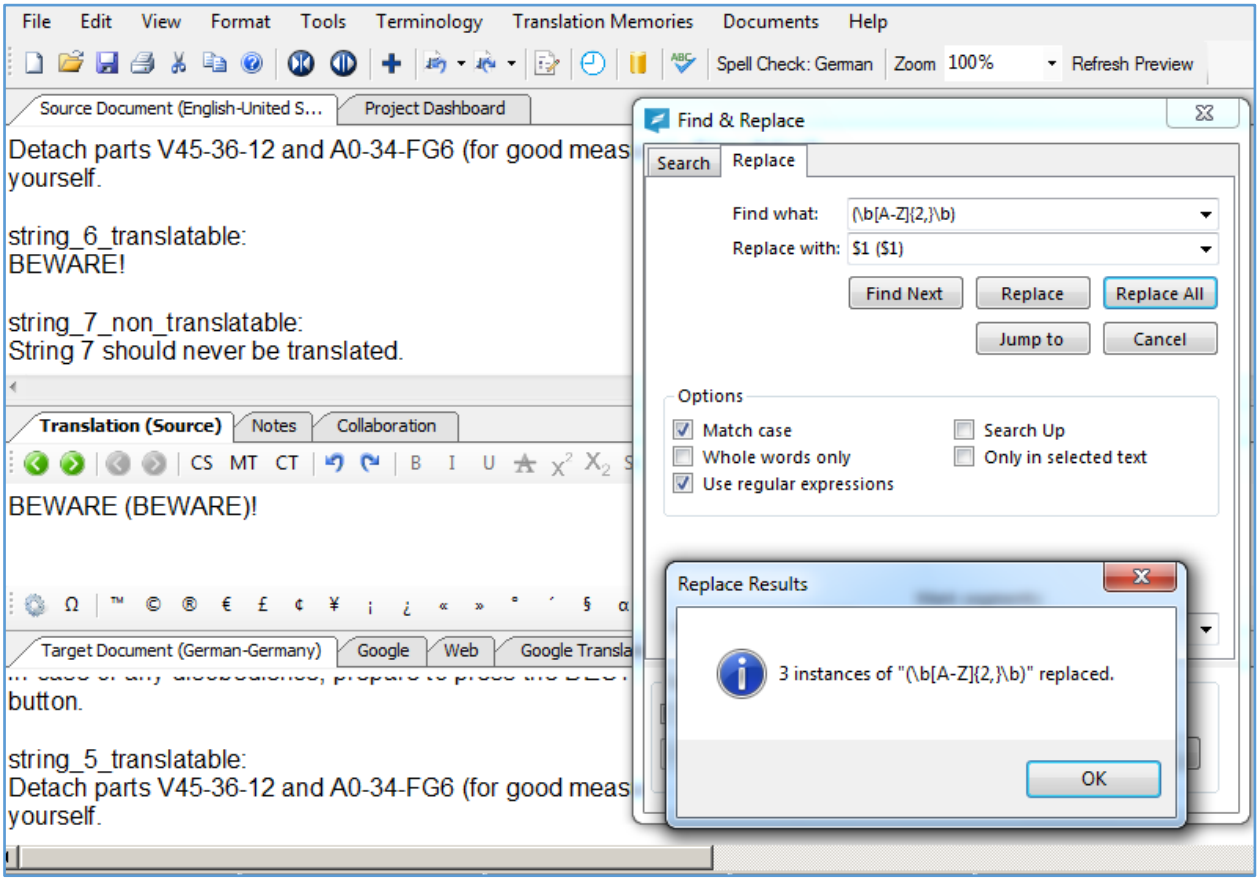

Figure 10-28. Fluency Now: Search and replace settings

MadCap Lingo 11 r2

Quadrant: Editor's Friends

Overall score: 37

Part of the MadCap suite of tools (the most well-known of which is probably the authoring and content management system MadCap Flare), MadCap Lingo is a mid-tier CAT program with enough functionality to satisfy many users' needs. In terms of its regex capabilities, it is also in the middle of the spectrum.

Text files

MadCap Lingo employs a very idiosyncratic approach to the tagging of text files that, coincidentally, goes well with our test case. However, only one regex per filter is allowed, and there is no way to protect placeables like our article numbers.

Custom file filter creation

Score: 5

The filter is built around the concepts of a *segment* (a translatable part of a string) and a *note* (an untranslatable part). Notes are captured by a custom regex and are then hidden in the Editor view. In our case, an ID section in the left part of strings is a classic example of a note in MadCap's sense.

How-to

At the project creation stage, we are invited to use a default filter or create our own:

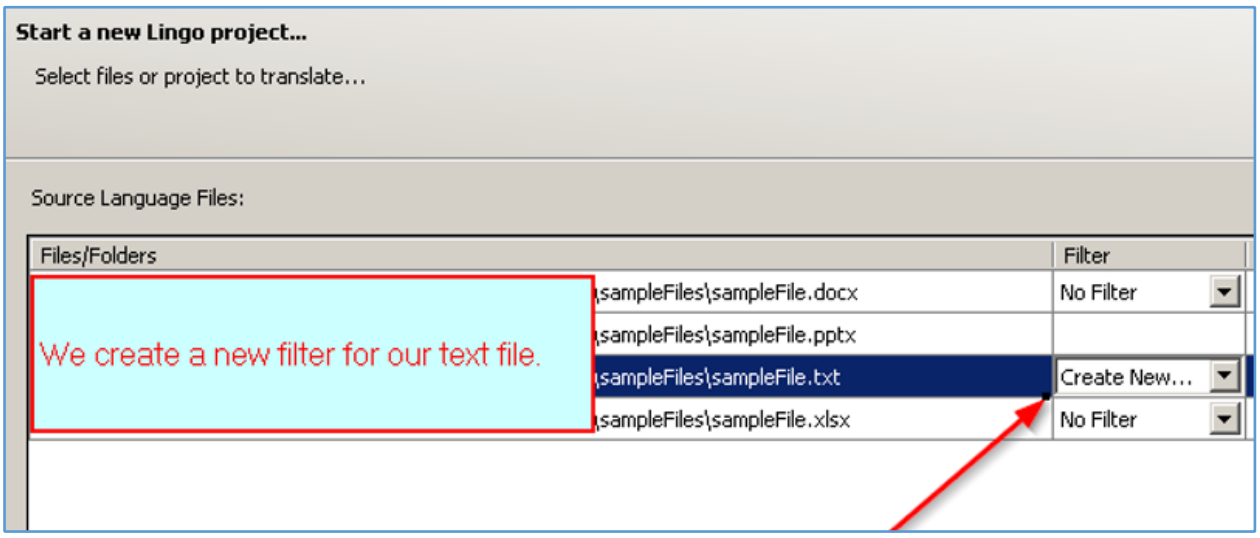

Figure 10-29. MadCap Lingo: New file filter creation

In a regex, we have to define a pattern for an untranslatable left part and put it in parentheses. The second part of our regex captures the translatable content between two lookarounds. The use of lookarounds allows us to get rid of the straight quotes before and after the translatable content.

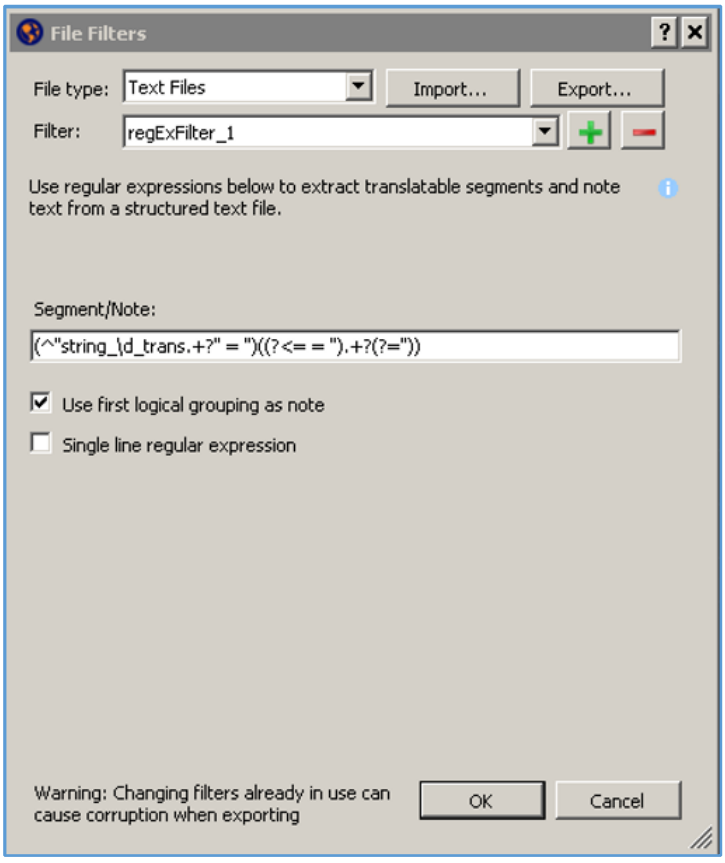

Figure 10-30. MadCap Lingo: Text file filter settings

If a string is not matched by the regex, it is skipped. This takes care of our non-translatable strings: their ID section is not matched, so they are left out.

After the new filter is applied, here is what we have in the Editor view:

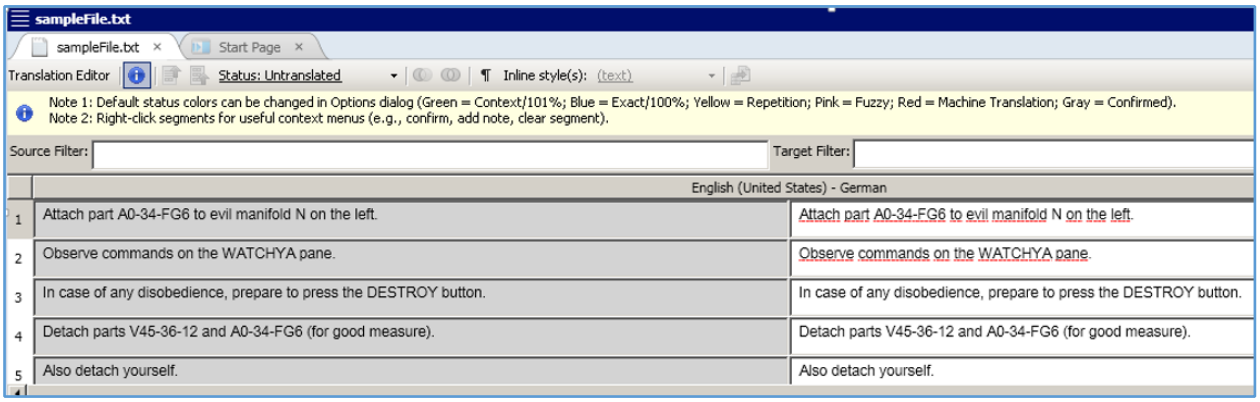

Figure 10-31. MadCap Lingo: Text file in Editor view

Non-translatable strings are hidden, but article numbers remain unprotected.

The MadCap Lingo custom filter functionality allowed us to achieve part of our goal, yet the restriction of only one regex per filter and lack of support for inline tags pushed the score down to 5 points.

Custom file filter preview

Score: 0

No preview is available.

Custom file filter reuse

Score: 10

Once created, a filter is available for all new files and projects.

Custom regex configuration creation

Score: 0

Custom regex configurations cannot be created.

Custom regex configuration preview

Score: 0

No preview is available.

Custom regex configuration reuse

Score: 0

Nothing to reuse.

The Big Three

Total score: 2

Only DOCX files can be to a degree processed via hidden text (2 points, as in all similar cases).

Segment filtering

Score: 10

MadCap Lingo's segment filter is sound. To set it up, you need to click the icon on the right and select the checkboxes. Regexes here are case-insensitive, so for my purposes I had to check both options:

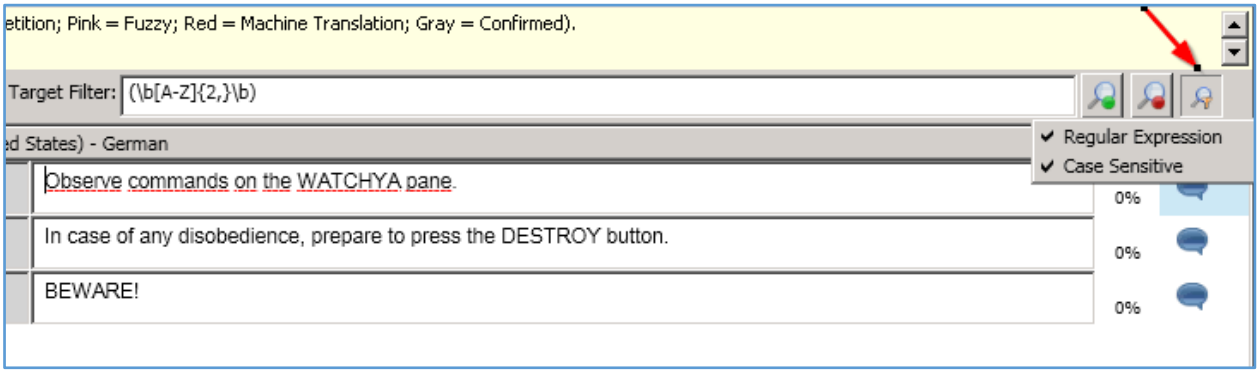

Figure 10-32. MadCap Lingo: Segment filter settings

Once a condition is typed in a filter field (see the Target Filter field in the figure above), the green magnifier icon must be clicked to apply the filter.

Search

Score: 10

The search field can be invoked by pressing *Ctrl-f* (*Ctrl-h* for search and replace). The search icon replaces filter magnifiers on the screen, which can be a little confusing. Otherwise, everything works as expected. Again, the Match case checkbox must be selected:

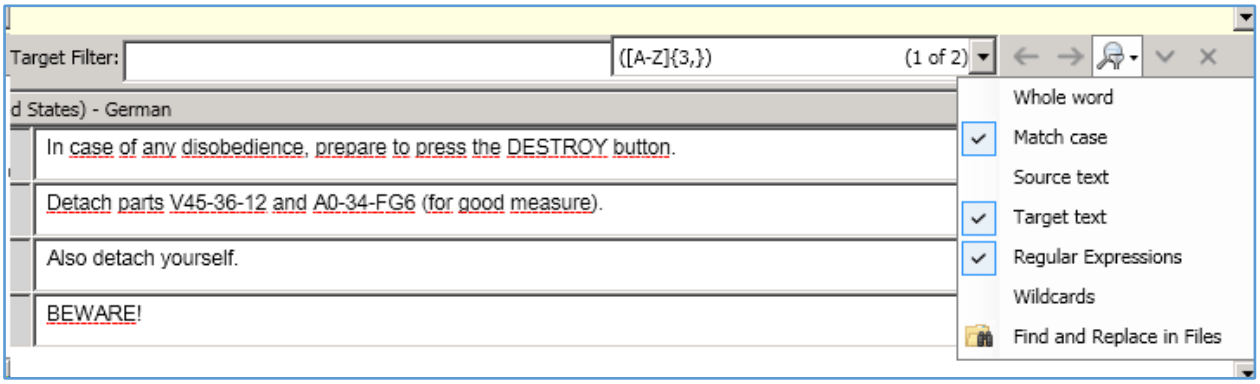

Figure 10-33. MadCap Lingo: Search settings

Replace

Score: 0

The replace function does not seem to support group backreferences, treating both $1 and \1, as well as some more fancy versions, as literals—that is, inserting $1 (or whatever was typed in the replace field) as a replacement text. This renders it next to useless in regex scenarios. The same problem emerged in several other cases so it can be considered a typical shortcoming of CAT tools' replace functionality implementation.

Matecat, Nucleus

Quadrant: Fledglings

Overall score (each): 2

These cloud solutions do not offer any regex-related capabilities that could be used in our test case. They do, however, support hidden text in DOCX, which merits 2 points.

memoQ 9.5.8 translator pro

Quadrant: Regular Beasts

Overall score: 136

memoQ is a true Regular Beast. Its powerful file preparation functionality featuring *Regex Taggers* and *cascading filters* makes it arguably the most capable CAT tool, at least as far as regexes are concerned. With memoQ, you can create very flexible configurations and easily reuse them with different file formats. memoQ's regex armor may have some slight cracks, but even if perfection has not been attained, excellence is definitely there.

File preparation

memoQ offers a versatile toolkit for creating complex file filters and regex configurations. First, it allows to change default file filters and save their custom versions, which we will do for our text file. Second, a special filter called *Regex Tagger* can be used to define regex configuration rules. Finally, default or custom file filters can be combined with one or several Regex Taggers to build so-called *cascading filters*. This mechanism is universal and applicable to different file formats. An additional benefit is memoQ's ability to retag source text of an existing project, without the need to recreate the project. For that, the same Regex Tagger functionality is used, available on the upper menu's Preparation tab. As far as I know, the only other CAT tool complete with such magic is Alchemy Catalyst.

Filters can be set up at the project creation stage (select Import with options) or separately on the Resource Console (to access it, click memoQ on the upper menu).

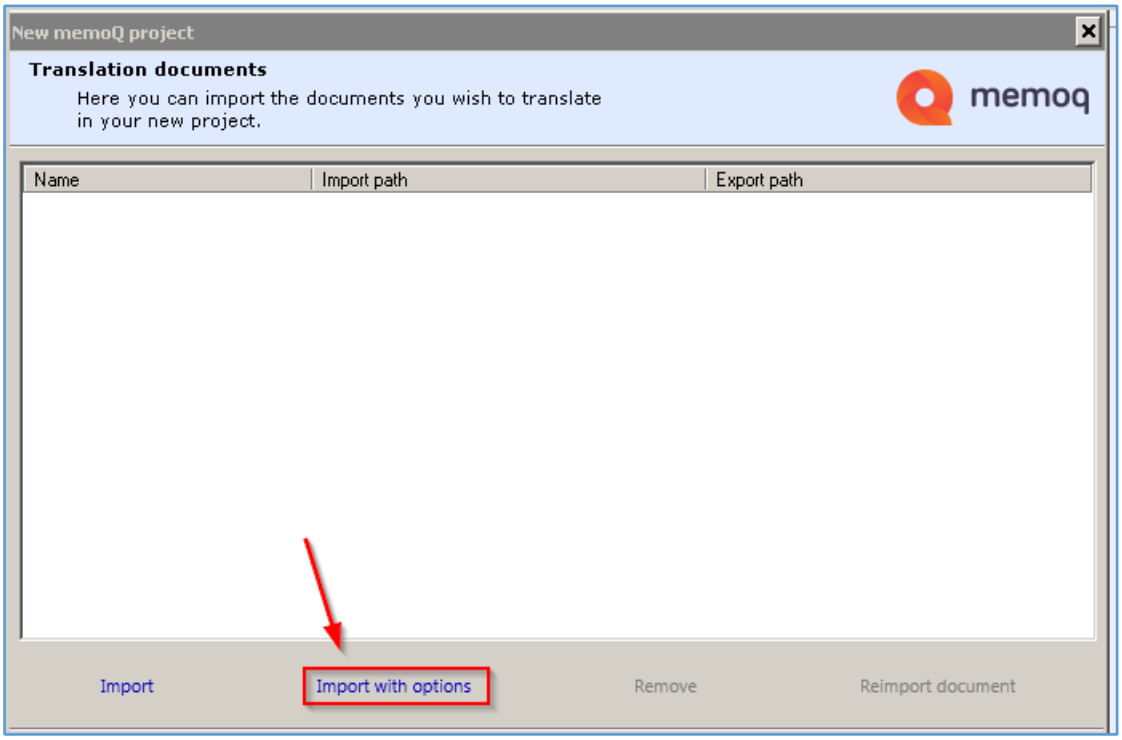

Figure 10-34. memoQ: Import with options start screen

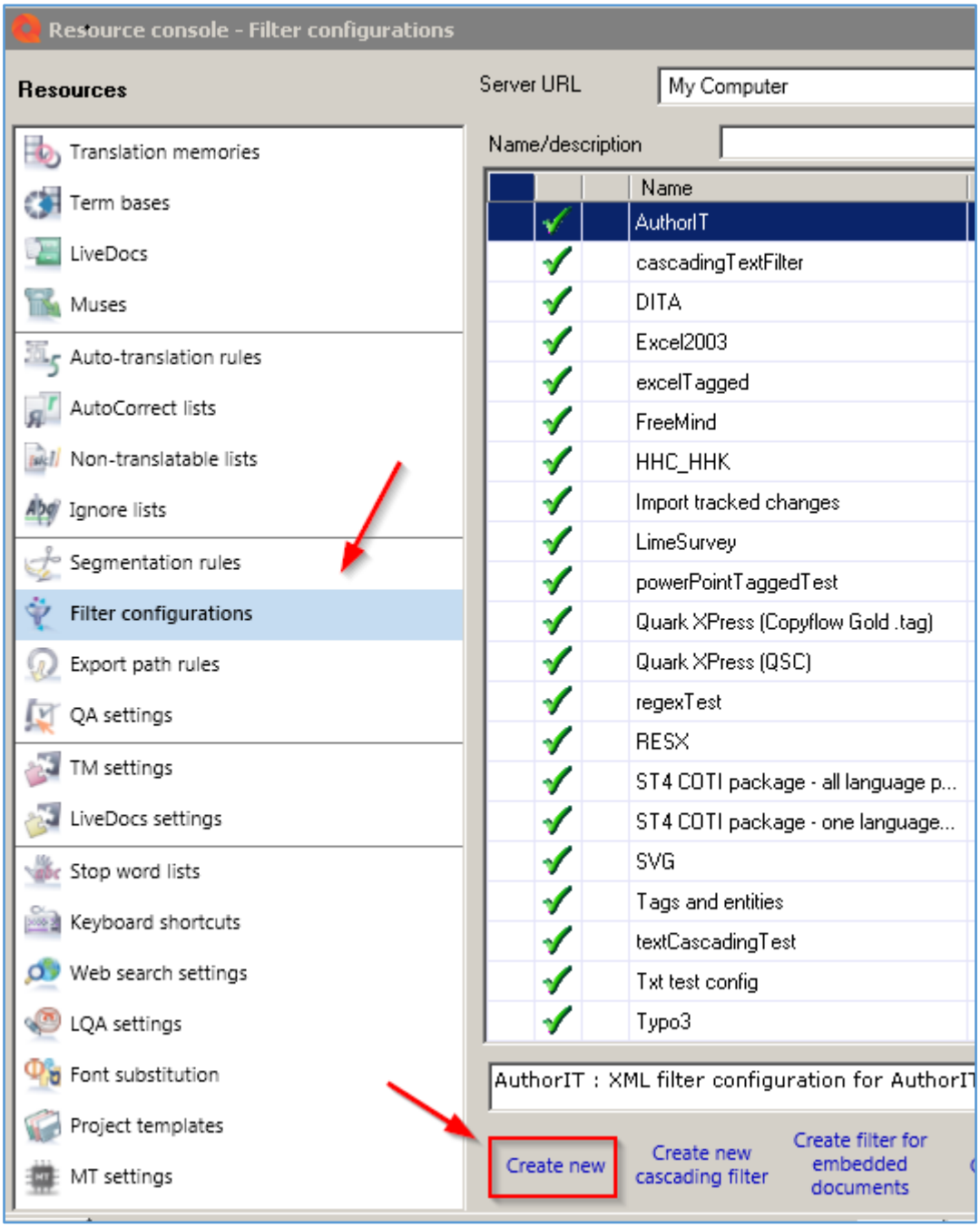

Figure 10-35. memoQ: New file filter creation

Text files

We can create a file filter for our text file based on the default *Regex text filter*. We can then add a Regex Tagger for article numbers and save the resulting configuration as a cascading filter.

Custom file filter creation

Score: 10

The most native way of dealing with regex-delimited text files in memoQ is via the Paragraph tab of the Regex text filter. By default, a paragraph means any new line in a text file, though this behavior can be configured if necessary. Rules defined on the Paragraph tab should capture the whole content of a line. Lines not captured by the rule will be skipped.

Paragraph rules in memoQ have a couple of peculiarities.

First, we need to use parentheses around parts of our regex, otherwise the rule will be rejected. In our case, parentheses are helpful as they tell the system which part of the line is to be translated. In other cases, there might be no need to use them, but you still have to, just to make the rule work.

Second, we are not allowed to use the ^ and $ anchors for the start and end of line. The reason is that paragraph rules match a whole line by design so anchors are deemed redundant.

