# Package: heddlr (via r-universe)

September 2, 2024

**Title** Dynamic R Markdown Document Generation

**Version** 0.6.1

**Description** Helper functions designed to make dynamically generating R Markdown documents easier by providing a simple and tidy way to create report pieces, shape them to your data, and combine them for exporting into a single R Markdown document.

**License** MIT + file LICENSE

**URL** https://github.com/mikemahoney218/heddlr, https://mikemahoney218.github.io/heddlr/

**BugReports** https://github.com/mikemahoney218/heddlr/issues

**Depends** R (>= 3.5.0)

**Imports** rlang (>= 0.1.2), utf8, yaml

**Suggests** covr, here, rmarkdown, roxygen2, testthat (>= 2.1.0), dplyr, tibble, tidyr (>= 1.0.0), nycflights13, ggplot2, knitr, purrr

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**VignetteBuilder** knitr

**Repository** https://mikemahoney218.r-universe.dev

**RemoteUrl** https://github.com/mikemahoney218/heddlr

**RemoteRef** HEAD

**RemoteSha** 2b05b946bf1b4c41bcf022896ece23402e0ddeb9

## Contents

---

`bulk_replace` *Bulk replace a pattern throughout multiple files.*

---

## Description

This function makes it easier to change a specific text string throughout a number of files, allowing you to ensure you're correcting all areas of your code at once.

## Usage

```
bulk_replace(files, pattern, replacement, dry.run = TRUE)
```

## Arguments

| | |
|---|---|
| `files` | A vector of filepaths to replace strings in. |
| `pattern` | The character string to be replaced. |
| `replacement` | A character string to replace all patterns with. |
| `dry.run` | Logical – describe the file changes that will be made (`TRUE`) or make them in the specified files (`FALSE`)? |

## Examples

```
library(heddlr)
temp_file <- tempfile("test", fileext = ".Rmd")
temp_patt <- "#"
export_template(temp_patt, temp_file)
bulk_replace(c(temp_file), "#", "##")
bulk_replace(c(temp_file), "#", "##", dry.run = FALSE)
```

create_yaml_header *Convert list objects into R Markdown YAML headers*

## Description

This function tweaks the behavior of `as.yaml` to return a string which can immediately be used as an R Markdown YAML header. It's designed to accept both deeply nested lists and simpler list formats to make reasoning about your header easier.

## Usage

```
create_yaml_header(
  ...,
  line.sep = c("\n", "\r\n", "\r"),
  indent = 2,
  unicode = TRUE,
  indent.mapping.sequence = FALSE,
  handlers = NULL
)
```

## Arguments

| | |
|---|---|
| `...` | A set of objects that will be combined into the YAML header. Objects may be provided as lists (the structure list("outputs" = "html_document") translates to outputs: html_document) or as single-item named vectors (passing "title" = "My Report" to ... will translate to title: "My Report"). |
| `line.sep`, `indent`, `unicode`, `indent.mapping.sequence`, `handlers` | Additional arguments to be passed to `as.yaml` |

## Value

Returns a string formatted for use as an R Markdown YAML header.

## See Also

Other manipulation functions: `heddle()`, `make_template()`, `provide_parameters()`, `use_parameters()`

## Examples

```
headerContent <- list(
  "title" = "Testing YAML",
  "author" = "Mike Mahoney",
  "output" = list(
    "flexdashboard::flex_dashboard" = list(
      "vertical_layout" = "fill",
      "orientation" = "rows",
      "css" = "bootstrap.css"
    )
```

```
  )
)
create_yaml_header(headerContent)
create_yaml_header(
  "title" = "testing",
  "params" = list("data" = "NA"),
  list("author" = "Mike Mahoney")
)
```

---

`export_template` *Safely export templates to file.*

---

## Description

This is a simple wrapper function around `as_utf8` and `writeLines`, letting users write their template strings to file without having to worry about file encodings. For more details on why UTF-8 encoding is necessary, check out Yihui Xie's post on the subject.

