log2seq

Mar 29, 2022

Contents

CHAPTER 1

log2seq

Log2seq is a python package to help parsing syslog-like messages into word sequences that is more suitable for further automated analysis. It is based on a procedure customizable with rules in order, using regular expressions.

- Document: https://log2seq.readthedocs.io
- Source: https://github.com/amulog/log2seq
- Bug Reports: https://github.com/amulog/log2seq/issues
- Author: Satoru Kobayashi
- License: BSD-3-Clause

1.1 Installation

You can install log2seq with pip.

```
pip install log2seq
```

1.2 Tutorial

Log2seq is designed mainly for preprocessing of automated log template generation. Many implementations of template generation methods requires input log messages in segmented format, but they only support simple preprocessing, using white spaces. Log2seq enables more flexible preprocessing enough for parsing practical log messages.

For example, sometimes you may face following format of log messages:

```
Jan  1 12:34:56 host-device1 system[12345]: host 2001:0db8:1234::1 (interface:eth0)␣
↪disconnected
```

This message cannot well segmented with `str.split()` or `re.split()`, because the usage of `:` is not consistent.

log2seq processes this message in multiple steps (in default):

1. Process message header (i.e., timestamp and source hostname)
2. Split message body into word sequence by standard symbol strings (e.g., spaces and brackets)
3. Fix words that should not be splitted later (e.g., ipv6 addr)
4. Split words by inconsistent symbol strings (e.g., `:`)

Following is a sample code:

```
mes = "Jan  1 12:34:56 host-device1 system[12345]: host 2001:0db8:1234::1␣
↪(interface:eth0) disconnected"

import log2seq
parser = log2seq.init_parser()

parsed_line = parser.process_line(mes)
```

Here, you can get the parsed information like:

```
>>> parsed_line["timestamp"]
datetime.datetime(2020, 1, 1, 12, 34, 56)

>>> parsed_line["host"]
'host-device1'

>>> parsed_line["words"]
['system', '12345', 'host', '2001:0db8:1234::1', 'interface', 'eth0', 'disconnected']
```

You can see `:` in IPv6 address is left as is, and other `:` are removed.

This example is using a default parser, but you can also customize your own parser. For defails, please see the document.

CHAPTER 2

Documentation

Release 0.3.1

Date Mar 29, 2022

2.1 Main module

2.1.1 LogParser

class log2seq.**LogParser**(*header_parsers*, *statement_parser*, *ignore_failure=False*)

Log parser object.

LogParser in log2seq consists of two different parsers: *HeaderParser* and *StatementParser*.

Parsed results are returned in one dict object. It consists of following parsed information.

- Header information (*value_name* as key)
- Statement part in string format ("message" as key)
- Segmented words in statement part ("words" as key)
- Separator symbols in ststement part ("symbols" as key)

Example

```
>>> mes = "Jan  1 12:34:56 host-device1 system[12345]: host 2001:0db8:1234::1␣
↪(interface:eth0) disconnected"
>>> import log2seq
>>> parser = log2seq.init_parser()  # get default LogParser
>>> parsed_line = parser.process_line(mes)
>>> parsed_line["timestamp"]  # timestamp parsed by HeaderParser
datetime.datetime(2020, 1, 1, 12, 34, 56)
>>> parsed_line["host"]  # Hostname item in HeaderParser
```

(continues on next page)

(continued from previous page)

```
'host-device1'
>>> parsed_line["message"]  # Statement part parsed by HeaderParser
'system[12345]: host 2001:0db8:1234::1 (interface:eth0) disconnected'
>>> parsed_line["words"]  # Segmented words parsed by StatementParser
['system', '12345', 'host', '2001:0db8:1234::1', 'interface', 'eth0',
↪'disconnected']
>>> parsed_line["symbols"]  # Separator symbols parsed by StatementParser
['', '[', ']: ', ' ', ' (', ':', ') ', '']
```

You can specify multiple *HeaderParser* as input. If so, *LogParser* try to parse a log message with them in order, and the first matched rule is used for the message.

