Package: yum (via r-universe)

October 24, 2024

Title Utilities to Extract and Process 'YAML' Fragments

Version 0.1.0

Description Provides a number of functions to facilitate extracting information in 'YAML' fragments from one or multiple files, optionally structuring the information in a 'data.tree'. 'YAML' (recursive acronym for ``YAML ain't Markup Language'') is a convention for specifying structured data in a format that is both machine- and human-readable. 'YAML' therefore lends itself well for embedding (meta)data in plain text files, such as Markdown files. This principle is implemented in 'yum' with minimal dependencies (i.e. only the 'yaml' packages, and the 'data.tree' package can be used to enable additional functionality).

License GPL-3

Encoding UTF-8

URL https://r-packages.gitlab.io/yum

BugReports https://gitlab.com/r-packages/yum/-/issues

RoxygenNote 7.1.1

Depends R (>= 3.0.0)

Imports yaml (>= 2.2)

Suggests covr, data.tree (>= 0.7), here, testthat

NeedsCompilation no

Author Gjalt-Jorn Peters [aut, cre]

Maintainer Gjalt-Jorn Peters <gjalt-jorn@userfriendlyscience.com>

Repository CRAN

Date/Publication 2021-07-16 19:20:03 UTC

Contents

`build_tree` *Convert the objects loaded from YAML fragments into a tree*

Description

If the data.tree::data.tree package is installed, this function can be used to convert a list of objects, as loaded from extracted YAML fragments, into a `data.tree::Node()`.

Usage

```
build_tree(
  x,
  idName = "id",
  parentIdName = "parentId",
  childrenName = "children",
  autofill = c(label = "id"),
  rankdir = "LR",
  directed = "false",
  silent = TRUE
)
```

Arguments

`x`	Either a list of YAML fragments loaded from a file with `load_yaml_fragments()`, or a list of such lists loaded from all files in a directory with `load_yaml_dir()`.
`idName`	The name of the field containing each elements' identifier, used to build the data tree when there are references to a parent from a child element.
`parentIdName`	The name of the field containing references to an element's parent element (i.e. the field containing the identifier of the corresponding parent element).

childrenName	The name of the field containing an element's children, either as a list of elements, or using the 'shorthand' notation, in which case a vector is supplied with the identifiers of the children.
autofill	A named vector where the names represent fields to fill with the values of the fields specified in the vector values. Note that autofill replacements are only applied if the fields to be autofilled (i.e. the names of the vector specified in autofill) do not already have a value.
rankdir	How to plot the plot when it's plotted: the default "LR" plots from left to right. Specify e.g. "TB" to plot from top to bottom.
directed	Whether the edges should have arrows ("forward" or "backward") or not ("false").
silent	Whether to provide (FALSE) or suppress (TRUE) more detailed progress updates.

Value

a data.tree::Node() object.

Examples

```
loadedYum <- yum::load_yaml_fragments(text=c(
"---",
"-",
"  id: firstFragment",
"---",
"Outside of YAML",
"---",
"-",
"  id: secondFragment",
"  parentId: firstFragment",
"---",
"Also outside of YAML"));
yum::build_tree(loadedYum);
```

---

delete_yaml_fragments *Delete all YAML fragments from a file*

Description

These function deletes all YAML fragments from a file, returning a character vector without the lines that specified the YAML fragments.

Usage

```
delete_yaml_fragments(
  file,
  text,
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  silent = TRUE
)
```

Arguments

`file`	The path to a file to scan; if provided, takes precedence over `text`.
`text`	A character vector to scan, where every element should represent one line in the file; can be specified instead of `file`.
`delimiterRegEx`	The regular expression used to locate YAML fragments.
`ignoreOddDelimiters`	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
`silent`	Whether to be silent (TRUE) or informative (FALSE).

Value

A list of character vectors.

Examples

```
yum::delete_yaml_fragments(text=c("---", "First YAML fragment", "---",
                                  "Outside of YAML",
                                  "---", "Second fragment", "---",
                                  "Also outside of YAML"));
```

`extract_yaml_dir` *Extract all YAML fragments from all files in a directory*

Description

These function extracts all YAML fragments from all files in a directory returning a list of character vectors containing the extracted fragments.

