

---

# **FastaChar Documentation**

***Release 0.2.4***

**Lucas Merckelbach, Luisa Borges**

**Jul 21, 2020**



---

## Contents:

---

<b>1</b>	<b>What is Fastachar for and how to use it?</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>9</b>
<b>3</b>	<b>Programming with fastachar</b>	<b>11</b>
<b>4</b>	<b>Regular expressions in FastaChar</b>	<b>15</b>
<b>5</b>	<b>References to the fastachar source code</b>	<b>17</b>
	<b>Bibliography</b>	<b>19</b>



This manual covers the use of the program **Fastachar**.

FastaChar is a software developed to extract molecular diagnostic characters from one or several taxonomically-informative DNA markers of a selected taxon compared to those of other taxa (as many as required by the user) in a single step. The input data consists of a single file with aligned sequences in the fasta format, which can be created using alignment software such as MEGA or GENEIOUS. The software is described by Merckelbach and Borges (2020) [[Merckelbach2020](#)].

The software was developed specifically to determine molecular diagnostic characters for the description of *Lyrodus mersinensis* by Borges and Merckelbach (2018) [[BORGES2018](#)], but it can be applied to any taxon. Since FastaChar is an intuitive and easy-to-use software, we hope it helps to standardize the use of molecular data and stimulate researchers to proceed to the final step of molecular taxonomy, that is, describe the new species.

**Fastachar** is written in Python3, and is released as open source software under the GPLv3 GNU Public License. The installation of the program is (also) covered in the README.rst file, that comes with the source code.



---

## What is Fastachar for and how to use it?

---

**Fastachar** is a graphical user interface to the **fastachar** python module that allows a user to compare pre-aligned DNA sequences. Sequences of different markers must be analysed individually (not concatenated). A typical application is to distinguish one species from a set of different, but closely related species, based on DNA sequences.

### 1.1 Example

Let's assume we have DNA sequences from specimens of cryptic species (a pair or more). After the discovery of the new species it is paramount to carry out the final step in taxonomy, their description. However, in this case, the morphological characters *per se* cannot be used to describe the new cryptic species. Therefore, molecular diagnostic characters (present in all members of a taxon and absent in all other taxa) can be obtained from the DNA sequences. These characters can be used to describe a species in a similar fashion to traditional morphological diagnostic characters used for species descriptions. To that end, the algorithm compares two sets of sequences, with one set consisting of a number of sequences of a taxon (e.g., a new species) and the other consisting of sequences of other taxa (e.g., congeneric or confamilial species). For each homologous position in the alignment (pre-aligned sequences), the algorithm tests for all characters of the sequence in the first set to be the same and to be different from all other characters of the sequences in the other set. When these conditions are met, the character in that position (nucleotide or amino acid) is marked as a *molecular diagnostic character*.

### 1.2 Preparation

The input for *fastachar* is a list of DNA sequences, formatted in the fasta format (see also [https://en.wikipedia.org/wiki/FASTA\\_format](https://en.wikipedia.org/wiki/FASTA_format)). The program assumes that the DNA sequences that are going to be compared already:

- are aligned, and
- written into a single file in fasta format.

There are several software programs to align sequences (e.g. Mega and Geneious).

## 1.3 Running Fastachar

On Windows, *fastachar* is run by executing the *fastachar.exe*, and on Linux, it is run by executing *fastachar* from the terminal console. Once started, a new window appears with three empty text boxes, labelled “Unselected species”, “Selected species list A” and “Selected species list B”, respectively. Below, there is a set of radio buttons to select the comparison operation, a button to execute the comparison (“Process”) and a button (“Clear output”) to clear the output that is generated and shown in the bottom text box, see the Figure.

