GENEALOGIES OF KNOWLEDGE

How to use 'Regular Expressions' to speed up the corpus building process

Purpose: A 'regular expression', or 'regex' for short, is a sequence of special text characters which can be used to describe a specific re-occuring pattern within a body of text.

By using regular expressions, the process of preparing and annotating texts for uploading to a corpus can be made much less time- and labour-intensive.

This document provides a brief introduction to regular expressions and the ways in which they can be helpful for the purposes of corpus-building. Additional information can be found in online tutorials such as: http://www.regular-expressions.info/

Regular expressions can be applied in text editing software such as *jEdit* or *Oxygen*. They are used to enhance the Search and Replace function (Ctrl + F) of this software and can help to identify and manipulate certain frequently occurring elements of a corpus text.

Literal and special characters

When using regular expressions, there are two kinds of character: 'literal' characters and 'special' characters.

All alphanumeric characters (the letters **a-z** and the numbers **0-9**) and some punctuation marks are 'literal' characters. Literal characters do not normally have any special function within the regular expression.

For example, if you type the word **democracy** into the search box, the software will simply find every instance in the text where these nine literal characters (**d**, **e**, **m**, **o**, **c**, **r**, **a**, **c** and **y**) are found next to each other (and in this order) within the body of the text.

'Special' characters, on the other hand, have a particular function within the regular expressions syntax.

Below are the special characters and their functions:

1. period or dot .

   The dot matches any (single) character, including blank spaces but except line break characters. It can be used in much the same way as a 'wildcard' token used in other search engine software.

   For example, typing **d.mocracy** into the search box matches **democracy**, **domocracy**, **d-mocracy**, etc.

   This can be useful for identifying and fixing OCR errors caused when the text has been scanned and converted into a text file.

2. the question mark ?

   Placing a question mark after a character renders it optional.

   For example, searching for **colou?r**, matches both **colour** and **color**.

3. square brackets []

   Square brackets can be placed in a search string to designate a 'character class'.

   For example, **[a-z]** will match any lower-case letter of the alphabet, **[A-Z]** will match any upper-case letter of the alphabet and **[a-zA-Z]** will match any letter of the alphabet, including both upper- and lower-case characters.

   [0-9] matches any numerical digit.

4. curly braces {}

   If you want to search for and find more than one character in a character class, then you must specify this using curly braces.

   For example, **[0-9]{2}** will match any sequence of exactly two digits, i.e. **12** or **98** or **45**.

   You can specify a minimum and a maximum using a comma to separate: **{min,max}**.

   For example, **[0-9]{2,}** will match any sequence of at least two digits, while **[0-9]{1,3}** will match any sequence of numbers between one and three digits long, i.e. **5** or **98** or **404**. This can be useful for finding and removing page numbers placed at the top/bottom of each page of a document.

5. the plus sign +

   The plus sign matches the preceding character or character class when it appears one or more times.

   For example, searching for **a+** will find **a**, **aaaa**, **aaaaaaaaaaaa** and **aaaaaaaaaaaaaaa**.

   [a-z]+ will find **a**, **aaaaaaaaaaa**, **bbbbbbb** and **greaagdgadfdfharhgbvxcvbvbbtttttb**

6. the asterisk or star *

   The asterisk is used to find instances where a specific character or character class is repeated any number of times, including zero times.

7. the caret ^

   If you want to omit ('negate') a character or character class from your search, you can place the ^ symbol before this character or character class in the square brackets.

   For example, **q[^u]** matches any **q** and the character that follows it (including a white space), as long as this second character is not a **u**.

   [^a-z] matches any character (including a white space) which is not a lower-case letter of the alphabet.

8. the vertical bar or pipe symbol |

   The vertical bar means 'EITHER/OR'.

   For example, if you want a regex that matches either **online discussion** or **on-line discussion**, you should search for **(online|on-line) discussion**.

9. the backslash \

   The backslash can be used if you want to cancel or 'escape' any special character's special function.