Usage

```
extract_yaml_dir(
  path,
  recursive = TRUE,
  fileRegexes = c("^[^\\.]+.*$"),
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)
```

Arguments

path	The path containing the files.
recursive	Whether to also process subdirectories (TRUE) or not (FALSE).
fileRegexes	A vector of regular expressions to match the files against: only files matching one or more regular expressions in this vector are processed. The default regex (^[^\.]+.*$) matches all files except those that start with a period (.).
delimiterRegEx	The regular expression used to locate YAML fragments.
ignoreOddDelimiters	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
encoding	The encoding to use when calling readLines(). Set to NULL to let readLines() guess.
silent	Whether to be silent (TRUE) or informative (FALSE).

Value

A list of character vectors.

Examples

```
### First get the directory where 'yum' is installed
yumDir <- system.file(package="yum");
### Specify the path of some example files
examplePath <- file.path(yumDir, "extdata");
### Show files (should be three .dct files)
list.files(examplePath);
### Load these files
yum::extract_yaml_dir(path=examplePath);
```

---

extract_yaml_fragments

Extract all YAML fragments from a file

Description

These function extracts all YAML fragments from a file, returning a list of character vectors containing the extracted fragments.

Usage

```
extract_yaml_fragments(
  text,
  file,
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)
```

Arguments

text, file	As text or file, you can specify a file to read with encoding encoding, which will then be read using base::readLines(). If the argument is named text, whether it is the path to an existing file is checked first, and if it is, that file is read. If the argument is named file, and it does not point to an existing file, an error is produced (useful if calling from other functions). A text should be a character vector where every element is a line of the original source (like provided by base::readLines()); although if a character vector of one element *and* including at least one newline character (\\n) is provided as text, it is split at the newline characters using base::strsplit(). Basically, this behavior means that the first argument can be either a character vector or the path to a file; and if you're specifying a file and you want to be certain that an error is thrown if it doesn't exist, make sure to name it file.
delimiterRegEx	The regular expression used to locate YAML fragments.
ignoreOddDelimiters	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
encoding	The encoding to use when calling readLines(). Set to NULL to let readLines() guess.
silent	Whether to be silent (TRUE) or informative (FALSE).

Value

A list of character vectors, where each vector corresponds to one YAML fragment in the source file or text.

Examples

```
extract_yaml_fragments(text="
---
First: YAML fragment
  id: firstFragment
---
Outside of YAML
---
Second: YAML fragment
  id: secondFragment
  parentId: firstFragment
---
Also outside of YAML
");
```

---

find_yaml_fragment_indices

Find the indices ('line numbers') of all YAML fragments from a file

---

Description

These function finds all YAML fragments from a file, returning their start and end indices or all indices of all lines in the (non-)YAML fragments.

Usage

```
find_yaml_fragment_indices(
  file,
  text,
  invert = FALSE,
  returnFragmentIndices = TRUE,
  returnPairedIndices = TRUE,
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  silent = TRUE
)
```

Arguments

`file`	The path to a file to scan; if provided, takes precedence over `text`.
`text`	A character vector to scan, where every element should represent one line in the file; can be specified instead of `file`.
`invert`	Set to `TRUE` to return the indices of the character vector that are *not* YAML fragments.
`returnFragmentIndices`	Set to `TRUE` to return all indices of the relevant fragments (i.e. including intermediate indices).
`returnPairedIndices`	Whether to return two vectors with the start and end indices, or pair them up in vectors of 2.
`delimiterRegEx`	The regular expression used to locate YAML fragments.
`ignoreOddDelimiters`	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
`silent`	Whether to be silent (TRUE) or informative (FALSE).

Value

A list of numeric vectors with start and end indices

Examples

```
### Create simple text vector with the right delimiters
simpleExampleText <-
  c(
    "---",
    "First YAML fragment",
```

```
    "---",
    "Outside of YAML",
    "This, too.",
    "---",
    "Second fragment",
    "---",
    "Also outside of YAML",
    "Another one outside",
    "Last one"
  );

yum::find_yaml_fragment_indices(
  text=simpleExampleText
);

yum::find_yaml_fragment_indices(
  text=simpleExampleText,
  returnFragmentIndices = FALSE
);

yum::find_yaml_fragment_indices(
  text=simpleExampleText,
  invert = TRUE
);
```

`flatten_list_of_lists` *Flatten a list of lists to a list of atomic vectors*

Description

This function takes a hierarchical structure of lists and extracts all atomic vectors, returning one flat list of all those vectors.