With that in mind, we can construct a rule as follows:

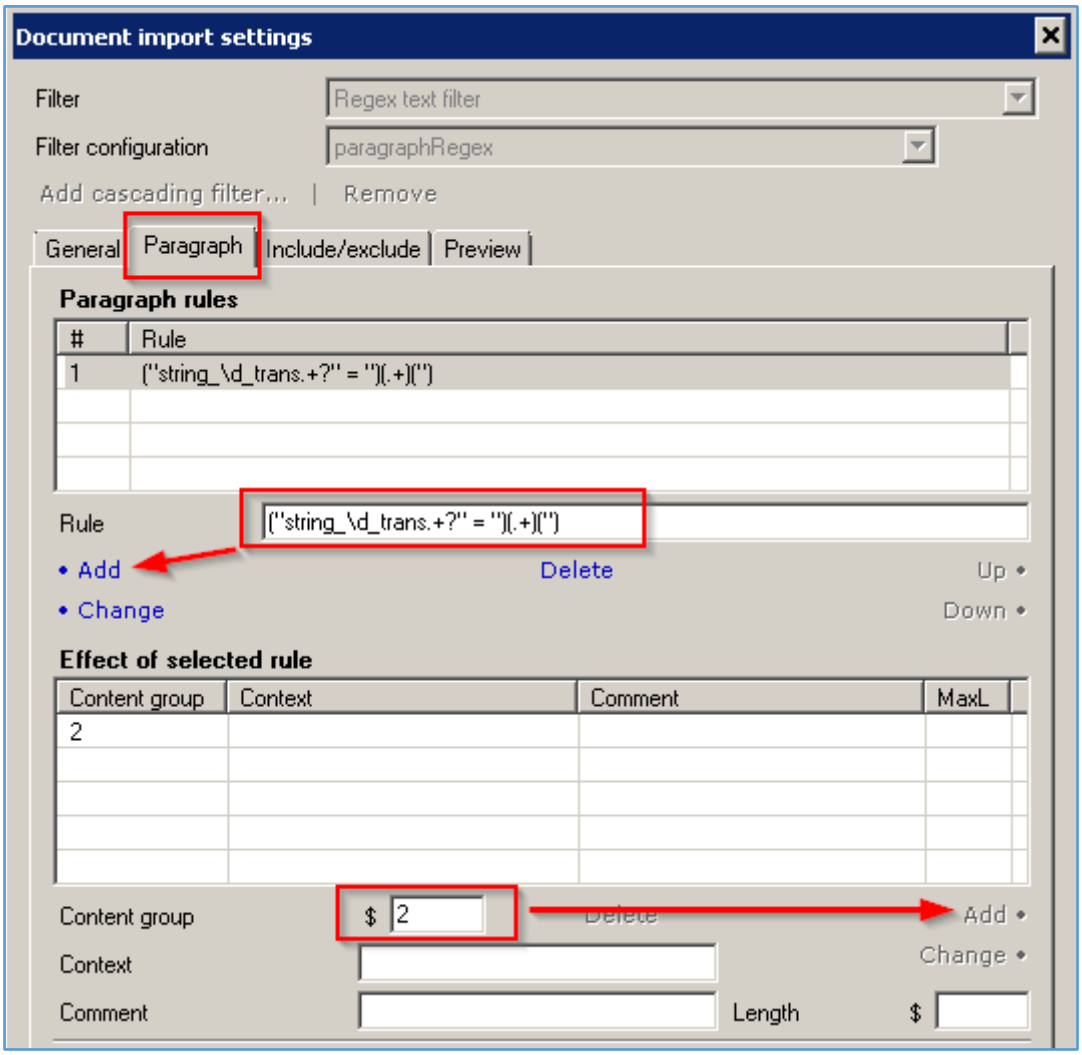

Figure 10-36. memoQ: Text file filter settings—paragraph rules

The process to create this rule consists of several steps:

- we take our standard regex rule capturing translatable lines;
- we remove the line anchors (^ and $) and then split the regex into three groups using parentheses: the first group captures an ID section, the second translatable part and the third a final straight quote;
- we tell memoQ that only the second group needs to be translated (see the bottom section on a screenshot above).

Our paragraph rule helped us filter out non-translatable strings. We still need to take care of article numbers. As mentioned before, the strategy to do that will be to use a Regex Tagger. However, it is also possible to add a rule for article numbers on the Include/exclude tab of the Regex text filter. I chose to go with a Regex Tagger based on two considerations: 1) to maintain consistency with the way we are going to process the Big Three and 2) because applying exclusion rules in the Regex text filter may lead to segmentation problems—memoQ may break sentences by the resulting tags treating them as the end of segment.

Custom file filter preview

Score: 10

The Preview tab clearly shows how our rules affect the outcome:

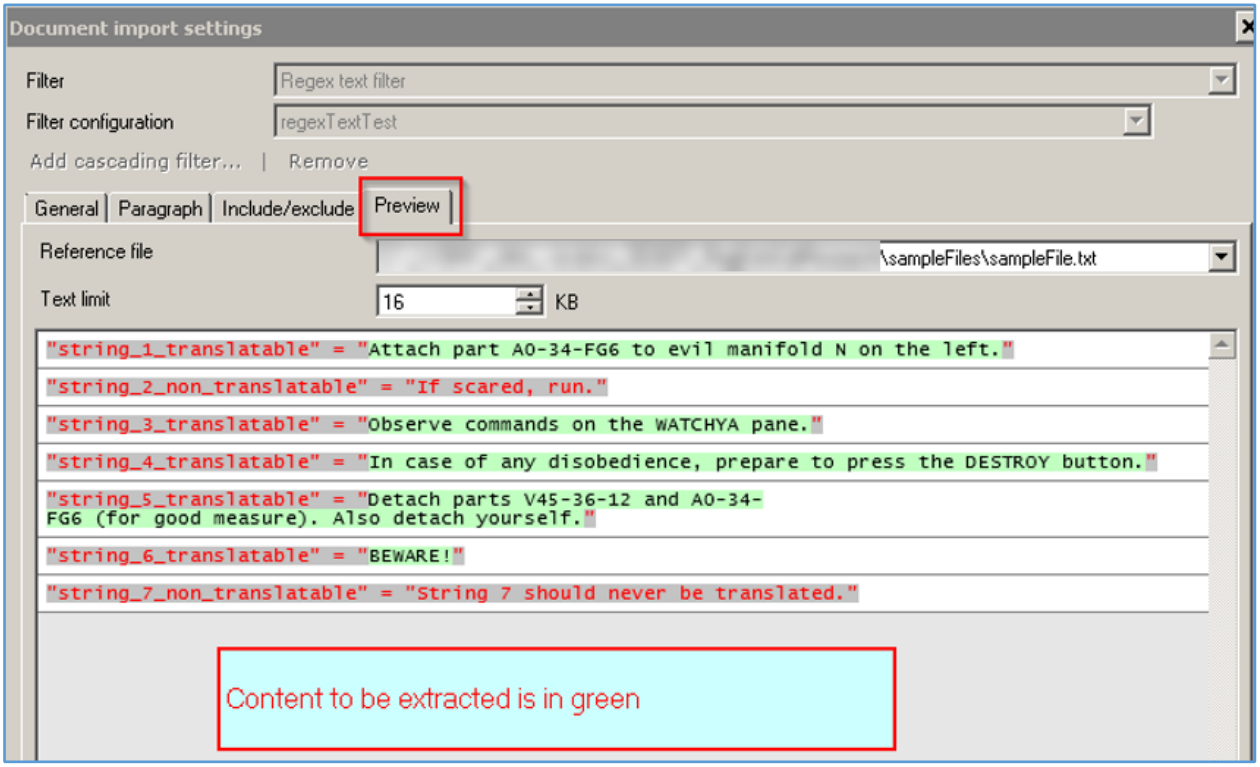

Figure 10-37. memoQ: Text file filter preview

Custom file filter reuse

Score: 10

Once created, a new filter becomes available for all new projects.

Custom regex configuration creation

Score: 10

To protect article numbers, we will create a custom Regex Tagger based on the default Regex Tagger filter (available among all other file filters). We will then add it to our custom text file filter to build a cascading filter. Our custom Regex Tagger will also take on a life of its own, so we will be able to use it in other cascading filters (for the Big Three).

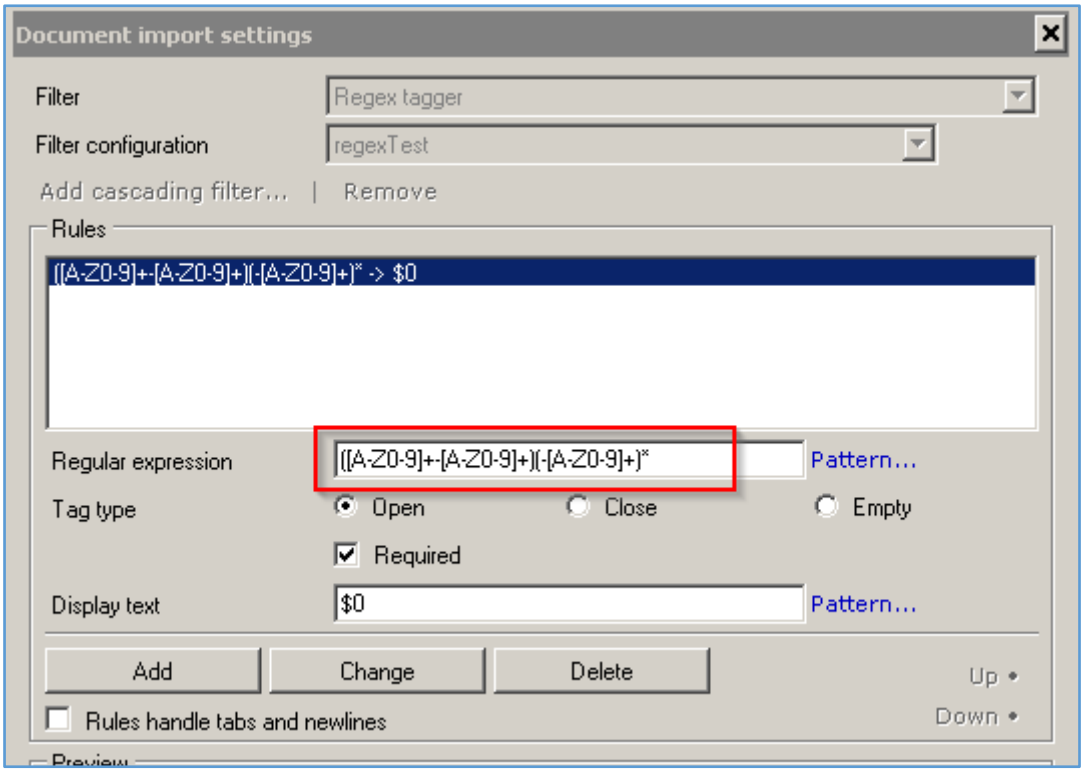

Figure 10-38: memoQ: Regex Tagger settings

Custom regex configuration preview

Score: 8

A preview is available at the bottom of the same window where a custom Regex Tagger is set up:

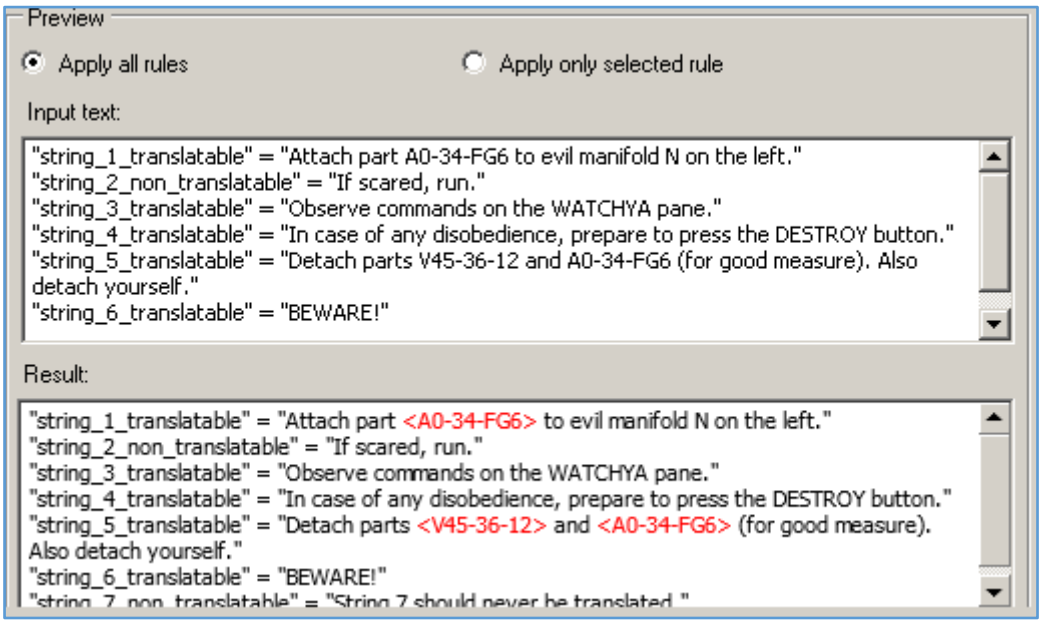

Figure 10-39. memoQ: Regex configuration preview

The reason I lowered the score is that a preview is only available for a pasted sample of a source file. You cannot load a file in its entirety. With larger files, it does make a difference: all possible cases can hardly be predicted, and so the ability to scroll through a whole of the document to see where additional tweaks to the rules might be needed is irreplaceable.

Custom regex configuration reuse

Score: 5

Once created, a customized Regex Tagger becomes available for all new projects.

Creating a cascading filter

To combine a custom file filter with a custom Regex Tagger, we need to create a new cascading filter. It can be done on the Resource Console:

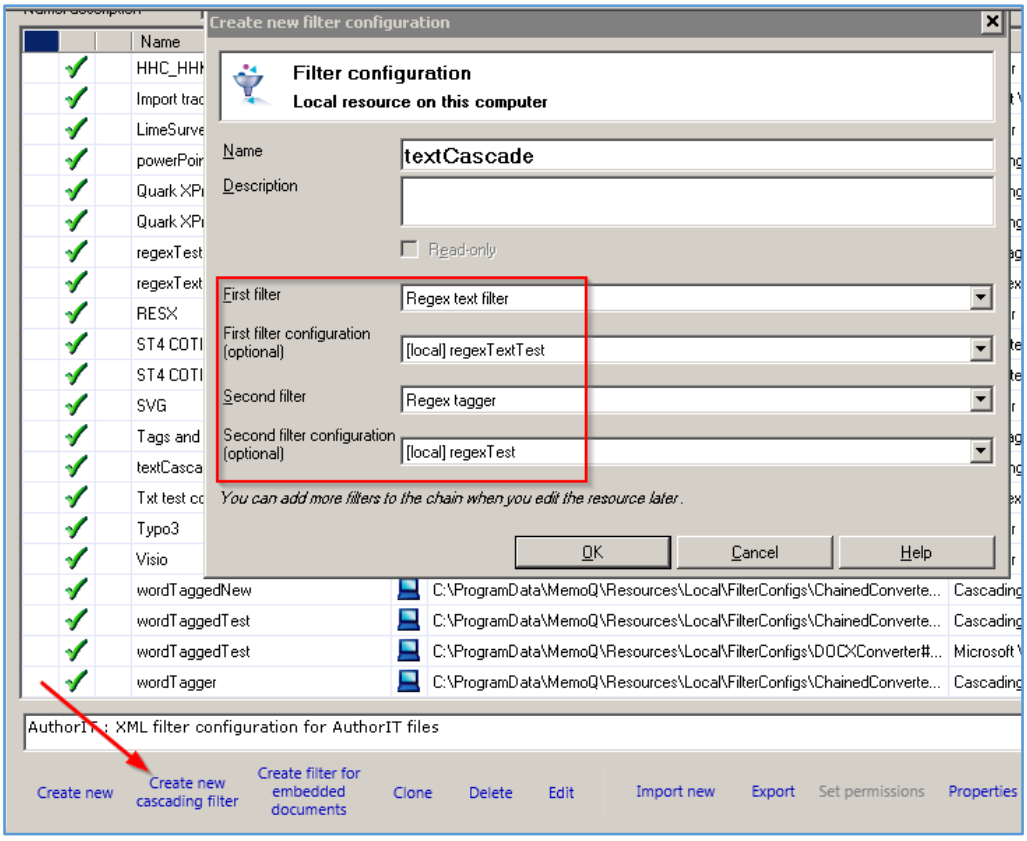

Figure 10-40. memoQ: New cascading filter creation

The Big Three

Total score: 53

The procedure for the Big Three is identical, except in our scenario we do not have to modify default file filters for DOCX, XLSX or PPTX. So we just build three more cascading filters, adding our custom Regex Tagger (the one we have created for a text file) to default file filters. For instance, to create a cascading filter for DOCX, our custom Regex Tagger has to be added to the default DOCX file filter.

Once the new cascading filters are created, they become available at the project creation stage:

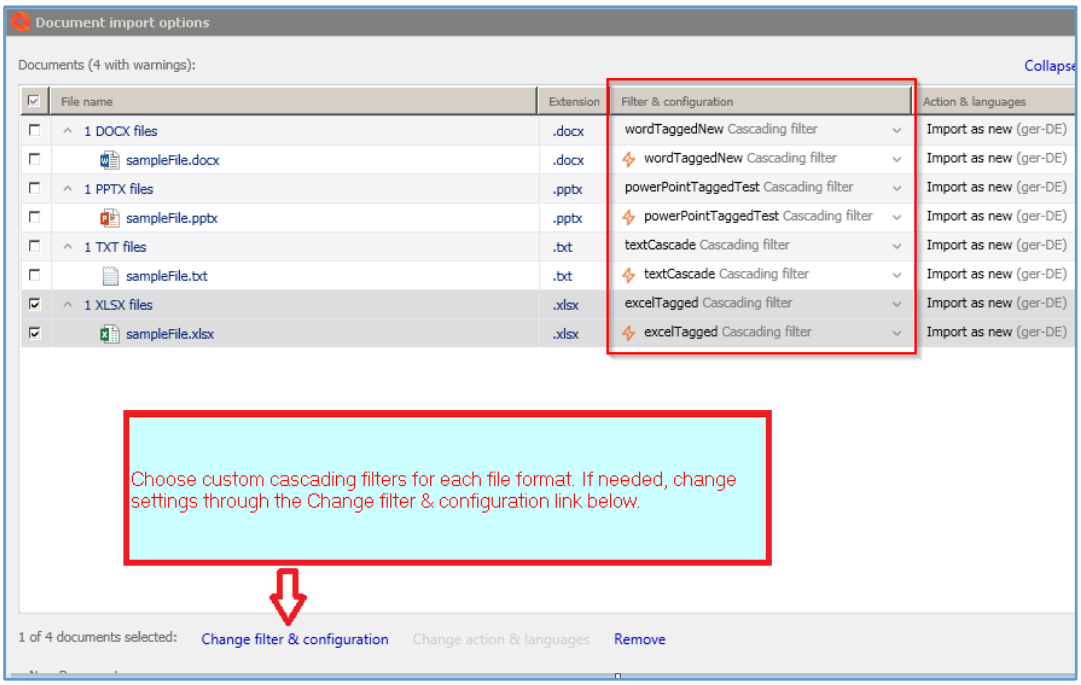

Figure 10-41. memoQ: Choosing custom cascading filters for text file and Big Three

The score for the Big Three is calculated as follows:

Custom regex configuration creation for the Big Three: 10 + 10 + 10 = 30

Custom regex configuration reuse for the Big Three: 5 + 5 + 5 = 15

Custom regex configuration preview for the Big Three: 8

Segment filtering

Score: 10

As one could expect, memoQ's segment filter is very clean. Settings are available under the cog icon in the Editor view:

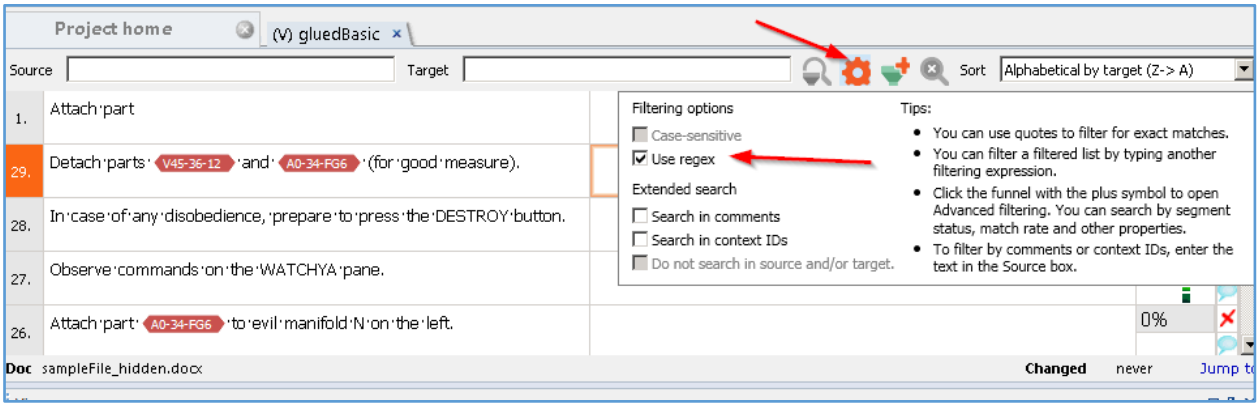

Figure 10-42. memoQ: Segment filter settings

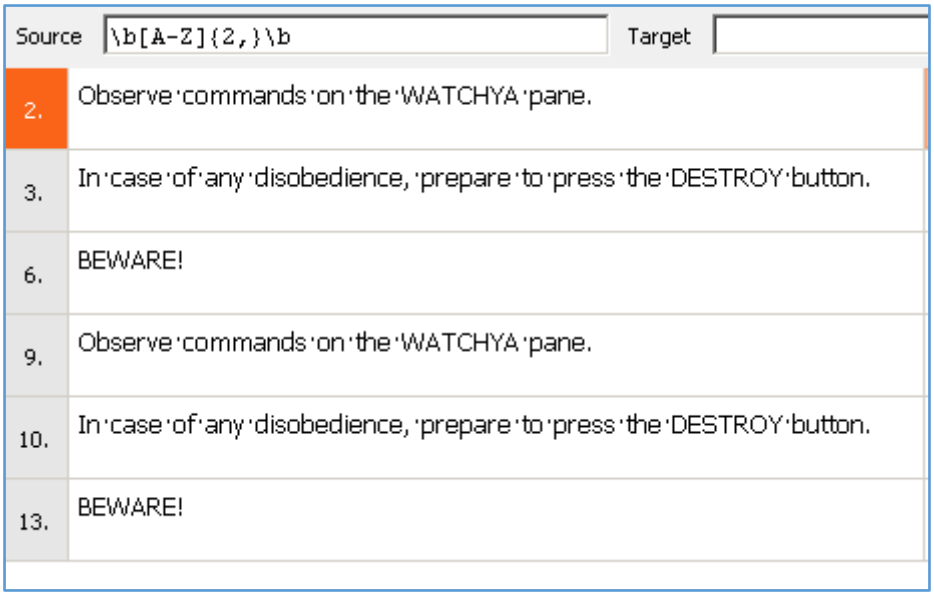

Figure 10-43. memoQ: Editor view with segment filter applied

Search and replace

Total score: 20

Search and replace functionality is also solid. You can quickly invoke it via *Ctrl-f* (or *Ctrl-h* to replace) and then click the Change these link to adjust settings. More experienced memoQ users can do without the link and use the icons on the right side of the Quick find and replace window.