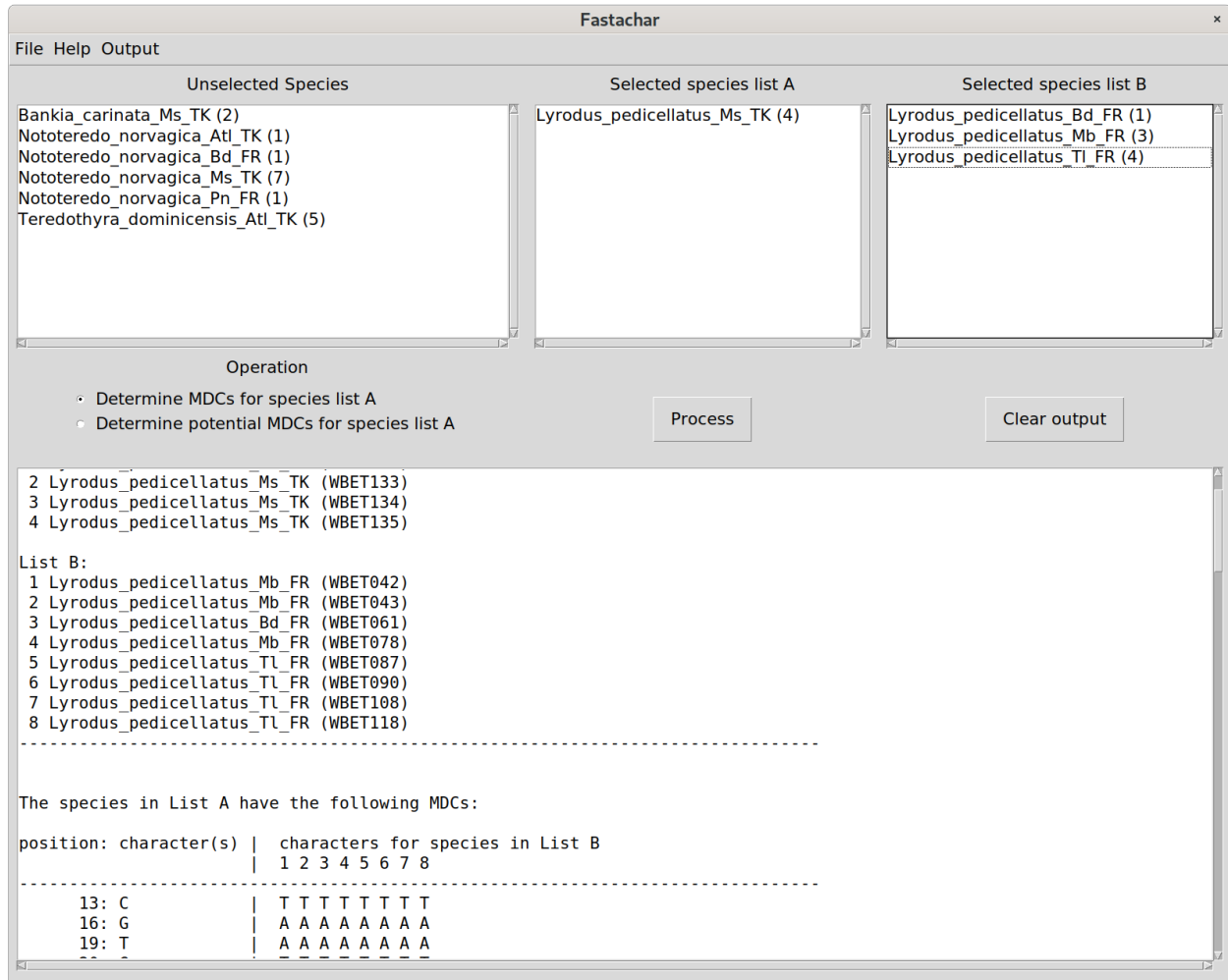


Fig. 1: Figure 1: Main window of *Fastachar*

### 1.3.1 Opening a fasta file

To start working, a fasta file is opened using:

```
File
└─ Open fasta file
```

and select a fasta file from the dialogue offered. If a valid fasta file is read, the text box *Unselected species* is populated with the names of the species found.

Alternatively, a fasta file can be opened using:

```
File
└─ Open fasta file /w preview
```

This allows the user to specify how the *fastachar* should interpret the header strings that precedes each string of sequence data. When opening a file with preview, first a pop-up window appears, similar to the one below

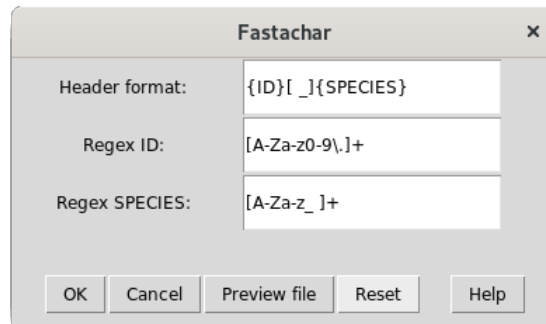


Fig. 2: Figure 2: A pop-up window allowing to specify regular expressions for parsing the fasta header strings

This window provides the user with a way to specify how the header strings are to be parsed. All three fields accept regular expressions, see [Regular expressions in FastaChar](#) for more information on regular expressions and a worked example.