Usage

```
flatten_list_of_lists(x)
```

Arguments

`x`	The list of lists.

Value

A list of atomic vectors.

Examples

```
### First create a list of lists
listOfLists <-
  list(list(list(1:3, 8:5), 7:7), list(1:4, 8:2));
yum::flatten_list_of_lists(listOfLists);
```

`is.odd` *Checking whether numbers are odd or even*

Description

Checking whether numbers are odd or even

Usage

```
is.odd(vector)

is.even(vector)
```

Arguments

`vector`	The vector to process

Value

A logical vector.

Examples

```
is.odd(4);
```

`load_and_simplify` *Load YAML fragments in one or multiple files and simplify them*

Description

These function extracts all YAML fragments from a file or text (`load_and_simplify`) or from all files in a directory (`load_and_simplify_dir`) and loads them by calling `load_yaml_fragments()`, and then calls `simplify_by_flattening()`, on the result, returning the resulting list.

Usage

```
load_and_simplify(
  text,
  file,
  yamlFragments = NULL,
  select = ".*",
  simplify = ".*",
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)

load_and_simplify_dir(
  path,
  recursive = TRUE,
  fileRegexes = c("^[^\\.]+.*$"),
  select = ".*",
  simplify = ".*",
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)
```

Arguments

`text`	As `text` or `file`, you can specify a `file` to read with encoding `encoding`, which will then be read using `base::readLines()`. If the argument is named `text`, whether it is the path to an existing file is checked first, and if it is, that file is read. If the argument is named `file`, and it does not point to an existing file, an error is produced (useful if calling from other functions). A `text` should be a character vector where every element is a line of the original source (like provided by `base::readLines()`); although if a character vector of one element *and* including at least one newline character (`\\n`) is provided as `text`, it is split at the newline characters using `base::strsplit()`. Basically, this behavior means that the first argument can be either a character vector or the path to a file; and if you're specifying a file and you want to be certain that an error is thrown if it doesn't exist, make sure to name it `file`.
`file`	As `text` or `file`, you can specify a `file` to read with encoding `encoding`, which will then be read using `base::readLines()`. If the argument is named `text`, whether it is the path to an existing file is checked first, and if it is, that file is read. If the argument is named `file`, and it does not point to an existing file, an error is produced (useful if calling from other functions). A `text` should be a character vector where every element is a line of the original source (like provided by `base::readLines()`); although if a character vector of one element *and* including at least one newline character (`\\n`) is provided as `text`, it is split at the newline characters using `base::strsplit()`. Basically, this behavior

means that the first argument can be either a character vector or the path to a file; and if you're specifying a file and you want to be certain that an error is thrown if it doesn't exist, make sure to name it `file`.

`yamlFragments` A character vector of class `yamlFragment` where every element corresponds to one line of the YAML fragments, or a list of multiple such character vectors (of class `yamlFragments`). Specify either `yamlFragments` (which, if specified, takes precedence over `file` and `text`), `file`, or `text` (`file` takes precedence over `text`).

`select` A vector of regular expressions specifying object names to retain. The default (`.*`) matches everything, so by default, all objects are retained.

`simplify` A regular expression specifying which elements to simplify (default is everything)

`delimiterRegEx` The regular expression used to locate YAML fragments.

`ignoreOddDelimiters`
Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.

`encoding` The encoding to use when calling `readLines()`. Set to NULL to let `readLines()` guess.

`silent` Whether to be silent (`TRUE`) or informative (`FALSE`).

`path` The path containing the files.

`recursive` Whether to also process subdirectories (`TRUE`) or not (`FALSE`).

`fileRegexes` A vector of regular expressions to match the files against: only files matching one or more regular expressions in this vector are processed. The default regex (`^[^\.]+.*$`) matches all files except those that start with a period (`.`).

Value

A list of objects, where each object corresponds to one item specified in the read YAML fragment(s) from the source file or text. If the convention of the `rock`, `dct` and `justifier` packages is followed, each object in this list contains one or more named objects (lists), where the name indicates the type of information contained. Each of those objects (lists) then contains one or more objects of that type, such as metadata or codes for `rock`, a decentralized construct taxonomy element for `dct`, and a justification, decision, assertion, or source for `justifier`.