Parameters

- **header_parsers** (*HeaderParser* or list of it) – one or multiple HeaderParser instance to use.
- **statement_parser** (*StatementParser*) – one StatementParser instance to use.
- **ignore_failure** (*bool, optional*) – If true, ignore non-matching lines with given header parsers.

process_header (*line*, *verbose=False*)

Parse header part in a log message.

This function uses all given HeaderParser rules. If all HeaderParser fails to match the input log message, it raises a *LogParseFailure* exception.

Parameters

- **line** (*str*) – A log message.
- **verbose** (*bool, optional*) – Show intermediate progress of applying header rules.

Returns parsed header data.

Return type dict

process_line (*line*, *verbose=False*)

Parse a log message (i.e., a line).

If all HeaderParser fails to match the input log message, it raises a LogParserFailure exception.

Parameters

- **line** (*str*) – A log message. Line feed code will be removed.
- **verbose** (*bool, optional*) – Show intermediate progress of applying rules.

Returns parsed data.

Return type dict

process_statement (*statement*, *verbose=False*)

Parse a log statement.

Parameters

- **statement** (*str*) – Statement part in a log message.
- **verbose** (*bool, optional*) – Show intermediate progress of applying rules.

Returns List of two components: words and symbols. See *statement.StatementParser.process_line()*.

Return type tuple

log2seq.**init_parser**(*header_parsers=None*, *statement_parser=None*)
Generate *LogParser* object.

If no arguments are given, this function generates LogParser with default configurations.

Parameters

- **header_parsers** (*HeaderParser* or list of it, optional) – one or multiple HeaderParser instance to use. If not given, use *preset.default_header_parsers()*.
- **statement_parser** (*StatementParser*) – one StatementParser instance to use. If not given, use *preset.default_statement_parser()*.

2.1.2 Exceptions

class log2seq.**ParserDefinitionError**
ParserDefinitionError is raised when the given rules are inappropriate (e.g., having syntax errors).

class log2seq.**LogParseFailure**
LogParserFailure is raised when the input log message not matched with all given HeaderParser rules.

If you want to pass such mismatching log messages, use try-except with this exception.

2.2 Header rules

2.2.1 HeaderParser

class log2seq.header.**HeaderParser**(*items*, *separator=None*, *full_format=None*, ***kwargs*)
Parser for header parts in log messages.

Header parts in log messages provides some items of meta-information. For example, default syslogd records messages with timestamps and hostnames as header information. The other parts (free-format statements) are parsed as statement part (with item *Statement*).

A HeaderParser rule is represented with a list of *Item*. Item is a component of regular expression patterns to parse corresponding variable item. HeaderParser automatically generates one regular expression pattern from the items, and tests that it matches the input log messages. If matched, HeaderParser extracts variables for the items.

In HeaderParser rule, one *Statement* item is mandatory.

If you want to extract timestamp in datetime.datetime format (i.e., using reformat_timestamp option), the items should include ones with special value names (see *value_name*):

- year (int)
- month (int)
- day (int)
- hour (int, optional)
- minute (int, optional)
- second (int, optional)
- microsecond (int, optional)
- tzinfo (datetime.tzinfo, optional)

Besides, you can also use aggregated items with following value names:

- datetime (datetime.datetime): all
- date (datetime.date): year, month, day
- time (datetime.time): hour, minute, second, microsecond, tzinfo

If some timestamp-related items are not given, please add corresponding values (in the specified type) in defaults. Note that "year" is missing in some logging framework (e.g., default syslogd configuration).

There are two options to define the placement of Items. One is "separator", which is an easier (and recommended) option. Separator defines separator characters between Items. The other is "full_format", which is similar to log_format in logparser[1]. It is a regular expression holed with Item replacers. For example, if full_format is r"<0> <1> <2> [<3>] <4>", <0> will be replaced with the first *Item* in items (The number corrsponds to the index of given items). If you need "<" and ">", escape it with a backslash. The number of replacers must be equal to the length of items. Note that optional Items must be manually enclosed with "(" and ")?" in the full_format regular expression. (e.g., r"<0> <1> <2> ([<3>])?<4>" where Item-3 is optional.)