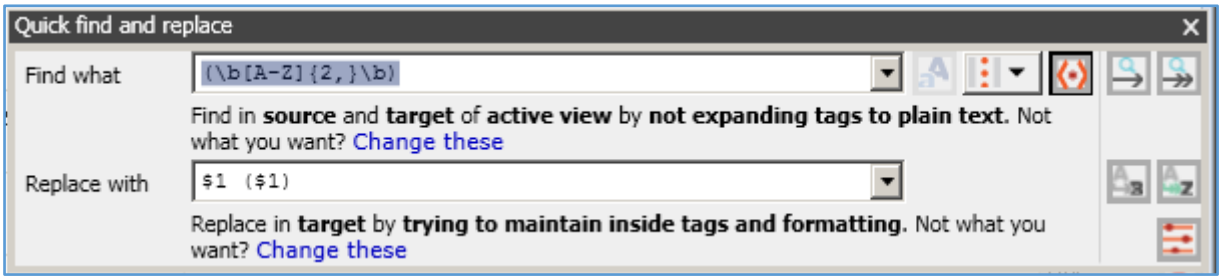

Figure 10-44. memoQ: Quick find and replace window

The advanced settings window (also available through Edit > Find And Replace > Advanced on the upper menu):

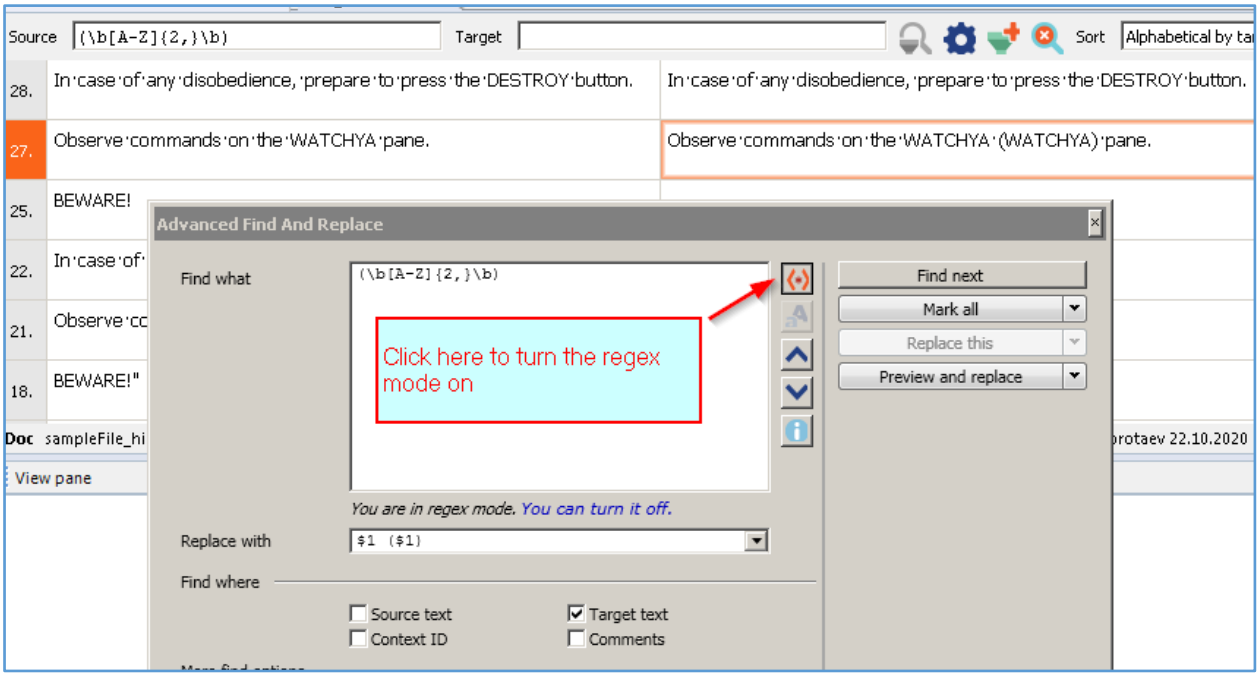

Figure 10-45. memoQ: Search and replace settings

Memsource + Memsource Editor for Desktop 20.21.3

Quadrant: Editor's Friends

Overall score: 52

Memsource is arguably the most popular cloud CAT tool on the market. It offers sophisticated workflow-related functionality and a bunch of connectors and integrations with other systems and environments. However, support for regexes in Memsource, while being above average for cloud tools, is not very impressive. Things start looking up if we consider the cloud part of Memsource together with *Memsource Editor for Desktop*. It is a relatively light-weight desktop tool that enhances user experience at linguistic stages like translation, editing and proofreading. The desktop editor can connect to the cloud and has been around since the very early days of the Memsource development. The inclusion of this tool goes against the principle of an integrated environment I relied on to select participating CAT systems for the report. Still, I eventually decided to include Editor for Desktop based on the following considerations: a) it is a Memsource product, not a third-party tool, b) it is free, c) it integrates seamlessly with the cloud and d) it is very widely used by editors and translators working with Memsource projects. However, I adjusted all Memsource scores earned thanks to Editor for Desktop's functionality by deducting 2 points from each such score.

File preparation

Memsource supports regexes for a number of file formats. Settings are available in the File Import Settings area:

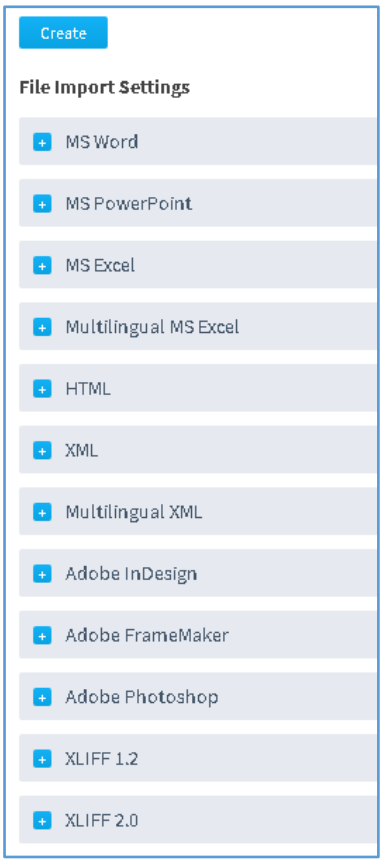

Figure 10-46. Memsource: File import settings area

Text files

To prepare a text file, we need to go to the TXT section and click the + icon. The Translatable text and Convert to Memsource tags fields appear. Using these fields, we can define both translatable text and non-translatables to be put into tags. Only one regex per field is allowed, but it can be sidestepped in some cases by using pipes (|) to combine regexes. Another important consideration to bear in mind is that Memsource only allows so-called *possessive quantifiers* for groups containing other quantifiers within them. Possessive quantifiers may significantly reduce computational strain when more complex regexes are applied, so this is how Memsource protects itself from overload. The regex for our article numbers falls under this category of complex regexes as we have groups with quantifiers there with other quantifiers applied to those groups. This means we have to modify it using possessive quantifiers to make it work in Memsource.

Custom file filter creation

Score: 8

The general logic is that we first define strings to be translated (as a whole), which leaves all non-translatable strings out. We then additionally define rules to put ID sections and article numbers in inline tags.

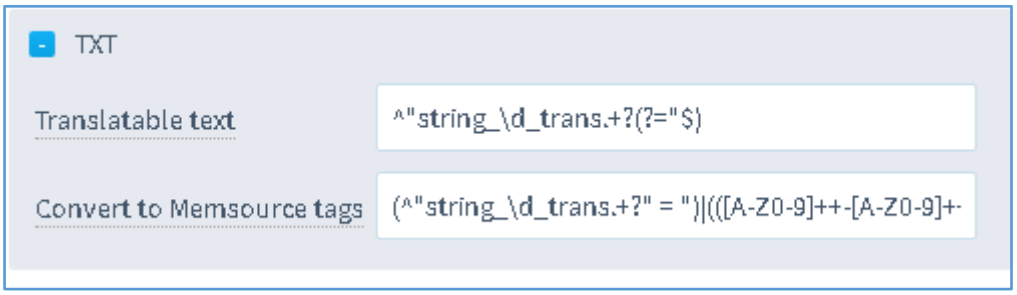

Figure 10-47. Memsource: Text file filter settings

I had to combine regexes for ID sections and article numbers using the OR operator (|) to overcome the *one field—one regex* restriction. It made the regex quite long so it did not fit into the visible area of the text field. Here it is in its entirety:

```
(^"string_\d_trans.+?" = ")|(([A-Z0-9]++-[A-Z0-9]++)(-[A-Z0-9]++)*+)
```

Additional + signs make quantifiers possessive as is required.

This is how our text file would look like in the Editor view:

#	Source: en-us
1	1 Attach part 2 to evil manifold N on the left.
2	1 Observe commands on the WATCHYA pane.
3	1 In case of any disobedience, prepare to press the DESTROY button.
4	1 Detach parts 2 and 3 (for good measure).
5	Also detach yourself.
6	1 BEWARE!

Figure 10-48. Memsource: Text file in Editor view

Though the layout with different rules for translatable text and inline tags offers some flexibility, the *one field—one regex* restriction is really limiting so I had to lower the score to 8.

Custom file filter preview

Score: 0

No preview is available.

Custom file filter reuse

Score: 0

Memsource offers two indirect ways of saving a custom file filter but neither is good enough considering our test case requirements.

First, changes can be made at the general Settings level. But this is hardly an option as it will only modify the default text filter, which is not our intention (we need to be able to choose between the default filter and a custom one for new projects).

Second, we can save a changed filter as part of a custom *project template*. We will then have to make sure that all new files where our new filter is to be applied are processed using this template. This can be an acceptable solution in some cases; however, many other things may need to be changed from project to project, and the use of a project template for the narrow purpose of applying a file filter seems like overkill.

So no points were awarded in this category.

Custom regex configuration creation

Score: 4

According to our scoring system, regex configurations inseparable from file filters merit half the score of the latter.

Custom regex configuration preview

Score: 0

No preview is available.

Custom regex configuration reuse

Score: 0

See above about custom file filter reuse.

The Big Three

Total score: 16

DOCXs and XLSXs are supported in a similar manner to plain text. Only the Convert to Memsource tags field is available, but this is enough for our purposes. Here is how the Excel filter can be configured:

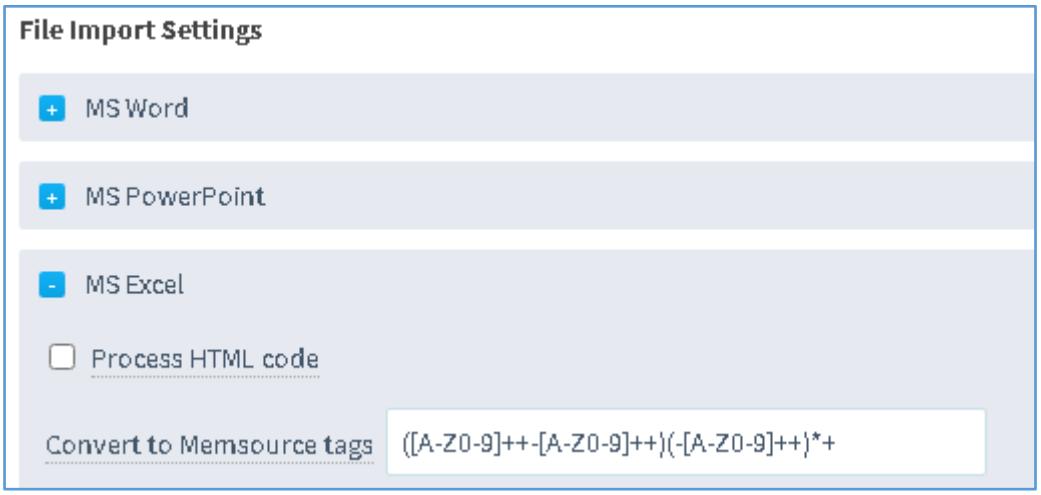

Figure 10-49. Memsource: Regex configuration for XLSX

PowerPoint files, however, are not supported.

No preview or reuse functionality is available (see the Custom file filter reuse section for more details).

The score is calculated as follows:

Custom regex configuration creation for the Big Three: 8 + 8 + 0 (PPTX) = 16

Segment filtering

Score: 8

The segment filter is good but only available in Memsource Editor for Desktop. The score would have been perfect, but 2 points were taken away to penalize for the violation of the integrated environment principle as discussed above.

The search is case-insensitive by default so the Case sensitive checkbox should be selected along with the Use regexp one:

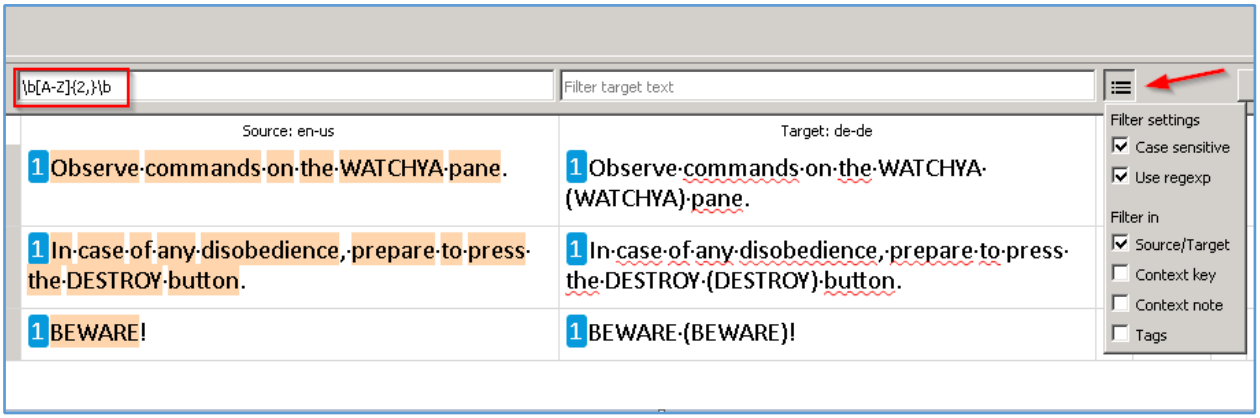

Figure 10-50. Memsource Editor for Desktop: Segment filter settings

Search and replace

Total score: 16

This functionality too can only be found in Memsource Editor for Desktop. Again, points were deducted from the potentially perfect score.

Note that to backreference groups, a backslash must be used in a replacement regex instead of a more common dollar sign.

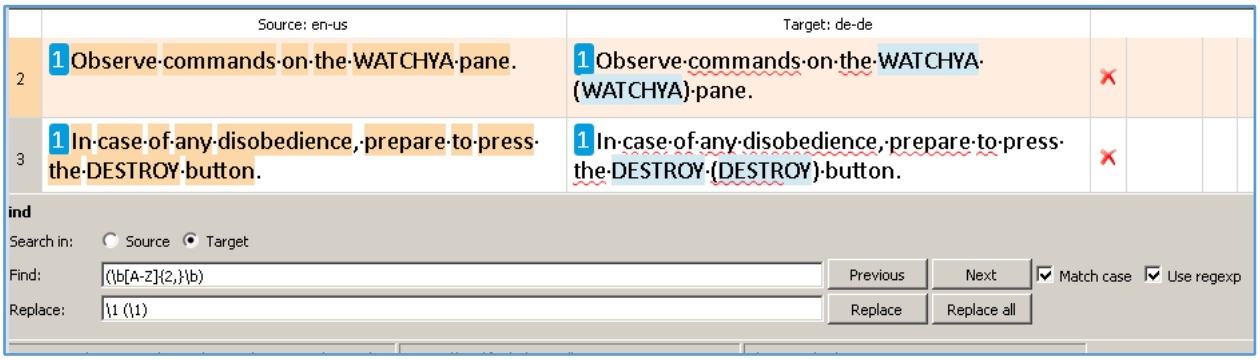

Figure 10-51. Memsource Editor for Desktop: Search and replace settings

OmegaT 4.3.2

Quadrant: Editor's Friends

Overall score: 29

OmegaT, a long-standing and well-known open-source system, is arguably the most mature CAT tool available free of charge. Considered by many a viable option (especially when a budget is tight), it is not, however, a very regex-friendly environment. The segment filtering and search and replace functions are solid, but the file preparation side lacks any regex-related capabilities that could be used in our test case.

File preparation

Total score: 2

Only hidden text protection for DOCX is available, which merits 2 points (as in all similar cases).

Segment filtering

Score: 10

Segment filters can be set using the *Ctrl-f* key combination or through Edit > Search Project on the upper menu. After all required checkboxes are selected, the Filter button should be pressed to filter segments. In a bit quirky twist, the Filter button only becomes active after the Search button is pressed. Note the Memory checkbox: it should also be checked for the whole thing to work.

The search is case-insensitive so the Case sensitive checkbox must be selected too.

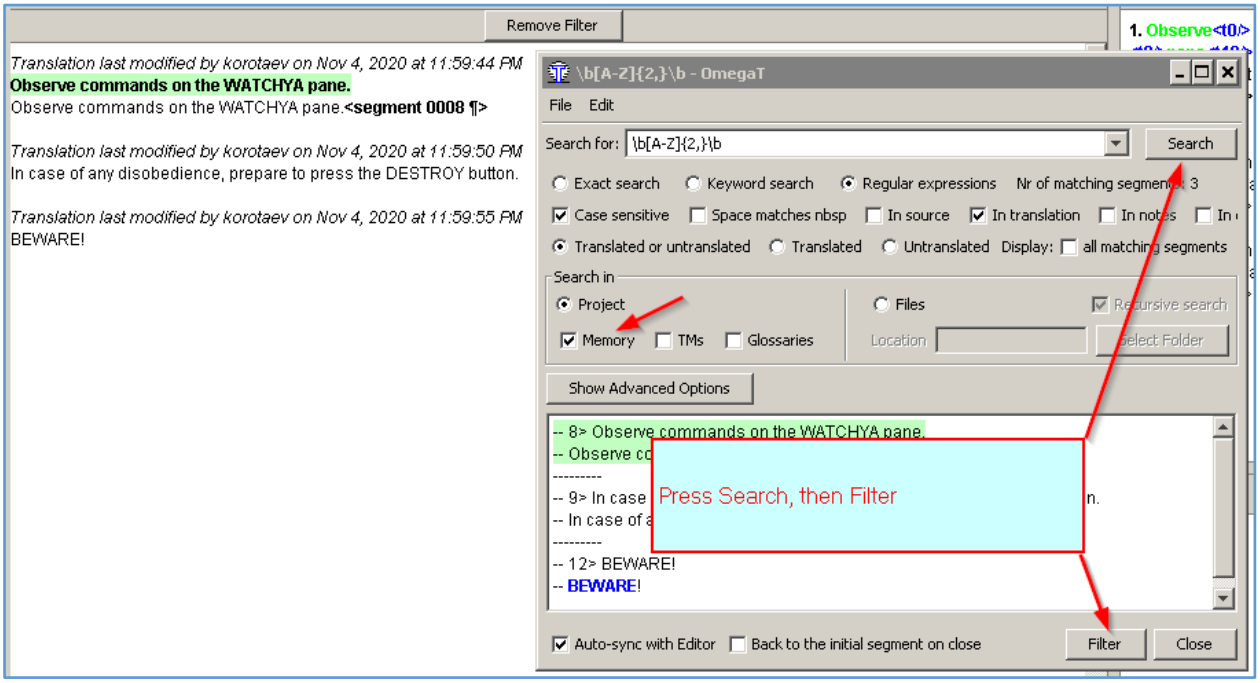
Remove Filter
Translation last modified by korotaev on Nov 4, 2020 at 11:59:44 PM
Observe commands on the WATCHYA pane.
Observe commands on the WATCHYA pane.<segment 0008 ¶>
Translation last modified by korotaev on Nov 4, 2020 at 11:59:50 PM
In case of any disobedience, prepare to press the DESTROY button.
Translation last modified by korotaev on Nov 4, 2020 at 11:59:55 PM
BEWARE!
\b[A-Z]{2,}\b - OmegaT
File Edit
Search for: \b[A-Z]{2,}\b
Search
Exact search
Keyword search
Regular expressions
Case sensitive
Space matches nbsp
In source
In translation
Translated or untranslated
Translated
Untranslated
Display:
all matching segments
Search in
Project
Files
Recursive search
Memory
TMs
Glossaries
Location
Select Folder
Show Advanced Options
-- 8> Observe commands on the WATCHYA pane.
-- 12> BEWARE!
-- BEWARE!
Press Search, then Filter
Auto-sync with Editor
Back to the initial segment on close
Filter
Close

Figure 10-52. OmegaT: Segment filter settings

Search

Score: 7

The search functionality can be invoked by pressing *Ctrl-k* or selecting Edit > Search and Replace on the upper menu. The navigation between occurrences is not supported (hence the reduction in score). Instead, all entries are shown in the field below, marked in blue. Search is case-insensitive so the Case sensitive checkbox should be selected.

Select the Auto-sync with Editor checkbox at the bottom to quickly jump to a segment from within the preview field.

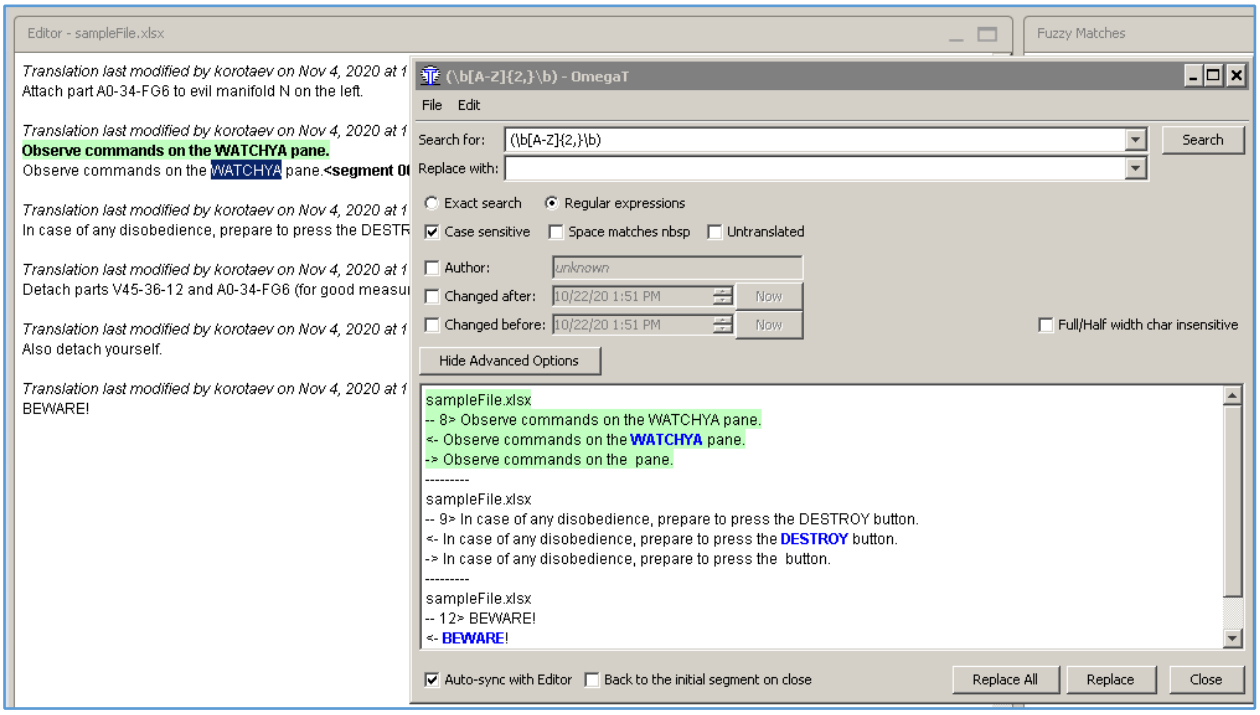
Editor - sampleFile.xlsx
Fuzzy Matches
Translation last modified by korotaev on Nov 4, 2020 at 1
Attach part A0-34-FG6 to evil manifold N on the left.
Translation last modified by korotaev on Nov 4, 2020 at 1
Observe commands on the WATCHYA pane.
Observe commands on the WATCHYA pane.<segment 0
Translation last modified by korotaev on Nov 4, 2020 at 1
In case of any disobedience, prepare to press the DESTR
Translation last modified by korotaev on Nov 4, 2020 at 1
Detach parts V45-36-12 and A0-34-FG6 (for good measu
Translation last modified by korotaev on Nov 4, 2020 at 1
Also detach yourself.
Translation last modified by korotaev on Nov 4, 2020 at 1
BEWARE!
(\b[A-Z]{2,}\b) - OmegaT
File Edit
Search for: (\b[A-Z]{2,}\b)
Search
Replace with:
Exact search
Regular expressions
Case sensitive
Space matches nbsp
Untranslated
Author: unknown
Changed after: 10/22/20 1:51 PM
Now
Changed before: 10/22/20 1:51 PM
Now
Full/Half width char insensitive
Hide Advanced Options
sampleFile.xlsx
-- 8> Observe commands on the WATCHYA pane.
<- Observe commands on the WATCHYA pane.
-> Observe commands on the pane.
---
sampleFile.xlsx
-- 9> In case of any disobedience, prepare to press the DESTROY button.
<- In case of any disobedience, prepare to press the DESTROY button.
-> In case of any disobedience, prepare to press the button.
---
sampleFile.xlsx
-- 12> BEWARE!
<- BEWARE!
Auto-sync with Editor
Back to the initial segment on close
Replace All
Replace
Close

Figure 10-53. OmegaT: Search settings

Replace

Score: 10

The replace window first shows the preview of an intended replacement:

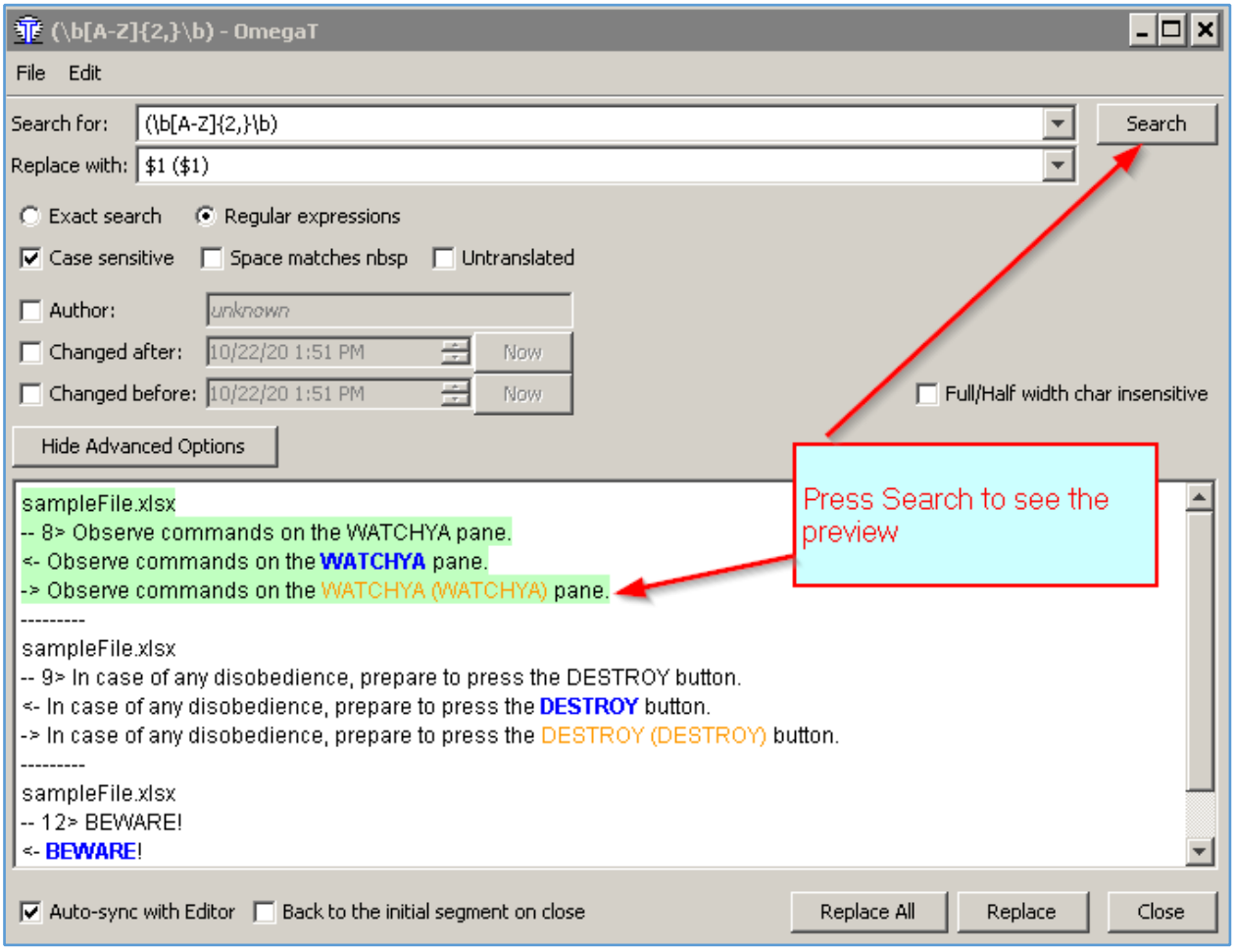

Figure 10-54. OmegaT: Replace settings

To start replacing in an interactive manner, you should press the Replace button and then use the buttons that would appear above the segments:

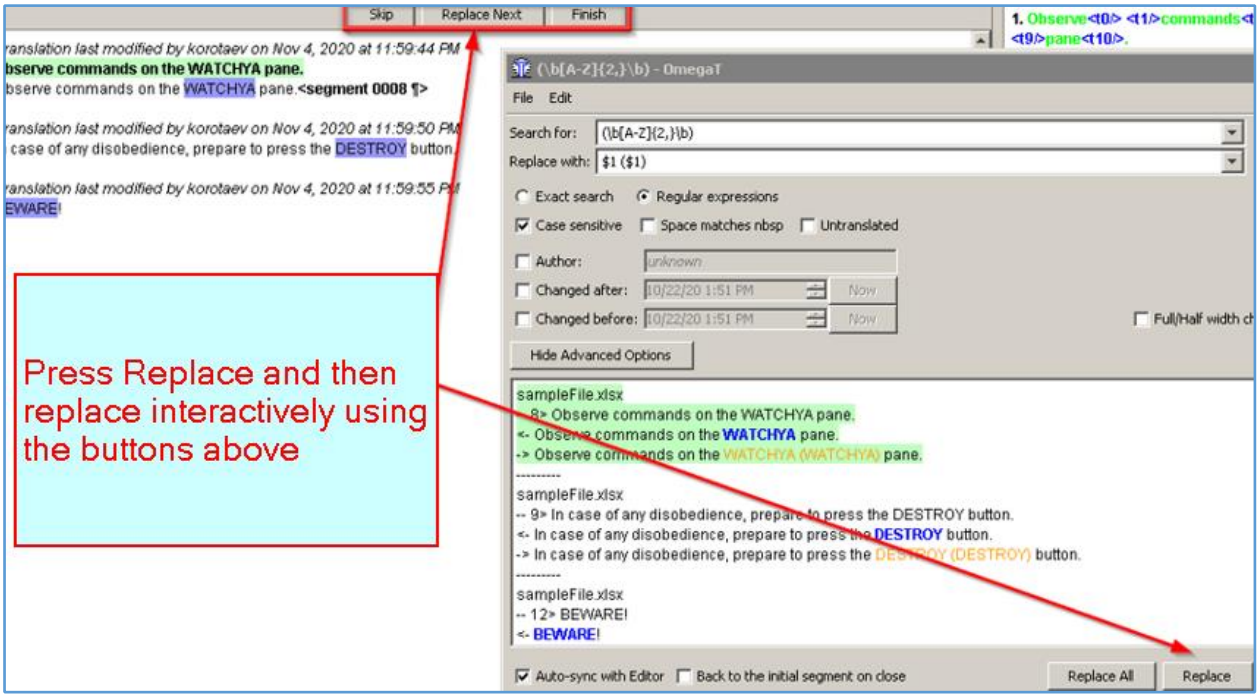

Figure 10-55. OmegaT: Replace Next functionality

SDL Trados Studio 2019 Professional

Quadrant: Regular Beasts

Overall score: 124.25

Trados Studio is a real juggernaut among CAT tools. The rich history of this system—which includes a total reinvention of the by-then already legendary *old* Trados after it had been purchased by SDL and merged with the latter's own CAT tool, SDLX, back in 2005—clearly manifests itself in the width and depth of its functionality. If not without its detractors, Trados Studio remains the most popular and arguably most functional of all CAT tools on the market.

Given all that, one could say that the system's regex functionality, historically, has slightly underperformed. Regexes have long been available on both file preparation and linguistic sides, but their implementation was not necessarily very consistent and user-friendly. However, with the introduction of the *Advanced Display Filter* for segment filtering and inclusion of the *embedded content* sections into XLSX and PPTX file types, the regex component has caught up and can now be considered one of the best in business along with memoQ's and Alchemy Catalyst's.

File preparation

File filters in Trados Studio are known as *file types*. They are available through FIle > Options or at the project creation stage. Users can create new file types to handle text files and modify many default file types through so-called *embedded content*. Together, new file types and embedded content provide a robust and flexible mechanism to protect non-translatable text and prepare files for translation. However, file filters and regex configurations (as they are understood in this report) are not clearly separated, and the reuse options are limited. All of this led to the reduction in scores in respective categories.

Text files

Trados Studio offers rich functionality to create new text file types and modify existing ones. In our scenario, we will create a new file type and then add regexes to define translatable (the Document structure section) and non-translatable (the Inline tags section) content.

Custom file filter creation

Score: 10

New file types can be created based on several predefined types—most notably, in the context of this report, *regular expression delimited text*.

How-to

First, we need to create a new file type based on regex delimiters. To do so, we add a new file type in the Options window:

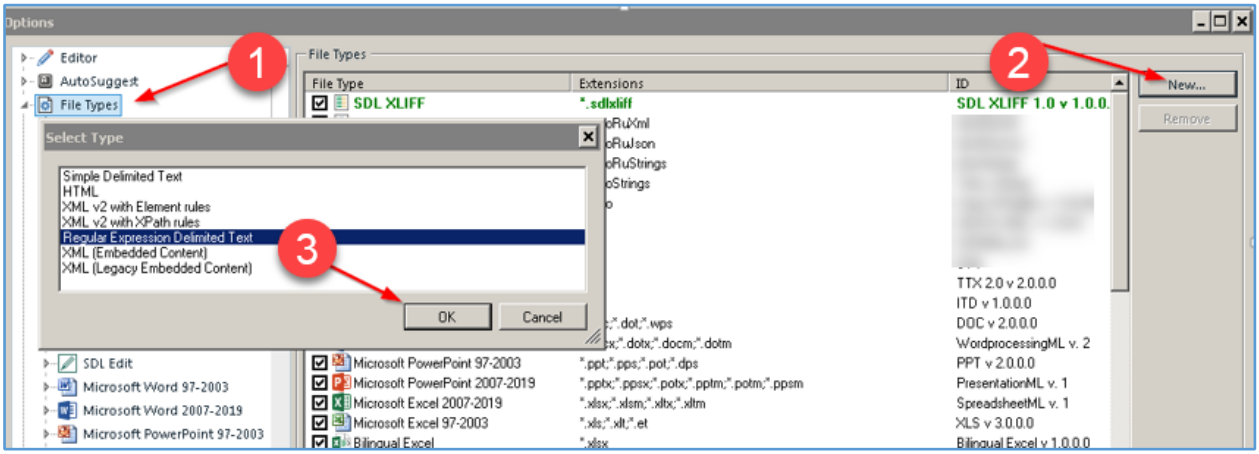

Figure 10-56. Trados Studio: New file type creation (step 1)

Then we name the new filter:

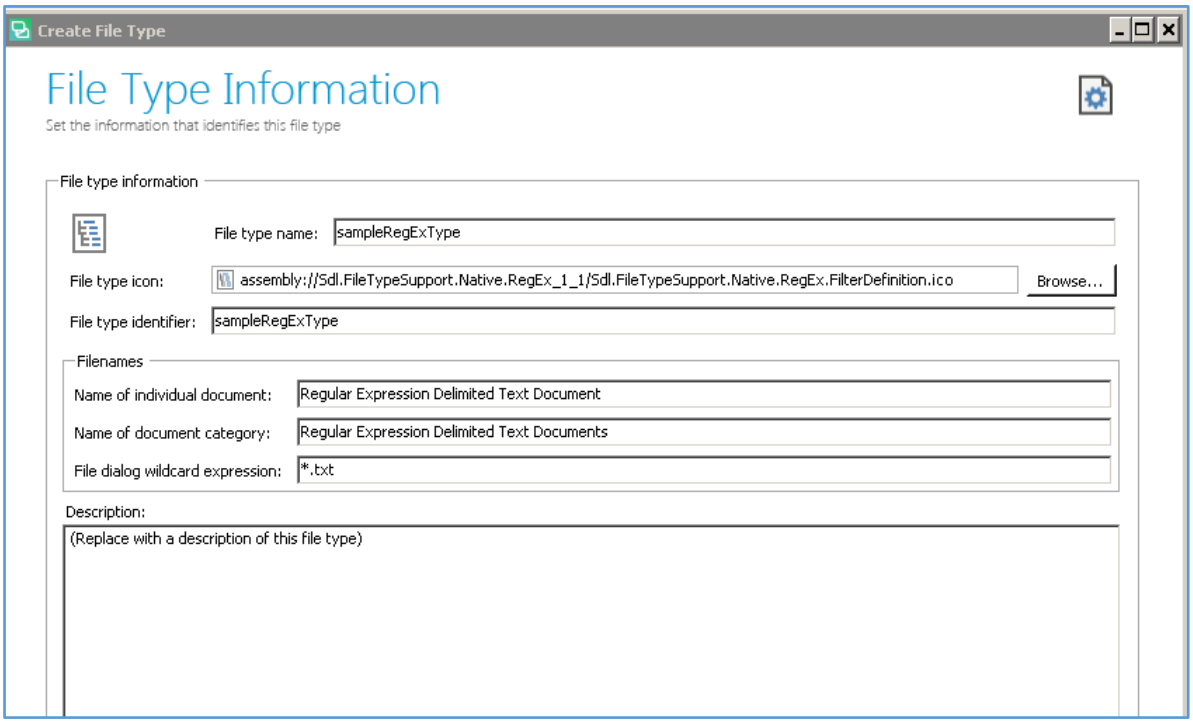

Figure 10-57. Trados Studio: New file type creation (step 2)

After that, the filter is available in the file types tree on the left, and we can add our regexes. We can start with defining boundaries for translatable content in the Document structure section:

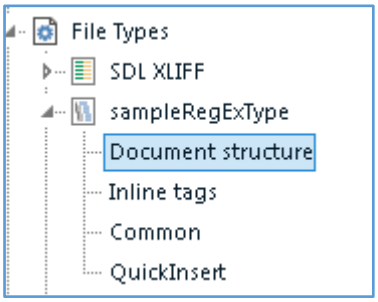

Figure 10-58. Trados Studio: Document structure node in file types tree

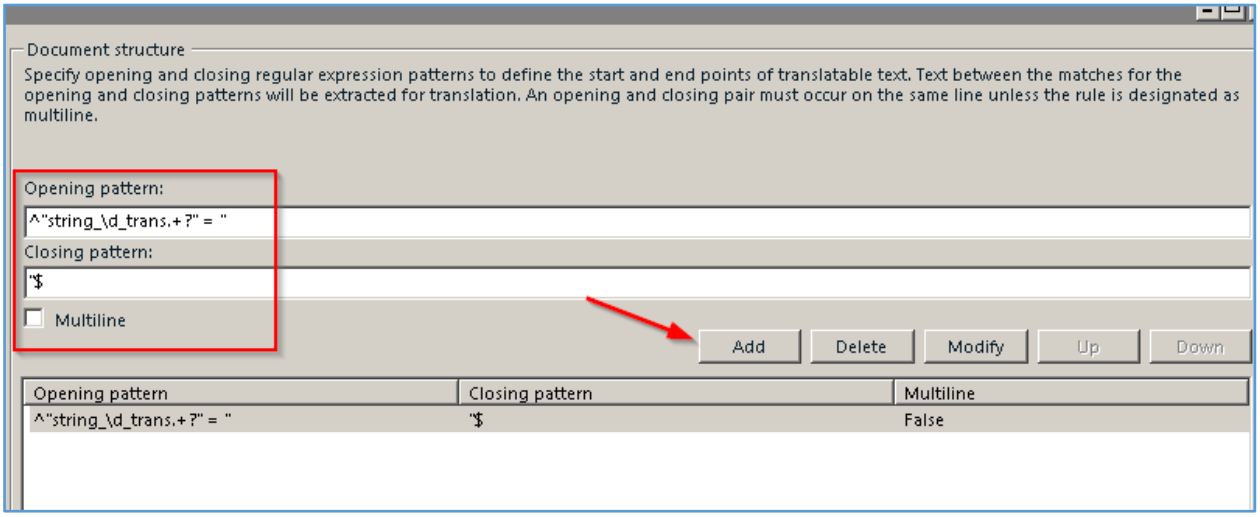

Figure 10-59. Trados Studio: Text file filter settings (document structure)

The Document structure section may contain a predefined regex with the opening pattern ^ and closing pattern $. It basically says that everything should be extracted. In our case, we need to delete it.

Next, we move to the Inline tags section and set the rule for article numbers. The Advanced button gives us the Include with text option, which means our placeholders will only be seen if there is other text in the segment. In our case, article numbers are only found between other words so it is irrelevant, but in a different project we may encounter segments consisting of placeholders only—this option prevents them from being displayed in the Editor view.

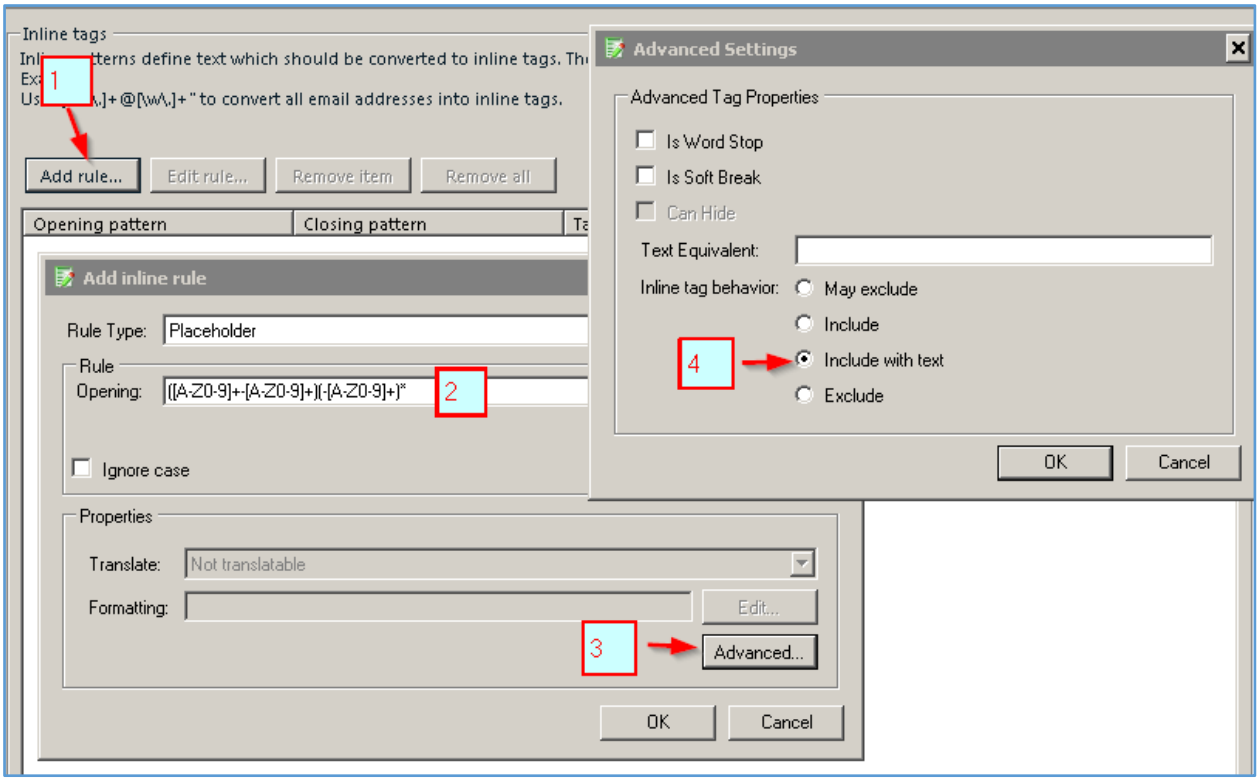

Figure 10-60. Trados Studio: Text file filter settings (inline tags)

Custom file filter preview

Score: 10

SDL Trados has arguably the best preview functionality among all CAT tools. For any file type, there is a Preview button available at the bottom that allows to quickly see the whole content of a given file with all current rules applied. A preview appears in a new window:

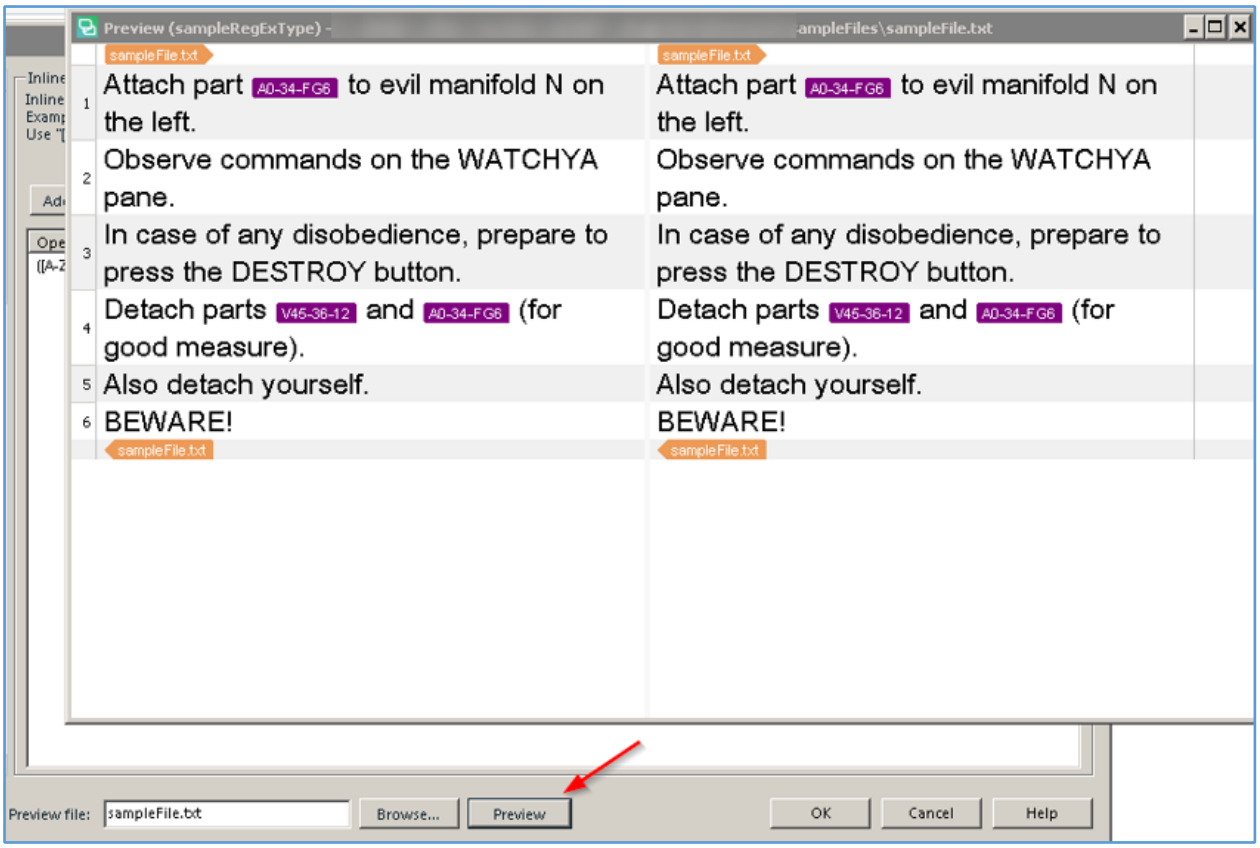

Figure 10-61. Trados Studio: Text file filter preview

Custom file filter reuse

Score: 7

The reuse capability is the only underperforming cog in Trados Studio's otherwise impressive regex machine. The chief flaw is lack of any drop-down functionality to select a file type for a project. The only way to make sure the right file type will be used is to move it up the list with the respective button or deselect other file types for the same file extensions.

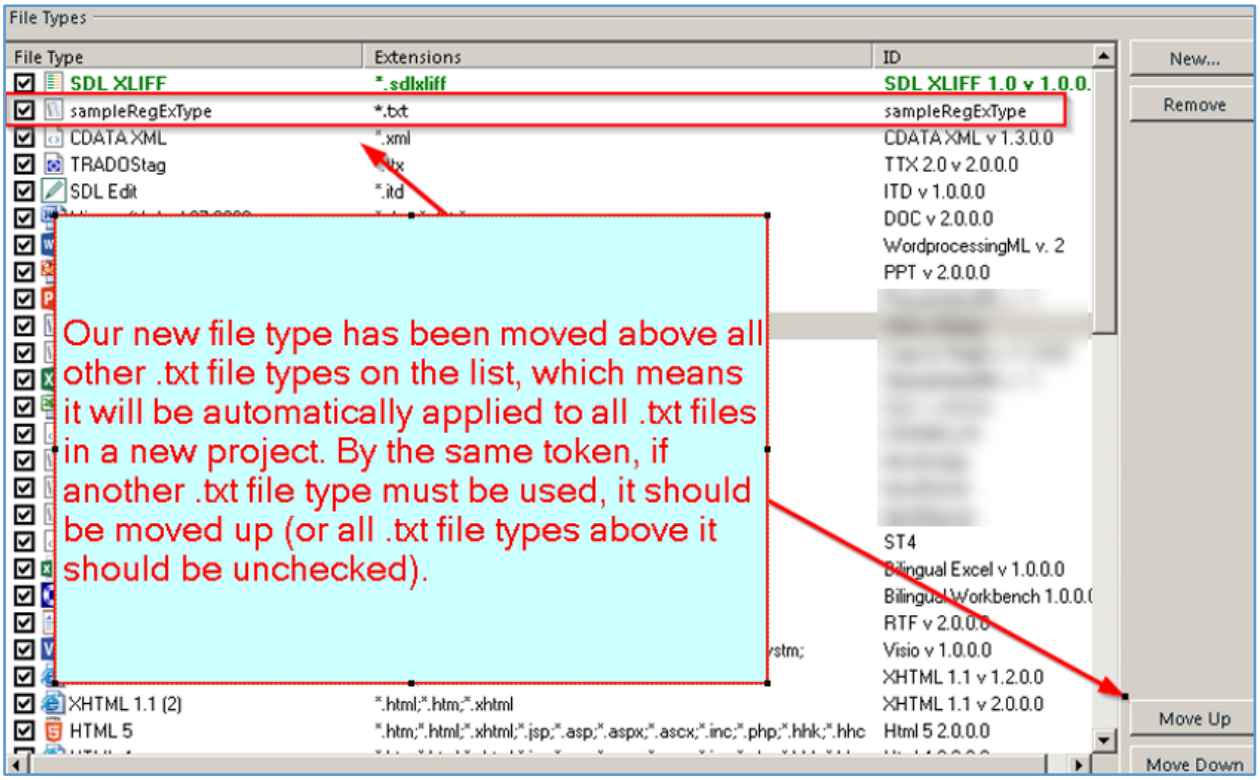

Figure 10-62. Trados Studio: Custom file filter moved up on file types list

The general logic is that, for any file to be translated, Trados goes through all available file types from top to bottom and employs the first file type matching the file extension. In our case, the extension would be *.txt*, and we have to make sure that our new file type is located above all other *.txt* file types. Next time, with another project, we might want to use another *.txt* file type. In this case, we will again have to either move it up the list or deselect all other *.txt* file types located above it. While workable, this process is clearly inferior to a straightforward selection of a required file filter from a list of options. For instance, we may have several different *.txt* files in the same project, with different file types to be applied to each of them. This cannot be done in Trados Studio, and the only solution would be to create a separate project for each of those files.