Examples

```
yum::load_and_simplify(text="
---
firstObject:
  id: firstFragment
---
Outside of YAML
---
otherObjectType:
  -
    id: secondFragment
    parentId: firstFragment
```

```
  -
    id: thirdFragment
    parentId: firstFragment
---
Also outside of YAML");
```

---

load_yaml_dir *Load all YAML fragments from all files in a directory*

Description

These function extracts all YAML fragments from all files in a directory returning a list of character vectors containing the extracted fragments.

Usage

```
load_yaml_dir(
  path,
  recursive = TRUE,
  fileRegexes = c("^[^\\.]+.*$"),
  select = ".*",
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)
```

Arguments

path	The path containing the files.
recursive	Whether to also process subdirectories (TRUE) or not (FALSE).
fileRegexes	A vector of regular expressions to match the files against: only files matching one or more regular expressions in this vector are processed. The default regex (^[^\.]+.*$) matches all files except those that start with a period (.).
select	A vector of regular expressions specifying object names to retain. The default (.*) matches everything, so by default, all objects are retained.
delimiterRegEx	The regular expression used to locate YAML fragments.
ignoreOddDelimiters	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
encoding	The encoding to use when calling readLines(). Set to NULL to let readLines() guess.
silent	Whether to be silent (TRUE) or informative (FALSE).

Details

These function extracts all YAML fragments from all files in a directory and then calls `yaml::yaml.load()` to parse them. It then returns a list where each element is a list with the parsed fragments in a file.

Value

A list of lists of objects.

Examples

```
### First get the directory where 'yum' is installed
yumDir <- system.file(package="yum");
### Specify the path of some example files
examplePath <- file.path(yumDir, "extdata");
### Show files (should be three .dct files)
list.files(examplePath);
### Load these files
yum::load_yaml_dir(path=examplePath);
```

---

load_yaml_fragments *Load all YAML fragments from a file*

Description

These function extracts all YAML fragments from a file and then calls `yaml::yaml.load()` to parse them. It then returns a list of the parsed fragments.

Usage

```
load_yaml_fragments(
  text,
  file,
  yamlFragments = NULL,
  select = ".*",
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)
```

Arguments

`text`	As `text` or `file`, you can specify a `file` to read with encoding `encoding`, which will then be read using `base::readLines()`. If the argument is named `text`, whether it is the path to an existing file is checked first, and if it is, that file is read. If the argument is named `file`, and it does not point to an existing file, an error is produced (useful if calling from other functions). A `text` should be a character vector where every element is a line of the original source (like

provided by base::readLines()); although if a character vector of one element *and* including at least one newline character (\\n) is provided as text, it is split at the newline characters using base::strsplit(). Basically, this behavior means that the first argument can be either a character vector or the path to a file; and if you're specifying a file and you want to be certain that an error is thrown if it doesn't exist, make sure to name it file.

file — As text or file, you can specify a file to read with encoding encoding, which will then be read using base::readLines(). If the argument is named text, whether it is the path to an existing file is checked first, and if it is, that file is read. If the argument is named file, and it does not point to an existing file, an error is produced (useful if calling from other functions). A text should be a character vector where every element is a line of the original source (like provided by base::readLines()); although if a character vector of one element *and* including at least one newline character (\\n) is provided as text, it is split at the newline characters using base::strsplit(). Basically, this behavior means that the first argument can be either a character vector or the path to a file; and if you're specifying a file and you want to be certain that an error is thrown if it doesn't exist, make sure to name it file.

yamlFragments — A character vector of class yamlFragment where every element corresponds to one line of the YAML fragments, or a list of multiple such character vectors (of class yamlFragments). Specify either yamlFragments (which, if specified, takes precedence over file and text), file, or text (file takes precedence over text).

select — A vector of regular expressions specifying object names to retain. The default (.*) matches everything, so by default, all objects are retained.

delimiterRegEx — The regular expression used to locate YAML fragments.

ignoreOddDelimiters — Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.

encoding — The encoding to use when calling readLines(). Set to NULL to let readLines() guess.

silent — Whether to be silent (TRUE) or informative (FALSE).

Value

A list of objects, where each object corresponds to one YAML fragment from the source file or text. If the convention of the rock, dct and justifier packages is followed, each object in this list contains one or more named objects (lists), where the name indicated the type of information contained. Each of those objects (lists) then contains one or more objects of that type, such as metadata or codes for rock, a decentralized construct taxonomy element for dct, and a justification for justifier.