- **Header format::** The **Header format** describes how each header is structured and must contain the strings `{ID}` and `{SPECIES}`. In the example given, the id precedes the species name and a space or an underscore separates the two strings.
- **Regex ID::** The value for the entry **Regex ID** is substituted for the string `{ID}` in the header format string. As this string should match any of the lab codes or IDs used in the fasta file headers, it will usually be a regular expression.
- **Regex SPECIES::** The value for the entry **Regex SPECIES** is substituted for the string `{SPECIES}` in the header format string. Also this string will usually be a regular expression.

After editing the regular expressions, the button *Cancel* cancels the modification, whereas the button *OK* accepts them. The button *Preview file* provides the user with a file chooser dialogue to select a fasta file. After this selection, the file is opened, and parsed. Each header is interpreted and how it fares is shown in a separate window:

In the example of Figure 3, we see in the left column (Header) the string as it appears in the fasta file. In the middle column, the parsed ID string is shown, and in the right column the species name. If the parsing fails completely, dashes only are shown. If the regular expressions do not match the format of the header strings, erroneous results are displayed.

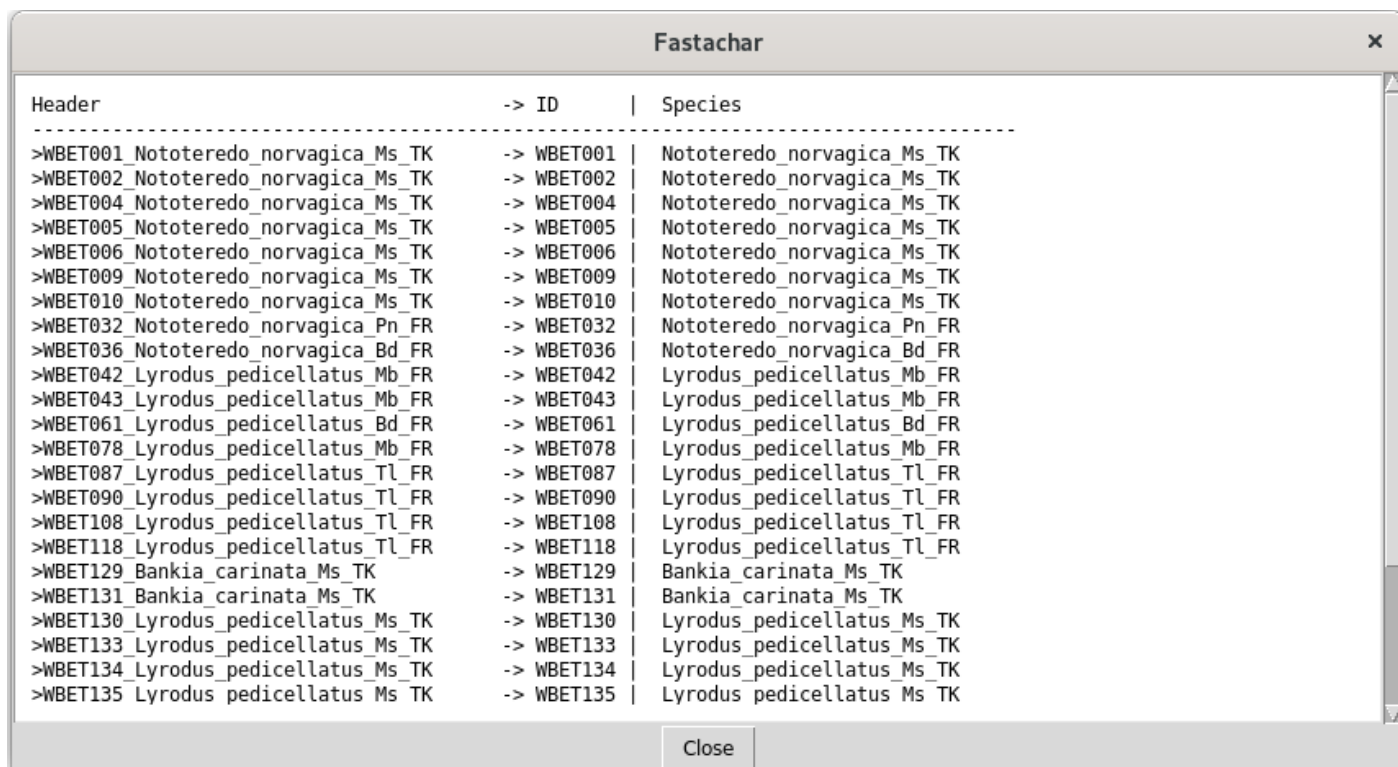
---

**Note:** If somehow the program is not capable of parsing the strings correctly, a work around would be to describe the header string as `{SPECIES}{ID}`, leave the regex for the ID blank, and for the SPECIES a regular expression `.+` is prescribed. FastaChar will now ignore any id's and consider the header of each sequence as a separate species.

---