Custom regex configuration creation

Score: 5

As we saw earlier, general file filters and more specific regexes for inline tags cannot be separated in Trados Studio. As in all similar cases, it halves the score in this category.

Custom regex configuration preview

Score: 10

A preview is always there for all file type sections.

Custom regex configuration reuse

Score: 1.75

Again, we need to halve the score, only this time the maximum value would be 2.5 (half of 5), and the value assigned in the file filter reuse category is 7, not 10. So the score is calculated as 2.5*7/10 = 1.75.

The Big Three

Total score: 50.5

Regex configurations for the Big Three are set in the Embedded content section of each file type:

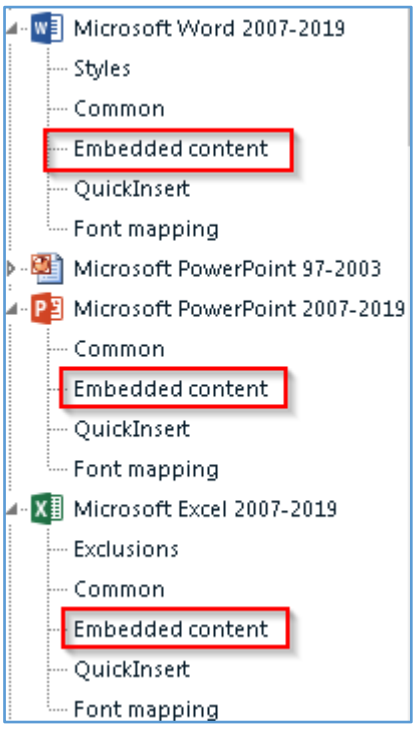

Figure 10-63. Trados Studio: Embedded content sections in file types tree

The embedded content engine allows us to define regex configurations to be applied to document elements selected in the Define Document Structure Information window. This last part can be confusing. The list of elements is quite long, and it may not be very obvious which of them needs to be chosen. In our case, we should select *cell* for XLSX, *paragraph* for DOCX and *slide* for PPTX. As an example, the PPTX settings are shown below:

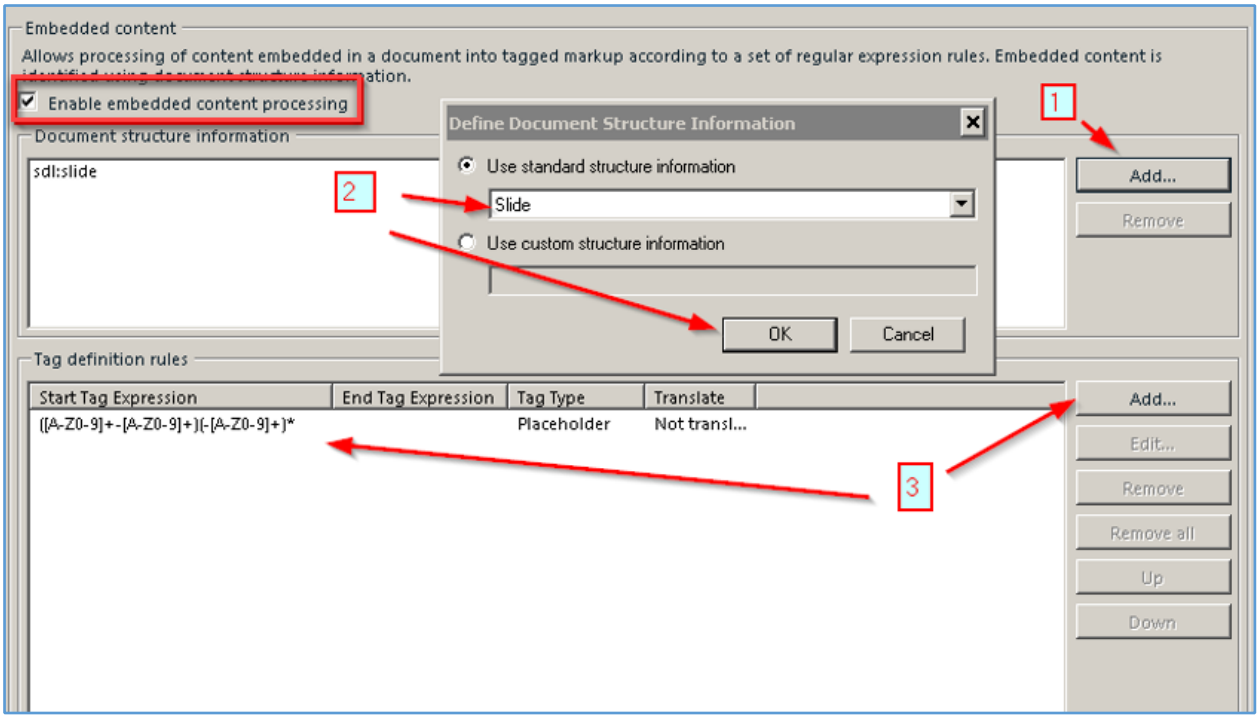

Figure 10-64. Trados Studio: Regex configuration for PPTX

The way a new regex is added is the same as for the Inline tags section of a new text file type (see above).

The preview functionality is also the same, as are the reuse limitations: modified file types cannot be saved as separate configurations, so any change to a file type affects how all files of this type will be processed. For instance, once we have added the rule to protect article numbers to the PowerPoint embedded content section, all PPTX files are processed using this rule. We can alter this behavior by modifying file types for a particular project and then saving changes as part of the project template. However, this does not constitute a clean reuse mechanism as is discussed in the Custom file filter reuse subsection of the Memsource section. The score in the reuse category was thus reduced proportionally to the custom text file filter reuse category—the maximum score of 5 was multiplied by 0.7 for the resulting value of 3.5.

The total score for the Big Three is calculated as follows:

Custom regex configuration creation for the Big Three: 10 + 10 + 10 = 30

Custom regex configuration reuse for the Big Three: 3.5 + 3.5 + 3.5 = 10.5

Custom regex configuration preview for the Big Three: 10

Segment filtering

Score: 10

Trados Studio has not just one but two segment filters. Admittedly, it is a bit convoluted structure, but together the two filters cover all required bases.

The more refined but less functional filter is located on the upper menu's Review tab. Note that this filter supports regexes by default, and this behavior cannot be changed. So we need to escape all special characters used in regexes with a backslash if we need them as literals. For instance, when searching for a dot, instead of just "." we should type in "\.". Straight quotes in the previous sentence are not part of the query and are used to separate the query text from everything else.

The Case Sensitive icon should be clicked as the filter is case-insensitive by default. Note also the In Target/In Source switch next to it:

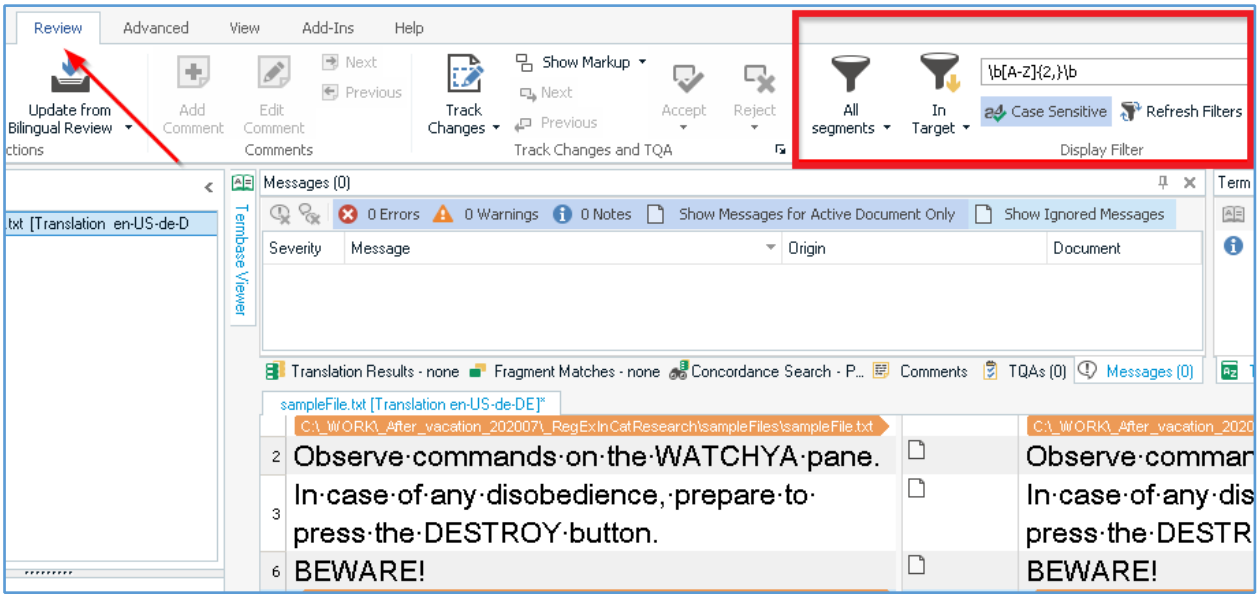

Figure 10-65. Trados Studio: Segment filter settings (default filter)

A peculiarity of this filter is that, when it is applied, the slider is scrolled down to the very bottom of the screen so it may seem as if no segments have been found. You need to scroll back up to find them.

The other option is called *Advanced Display Filter* and can be found on the View tab:

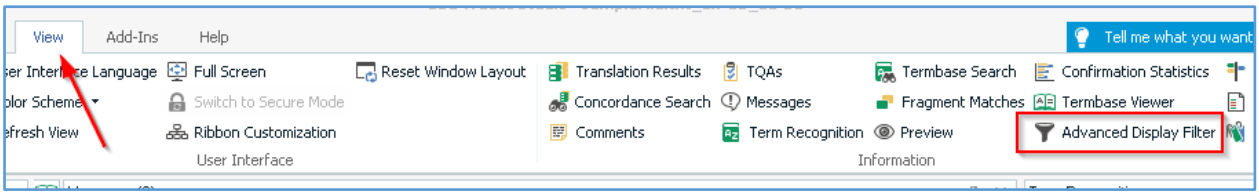

Figure 10-66. Trados Studio: Advanced Display Filter location

This later addition to Trados functionality is more powerful than the default filter, though its usability is not perfect: it is put next to the Editor window consuming screen space and does not keep the history of queries so you have to retype them every time from scratch. On the plus side, the advanced filter gives you considerably more options, including the ability to switch regexes on and off as well as simultaneously filter both source and target.

Note also the Case Sensitive checkbox:

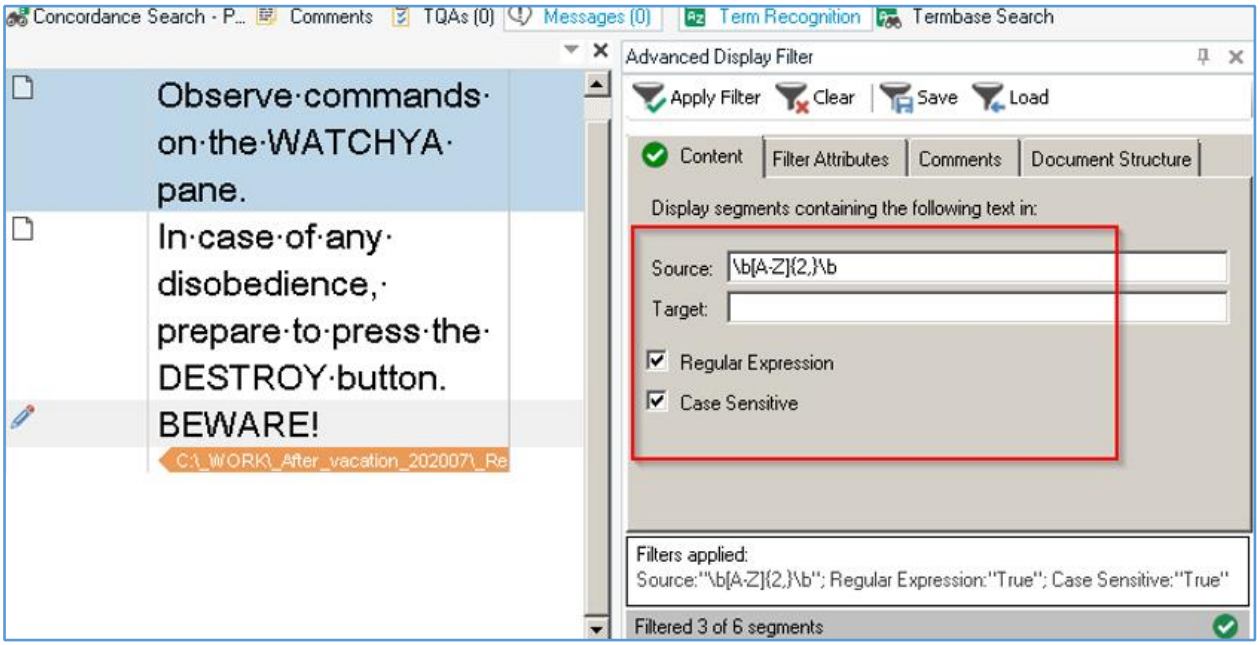

Figure 10-67. Trados Studio: Segment filter settings (advanced filter)

Search and replace

Total score: 20

The search and replace functions mostly work as expected though they have some minor quirks, especially when changes need to be done at the beginning or end of a sentence. As a general rule, filter first and then search and/or replace in the segments displayed by the filter. In Trados, it is a much faster and more reliable procedure than searching and replacing through a whole document.

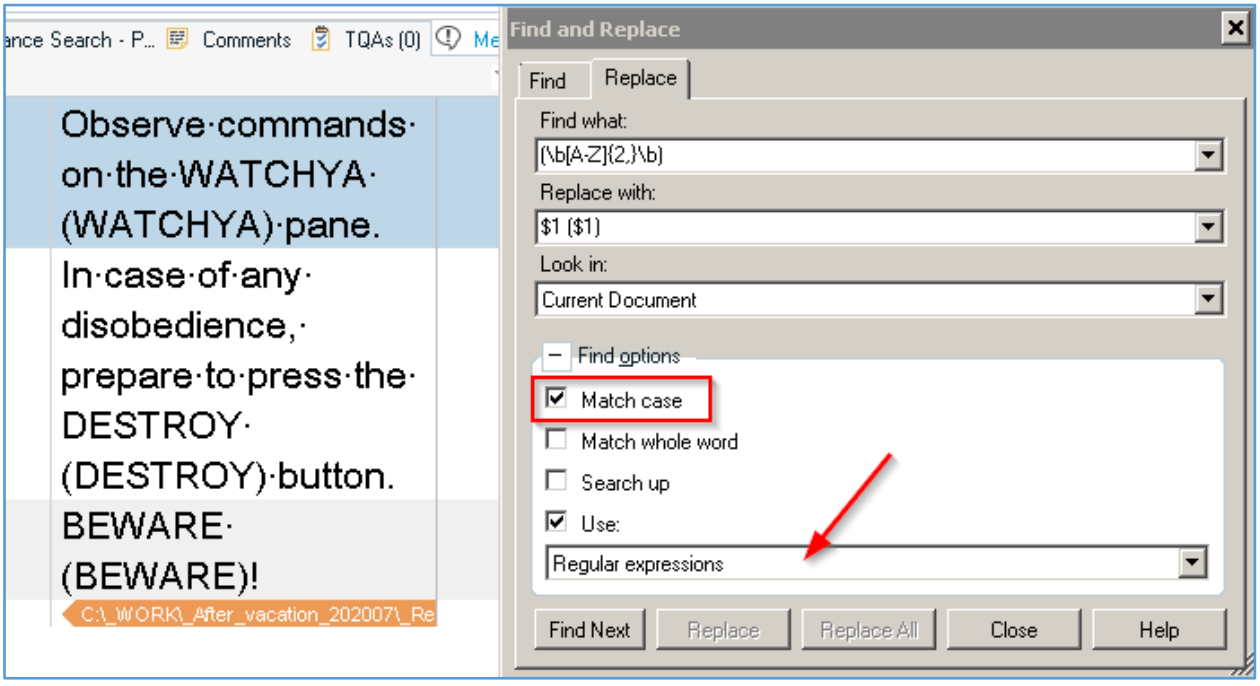

Figure 10-68. Trados Studio: Search and replace settings

Smartcat

Quadrant: Fledglings

Overall score: 2

A popular cloud environment known for its innovative approach to financial transactions, Smartcat does not offer sophisticated regex-related functionality. For the purposes of our test case, there is nothing on the linguistic side or for the preparation of the Big Three. The preparation of text files, however, can be managed to a degree, through so-called *placeholders* (and also XPath-based rules for XML). Unfortunately, the *.txt* extension is not supported, and attempts to deceive the system by changing it to one of the supported formats (like *.strings*) led to errors.

On the bright side, Smartcat supports hidden text in DOCX, which merits 2 points.

Swordfish IV

Quadrant: Editor's Friends

Overall score: 15

Swordfish, developed by the Uruguayan company Maxprograms, is an open-source tool which is rather a translation editor than a full-fledged CAT solution. It boasts some regex functionality on the linguistic side but nothing for the purposes of file preparation. All you can do is use predefined filters for different file formats.

Segment filtering

Score: 10

The segment filter is good. As in many other cases, regexes here are case-insensitive, so for our purposes I had to select the Case Sensitive Search checkbox:

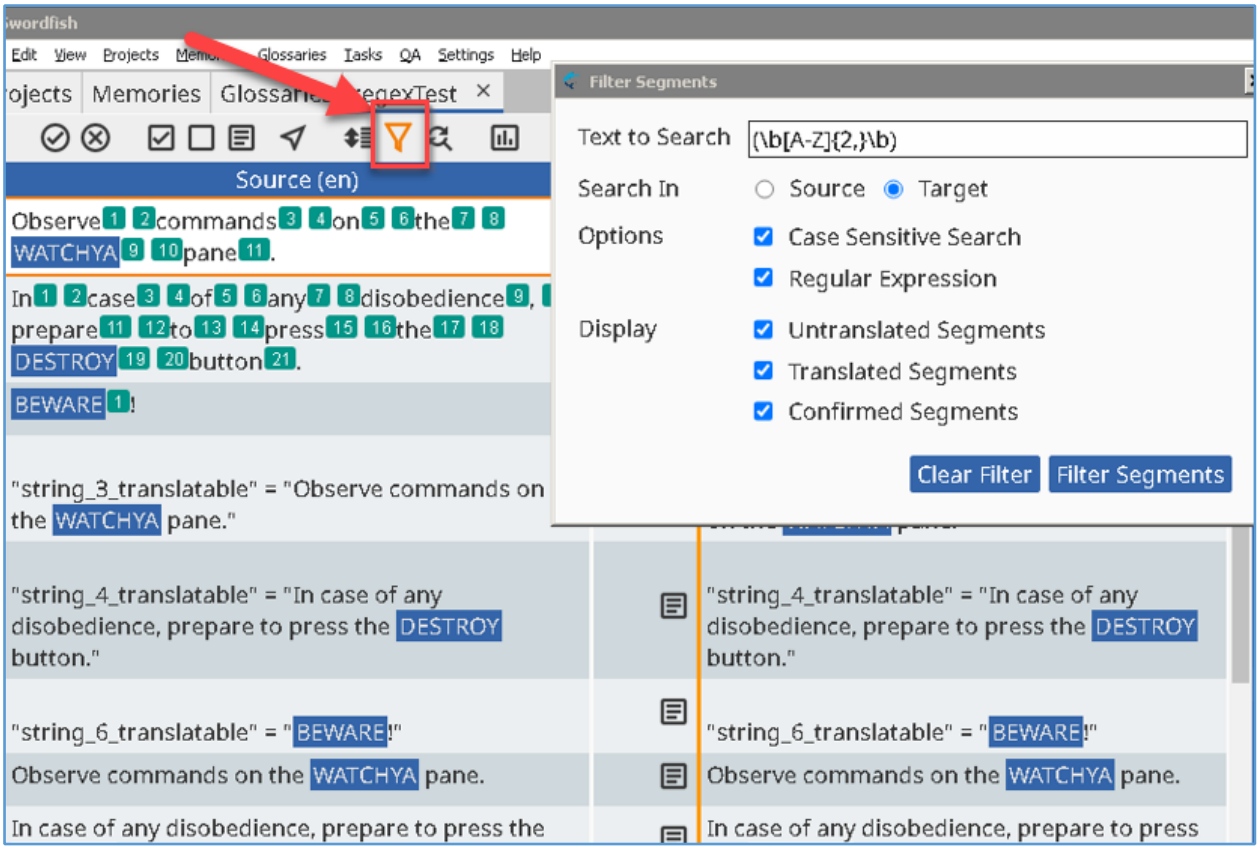

Figure 10-69. Swordfish: Segment filter settings

Search and replace

Total score: 5

The search functionality is not implemented. You can filter segments based on text conditions but cannot navigate between occurrences.

As for the replace, only batch operations are supported, and they cannot be undone. 5 points is all I could award for that.

translate5

Quadrant: Fledglings

Overall score: 8

translate5 is an open-source cloud CAT tool developed by Marc Mittag. While not being very advanced regex-wise, translate5 still has some functionality worth being reviewed.

File preparation

Total score: 2

translate5 allows to modify its file filters using the *Okapi* framework and the *Rainbow* tool built on top of it (through batch configuration files with the *.bconf* extension). Though no points could be awarded for this complex solution, the very possibility of such modifications is a plus. To be fair, Rainbow only supports regex rules for plain text, not for the Big Three, so the potential here is limited anyway.

By default, translate5 protects hidden text in DOCX, which earned it 2 points.

Segment filtering

Score: 0

Segments can be filtered based on text conditions but regexes are not supported.

Search and replace

Total score: 6

The search function supports regexes but many typical regex components are *black-listed* (according to translate5' help page). Even without that, regexes in translate5 must be MySQL-compatible, which is limiting in and of itself. In particular, word boundaries and group backreferences are not supported. For our purposes, I managed to achieve acceptable results going without word boundaries. Note that the search is case-insensitive so the Match case checkbox should be selected. The screenshot below is also somewhat misleading in that it shows the Replace tab. In our case, it was used to perform a search operation:

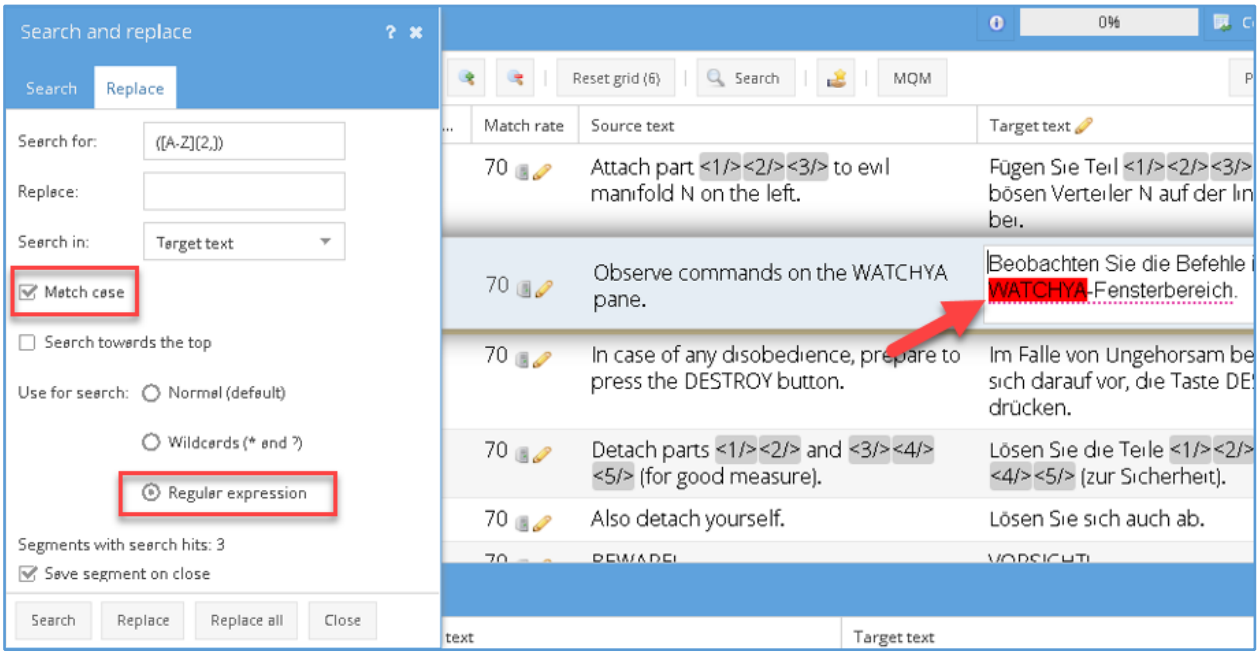

Figure 10-70: translate5: Search settings

The score for the search category was affected by the restrictions (I pushed it down to 6). As for the replace, without group backreferences it renders useless in our scenario and thus does not merit any points.