   For example, if you want to search the text for a full stop, you must place a backslash in front of this character **\.**, otherwise the software will treat the full stop as a special character and match every single character in the document.

The backslash is also used to give certain literal characters ‘special’ functions.

\n matches a new line or ‘line break’;

\t matches a tabbed space.

\s matches any white space, including tabs and line breaks;

\w matches a ‘word character’ (alphanumeric characters plus underscore);

\d matches a single digit.

10. parentheses () and the dollar sign $

Parentheses are used to ‘group’ certain characters.

For instance, in the example given above for the vertical line, we used parentheses to tell the search engine to ‘group’ **online** and **on-line** as alternatives in the EITHER/OR query.

In combination with the dollar sign, they can also be used to ‘grab’ a certain set of characters and re-use these same characters via the replace function.

The dollar sign is used to represent any text that you have ‘grabbed’ in the search string and that you want to reuse in the replace string. Grabbed sections are numbered left to right in sequence: **$1**, **$2**, **$3**, **$4**…

See the example given below regarding endnotes for an illustration of how to use parentheses and dollar signs.

Applying regular expressions to corpus text preparation

Regular expressions cannot be used to solve every kind of problem in the process of text annotation. However, they are particularly useful for certain kinds of operation. Below are a series of concrete examples to illustrate the ways in which regular expressions can be applied to corpus text preparation. Please note that every document is different so these regular expressions will likely need to be adapted to each particular case.

- *Paragraphing:*

One of the most common and simplest uses for regular expressions is for removing ‘white space’ and blank lines from the xml document. For example, if after every paragraph in the document there are two (or more) line breaks instead of just one, you can simply Search **\n{2,}** and Replace this with **\n**.

Likewise, tabbed spaces at the beginning of each paragraph can be quickly removed by Searching **\n\t** and Replacing this with **\n**.

<p/> tags can be inserted at the end of every paragraph by Searching for **([^>])\n** and Replacing this with **$1<p/>\n**. Note that by 'negating' the **>** character using the caret **^**, we avoid placing a paragraph tag after content already annotated with another tag: for example, the software will not add a **<p/>** tag after a heading that has already been annotated with **<shead>**…**</shead>** tags.

- *Removing hyphens in the middle of a word:*

Some publishers may regularly split and hyphenate words located at the end of a line in order to improve the layout of the text on the printed page. For the corpus file, however, these hyphens will need to be removed and regular expressions provide an easy way of doing this. Simply Search for **([a-z])- ([a-z])** and Replace this with **$1$2**. This will remove both the hyphen and the space after the hyphen from the middle of the word: **demo- cracy** becomes **democracy**.

- *Inserting <url> tags:*

Since all urls follow a very similar pattern, these can easily be marked up using regular expressions. Search for **https?://[^ <>]+** and Replace this with **<url>$0</url>**. (Note that **$0** re-places back into the text anything found via the entire Search function.

- *Endnotes:*

The annotation of endnotes can be made much easier using regular expressions. If, when looking at the original text, you notice that every endnote (a) begins after a line break, (b) starts with a one- or two-digit number, and (c) ends with a line break, you can type **\n([0-9]{1,2}.*?)\n** in the Search box and Replace this with **\n<endnote>$1</endnote>\n**

This tells the software to take whatever it 'grabbed' between the parentheses in the Search function, and to re-insert this selection back into the text, between the <endnote>…</endnote> tags.

Note that in this case the question mark renders the regular expression 'non-greedy'. This means that the software will only match the smallest amount of text it can, while still satisfying all of its elements.

- *Headers and page numbers:*

If a text has been created by running OCR software on a scanned copy of a physical book, you will likely find that it still contains the header (and perhaps the footer) for every page of the document:

2   Herodotus Book 1

[…]

Herodotus' Introduction   3

[…]

4 Herodotus Book 1

[…]

Herodotus' Introduction 5

[…]

As this only contains information regarding the formatting of the original book (page number and chapter/book title), it should be removed from the corpus text.