### 1.3.2 Selecting species for lists A and B

Select a species name by left-clicking. A multiple selection can be made by clicking again with *ctrl* pressed, which also selects the item clicked. If instead of *ctrl* the *shift* key is pressed, all the items in between are selected as well.



Header	ID	Species
>WBET001_Nototerredo_norvagica_Ms_TK	-> WBET001	Nototerredo_norvagica_Ms_TK
>WBET002_Nototerredo_norvagica_Ms_TK	-> WBET002	Nototerredo_norvagica_Ms_TK
>WBET004_Nototerredo_norvagica_Ms_TK	-> WBET004	Nototerredo_norvagica_Ms_TK
>WBET005_Nototerredo_norvagica_Ms_TK	-> WBET005	Nototerredo_norvagica_Ms_TK
>WBET006_Nototerredo_norvagica_Ms_TK	-> WBET006	Nototerredo_norvagica_Ms_TK
>WBET009_Nototerredo_norvagica_Ms_TK	-> WBET009	Nototerredo_norvagica_Ms_TK
>WBET010_Nototerredo_norvagica_Ms_TK	-> WBET010	Nototerredo_norvagica_Ms_TK
>WBET032_Nototerredo_norvagica_Pn_FR	-> WBET032	Nototerredo_norvagica_Pn_FR
>WBET036_Nototerredo_norvagica_Bd_FR	-> WBET036	Nototerredo_norvagica_Bd_FR
>WBET042_Lyrodus_pedicellatus_Mb_FR	-> WBET042	Lyrodus_pedicellatus_Mb_FR
>WBET043_Lyrodus_pedicellatus_Mb_FR	-> WBET043	Lyrodus_pedicellatus_Mb_FR
>WBET061_Lyrodus_pedicellatus_Bd_FR	-> WBET061	Lyrodus_pedicellatus_Bd_FR
>WBET078_Lyrodus_pedicellatus_Mb_FR	-> WBET078	Lyrodus_pedicellatus_Mb_FR
>WBET087_Lyrodus_pedicellatus_Tl_FR	-> WBET087	Lyrodus_pedicellatus_Tl_FR
>WBET090_Lyrodus_pedicellatus_Tl_FR	-> WBET090	Lyrodus_pedicellatus_Tl_FR
>WBET108_Lyrodus_pedicellatus_Tl_FR	-> WBET108	Lyrodus_pedicellatus_Tl_FR
>WBET118_Lyrodus_pedicellatus_Tl_FR	-> WBET118	Lyrodus_pedicellatus_Tl_FR
>WBET129_Bankia_carinata_Ms_TK	-> WBET129	Bankia_carinata_Ms_TK
>WBET131_Bankia_carinata_Ms_TK	-> WBET131	Bankia_carinata_Ms_TK
>WBET130_Lyrodus_pedicellatus_Ms_TK	-> WBET130	Lyrodus_pedicellatus_Ms_TK
>WBET133_Lyrodus_pedicellatus_Ms_TK	-> WBET133	Lyrodus_pedicellatus_Ms_TK
>WBET134_Lyrodus_pedicellatus_Ms_TK	-> WBET134	Lyrodus_pedicellatus_Ms_TK
>WBET135_Lyrodus_pedicellatus_Ms_TK	-> WBET135	Lyrodus_pedicellatus_Ms_TK

Fig. 3: Figure 3: Popup window showing the results of the header parsing.