Translation Workspace XLIFF Editor

Quadrant: Fledglings

Overall score: 28.75

Translation Workspace (TWS) is a proprietary CAT tool developed by Lionbridge, one of the world's largest translation companies. It is mostly used by Lionbridge subcontractors and rarely comes to mind as a tool of choice in other scenarios. However, Lionbridge's ubiquitous presence in the field of translation and localization makes TWS a very common and well-known tool in the industry.

Strictly speaking, TWS is a server-based environment as you can only use a desktop XLIFF Editor if you have an account with Lionbridge and live connection to their server. Still, files can be prepared for translation and then translated without leaving Editor, which qualified TWS to be included in this report.

On the whole, TWS is skewed toward the file preparation side where its regex capabilities are quite solid (as long as we talk about text files). In contrast, segment filtering and search and replace functionality hardly include any support for regexes.

Text files

TWS offers a capable, if a little convoluted, framework for tagging plain text. User settings are stored locally in an *./mx* file, which can even be edited directly if you suddenly start feeling a little geeky. In this report, however, we will reduce ourselves to UI-supported operations.

Custom file filter creation

Score: 10

How-to

At the project creation stage, the Configure Filters window appears where we can get down to business on the TEXT tab.

First we create a new filter:

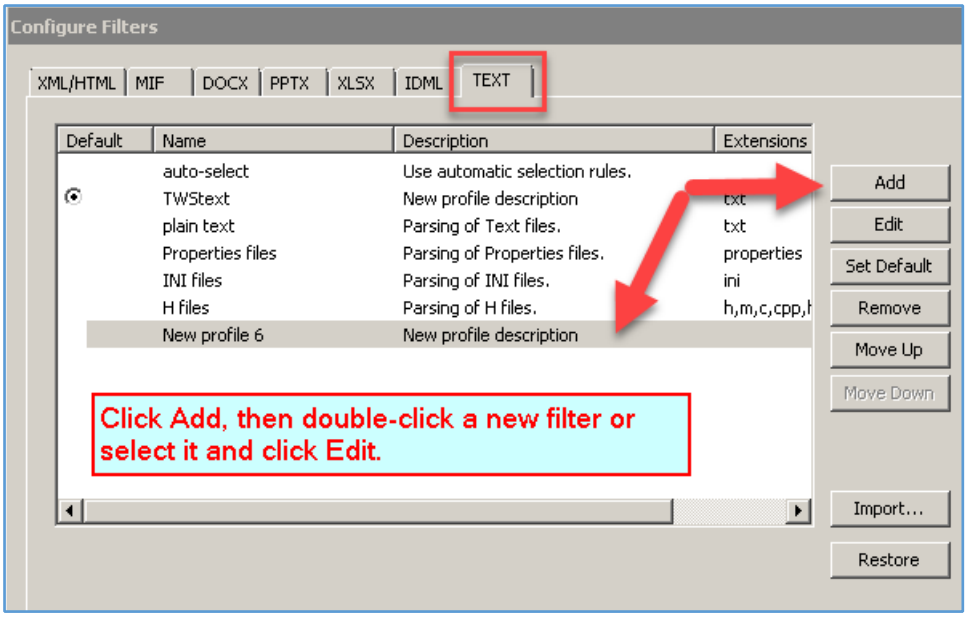

Figure 10-71. TWS: Text file filter creation (step 1)

Then we give it a name and add file extensions (*txt* in our case):

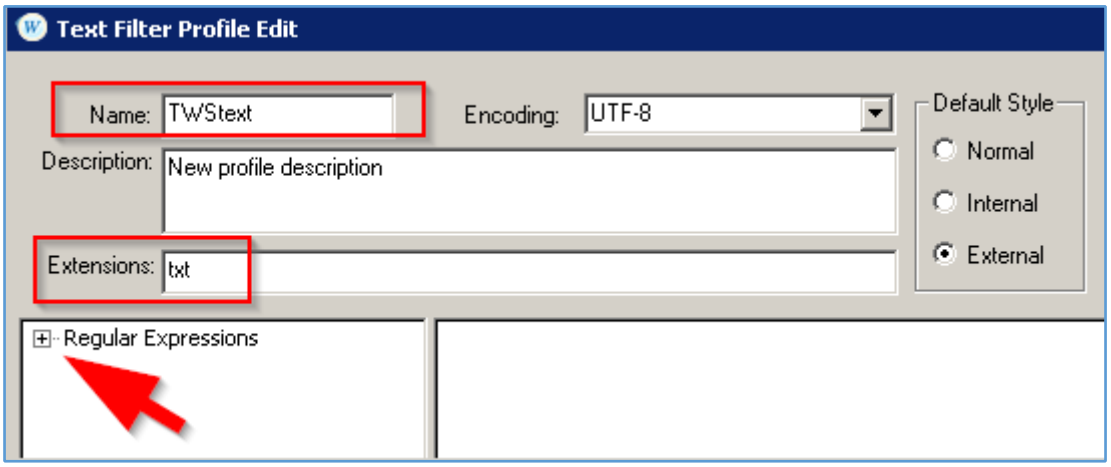

Figure 10-72. TWS: Text file filter creation (step 2)

Regex rules are organized in a tree structure with parent—child relationships. When we need to add a rule, we right-click the tree and append a new node:

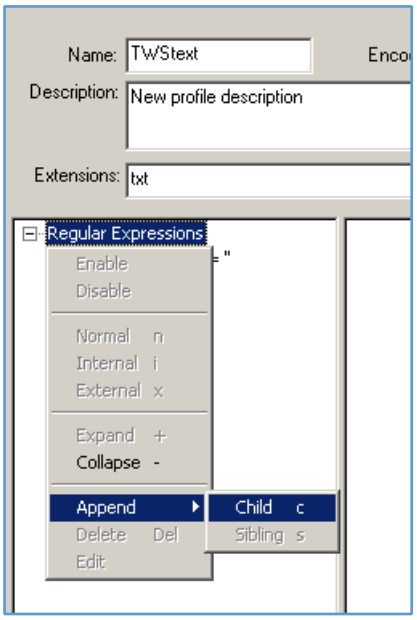

Figure 10-73. TWS: Appending child rule in custom text file filter

In our case, two rules must be constructed.

First, we have to define *delimiters* which denote the left and right boundaries of the translatable content. For our left delimiter we need to use something that would differentiate translatable strings from non-translatable ones. It is very similar to the technique used in a Wordfast rules file. A digit followed by *_translatable* would do the job, as there is a *non_* component in between them in non-translatable strings.

Here is our first rule:

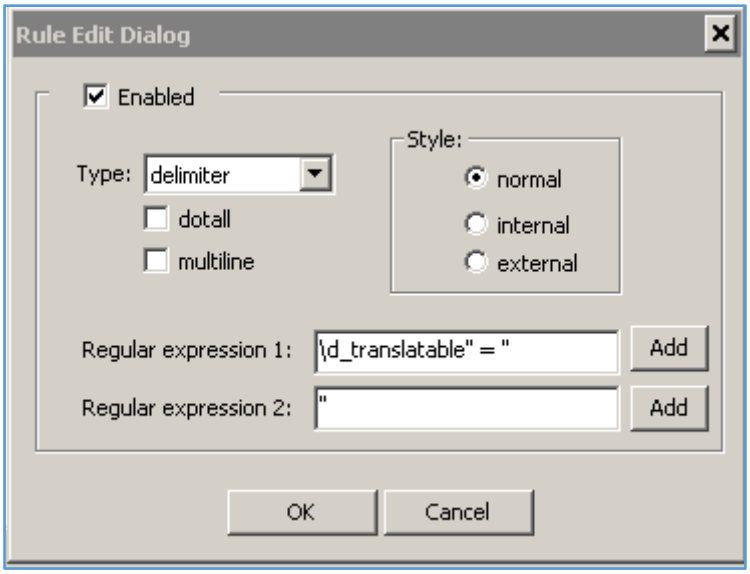

Figure 10-74. TWS: Text file filter settings (delimiters)

Second, we append a child node and define another rule, for article numbers. This time, it is not a delimiter but a *block*, that is a placeholder to be put in tags:

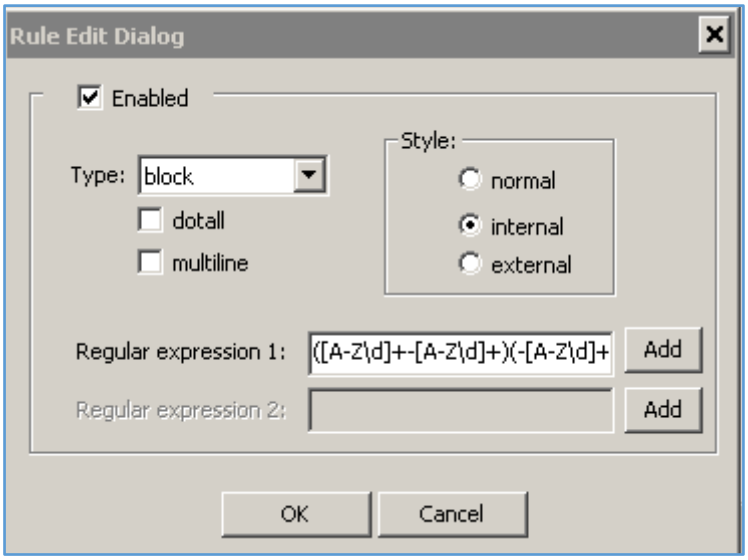

Figure 10-75. TWS: Text file filter settings (block)

And here is our new filter in all its glory:

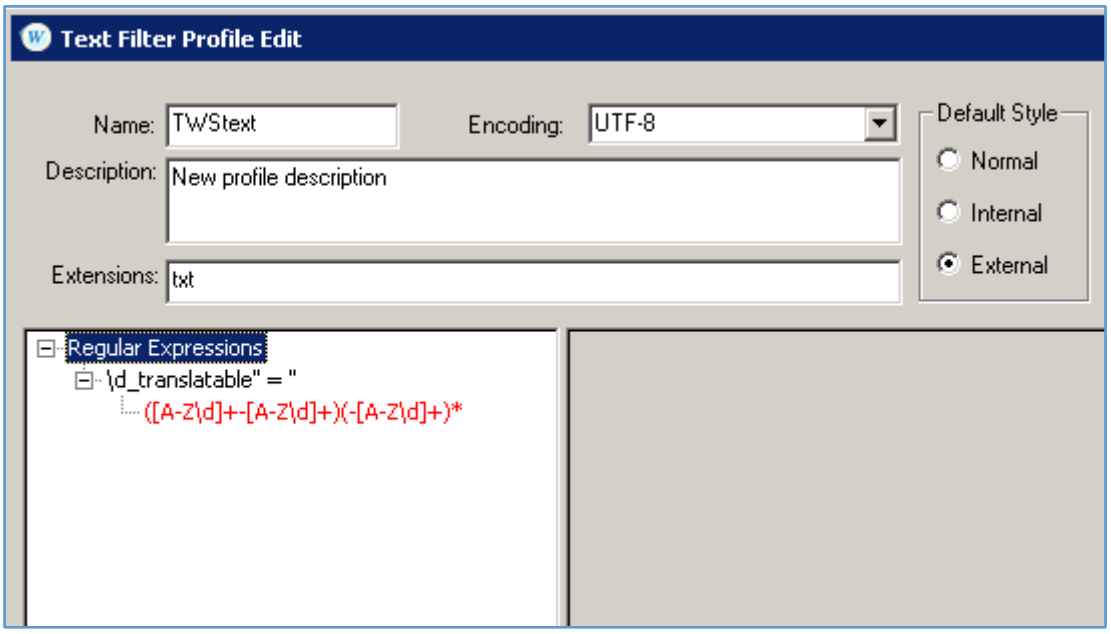

Figure 10-76. TWS: Text file filter settings (final view)

Back to the Configure Filters window, we now need to set this newly created filter as default for the *.txt* extension. A default filter is marked with a radio button on the left. We need to click the Clear Default button, then select our filter and click the Set Default button (which appears in place of the Clear Default button).

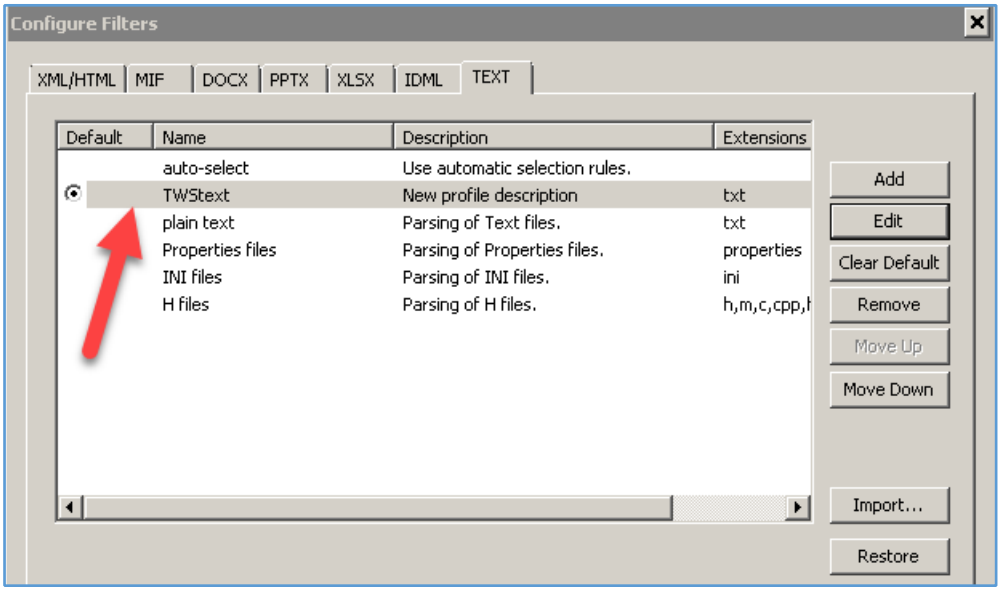

Figure 10-77. TWS: New text file filter set as default

Finally, we save changes:

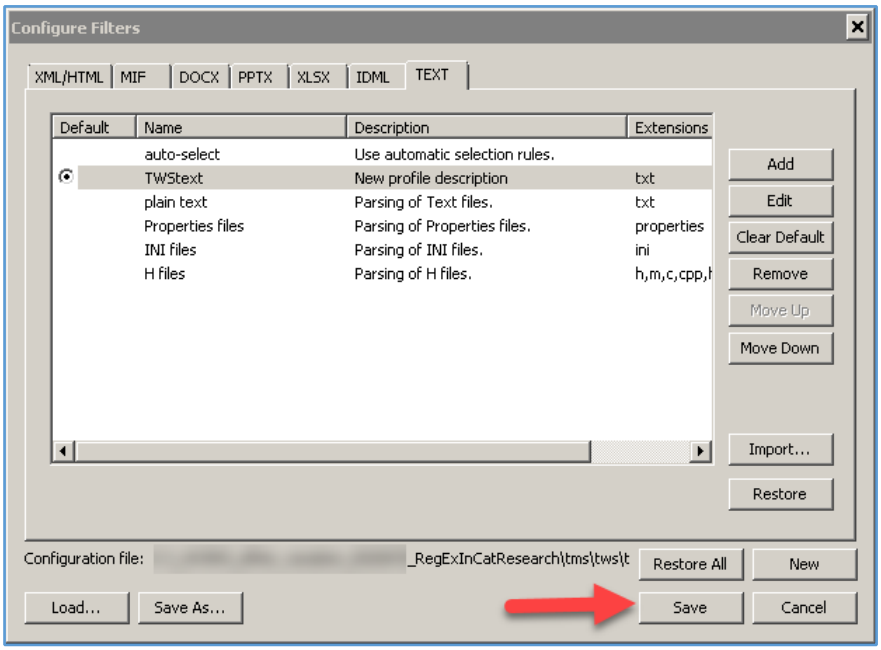

Figure 10-78. TWS: Saving new text file filter settings

Once saved, a custom file filter is included in the *.lmx* file that stores all settings.

And here is how our text file would look like in the Editor view, with the new filter applied:

Data Type: x-text **Slang:** en-us **Tlang:** de-de
Attach part «1» to evil manifold N on the left.
Observe commands on the WATCHYA pane.
In case of any disobedience, prepare to press the DESTROY button.
Detach parts «1» and «2» (for good measure).
Also detach yourself.
BEWARE!

Figure 10-79: TWS: Text file in Editor view

Custom file filter preview

Score: 0

No preview is available.

Custom file filter reuse

Score: 7

Once created, a file filter is available as part of the *.lmx* settings file (save it after each modification of the filter). If you do not see your custom filter on the list, load the *.lmx* file:

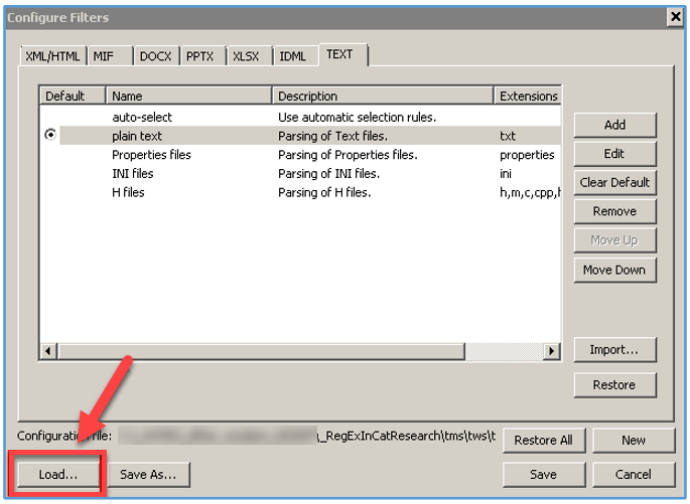

Figure 10-80. TWS: Loading of previously saved .lmx settings file

However, the ability to choose a custom file filter for a project does not amount to good reuse functionality. Only one file filter per project can be selected for all files with the same extension, and the procedure of setting new default types is not very user-friendly. See a more detailed discussion of this limitation, including the explanation of the score, in the Custom file filter reuse subsection of the Trados Studio section.

Custom regex configuration creation

Score: 5

General file filters and more specific regexes for inline tags cannot be separated in TWS. Neither can we use several filters at once. Technically, though, we still can create many different filters with the same parent structure and varying child regexes for inline tags. According to our scoring system, it merits 5 points.

Custom regex configuration preview

Score: 0

No preview is available.

Custom regex configuration reuse

Score: 1.75

See the Custom regex configuration reuse subsection of the Trados Studio section for the explanation of the score.

The Big Three

Total score: 0

Settings for Office formats are very limited. Even the hidden text protection is not available for DOCX.

Segment filtering

Score: 0

No regex-based filtering is supported.

Search and replace

Total score: 5

Search and replace functions do not support regexes. 5 points are given for a batch replace feature, which is available through the upper menu: Tools > Batch Find and Replace. Batch replace operations cannot be undone, and the behavior of the feature is not always reliable.

Wordbee

Quadrant: Manager's Helpers

Overall score: 95.5

For a cloud solution, Wordbee has a surprisingly rich functionality on the file preparation side. Its *regular expression tool* used to verify custom regexes is quite good even by desktop solution standards.

On the linguistic side, though, things look bleaker, with segment filtering being the only capability supporting regexes.

Text files

Wordbee does not differentiate between file filters and regex configurations as we understand them in this report. However, it allows to create pretty flexible rules, for both entire segments and individual placeables. What is even better, rules can be created not only for text files but also for the Big Three (and a number of other formats). This really sets Wordbee apart among its cloud counterparts.

Custom file filter creation

Score: 10

How-to

To start creating a new filter, we need to go to Settings and then click the fitting file format:

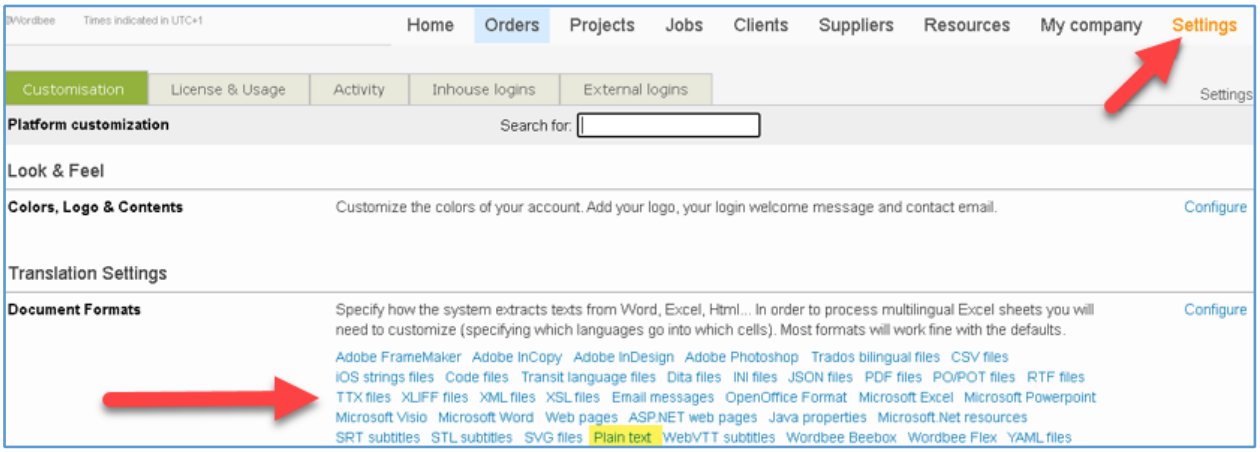

Figure 10-81. Wordbee: File filter selection

In our case, we click *Plain text*. Then we add a new filter and create rules for whole segments (to hide non-translatable strings) and for placeables within strings to be translated.