Parameters

- **items** (list of *Item*) – header format rule.
- **separator** (*str, optional*) – Separators for header part. Defaults to white spaces.
- **full_format** (*str, optional*) – Place format of header part. If given, argument separator is ignored.
- **defaults** (*dict, optional*) – Default values, used for missing values (for optional or missing items) in log messages.
- **reformat_timestamp** (*bool, optional*) – Transform time-related items into a timestamp in datetime.datetime object. Set false if log messages do not have timestamps.
- **astimezone** (*datetime.tzinfo, optional*) – Convert timestamp to given new timezone by calling datetime.datetime.astimezone(). Effective only when reformat_timestamp is True.

Reference: [1] logparser: https://github.com/logpai/logparser

process_line(*line*)

Parse header part of a log message (i.e., a line).

Parameters **line** (*str*) – A log message without line feed code.

Returns Parsed items.

Return type dict

2.2.2 Basic items

class log2seq.header.**Item**(*optional=False*, *dummy=False*)

Base class of items, components of header parts.

Parameters

- **optional** (*bool, optional*) – This item is optional. Not all inputs need this item in their header parts. If true, Item.pick() returns None if no corresponding part found.
- **dummy** (*bool, optional*) – Dummy items do not extract any values. If true, log2seq does not try extracting a value for this item, and Item.pick() will not be called for this item.

For example, if a header part have multiple same value (e.g., year in top and middle), one of them should be dummy for avoiding re groupname duplication.

get_regex()
Get regular expression pattern string of this *Item* instance.

match_name
Match name of this Item.

Match name is used to distinguish the extracted values in re MatchObject. Match name cannot be duplicated in a set of ParserHeader items.

Type str

pattern
Get regular expression pattern string for this *Item class*.

Type str

pick(*mo*)
Get value name and the extracted values from re MatchObject in appropriate format.

Parameters **mo** – MatchObject for combined pattern of *HeaderParser*.

Returns *value_name* and the value extracted by *Item.pick_value()*.

Return type tuple

pick_value(*mo*)
Get a value from re MatchObject in appropriate format.

Parameters **mo** – MatchObject for combined pattern of *HeaderParser*.

Returns Extracted value for this *Item*. Any type, depending on the class. If not specified, a matched string value is returned as is.

test(*string*)
Test this Item will match the input string or not. Note that this function is only for debugging your parser script (because it generates internal re.Pattern for every call).

Parameters **string** – Input string to test matching.

Returns re.Match or None

value_name
Value name of this *Item*.

Value name is used as the keys of return value of *HeaderParser*. Also, timestamps are reformatted with specific value names.

Type str

class log2seq.header.**ItemGroup**(*items*, *separator=None*, *optional=False*)
ItemGroup enables us a hierarchical parsing of Items. One typical use is defining an optional part including multiple Items appearing together. Another use is using different separator definition in the ItemGroup part.

class log2seq.header.**Statement**(*optional=False*, *dummy=False*)
Item for statement part. Usually it includes strings except all other items with greedy match.

2.2.3 Timestamp items

class log2seq.header.**UnixTime**(*optional=False*, *dummy=False*)
Item for unixtime integer.

e.g., `1551024123` for 2019-02-25 01:02:03

class log2seq.header.**DatetimeISOFormat**(*optional=False*, *dummy=False*)
Item for datetime in ISO8601 (or RFC3339) format. Datetime information (year, month, day, hour minute, second) are always included. Microseconds and timezone are optionally extracted.

e.g., `2112-09-03T11:22:33`

e.g., `2112-09-03T11:22:33.012345+09:00`

class log2seq.header.**Date**(*optional=False*, *dummy=False*)
Item for date, including year, month, and day. Represented in eight-letter numeric string separated with two hyphens. Similar to the formar part of DatetimeISOFormat.

e.g., `2112-09-03`

class log2seq.header.**Time**(*optional=False*, *dummy=False*)
Item for time, including hour, minute, and second. It can also include microsecond and timezone, as like *DatetimeISOFormat*.

e.g., `11:22:33`

class log2seq.header.**MonthAbbreviation**(*optional=False*, *dummy=False*)
Item for abbreviated month names. Strings with first capitalized 3 characters will match (e.g., `Jan`, `Feb`, `Mar`, ...).