Looking at the document, we can see that these headers follow the same patterns: the even numbered pages begin with a one-, two- or three-digit number, followed by a series of blank spaces, followed by the words 'Herodotus Book' and then a one-digit number. They also always begin on a new line and end with a line break.

The odd-numbered pages, on the other hand, include the chapter title, followed by a series of blank spaces, followed by a one-, two- or three-digit number.

So, to remove these, we can simply type the following two regexes in the Search box and leave the Replace box blank (replace with nothing = delete).

For the even-numbered pages:

```
\n[0-9]{1,3}(\s)+(Herodotus Book)\s[0-9]\n
```

And for the odd-numbered pages:

```
\n(Herodotus' Introduction)(\s)+[0-9]{1,3}\n
```

```
\n(The Reign of Croesus of Lydia)(\s)+[0-9]{1,3}\n
```

etc.

- *Removing/modifying existing xml tags:*

Some texts in the corpus (and particularly those included in the premodern corpus) may have already been marked up in xml by another research project. For example, many of the texts available from the website of the Perseus Digital Library project (http://www.perseus.tufts.edu/hopper/) look something like this:

```
<TEI.2>
<text lang="en">
<body>
<div1 type="work" n="Ath. Pol." org="uniform" sample="complete">
<div2 type="chapter" n="3" org="uniform" sample="complete">
<milestone n="1" unit="section"/>
<p>
```

```
The form of the ancient constitution that existed before Draco was as follows. Appointment to the
supreme offices of state went by birth and wealth; and they were held at first for life, and afterwards
for a term of ten years.
<milestone n="2" unit="section"/>
The greatest and oldest of the offices were the King, the War-lord and the Archon. Of these the office
of King was the oldest, for it was ancestral. The second established was the office of War-lord, which
was added because some of the Kings proved cowardly in warfare (which was the reason why the
Athenians had summoned Ion to their aid in an emergency).
<note anchored="yes" resp="Rackham" place="unspecified">
<bibl n="Aristot. Ath. Pol. fr. 1" default="NO" valid="yes">Aristot. Ath. Pol. Fr. 1</bibl>
above.
</note>
<milestone n="3" unit="section"/>
The last of these three offices established was that of the Archon, the institution of which is dated by a
majority of authorities in the time of Medon,
<note anchored="yes" resp="Rackham" place="unspecified">
Son of Codrus (see
<bibl n="Aristot. Ath. Pol. fr. 7" default="NO" valid="yes">Aristot. Ath. Pol. Fr. 7</bibl>
above) and life-archon.
</note>
[…]
```

Since most of these tags do not correspond to elements in the goktext.dtd, this text would not validate correctly if we were to simply upload it to the corpus in this format. However, the task of removing and/or modifying these existing tags is made much easier with regular expressions.

For example, to remove the <milestone> tags, you can simply Search for **<milestone n="[0-9]{1,3}" unit="section"/>** and Replace this with **\n.**

Similarly, the <bibl> tags can be removed by searching for **<bibl n=".*?>.*?</bibl>** and Replacing this with a blank space.

The <div2> tags mark the beginning and end of each chapter of the work so these can usefully be converted into <chapter> tags by Searching **<div2 type="chapter" n="([0-9]{1,3})" .*?>** and Replacing this with **<chapter n="$1">**.

In addition, you would also need to turn all the <note> tags into **<footnote>** tags by Searching for **<note .*?>** and Replacing this with **<footnote>**. You would also need to change the closing tags by Searching for **</note>** and Replacing this with **</footnote>.**

Notes and tips

To enable regular expressions in *jEdit*, you must click on the small tick box labelled 'Regular Expressions' within the Search and Replace window.

If a regular expression is not working in the way that you expect, it is often helpful to break it down into smaller and less complicated expressions, and to test these individually, in order to help identify the problem.

It is best to use the period or dot character sparingly. Often, a character class or negated character class is faster and more precise.