In order to move them into either list A or list B, drag the selected items from the *Unselected species* text box to the target text box whilst holding the right-mouse button pressed.

### 1.3.3 Selecting the operation

Once the selection is made, the comparison operation is to be selected. Two operations are implemented:

- Determining MDCs for species list A
- Determining potential MDCs for species list A

After selecting the operation, the operation is executed by clicking the *Process* button, and a report appears in the lower text box, see Figure 1.

The output lists the path of the input fasta file (not shown in Figure 1), the species' names and IDs of the sequences in list A, and list B. If the species in list A have any molecular diagnostic characters, then they are listed by their position, their value, and the values of the sequences in list B for the same position. Note that masked characters, if any, are left blank.

A molecular diagnostic character is the character at position  $k$  of the sequences in list A, for which holds that:

- 1) all characters in A are identical for this position, and
- 2) all characters in B for are different from those in A for this position.

For a potential diagnostic character, the second condition is met only. For a precise definition, the user is referred to the accompanying paper, see [Merckelbach2020].

### 1.3.4 Case files

To facilitate repeated operations on a specific file, or storing the specifics of a case for future reference, a so-called case file can be used. When writing a case file via:

```
File
└─ Save case file
```

the following information can be stored:

- the fasta file read
- the regular expressions used for reading
- the selection made
- the operation selected.

A previously saved case file can then be loaded by

```
File
└─ load case file
```

### 1.3.5 Output

Multiple operations as well as species selections can be processed and the output will be appended to the lowest text box. The output can be cleared using the *Clear output* button.

To save the output to file, select from the menu:

```
Output
└─ Save report (txt)
```

to write the output of the last operation as shown in a text file, or

```
Output
└─ Save report (xls)
```

to write the output in an excel file, with a tab for each processing operation.

### 1.3.6 Help

The user interface also provides help and information on the licensing from the menu entry:

```
Output
└─ Help
```

and

```
Output
└─ About
```

respectively.



## CHAPTER 2

---

### Installation

---

**Fastachar** is written in Python3 and should run on all major platforms, including linux and windows. In order to run **Fastachar** a working copy of the python3 interpreter is required (including the tkinter windowing toolkit).

### 2.1 Linux

As of the beginning of 2020, support for python2 is officially dropped. This means that most likely python points to python version 3. In the documentation below, python3 is used explicitly, but on recent linux distributions, the '3' can probably be left out.

Usually python3 is included in most linux distributions. A simple test is to open a terminal and try to run python by:

```
$ python3
```

which should give you the python interpreter (including its version number). If it is verified that this works, test for the presence of the tkinter windowing toolkit:

```
>>> import tkinter
```

If this does not raise an exception, then you are all good and you can exit the interpreter (ctl-D). Otherwise you will need to install tkinter yourself, which is probably best done via your distribution's package manager.

**Fastachar** can be installed from pypi, using pip from the command line:

```
$ pip3 install fastachar
```

or from a tar-gzipped file downloaded from [FastaChar github repository](#). After extracting the .tgz file and cd-ing into the newly created directory, you run:

```
$ python3 setup.py build && sudo python3 setup.py install && sudo python3 setup.py_
↪clean
```

Either installation method for **Fastachar** should take care of installing the dependencies (xlwt) correctly.

---

**Note:** In recent versions of linux python3 may be installed as default, and commands such as `pip3` and `python3` should be spelled without the suffix 3.

---

## 2.2 Windows

Usually python is not installed by default on a Windows computer and needs to be installed by the user. In this readme the official python distribution will be used, but other python packages exist and may work equally well, too.

To install Python3, visit <https://www.python.org/downloads/windows/> and select the (latest) python3 version. (Do not select Python2 as this is not supported by **FastaChar**.) When downloading the Python3 distribution, make sure you select the proper version for your computer hardware:

- Download Windows x86 executable installer for a 32 bit system
- Download Windows x86-64 executable installer for a 64 bit system.

You can check from the **control panel/system** which version your computer is running on, when in doubt.

When, the file is downloaded, run it and follow the default installation (which includes pip and tkinter, which are both needed for successful operation of **FastaChar**). It is recommend however to check the box to add Python3 to the PATH environment variable.