class log2seq.header.**DemicalSecond**(*optional=False*, *dummy=False*)
Item for demical seconds.

e.g., `678` as milliseconds

e.g., `123456` as microseconds

class log2seq.header.**TimeZone**(*optional=False*, *dummy=False*)
Item for timezone.

e.g., `+0900`

class log2seq.header.**DateConcat**(*no_century=False*, ***kwargs*)
Item for date without separators.

e.g., `20190225` for 2019-02-25

e.g., `190225` for 2019-02-25 (no_century is True)

Parameters **no_century** (*bool, optional*) – If true, abbreviate year by removing century.

class log2seq.header.**TimeConcat**(*optional=False*, *dummy=False*)
Item for time without separators.

e.g., `010203` for 01:02:03

2.2.4 Variable items

class log2seq.header.**NamedItem**(*name*, ***kwargs*)
A base class of namable items. Namable items requires an argument for the name. The name is used as match name and value name. The name should not be duplicated with match names of other items (including unnamable items) in one *HeaderParser* rule.

Parameters **name** (*string*) – name of *Item* instance, used as match name and value name.

class log2seq.header.**Digit**(*name*, ***kwargs*)
NamedItem for a digit value.

class log2seq.header.**Hostname**(*name*, ***kwargs*)
NamedItem for a hostname (or IPaddress) string.

Check Hostname.pattern to see the accepted names. If your hostname does not match the pattern, consider using UserItem. (This is because hostname can include various values depending on the devices or OSes.)

class log2seq.header.**UserItem**(*name*, *pattern*, *strip=None*, ***kwargs*)
Customizable *NamedItem*.

The pattern is described in Python Regular Expression Syntax (re). Some special characters are not allowed to use for this Item because HeaderParser generates a single re.Pattern by automatically combining the given set of items.

- Optional parts, such as `?`
- `^` and `$`

Parameters

- **name** – same as NamedItem.
- **pattern** – regular expression pattern of this Item instance.
- **strip** (*str, optional*) – specified characters will be stripped with str.strip() in the parsed object.

2.3 Statement rules

2.3.1 StatementParser

class log2seq.statement.**StatementParser**(*actions*)

Parser for statement parts in log messages.

Statement parts in log messages describe the event in free-format text. This parser will segment the text into words and theire separators. The words are parsed as 'words' item, and the separators are parsed as 'symbols' item.

The behavior of this parser is defined with a list of actions. The actions are sequentially applied into the statement, and separate it into a sequence of words (and separator symbols).

Parameters **actions** (*list of any action*) – Segmentation rules. The rules are sequentially applied to the input statement.

process_line(*statement: str*, *verbose: bool = False*)

Parse statement part of a log message (i.e., a line).

Parameters

- **statement** (*string*) – String of statement part.
- **verbose** (*bool, optional*) – Show intermediate progress of applying actions.

Returns List of two components: words and symbols. First component, words, is list of words. Second component, symbols, is list of separator string symbols. The length of symbols is always len(words)+1, which includes one before first word and one after last word. Some symbols can be empty string.

Return type tuple

2.3.2 Standard actions

class log2seq.statement.**Split**(*separators*)

Split statement (or its parts) by given separators.

For example, separators of white space and dot translates

```
['This is a statement.'] -> ['This', 'is', 'a', 'statement']
```

The removed separators (white space and dot in this case) will not be considered in further actions.

Example

```
>>> parser = StatementParser([Split(" .")])
>>> parser.process_line("This is a statement.")
(['This', 'is', 'a', 'statement'], ['', ' ', ' ', ' ', '.'])
```

Parameters **separators** (*str*) – separator symbol strings. If iterable, they imply joined and used all for segmentation. Escape sequence is internally added, so you don't need to add it manually.

class log2seq.statement.**Fix**(*patterns*)

Add Fixed flag to matched parts.

Fixed parts will not be segmented by following actions. Fixed parts are selected by regular expression of given pattern (see re).