## Usage

```
export_template(
  template,
  filename,
  sep = "",
  filename.is.string = TRUE,
  strip.carriage.returns = TRUE
)
```

## Arguments

| | |
|---|---|
| `template` | The template string to be written out |
| `filename` | The path to write the template to, passed to `writeLines`. Also accepts `stdout` (and likely other similar functions) with a warning. |
| `sep` | Separator to use between lines written, passed to `writeLines`. Defaults to no separator, as templates are generally already spaced appropriately. |
| `filename.is.string` | A logical value indicating whether or not the filename parameter is expected to be a string (that is, a character vector). Setting the value to FALSE disables the warning when a non-character argument is passed, but this is unsupported functionality. |
| `strip.carriage.returns` | A logical value indicating whether or not to strip carriage feed characters, should any exist. This preserves line spacing when writing out files originally written on Windows; otherwise `writeLines` appears to not recognize lines as ending with a newline and inserts one, resulting in 2x the number of line breaks as anticipated. |

### Details

Note that this function is effectively the inverse of `import_pattern` – `export_template(import_pattern("out.txt"), "out.txt")` should always result in an unchanged file, and exceptions to this rule would be considered bugs.

### Value

Returns the input template invisibly.

### Examples

```
pattern_file <- tempfile("out", tempdir(), ".Rmd")
export_template("my sample pattern", pattern_file)
```

---

## extract_draft *Extract multiple patterns into a single draft object*

---

### Description

When working with multiple patterns that will be woven into a template, it makes sense to have all patterns stored in a central object. This function creates that object from a named vector of filenames to be used in further generation, importing the files via `extract_pattern`.

### Usage

```
extract_draft(filepath, ...)
```

### Arguments

| | |
|---|---|
| `filepath` | A valid character string to the plaintext file containing the pattern. |
| `...` | Keywords to be used by `extract_pattern` to extract each pattern. If arguments to ... are named, the returned draft will have the same names. |

### Value

Returns a list (the same length as ...) containing the extracted patterns.

### See Also

Other import functions: `extract_pattern()`, `import_draft()`, `import_pattern()`

### Examples

```
pattern_file <- tempfile("out", tempdir(), ".Rmd")
export_template("EXTRACT my sample pattern EXTRACT", pattern_file)
extract_draft(pattern_file, "one" = "EXTRACT")
```

## extract_pattern *Extract patterns from larger documents*

### Description

This function loads a file and scans it for a given keyword which signposts the beginning and end of a pattern. It then extracts all the text between the keywords for manipulation as a pattern. For extracting multiple patterns at once from a single file, check out `extract_draft`.

### Usage

```
extract_pattern(filepath, keyword, preserve = FALSE)
```

### Arguments

| | |
|---|---|
| `filepath` | A valid character string to the plaintext file containing the pattern. |
| `keyword` | A placeholder which signposts the beginning and end of the pattern to be extracted. |
| `preserve` | A boolean (`TRUE`/`FALSE`) value indicating whether or not keywords should be included in the extracted pattern (`TRUE`) or not (`FALSE`); default `FALSE`. |

### Value

A character string, typically used to assemble a draft.

### See Also

Other import functions: `extract_draft()`, `import_draft()`, `import_pattern()`

### Examples

```
pattern_file <- tempfile("out", tempdir(), ".Rmd")
export_template("EXTRACT my sample pattern EXTRACT", pattern_file)
extract_pattern(pattern_file, "EXTRACT")
```

## heddle *Transform pattern objects into template pieces*

### Description

This function replicates pattern objects, replacing placeholder keywords in each iteration with values from the provided data. This allows efficiently creating R Markdown documents with many repeating pieces which may shift alongside the underlying data.