Examples

```
yum::load_yaml_fragments(text="
---
-
```

```
  id: firstFragment
---
Outside of YAML
---
-
  id: secondFragment
  parentId: firstFragment
---
Also outside of YAML");
```

---

load_yaml_list *Load all YAML fragments from all character vectors in a list*

Description

These function extracts all YAML fragments from character vectors in a list, returning a list of character vectors containing the extracted fragments.

Usage

```
load_yaml_list(
  x,
  recursive = TRUE,
  select = ".*",
  delimiterRegEx = "^---$",
  ignoreOddDelimiters = FALSE,
  encoding = "UTF-8",
  silent = TRUE
)
```

Arguments

x	The list containing the character vectors.
recursive	Whether to first unlist the list (TRUE) or not (FALSE).
select	A vector of regular expressions specifying object names to retain. The default (.*) matches everything, so by default, all objects are retained.
delimiterRegEx	The regular expression used to locate YAML fragments.
ignoreOddDelimiters	Whether to throw an error (FALSE) or delete the last delimiter (TRUE) if an odd number of delimiters is encountered.
encoding	The encoding to use when calling readLines(). Set to NULL to let readLines() guess.
silent	Whether to be silent (TRUE) or informative (FALSE).

Details

This function calls `yaml::yaml.load()` on all character vectors in a list. It then returns a list where each element is a list with the parsed fragments in a file.

Value

A list of lists of objects.

Examples

```
yamlList <- list(c(
"---",
"-",
"  id: firstFragment",
"---"), c(
"---",
"-",
"  id: secondFragment",
"  parentId: firstFragment",
"---"));
yum::load_yaml_list(yamlList);
```

`simplify_by_flattening` *Simplify the structure of extracted YAML fragments*

Description

This function does some cleaning and simplifying to allow efficient specification of elements in the YAML fragments.

Usage

```
simplify_by_flattening(x, simplify = ".*", .level = 1)
```

Arguments

`x`	Extracted (and loaded) YAML fragments
`simplify`	A regular expression specifying which elements to simplify (default is everything)
`.level`	Internal argument to enable slightly-less-than-elegant 'recursion'.

Value

A simplified list (but still a list)

Examples

```
yamlFragmentExample <- '
---
source:
  -
    id: src_1
    label: "Label 1"
  -
    id: src_2
    label: "Label 2"
assertion:
  -
    id: assertion_1
    label: "Assertion 1"
  -
    id: assertion_2
    label: "Assertion 2"
---
';
loadedExampleFragments <-
  load_yaml_fragments(yamlFragmentExample);
simplified <-
  simplify_by_flattening(loadedExampleFragments);

### Pre simmplification:
str(loadedExampleFragments);

### Post simmplification:
str(simplified);
```

---

vecTxt *Easily parse a vector into a character value*

Description

Easily parse a vector into a character value

Usage

```
vecTxt(
  vector,
  delimiter = ", ",
  useQuote = "",
  firstDelimiter = NULL,
  lastDelimiter = " & ",
  firstElements = 0,
  lastElements = 1,
  lastHasPrecedence = TRUE
```

```
)

vecTxtQ(vector, useQuote = "'", ...)
```

Arguments

`vector`	The vector to process.
`delimiter`, `firstDelimiter`, `lastDelimiter`	The delimiters to use for respectively the middle, first `firstElements`, and last `lastElements` elements.
`useQuote`	This character string is pre- and appended to all elements; so use this to quote all elements (`useQuote="'"`), doublequote all elements (`useQuote='"'`), or anything else (e.g. `useQuote='|'`). The only difference between `vecTxt` and `vecTxtQ` is that the latter by default quotes the elements.
`firstElements`, `lastElements`	The number of elements for which to use the first respective last delimiters
`lastHasPrecedence`	If the vector is very short, it's possible that the sum of firstElements and lastElements is larger than the vector length. In that case, downwardly adjust the number of elements to separate with the first delimiter (`TRUE`) or the number of elements to separate with the last delimiter (`FALSE`)?
`...`	Any addition arguments to `vecTxtQ` are passed on to `vecTxt`.

Value

A character vector of length 1.

Examples

```
vecTxtQ(names(mtcars));
```

Index