Example

```
>>> p = StatementParser([Split(" "), Fix(r".+\.txt"), Split(".")])
>>> p.process_line("parsing sample.txt done.")
(['parsing', 'sample.txt', 'done'], ['', ' ', ' ', '.'])
```

Parameters **patterns** (*str or list of str*) – Regular expression patterns. If multiple patterns are given, they are matched with every word in order.

2.3.3 Extended actions

class log2seq.statement.**FixIP**(*address=True*, *network=True*)

Add Fixed flag to the parts of IP addresses.

This class use ipaddress library instead of regular expression.

Parameters

- **address** – match IP addresses, defaults to True
- **network** – match IP networks, defaults to True

class log2seq.statement.**FixPartial**(*patterns*, *fix_groups*, *recursive=False*, *remove_groups=None*, *rest_remove=False*)

Extended Fix action to accept complicated patterns.

Usual *Fix* consider the matched part as a word, and fix it. In contrast, *FixPartial* allow the matched part to include multiple fixed words or separators.

Usecase 1:

e.g., `source 192.0.2.1.80 initialized.`

If you intend to consider `192.0.2.1.80` as a combination of two different word: IPv4 address `192.0.2.1` and port number `80`, this cannot be segmented with simple Fix and Split actions. Following example with *FixPartial* can fix these two variables.

Example

```
>>> pattern = r'^(?P<ipaddr>(\d{1,3}\.){3}\d{1,3})\.(?P<port>\d{1,5})$'
>>> parser = StatementParser([Split(" "), FixPartial(pattern, fix_groups=["ipaddr
↪", "port"]), Split(".")])
>>> parser.process_line("source 192.0.2.1.80 initialized.")
(['source', '192.0.2.1', '80', 'initialized'], ['', '', '.', '', '.'])
```

Usecase 2:

e.g., `comment added:  "This is a comment description".`

If you intend to consider the comment (strings between parenthesis) as a word without segmentation, this cannot be achieved with simple Fix and Split actions. Following example with *FixPartial* can fix the comment part.

Example

```
>>> pattern = r'^.*?(?P<left>")(?P<fix>.+?)(?P<right>").*$'
>>> parser = StatementParser([FixPartial(pattern, fix_groups=["fix"], \
... remove_groups=["left", "right"], rest_remove=False), Split(' :.')])
>>> parser.process_line('comment added: "This is a comment description".')
(['comment', 'added', 'This is a comment description'], ['', ' ', ': "', '".'])
```

Parameters

- **patterns** (*str*) – Regular expression patterns. If multiple patterns given, the first matched pattern is used to Fix the part (other patterns are ignored).
- **fix_groups** (*str or list of str*) – Name groups in the patterns to fix. e.g., ["ipaddr", "port"] for Usecase 1. Unspecified groups are not fixed, so you can use other group names to other re functions like back references.
- **recursive** (*bool, optional*) – If True, the patterns will be searched recursively.
- **remove_groups** (*str or list of str, optional*) – Name groups in the patterns that should be considered as separators.
- **rest_remove** (*bool, optional*) – This option determines how to handle strings outside the fixed groups. e.g., 'comment added: "' and '".' in Usecase 2. Defaults to False, which means they are left as parts for further actions. In contrast, if True, they are considered as separators and will not be segmented or fixed further.

class log2seq.statement.**FixParenthesis**(*patterns*, *recursive=False*)

Extended FixPartial easily used to fix strings between parenthesis.

The basic usage is similar to FixPartial, but this class is designed especially for parenthesis, and the format of patterns is simpler. For example, FixParenthesis with pattern `['"', '"']` work samely as FixPartial with pattern `r'^.*?"(?P<fix>.+?)".*$'`.

Each pattern is a 2-length list of left and right parenthesis. The left and right pattern can consist of multiple characters, such as `["<!--", "-->"]`.

Example

```
>>> parser = StatementParser([FixParenthesis(['"', '"']), Split(' .:"')])
>>> parser.process_line('comment added: "This is a comment description".')
(['comment', 'added', 'This is a comment description'], ['', ' ', ': "', '".'])
```

Note: If a statement has multiple pairs of parenthesis, you need to add multiple FixParenthesis action to StatementParser actions. This is because FixParenthesis accept only one fix_group to extract in the action.

class log2seq.statement.**Remove**(*patterns*)

Add Separator flag to matched parts.