## Usage

```
heddle(data, pattern, ..., strip.whitespace = FALSE)
```

## Arguments

| | |
|---|---|
| `data` | Input dataframe to pull replacement values from. Accepts either vector or dataframe inputs. |
| `pattern` | The base pattern, as either an atomic vector (which will be recycled for every value in your data) or a vector of the same length as your data (which will be applied element-wise to your data, so that `data[[1]]` will replace `pattern[[1]]`). |
| `...` | Values indicating what placeholders in the pattern should be replaced – see "Specifying replacement values" below for more. |
| `strip.whitespace` | A boolean (TRUE/FALSE) value indicating if whitespace should be removed from the replacement variable. Toggle this on if you're using the variable in chunk labels or similar places. |

## Value

Returns a character vector of the pattern with placeholders replaced by variables.

## Specifying replacement values

`heddle` can accept multiple different values for `...`, depending on how you call it.

If `data` is a vector (which is the case when either calling `heddle` on a vector directly, or using it in a `mutate`) call, `...` should be unnamed strings matching the values to be replaced. If any argument passed to `...` isn't found in the pattern, a warning will be raised – use `NA` to replicate patterns without replacing any values.

If `data` is a dataframe (which is the case both when calling `heddle` on a dataframe directly or using it in combination with `nest` and `map`), `...` should be a set of name = variable pairs, with the name matching the keyword to be replaced by that variable. Names should be quoted, variable names don't need to be. As with vectors, if any argument passed to `...` isn't found in the pattern, a warning will be raised.

## See Also

Other manipulation functions: `create_yaml_header()`, `make_template()`, `provide_parameters()`, `use_parameters()`

## Examples

```
# When passed a vector, heddle replaces all placeholders passed to ...
# with each value
spList <- unique(iris$Species)
heddle(spList, "SPECIES CODE GWAR ", "GWAR")
heddle(spList, "SPECIES CODE GWAR ", "GWAR", "CODE")
heddle("test string", "pattern tk", "tk", strip.whitespace = TRUE)
```

```
# When passed a dataframe, heddle uses "Name" = Variable syntax to determine
# which values should replace which placeholders
spList <- data.frame(Species = c(unique(iris$Species), "test string"))
heddle(spList, "SPECIES CODE GWAR ", "GWAR" = Species)
heddle(spList, "SPECIES CODE GWAR ", "GWAR" = Species, "CODE" = Species)
```

---

`import_draft` *Import multiple patterns into a single draft object*

---

## Description

When working with multiple patterns that will be woven into a template, it makes sense to have all patterns stored in a central object. This function creates that object from a named vector of filenames to be used in further generation, importing the files via `import_pattern`.

## Usage

```
import_draft(...)
```

## Arguments

| | |
|---|---|
| ... | A named vector of filenames which will be imported as patterns stored in the returned draft, with the names used as indices. Files should be plain text. |

## Value

Returns a list (the same length as ...) containing the imported patterns.

## See Also

Other import functions: `extract_draft()`, `extract_pattern()`, `import_pattern()`

## Examples

```
pattern_file <- tempfile("out", tempdir(), ".Rmd")
export_template("my sample pattern", pattern_file)
import_draft("sample_pattern" = pattern_file)
```

## import_pattern *Quickly import plaintext files.*

### Description

Longer, multi-chunk patterns can benefit from being developed in files independent of the rest of a draft. This is a quick wrapper function to import those patterns as objects for assembly into a draft.

### Usage

```
import_pattern(filepath)
```

### Arguments

| | |
|---|---|
| filepath | A valid character string to the plaintext file containing the pattern. |

### Value

A character string, typically used to assemble a draft.

### See Also

Other import functions: extract_draft(), extract_pattern(), import_draft()

### Examples

```
pattern_file <- tempfile("out", tempdir(), ".Rmd")
export_template("my sample pattern", pattern_file)
import_pattern(pattern_file)
```

## make_template *Linearly combine template elements into templates*

### Description

Applying heddle can leave your template pieces stored as either string objects, vectors (standalone or in a dataframe), or nested vectors (if applied using map()). This function takes those elements and combines them into a single exportable template.