Once Python3 is installed, **FastaChar** can be installed using pip. This requires a dos prompt (go to **Start/search for programs** and enter **cmd**, which should give an entry to the dos-command line. Type in the dos-prompt:

```
pip3 install fastachar
```

or:

```
py -3 pip -m install fastachar
```

which should also install the dependency xlwt.

---

## Programming with fastachar

---

The graphical user interface is intended to provide easy access to the functionality offered by the **fastachar** module. Rather than using the graphical interface, the user can also create her/his own python scripts.

### 3.1 Example script

```
1  # Example script how to do an analysis of a fasta file accessing the
2  # fasta module directly, and not using a graphical interface. The
3  # example script reads a fasta file, and divides all the species in a
4  # two sets, one that with species names that match a regular
5  # expression, and a set with sequences that does not match the regular
6  # expression. Then, for set A, the differences within this set as well
7  # as its unique characters are computed. Finally, the results are
8  # reported and dumped on the terminal.
9  import sys
10 sys.path.insert(0, '..')
11 import fastachar
12
13
14 filename = "../data/COI_sequences_MUSCLE.fas"
15
16 alignment = fastachar.fasta_io.Alignment()
17 # The sequences in this alignemnt typically look like this:
18 # >WBET001_Nototeredo_norvagica_Ms_TK
19
20 # that is, an ID, followed by an underscore and a species name. In
21 # order to parse this sequence header correctly, we must tell the
22 # alignment reader how this header is constructed.
23 # See http://www.rexegg.com/regex-quickstart.html for a reference table.
24 alignment.set_fasta_hdr_fmt(header_format='{ID}_{SPECIES}',
25                             IDregex = '[A-Z0-9]+',
26                             SPECIESregex = '[A-Z][a-z_]+')
27
```

(continues on next page)

(continued from previous page)

```

28  errno, errmsg = alignment.load(filename)
29  if errno: # we have a non-zero error, so something went wrong. Print
30            # the corresponding message to give us a clue
31      print(errmsg)
32  else:
33      # all well.
34      species = alignment.get_species_list()
35      print("Species in this file:")
36      for s in species:
37          print("{:30s}".format(s))
38      print()
39
40      # Divide all the species in two groups, set A that matches the regex,
41      # and set B that does not. Notice we can use regular expressions here too.
42      lst_A, lst_B = alignment.select_two_sequence_sets("Lyrodus.pedicellatus.*[mM][Ss]
↪")
43
44      # We could also use other methods to extract specific sequences.
45      # Let's investigate Lyrodus pedicellatus. We suspect that the
46      # sequences found in Turkey, they end with TK might be different
47      # from those found in France (ending in Fr). So we select all
48      # Lyrodus species, but exclude those ending in TK, for lst_A, and
49      # do the same for lst_B, but invert the selection.
50
51      lst_A = alignment.select_sequences(regex='Lyrodus[_ ]pedicellatus.*',
52                                       invert=False,
53                                       exclude='.*[Tt][Kk]')
54      lst_B = alignment.select_sequences(regex='Lyrodus[_ ]pedicellatus.*',
55                                       invert=True,
56                                       exclude='.*[Tt][Kk]')
57
58
59      S = fastachar.fasta_logic.SequenceLogic()
60
61      # Compute the unique characters in A with respect to B
62      method = "MDC"
63
64      mcds = S.compute_mdcs(lst_A, lst_B, method)
65
66      # Report the results to the terminal.
67      reportxls = fastachar.fasta_io.ReportXLS()
68      report = fastachar.fasta_io.Report(filename, reportxls = reportxls)
69      report.report_header(lst_A, lst_B, method)
70      report.report_mdcs("List A", lst_A, lst_B, mcds, method)
71
72      # compute non unique characters in B
73      nucs = S.list_non_unique_characters_in_set(lst_B)
74
75      report.report_header(lst_A, lst_B, method='nucs')
76      report.report_nucs("List B", lst_B, nucs)
77
78      #reportxls.save('test.xls')
79
80
81
82

```

The advantage of using scripts such as the one above, is that it is easier to redo an analysis, modify an existing one, or batch analyses a number of fasta files.

The API for the class `SequenceData` can be consulted `modindex`.



---

### Regular expressions in FastaChar

---

#### 4.1 What is a regular expression anyway?

Adapted from wikipedia :

The phrase regular expressions, also called regexes, is often used to mean the specific, standard textual syntax for representing patterns for matching text. Each character in a regular expression (that is, each character in the string describing its pattern) is either a metacharacter, having a special meaning, or a regular character that has a literal meaning.