Separator parts will be ignored by following actions. Separator parts are selected by regular expression of given pattern (see re).

Parameters **patterns** (*str or list of str*) – Regular expression patterns. If multiple patterns are given, they are matched with every word in order.

class log2seq.statement.**RemovePartial**(*patterns*, *remove_groups*, *recursive=False*)
Extended Remove action to accept complicated patterns.

Usual *Remove* consider the matched part as a separator. In contrast, *RemovePartial* allow partially removing separators from a word matching with the given patterns.

Example

```
>>> rpattern = r'^.*([^:](?P<colon>:))$'
>>> fpattern = r'^\d{2}:\d{2}:\d{2}\.\d{3}$'
>>> rules = [Split(" "), RemovePartial(rpattern, remove_groups=["colon"]),␣
↪Fix(fpattern), Split(":")]
>>> parser = StatementParser(rules)
>>> parser.process_line("2000 Mar 4 12:34:56.789: message: duplicated header")
(['2000', 'Mar', '4', '12:34:56.789', 'duplicated', 'header'], ['', ' ', ' ', ' ',
↪ ': ', ' ', ''])
```

class log2seq.statement.**ConditionalSplit**(*patterns*, *separators*)
Split parts matching the given patterns by given separators.

Example

```
>>> parser = StatementParser([
>>>     Split(" ()"),
>>>     RemovePartial(r'^.*[^:](?P<colon>:)$', remove_groups=["colon"]),
>>>     ConditionalSplit(r'^%[A-Z]+-\d+(-[A-Z]+-\d+)?$', r'%-')
>>> ])
>>> parser.process_line("%KERNEL-4-EVENT-7: host h1-i2.example.org scored -0.035␣
↪value (20.0%)")
['KERNEL', '4', 'EVENT', '7', 'host', 'h1-i2.example.org', 'scored', '-0.035',
↪'value', '20.0%']
```

Parameters

- **patterns** (*str*) – Regular expression patterns. If multiple patterns given, this action will split parts matching at least one of them.
- **separators** (*str*) – separator symbol strings. If iterable, they imply joined and used all for segmentation. Escape sequence is internally added, so you don't need to add it manually.

2.4 Presets

2.4.1 log2seq.preset

log2seq.preset is a submodule to provide some settings for frequently used log formats.

log2seq.preset.**apache_errorlog_parser**()
Generate *LogParser* for Apache error logs in default format.

e.g., `[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client denied by server configuration: /export/home/live/ap/htdocs/test`

e.g., `[Fri Sep 09 10:42:29.902022 2011] [core:error] [pid 35708:tid 4328636416] [client 72.15.99.187] File does not exist: /usr/local/apache2/htdocs/favicon.ico`

Returns *LogParser*

Reference: Log Files - Apache HTTP Server Version 2.4: https://httpd.apache.org/docs/2.4/en/logs.html

log2seq.preset.**default**()
Generate *LogParser* of default settings.

It consists of *default_header_parsers()* and *default_statement_parser()*. *init_parser()* generates same instance without any arguments.

Returns *LogParser*

log2seq.preset.**default_header_parsers**()
Generate list of *HeaderParser* with default settings.

The default header parsers consists of 2 different rules.

- Rule 1 (designed for syslogd default format)
  - year (`Digit`, optional)
  - month (*MonthAbbreviation*)
  - day (*Digit*)
  - time (*Time*)
  - host (`String`)
  - statement (*Statement*)
- Rule 2 (designed for default asctime format of python logging)
  - date (*Date*)
  - time (*Time*)
  - host (`String`)
  - statement (*Statement*)

Returns list of *HeaderParser*

log2seq.preset.**default_statement_parser**()
Generate *StatementParser* with default settings.

The default parser consists of 4 step actions.

1. *Split* with standard symbols including white space and parenthesis
2. *FixIP* to fix IP addresses (including network address)
3. *Fix* with timestamps and MAC addresses
4. *Split* with `:`

Returns *StatementParser*

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

l

Index

U

V