Package 'tabulog'

May 8, 2026

Type Package

Title Parsing Semi-Structured Log Files into Tabular Format

Version 0.1.1

Author Austin Nar

Maintainer Austin Nar <austin.nar@gmail.com>

Description Convert semi-structured log files (such as 'Apache' access.log files) into a tabular format (data.frame) using a standard template system.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports yaml

Suggests lubridate, knitr, readr

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-08-09 13:00:02 UTC

Contents

`default_classes` *Default parser classes*

Description

List or parser classes provided 'out-of-the-box'. These can be used without further definition in any templates, or can be overridden.

Usage

```
default_classes(file = system.file("config/parser_classes.yml", package =
  "tabulog"), formatters = .default_formatters())
```

Arguments

`file`	Yaml file of parser classes to load. Defaults to included package file.
`formatters`	Named list of formatter functions to be associated with parsers. Default formatters are provided for default parser classes

Details

Parser classes are provided for the following

- ip: For matching ip addresses
- quote: For matching any string quoted by double-quotes
- url: For matching a standard http(s) url
- int: For matching any integer
- double: For matching any numeric value (including integers)

Value

A named list of the default parser classes provided "out of the box". Users should not need to use this in their code, and is mostly used for use in other internal functions. It is only visible to users so they can call it and see what classes are available by default.

Examples

```
default_classes()
```

format.parser *Encode for printing*

Description

Format a parser object for printing

Usage

```
## S3 method for class 'parser'
format(x, ...)
```

Arguments

x	parser to be formatted
...	other arguments to be passed to format.character

Examples

```
# No name, default formatter
format(parser('[0-9]+'))
# Custom name and formatter
format(parser('[0-9]+]', as.integer, name='int'))
```

formatter *Formatters*

Description

Get or set the formatter for a parser

Usage

```
formatter(x)

formatter(x) <- value
```

Arguments

x	parser
value	formatter function to be set

Value

The formatter attribute (should be a function) for the passed object (usually a parser object)

Examples

```
p <- parser('[0-9]+]')

# Default formatter
formatter(p)

# Set formatter
formatter(p) <- as.integer

# Custom formatter
formatter(p)
```

---

name *Parser Names*

Description

Get or set the name for a parser

Usage

```
name(x)

name(x) <- value
```

Arguments

x	parser
value	Name to be set

Value

The name attribute (should be a character) for the passed object (usually a `parser` object)

Examples

```
p <- parser('[0-9]+]')

# Default name (NULL)
name(p)

# Set name
name(p) <- 'int'

# Custom name
name(p)
```

parser *Parser Objects*

Description

Create or test for parser objects. These objects will be used by templates to identify a field within a log file.

Usage

```
parser(x, f, name = NULL)

is.parser(x)
```

Arguments

x	A regex string, a parser, or a list of either; Or object to be tested
f	A function to format the captured output, or a named list of such functions if x is a list
name	An optional name for the parser

Details

Parser objects contain 3 things:

1. A regex expression that matches the given field
2. A 'formatter'; a function that will in some way modify the captured text
    - By default, this the identity function
3. (Optional) A name for the parser

Value

`parser` and its S3 methods coerce `x` to a `parser` object, returning said parser object. `is.parser` returns TRUE or FALSE

Examples

```
# Captures integers
parser('[0-9]+')

# Captures integers, cast to integers
parser('[0-9]+', as.integer)

# List of parsers, all named (inferred from list names), some with parsers
parser(
  list(
    ip = '[0-9]{1,3}(\\.[0-9]{1,3}){3}',
```

```
    int = '[0-9]+',
    date = '[0-9]{4}\\-[0-9]{2}\\-[0-9]{2}'
  ),
  list(int = as.integer, date = as.Date)
)

is.parser(parser('[0-9]+')) #TRUE
is.parser(100)              #FALSE
```

`parse_logs` *Parse Log Files*

Description

Parse a log file with a provided template and a set of classes

Usage

```
parse_logs(text, template, classes = list(), ...)

parse_logs_file(text_file, config_file, formatters = list(), ...)
```

Arguments

`text`	Character vector; each element a log record
`template`	Template string
`classes`	A named list of parsers or regex strings for use within the template string
`...`	Other arguments passed onto `regexpr` for matching regular expressions.
`text_file`	Filename (or readable connection) containing log text
`config_file`	Filename (or readable connection) containing template file
`formatters`	Named list of formatter functions for use of formatting `classes`

Details

'`template` should only be a template string, such as 'ip ip_address [date access_date]...'.

`config_file` should be a yaml file or connection with the following fields

- template: Template String
- classes: Named list of regex strings for building classes

`text` should be a character vector, with each element representing a a log record

`text_file` should be a file or connection that can be split (with readLines) into a character vector of records

`classes` should be a named list of parser objects, where names match names of classes in template string, or a similarly named list of regex strings for coercing into parsers

`formatters` should be a named list of functions, where names match names of classes in template string, for properly formatting fields once they have been captured

Value

A data.frame with each field identified in the template string as a column. For each record in the passed text, the fields were extracted and formatted using the parser objects in `default_classes()` and `classes`.

Examples

```
# Template string with two fields
template <- '{{ip ipAddress}} - [{{date accessDate}}] {{int status }}'

# Two simple log records
logs <- c(
  '192.168.1.10 - [26/Jul/2019:11:41:10 -0500] 200',
  '192.168.1.11 - [26/Jul/2019:11:41:21 -0500] 404'
)

# A formatter for the date field
myFormatters <- list(date = function(x) lubridate::as_datetime(x, format = '%d/%b/%Y:%H:%M:%S %z'))
# A parser class for the date field
date_parser <- parser(
  '[0-3][0-9]\\/[A-Z][a-z]{2}\\/[0-9]{4}:[0-9]{2}:[0-9]{2}:[0-9]{2}[ ][\\+|\\-][0-9]{4}',
  myFormatters$date,
  'date'
)

# Parse the logs from raw data
parse_logs(logs, template, list(date=date_parser))

# Write the logs and to file and parse
logfile <- tempfile()
templatefile <- tempfile()
writeLines(logs, logfile)
yaml::write_yaml(list(template=template, classes=list(date=date_parser)), templatefile)
parse_logs_file(logfile, templatefile, myFormatters)
file.remove(logfile)
file.remove(templatefile)
```

`print.parser` *Print*

Description

Print a `parser` object. Underlying method uses `cat`.

Usage

```
## S3 method for class 'parser'
print(x, ...)
```

Arguments

x	parser to be printed
...	Other arguments; ignored

Value

x, invisibly

Examples

```
# No name, default formatter
print(parser('[0-9]+'))

#Custom name and formatter
print(parser('[0-9]+]', as.integer, name='int'))
```

Index