For example, in the regex 'a.', a is a literal character which matches just 'a', while '.' is a metacharacter that matches every character except a newline. Therefore, this regex matches, for example, 'a ', or 'ax', or 'a0'. Together, metacharacters and literal characters can be used to identify text of a given pattern, or process a number of instances of it.

Pattern matches may vary from a precise equality to a very general similarity, as controlled by the metacharacters. For example, '.' is a very general pattern, [a-z] (match all lower case letters from 'a' to 'z') is less general and a is a precise pattern (matches just 'a'). The metacharacter syntax is designed specifically to represent prescribed targets in a concise and flexible way to direct the automation of text processing of a variety of input data, in a form easy to type using a standard ASCII keyboard.

In **FastaChar** we use regular expressions to parse the header strings belonging to the sequences in fasta files.

Extensive information on regular expression can be found in sources on the internet, for example <https://www.rexegg.com/regex-quickstart.html>.

#### 4.2 How do we use regular expressions in FastaChar?

When **FastaChar** reads a fasta file with aligned sequences, this file can have a number of sequences pertaining to one taxon. For the analysis we would like to compare the sequences of this taxon with those of other taxa. In order to select all the sequences of a given taxon and label them with one name, the species name, **FastaChar** needs some way of knowing how to interpret the headers in the fasta files. This may be best illustrated using an example.

### 4.2.1 Example

Let us say we have a fasta file with the following entry:

```
>WBET001_Nototeredo_norvagica_Ms_TK
TACTTTGTATTTTATTTTCTATTGAGCGGGTTGGT.....
```

Here we see that the first line is the header, as it starts with a “>”. The header string is apparently composed of the lab id followed by the species description, using an underscore to separate them. The header format now becomes:

```
{ID}_{SPECIES}
```

In order to specify the regular expression for the ID string, we need to know how the other id’s in this file look like. If we know that all lab codes start with ‘WBET’, we could specify something like:

```
WBET[0-9]+
```

which should be interpreted as the id starts with WBET and is followed by at least one numerical, but nothing else. The WBET part is taken literal. The part between [ ] represents the position of one single character. In this case this character can be any in the range from 0 to 9. The + means that the preceding character (or possible characters) can be repeated.

This would work for this example, but when a different file is opened, then this expression might not match. As an alternative approach we could be more general. So we may say that the id may contain alphanumeric characters and a period, and at least one character. This translates to:

```
[A-Za-z0-9\.] +
```

Now the first character can be anything from upper case and lower case letters that appear in the (English) alphabet, any digits from 0 - 9 and a period. The symbol . has a special meaning in regular expressions, so that if the literal symbol is meant, it must be escaped by a backslash. What follows of the id string should be a character that follows the same restriction as the first character does, as indicated by the + symbol.

This representation would match our example WBTE001, but also PC025239.1, or ZSM20100595. Similar considerations apply to how the regular expression should be described for the species string.

## 4.3 Can we disable the use of regular expressions?

It can be, of course, that species names and lab codes within a single fasta file do not adhere to a specific format, or they are formatted in such a way that it is not easy to find a regular expression pattern that works for all entries. The solution in such a case would be to specify a regular expression that captures all.

The header format then reads:

```
{SPECIES}
```

the field for the regular expression for the ID or lab code, Regex ID, can be left empty, and the regular expression for the species, Regex SPECIES becomes:

```
. *
```

which captures all characters.

---

### References to the fastachar source code

---

- [genindex](#)
- [modindex](#)
- [search](#)



---

## Bibliography

---

- [Merckelbach2020] Merckelbach, L. M., & Borges, L. M. S. (2020). Make every species count: fastachar software for rapid determination of molecular diagnostic haracters to describe species. *Molecular Ecology Resources*, 00:1–8. <https://doi.org/10.1111/1755-0998.13222>.
- [BORGES2018] Borges, L. M. S., & Merckelbach, L. M. (2018). *Lyrodus mersinensis* sp. nov. (Bivalvia: Teredinidae) another cryptic species in the *Lyrodus pedicellatus* (Quatrefages, 1849) complex. *Zootaxa*, 4442(3), 441–457. <https://doi.org/10.11646/zootaxa.4442.3.6>