### Usage

```
make_template(data, ...)
```

### Arguments

| | |
|---|---|
| data | The dataframe containing variables to be combined. |
| ... | The variables to be combined into a template object. |

## Value

Returns the collapsed template as a character string.

## See Also

Other manipulation functions: `create_yaml_header()`, `heddle()`, `provide_parameters()`, `use_parameters()`

## Examples

```
# When passed vectors, make_template flattens each vector into a single
# string and then combines its arguments from left to right
spList <- data.frame(Species = c(unique(iris$Species), "test string"))
make_template(
  heddle(spList, "SPECIES CODE GWAR ", "GWAR" = Species),
  heddle(spList, "SPECIES CODE GWAR ", "GWAR" = Species)
)

# When passed variables in a dataframe, make_template collapses each column
# in turn, then combines the output strings from left to right
spList <- data.frame(Species = c(unique(iris$Species), "test string"))
spList$template <- heddle(spList, "SPECIES CODE GWAR ", "GWAR" = Species)
make_template(spList, template)
make_template(spList, template, template)

# When passed nested columns, heddlr collapses each cell into a string,
# then collapses each column into a string, and then combines the outputs
# from left to right
make_template(tidyr::nest(spList, nested = template), nested)
```

---

# provide_parameters *Easily provide parameters to R Markdown render calls*

---

## Description

R Markdown documents allow you to pass almost any object – including large data frames and functions – to the document as parameters, letting you only define them once to use them in both your document generator and the generated document. This function makes it slightly easier to do so, by automatically creating a named list from provided objects rather than requiring a named list. This function is a stripped-down variant of [tibble::lst].

## Usage

```
provide_parameters(...)
```

## Arguments

| | |
|---|---|
| `...` | Objects to be included as parameters. Objects should be unquoted and exist in the current session environment. |

## See Also

Other manipulation functions: `create_yaml_header()`, `heddle()`, `make_template()`, `use_parameters()`

## Examples

```
template <- make_template(
  "---\ntitle: Example\noutput: html_document\n---\n",
  "\nThe random number is `r random_number`.\n"
)
template <- use_parameters(template, "random_number")
pattern_file <- tempfile("out", tempdir(), ".Rmd")
export_template(template, pattern_file)

random_number <- rnorm(1)
if (rmarkdown::pandoc_available()) {
  rmarkdown::render(pattern_file, params = provide_parameters(random_number))
}
```

---

# use_parameters *Automatically include session objects as report parameters*

---

## Description

R Markdown documents allow you to pass almost any object – including large data frames and functions – to the document as parameters, letting you only define them once to use them in both your document generator and the generated document. This function makes it slightly easier to do so, by adding your objects to the YAML header and then initializing them so you can use the same object names in your generated document as in your generator.

## Usage

```
use_parameters(template, ..., init.params = TRUE, is.file = FALSE)
```

## Arguments

| | |
|---|---|
| `template` | An atomic (`length(template) == 1`) character vector containing either the template to manipulate OR the path to the file storing the template, which will be imported via import_pattern. |
| `...` | Objects to be included as parameters. Objects should be unquoted and exist in the current session environment. This function currently will always assign parameters NA as a default value, and does not yet provide an option to override that. |
| `init.params` | A boolean (`TRUE/FALSE`) value indicating if a chunk initalizing the parameters (that is, assigning them via `object <- params$object`) should be included. Default `TRUE`. |
| `is.file` | A boolean value indicating if the template argument is a vector containing the template (`FALSE`, default) or the path to the template file (`TRUE`). |

## See Also

Other manipulation functions: `create_yaml_header()`, `heddle()`, `make_template()`, `provide_parameters()`

## Examples

```
template <- make_template("---\ntitle: Cool Report\noutput: html_document\n---\n")
use_parameters(template, data)
```

# Index