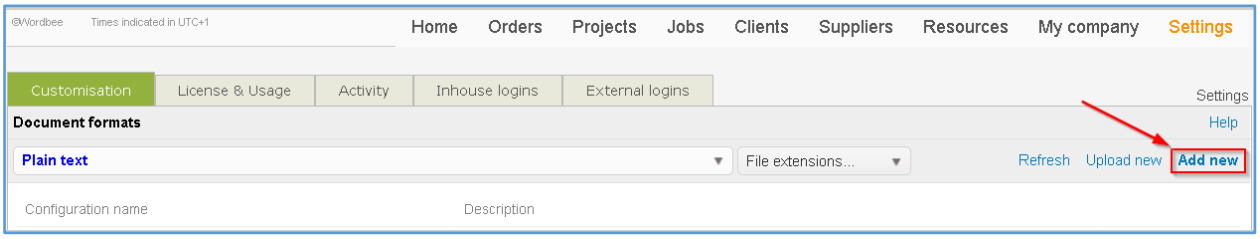

Figure 10-82. Wordbee: Adding new rule for text file filter

Rules are created on the Do Not Translate tab. In the Segments area we define non-translatable strings (note also the red No in the Translate column), and in the Words or terms area rules for ID sections, article numbers and closing straight quotes:

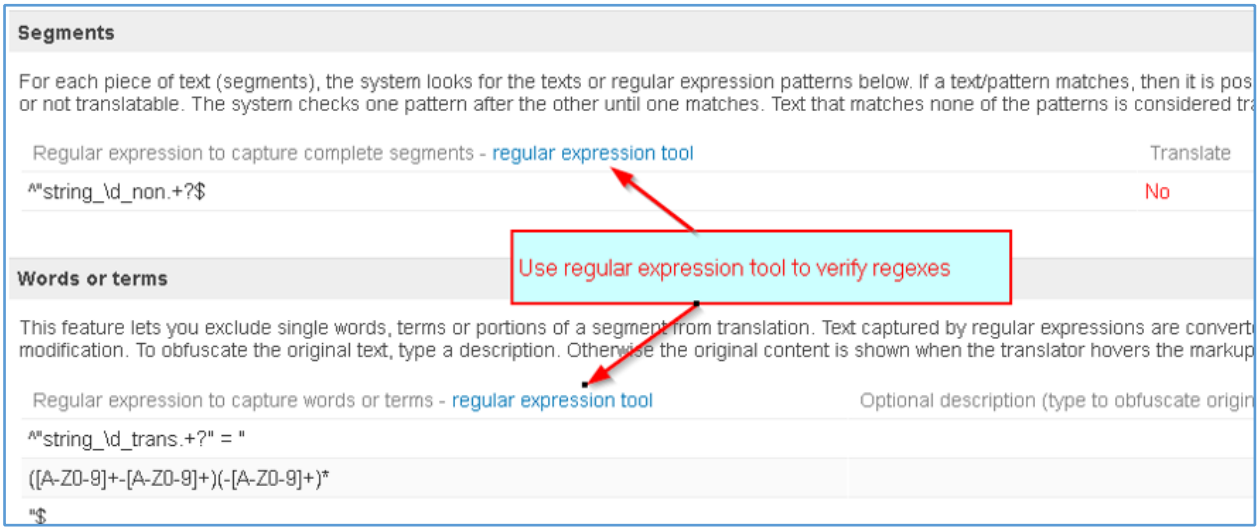

Figure 10-83. Wordbee: Text file filter settings

With a new filter applied, our text file looks as follows in the Editor view:

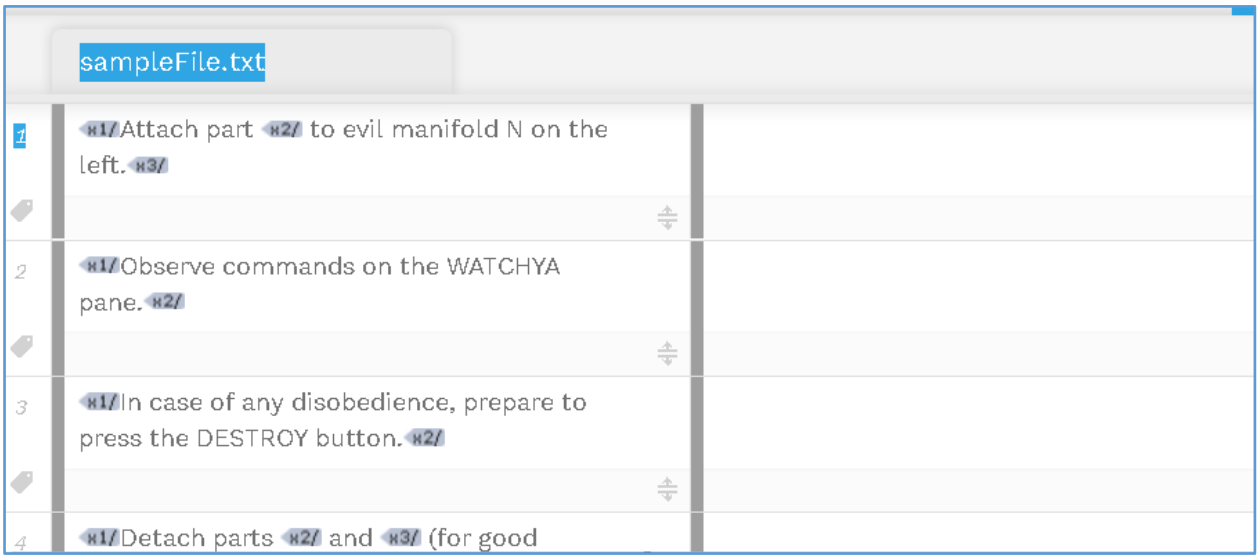

Figure 10-84. Wordbee: Text file in Editor view

Custom file filter preview

Score: 5

A limited preview is available within the *regular expression tool*. You can paste up to three samples for each regex you are testing. In many cases that would be enough for this particular regex, but you will not be able to see all your rules applied at once to the source file. Still, some points had to be awarded for this functionality.

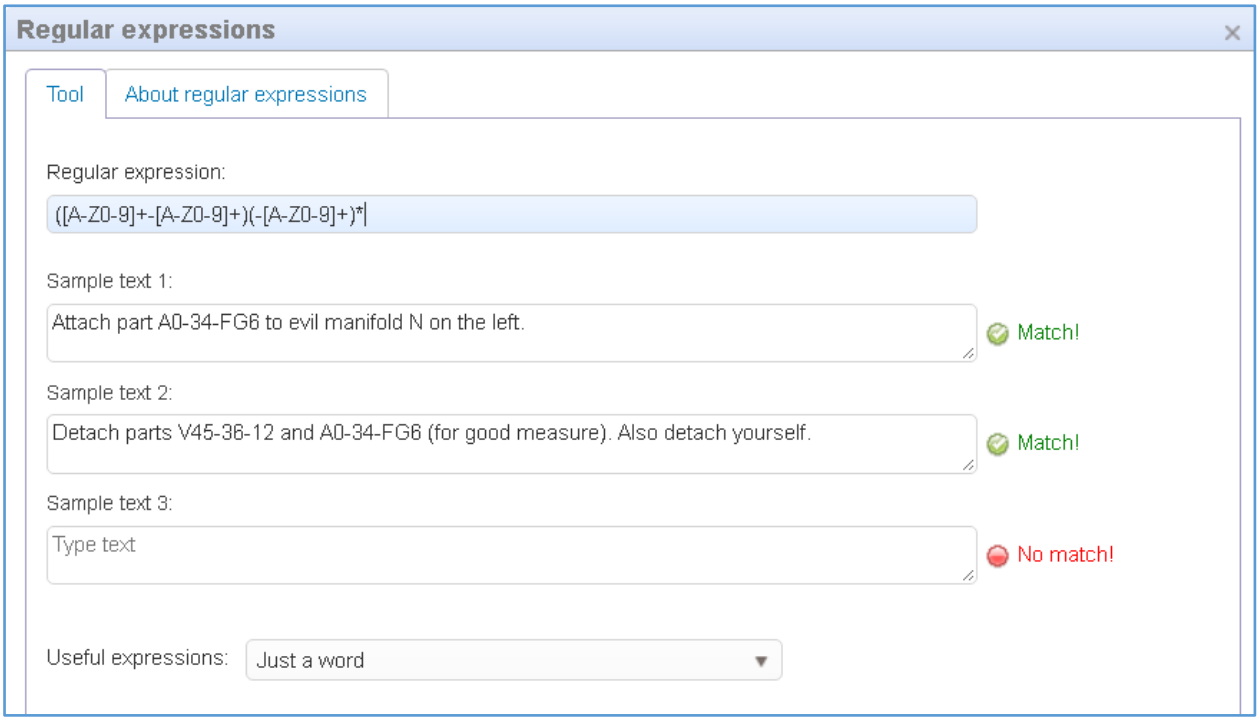

Figure 10-85. Wordbee: Text file filter preview

Custom file filter reuse

Score: 10

Once created, a new custom file filter is available for all new projects.

Custom regex configuration creation

Score: 5

As there is no clear differentiation between file filters and regex configurations, I awarded 5 points as I did in all similar cases.

Custom regex configuration preview

Score: 5

See above on the limitations of the preview.

Custom regex configuration reuse

Score: 2.5

Half the top score as file filters and regex configurations are merged.

The Big Three

Total score: 50

The Big Three are handled just the same as text files, by creating a new file filter and then adding regex rules for segments and placeables. In our case, we only add one rule for each of the Big Three formats:

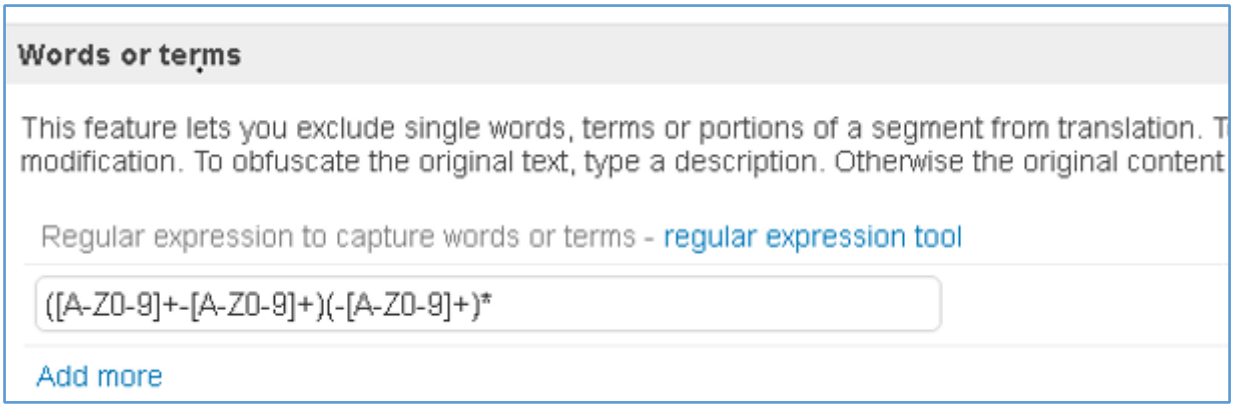

Figure 10-86. Wordbee: Regex configuration settings for Big Three

Below is the total score calculation for the Big Three:

Custom regex configuration creation for the Big Three: 10 + 10 + 10 = 30

Custom regex configuration reuse for the Big Three: 5 + 5 + 5 = 15

Custom regex configuration preview for the Big Three: 5

Segment filtering

Score: 8

Wordbee's segment filtering works, but its usability leaves much to desire. For example, all settings are lost each time you clear a filter so you have to reproduce them for your next filtering operation (the Regex checkbox has to be reselected, etc.). You cannot check several checkboxes at once in a drop-down next to the Find text field, it collapses after each selection. The settings area is not removed from the screen after a filter is applied, so you cannot see your filtered segments underneath it. It also seems that the regex functionality is not implemented consistently: \b for word boundaries did not work so I had to go with a less accurate [A-Z]{2,}.

Note that the search is case-insensitive so the Case sensitive checkbox should be selected.

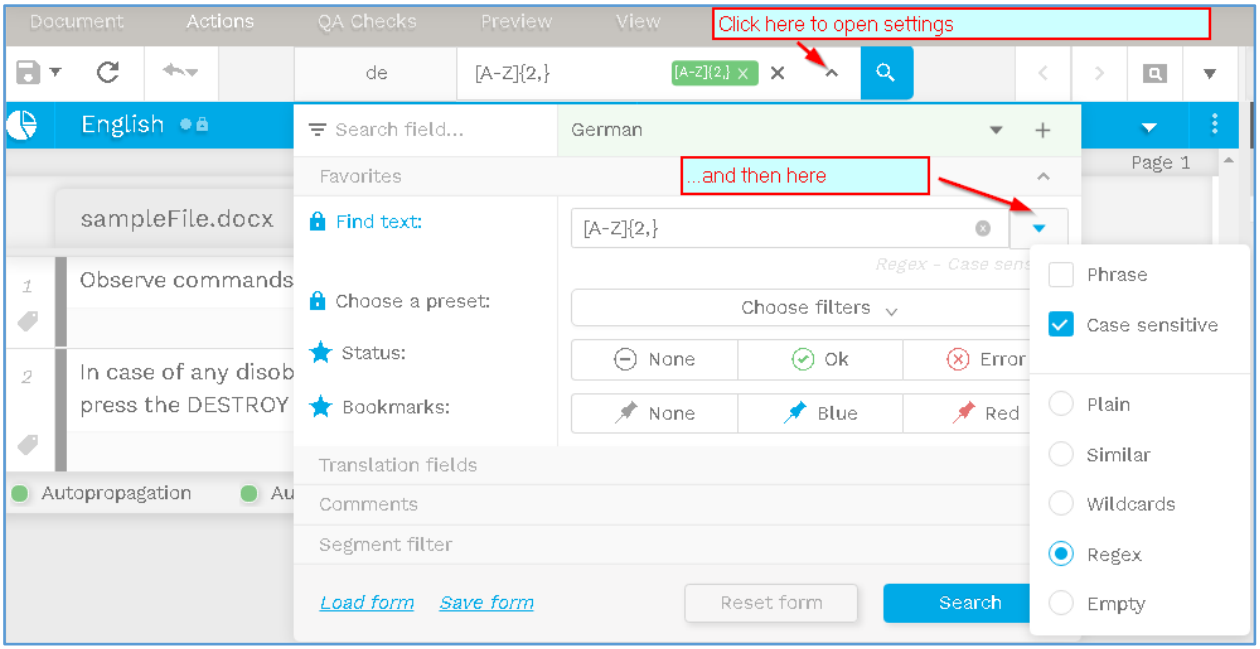

Figure 10-87. Wordbee: Segment filter settings

I deducted two points from the perfect score for these inconveniences.

Search and replace

Total score: 0

The search function is integrated with segment filtering, there is no navigation between occurrences. As for the replace function, only batch operations are available, but even this option does not seem to support group backreferences, which renders it useless for our purposes.

Wordfast Pro 5

Quadrant: Editor's Friends

Overall score: 59.5

A very well-known and popular tool, Wordfast Pro is not, however, a real regex powerhouse. While its linguistic regex capabilities (segment filtering and search and replace) are solid, the file preparation side is quite limited. Some things can be done via manual modification of so-called *rules files*, but even this slightly awkward method will not give us all the results we might want.

Text files

To modify rules for plain text, we will have to create a special *rules file* and then save it with the *.properties* extension. In can be done in any text editor. I mostly use Notepad++, which is free and very rich in functions. But even a basic tool like Windows Notepad will do.

Wordfast's regex functionality is not very well documented, but a chapter on rules files provides enough information. You can find it here: Building a Rules File for Tagged Text Translation in Wordfast Pro. Using this instruction, I was able to create rules that satisfied conditions of our test case.

Custom file filter creation

Score: 10

How-to

First we prepare a *.properties* file. For our purposes, it will look like this (CRLFs are end-of-line marks displayed in Notepad++, they are not part of the regexes):

```
paragraphPrefix.1=[0-9]_translatable" = "CRLF
paragraphSuffix.1="CRLF
internalTag.1=([A-Z0-9]+-[A-Z0-9]+)(-[A-Z0-9]+)*
```

Figure 10-88. Wordfast: Rules file

On the left, we have predefined labels telling the system what a regex does. On the right, after an equal sign (=), we add a regex itself. No additional quotes around a regex are needed.

The *paragraphPrefix* label and the *paragraphSuffix* label define boundaries encompassing the translatable text. To exclude non-translatable strings, we must find a way to uniquely identify ID sections in translatable strings. It can be achieved by including a digit followed by *_translatable* in a prefix, as there is the *non_* component in between them in non-translatable strings. For a suffix, a straight quote will suffice, since all strings end with it.

The *internalTag* label lets us define a rule for inline tags. Here we can use our standard regex for article numbers.

Having all required rules defined, we save the file with any name we like. The extension should be *.properties*.

Then, at the project creation stage, we create a new filter:

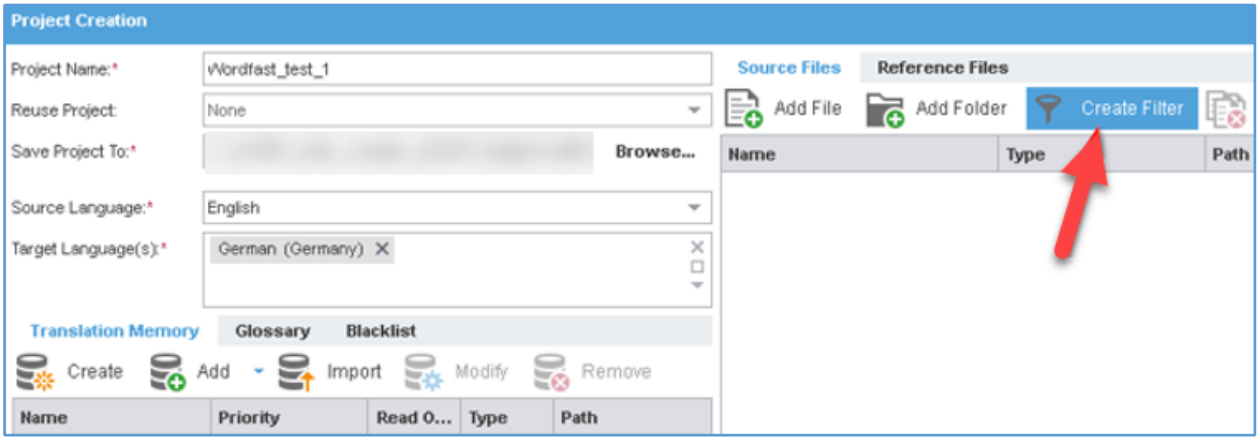

Figure 10-89. Wordfast: New text file filter creation (step 1)

Somewhat misleadingly, we need to choose *xml* as our base configuration, despite the fact that we are going to process *.txt* files:

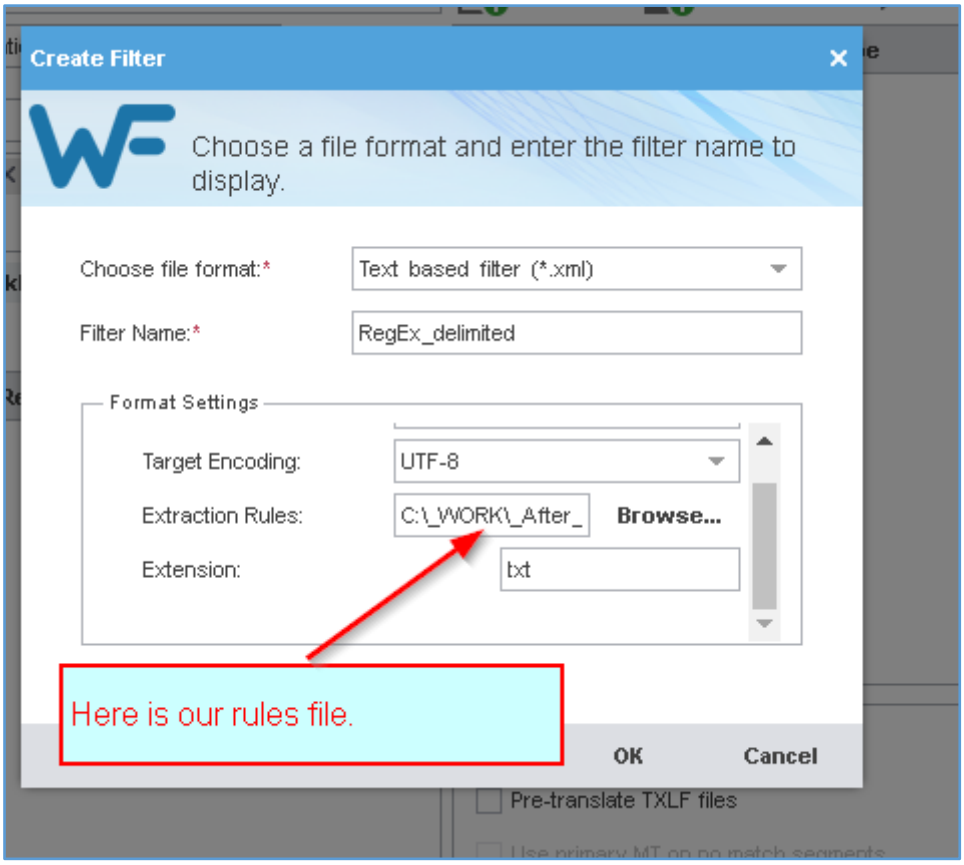

Figure 10-90. Wordfast: New text file filter creation (step 2)

And here is how our text file looks like in the Editor view, with all tags applied:

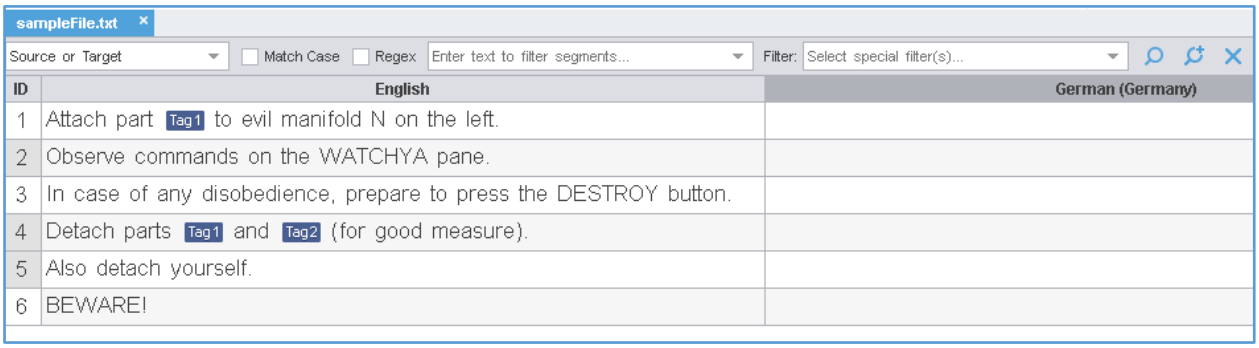

Figure 10-91. Text file in Editor view

Custom file filter preview

Score: 0

No preview is available.

Custom file filter reuse

Score: 10

Once created, a file filter is available on the list of filters for any new text file:

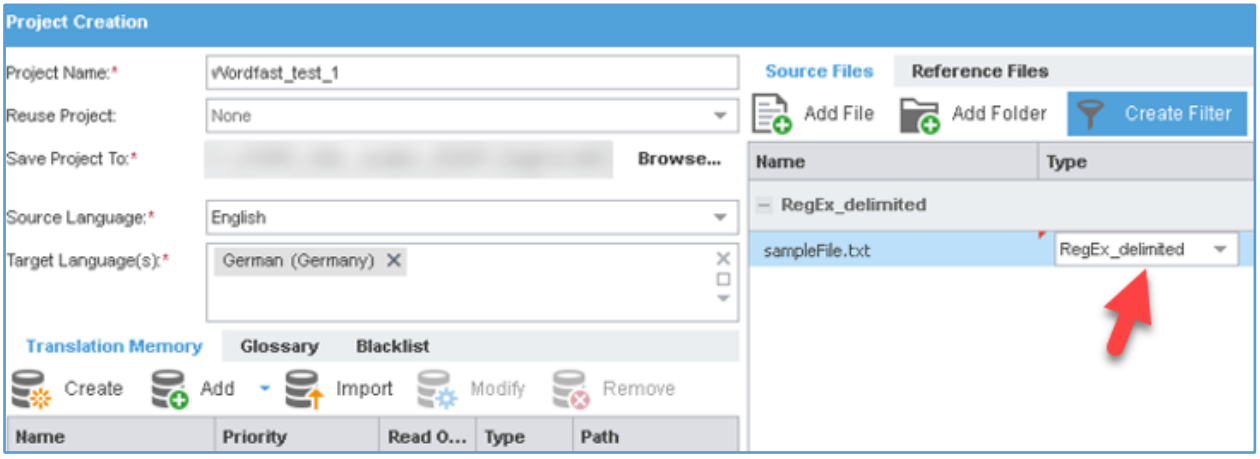

Figure 10-92. Wordfast: Text file filter selection

Custom regex configuration creation

Score: 5

As we have seen earlier, general file filters and more specific regexes for inline tags cannot be separated in Wordfast Pro. According to our scoring system, it merits 5 points.

Custom regex configuration preview

Score: 0

No preview is available.

Custom regex configuration reuse

Score: 2.5

The top score of 5 was halved, as in all similar cases where file filters cannot be separated from regex configurations.

The Big Three

Total score: 2

Next to nothing is going here for us. The default Excel filter can supposedly be configured via an external *.xml* file (along the lines of rules files for plain text that we analyzed above). Unfortunately, despite my best effort, I could not get it to work.

Word documents can be controlled using hidden text, but that is all.

PowerPoint files are not configurable, as long as regexes are involved.

So I only awarded 2 points for custom regex configuration for DOCX, as I did in all similar cases.

Segment filtering

Score: 10

Segment filters in Wordfast Pro are sound. Here is how you set them up:

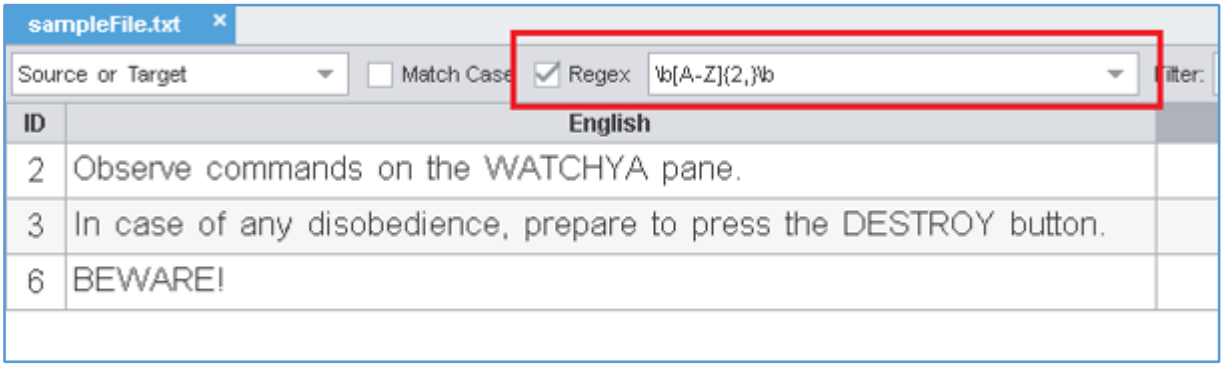

Figure 10-93. Wordfast: Segment filter settings

Search and replace

Total score: 20

The search and replace functions, invoked via *Ctrl-f* (*Ctrl-h* for replace) or on the upper menu, are good too. The settings are shown below:

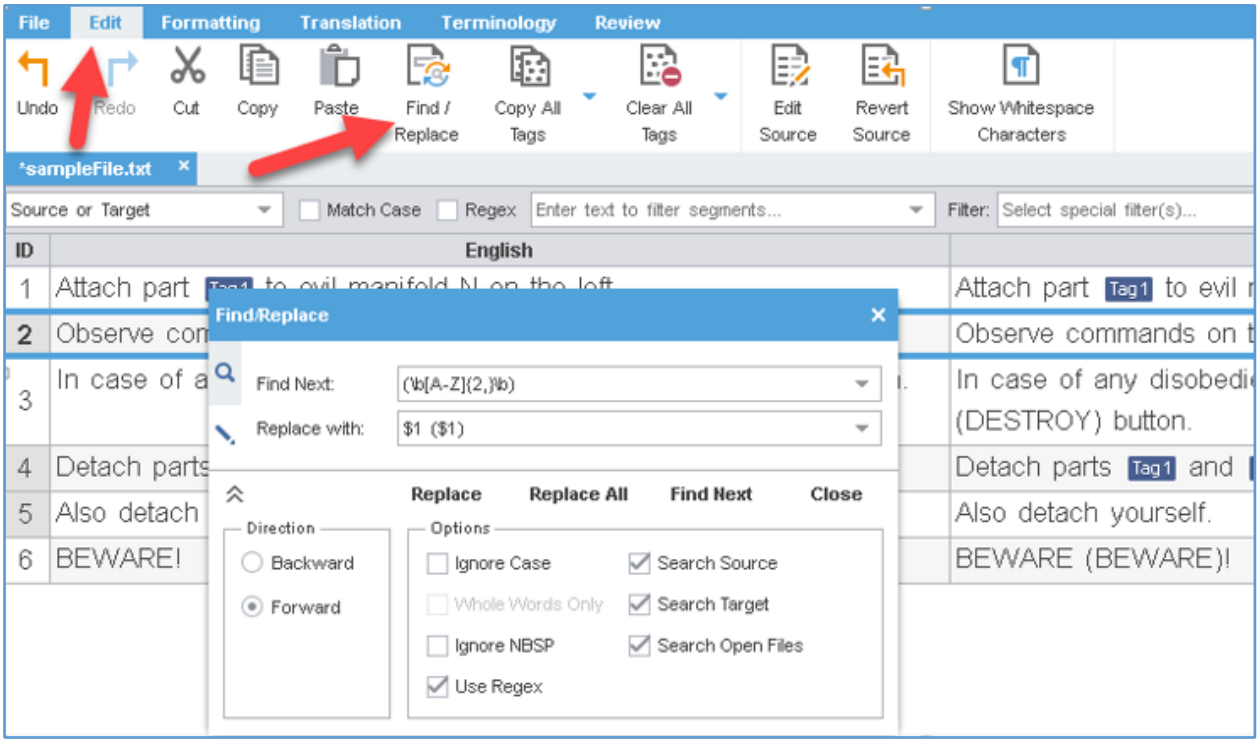

Figure 10-94. Wordfast: Search and replace settings

XTM

Quadrant: Fledglings

Overall score: 2

Despite being one of the most popular and respectable cloud-based CAT tools, XTM is not equipped with regex capabilities. Surprisingly, regexes cannot even be used on the linguistic side, for segment filtering and search and replace operations. As for the file preparation stage, the system is built on open industry standards and relies on ITS rules for text extraction. Theoretically, you can create your own ITS rules and add them to the system.

The functionality to do that can be found in the Analysis Manager area of your account's settings, on the Content tab:

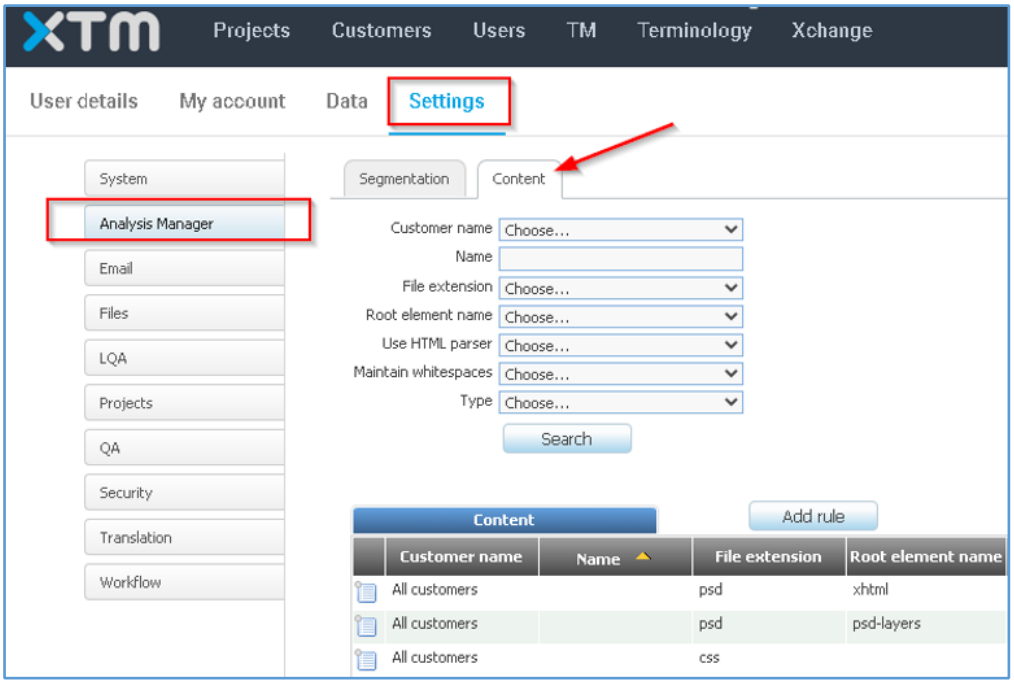

Figure 10-95. XTM: Analysis Manager area

However, modifying ITS rules is a very involved process, and even XTM's own manual recommends to request help from their support specialists. It is also unclear how ITS rules, which rely on XPath for finding nodes with translatable text, can be modified to process plain text without any mark-up. Under the hood, XTM seems to have a very viable engine; unfortunately, it does not expose much regex functionality to a user.

As many other systems, XTM puts hidden text in DOCX in tags, so 2 points were awarded for that.

11. memoQ vs. SDL Trados Studio: detailed comparison

As shown in this report, memoQ and Trados Studio are clear leaders among general-purpose CAT tools as far as regex functionality is concerned. The only other system with comparable regex capabilities is Alchemy Catalyst, but it is rather localization software than a general-purpose system.

Based on our scoring system, memoQ proved its edge over Trados Studio. However, the margin was not very wide, and it could be interesting to see how these two tools would stack up against each other under a more nuanced set of comparison criteria.

To find out, we will look into the following categories:

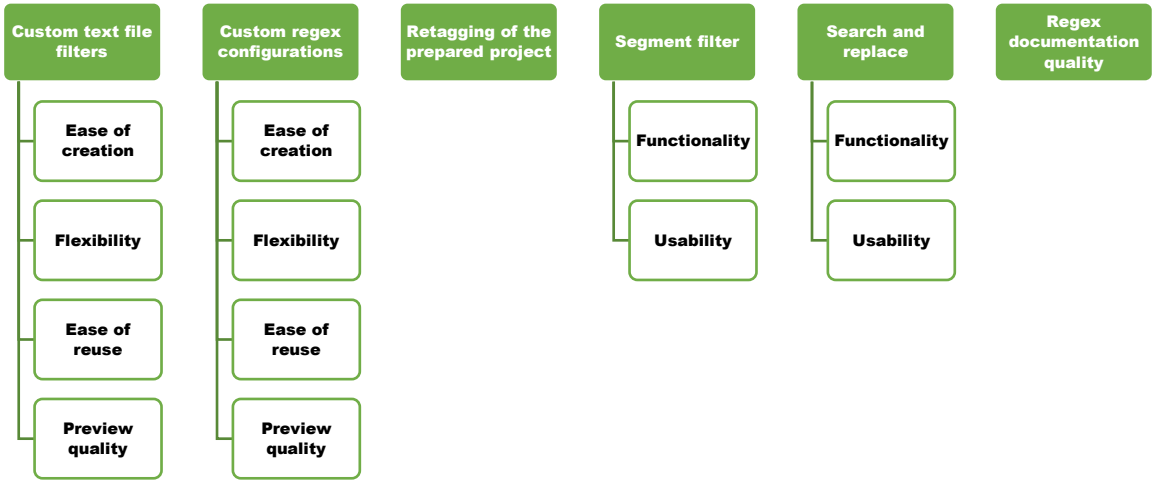

Figure 11-1. memoQ vs. Trados Studio: Comparison criteria

Instead of scores, this one-on-one comparison uses the ternary system of + – = (think chess). Whoever gets more pluses, wins.

Custom text file filters

Ease of creation

The contestants are tied here. Creating a new filter is a breeze in both tools. We will not go through screenshots again, see respective quick tutorials for details.

memoQ: =

Trados Studio: =

Flexibility

A file filter can be considered flexible if it allows to create complex configurations with options to define both translatable and non-translatable content. In other words, a flexible filter gives a user the power to either include certain content or exclude it from translation.

In my view, it is draw again. memoQ has a more extensive set of options around its paragraph and inclusion/exclusion rules. But Trados Studio's combination of document structure rules and inline tags is also pretty powerful.

memoQ: =

Trados Studio: =

Ease of reuse

memoQ is a clear winner in this category. The reuse capabilities (or rather lack thereof) are the Achilles heel of Trados Studio's otherwise robust file filter functionality. In memoQ, any file filter is readily available for any file in a project. In Trados, file filters (file types in Trados terminology) are applied automatically, based on a file's extension. Trados allows to create different file types for the same extension, but the only way to make it use a filter of your choice is to manually move this filter up the list of all available filters for this extension or deselect, also manually, other applicable options. It is a convoluted process, especially when different file types are used on a regular basis. Another unfortunate consequence of this approach is that only one file type can be used for any given file extension within one project: for instance, we cannot have two different file types for *.txt* files used at once.

memoQ: +

Trados Studio: –

Preview quality

While reuse is the weakest part of Trados Studio's file filter system, the preview function can be confidently called its best feature. However, memoQ is also equipped with a very reliable preview engine, especially when it comes to text files. The main differentiator with previews, in my opinion, is the ability to load a whole source file to see how rules will affect its display in the Editor view. This is what sets Trados Studio apart in the Big Three categories—but not with text files, where memoQ offers similar functionality. So, personal preferences aside, I called it a tie.

memoQ: =

Trados Studio: =

Custom regex configurations

Ease of creation

Both tools offer convenient mechanisms to set up regex configurations: memoQ via the Regex Tagger and Trados Studio using the embedded content engine. Nobody's game.

memoQ: =

Trados Studio: =

Flexibility

Regex configurations are usually created to protect non-translatable content, and the contestants are on par with each other in this regard. Both allow to create different combinations of tags and apply as many consecutive rules as necessary. memoQ supports different types of tags (*open*, *close* and *empty*) whereas Trados Studio offers the choice between placeholders (one tag) and tag pairs.

However, for regex configurations, flexibility must be first and foremost defined in terms of a configuration's ability to be used with different underlying file filters. Regex configurations are fine tuners; they are useless without a file filter they help modify. And this is where memoQ, with its cascading filters, reigns supreme. Regex Taggers can be paired with any other file filters, which creates almost limitless opportunities. Trados Studio, on the other hand, only supports embedded content for a number of file types. If a user wants the same regex rule to be applied to both PowerPoint and Word documents, this rule has to be reproduced in the embedded content sections for both. In contrast, Regex Taggers can be used across different file filters, without the need to change anything in filters themselves.

Note. To be fair, Trados Studio does have functionality that in some cases is capable of providing a higher degree of flexibility. It is called *Embedded Content Processors*. They can be found on the same list with other file types. Embedded content processors allow to set up custom rules, just like standard embedded content sections do. Processors can be used with several specialized text file types, like XML, HTML, JSON, etc. Translatable content is first extracted using a main file filter, and then an associated processor applies its rules to the extracted text. As far as I can see, the concept of embedded content processors is strategically prioritized by SDL's development team, so they are probably going to become more versatile and compatible with more file types in future releases. If so, it will add an important layer of granularity to the Trados Studio file type system.

Despite the note above, this one clearly goes to memoQ.

memoQ: +

Trados Studio: –

Ease of reuse

Again, no real fight here. memoQ's Regex Taggers and cascading filters can be easily reused any time you want them, including alongside default system filters they are based on. You can use a DOCX filter and a cascading filter originating from it in the same project.

In Trados Studio, a modified (through embedded content) system filter becomes a new default option, and there is no way of bypassing it other than by creating custom configurations and storing them in project templates. As we discussed earlier, this is not a complete solution; for one, you may need to use different rules on a file-to-file basis even within the same project.

An easy plus for memoQ.

memoQ: +

Trados Studio: –

Preview quality

Unlike text file filters, memoQ's Regex Taggers do not allow to load a whole file to see the effect of rules on how text will be displayed in the Editor view. Instead, a portion of a document can be manually pasted to a field below the rule, and it is only this portion that can be previewed. It is an acceptable implementation, and yet Trados Studio's is better. Consistently, across all file filters and embedded content sections, Trados allows you to load a source file in its entirety to see where your configuration might need a patch or two. Granted, it may take some time to load a larger file, but, in my view, it is a small price to pay for the luxury of having all content before your eyes.

Trados Studio scores this time.

memoQ: –

Trados Studio: +

Retagging of the prepared project

Sometimes, despite the best effort, an important detail can go unnoticed, only to be discovered later, after the project has already been created. In most CAT tools, including Trados Studio, it means going back to the file filter window, making changes in your rules and then recreating the project from scratch. This exercise has a potential of getting pretty old pretty quickly, especially when large source files are involved.

In memoQ (and also in Catalyst), this problem is solved in the most radical way possible: the retagging—or, in the terminology of this report, the creation of custom regex configurations—is available directly in the Editor view. All you need to do is click the Regex Tagger button on the Preparation tab:

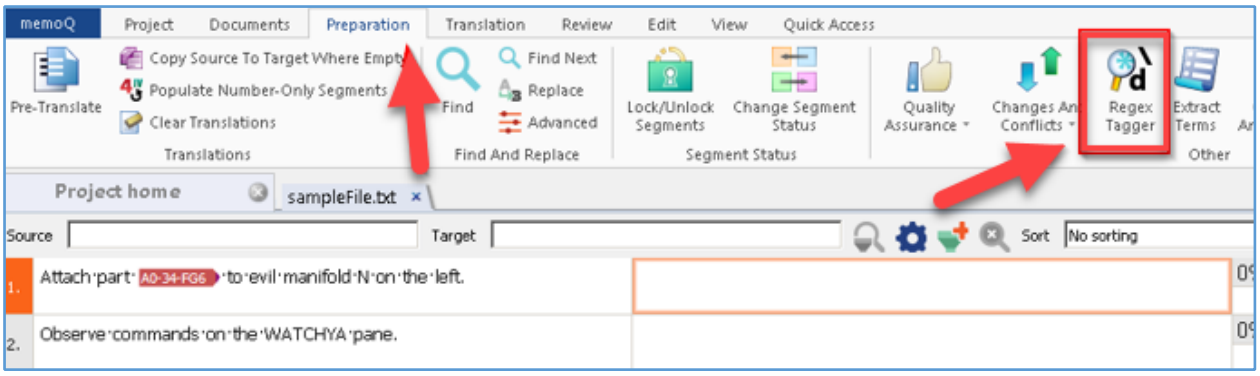

Figure 11-2. memoQ: Using Regex Tagger in Editor view

The standard Regex Tagger window appears where rules can be defined (see the Custom regex configuration creation section in the memoQ quick tutorial). After that, rules are applied to the source text, without the need to recreate the project.

It is a truly great feature, and I hope more CAT tools will be offering similar functionality in the future.

memoQ: +

Trados Studio: –

Segment filter

Functionality

A segment filter's functionality, as far as regexes go, should be judged by its ability to simultaneously filter both source and target. Trados Studio used to lag behind in this regard as its default filter only allowed to filter either source or target but not both at once. However, with the introduction of the Advanced Display Filter, this problem was solved.

memoQ is also solid in this department, so it seemed fair to call it a tie.

memoQ: =

Trados Studio: =

Filtering by both source and target

Below is a little demonstration of how segment filters can be used in both tools to display only segments that have upper-case words in source *and* no such words in target.

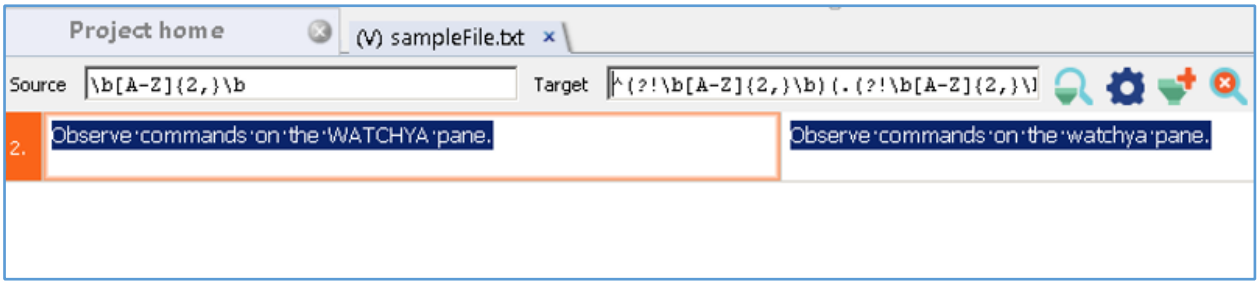

Figure 11-3. memoQ: Segment filter to find source with upper-case words where target does not have such words

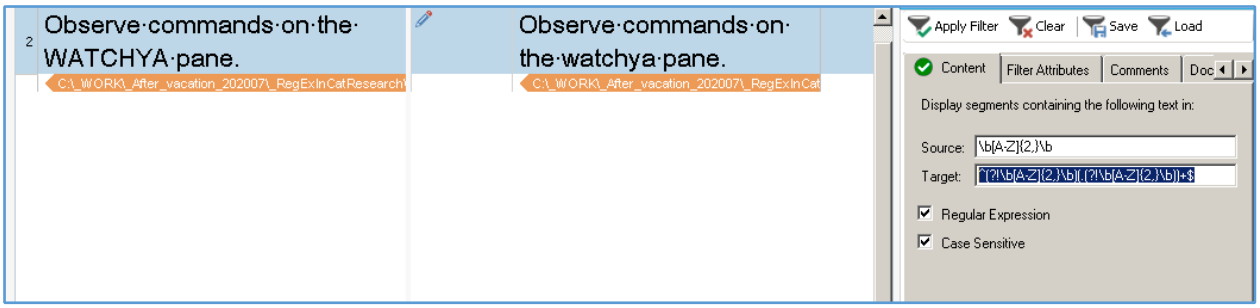

Figure 11-4. Trados Studio: Segment filter to find source with upper-case words where target does not have such words

The regexes are identical for both tools. Here they are:

Source: `\b[A-Z]{2,}\b`

Target: `^(?!\b[A-Z]{2,}\b)(.(?!\b[A-Z]{2,}\b))+$`

Usability

It is a hard one as both tools have room for improvement here.

memoQ's main advantage is that the filter is located right above the segments. It is always there, easily available whenever you may need it. In contrast, Trados Studio's default filter is tied to the Review tab on the upper menu and is hidden when this tab is not active (which is a very common case). Trados's Advanced Display Filter, on the other hand, can be pinned to the side of the screen and stay there until the end of your session, but 1) it still has to be done manually each time you start working in the Editor view and 2) the filter is quite bulky and eats up a lot of space, which can be an issue if you work on a smaller screen.

Based on the above, memoQ seems to be ahead in this category. But it is dragged down by the absence of queries history. There is no drop-down list with previously used queries so you have to retype or copy and paste them every time they need to be reused. This minor yet very annoying usability issue really diminishes user experience, especially on larger projects.

Trados Studio's default filter is free of this defect, but again: it is unclear if it is a tangible advantage as this filter is functionally limited and in most cases the Advanced Display Filter must be used instead. Unfortunately, the latter lacks the drop-down history, just like memoQ's.

Overall, the scale is still slightly tipped in favor of memoQ, which is also helped by the very fact that Trados, in a somewhat cumbersome way, has two filters with overlapping functionality instead of one. The margin is not wide, however.

memoQ: +

Trados Studio: –

Search and replace

Functionality

In most other regards, it is neck and neck, but memoQ offers a very useful feature that Trados Studio does not have: search and replace within tags. Sometimes, the ability to navigate between tags based on their content can be a lifesaver, so this feature alone definitely tips the balance in memoQ's favor.

memoQ: +

Trados Studio: –

Usability

Personal preferences aside, the contestants are more or less equal in this respect. memoQ's Advanced Search And Replace window is a bit bulky, and Trados's Find and Replace window tends to get lost at times among other open windows so you need to thoroughly alt-tab your way to find it. However, these are rather minor issues, and it did not seem fair to call this category for either contestant.

memoQ: =

Trados Studio: =

Regex documentation quality

Trados Studio has an extensive online knowledge base, but many sections of it lack important details and examples. This is particularly true in respect of regexes: information is scarce, and a user is oftentimes left guessing about the implementation specifics. To a large degree, this is made up for by the community input, including the fantastic blog by Paul Filkin (a.k.a. *multifarious*). However, I only considered official documentation, and in this realm, memoQ is way ahead. Its help pages contain multitudes of examples, use cases and detailed instructions. I am not sure if someone without any prior knowledge of regexes could start using them just from reading memoQ's help, but I would not be surprised to learn about such case. To sum it up, with Trados, help content often seems an afterthought, a boring part that just had to be done; whereas memoQ's help system creates an impression of something well thought-out and aimed at helping users.

This one goes to memoQ.

memoQ: +

Trados Studio: –

Winner

As our analysis shows, memoQ keeps the edge over Trados Studio in most categories. It is not a one-way street, and Trados got the better of its competitor or at least held its ground in a number of subcategories. But, all things considered, memoQ is ahead.

The final score (one point for each plus and half a point for each tie) is:

memoQ vs. Trados Studio 10:4

Below is the breakdown by categories:

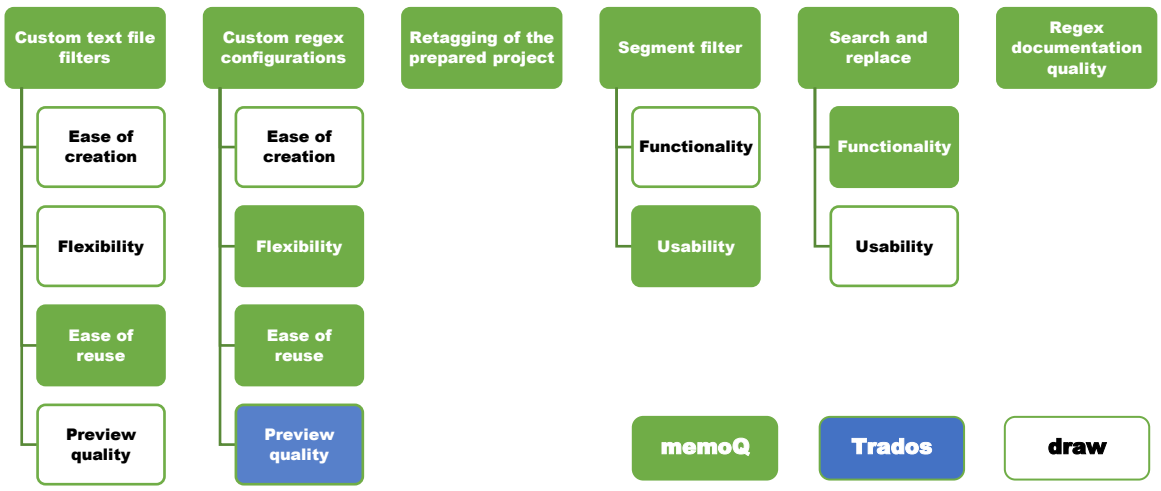

Figure 11-5. memoQ vs. Trados Studio: Comparison results

12. Appendix. Sample files

If you would like to recreate any of the experiments described in this report, you can use the very same files I did. Just copy the text below, paste it into a new text file or a blank Word, Excel or PowerPoint document and save the files with any names you like.

Text file

"string_1_translatable" = "Attach part A0-34-FG6 to evil manifold N on the left."

"string_2_non_translatable" = "If scared, run."

"string_3_translatable" = "Observe commands on the WATCHYA pane."

"string_4_translatable" = "In case of any disobedience, prepare to press the DESTROY button."

"string_5_translatable" = "Detach parts V45-36-12 and A0-34-FG6 (for good measure). Also detach yourself."

"string_6_translatable" = "BEWARE!"

"string_7_non_translatable" = "String 7 should never be translated."

The Big Three

Attach part A0-34-FG6 to evil manifold N on the left.

Observe commands on the WATCHYA pane.

In case of any disobedience, prepare to press the DESTROY button.

Detach parts V45-36-12 and A0-34-FG6 (for good measure). Also detach yourself.

BEWARE!