Package 'augere.core'

June 4, 2026

Version 0.99.3

Date 2026-04-11

Title Core Utilities for Automatically Generated Reports

Description Provides utility functions for automatic generation of reports in the augere framework. This mostly involves parsing and dynamically editing Rmarkdown files, controlling inputs into the augere pipelines.
Each pipeline generates and executes a self-contained Rmarkdown file for a common bioinformatics analysis;
see downstream packages like augere.de and augere.screen for examples of some analysis pipelines.

License MIT + file LICENSE

Imports utils, grDevices

Suggests testthat, knitr, rmarkdown, BiocStyle, limma, edgeR, airway, S4Vectors, SummarizedExperiment, jsonlite, alabaster.base

VignetteBuilder knitr

RoxygenNote 7.3.3

Encoding UTF-8

biocViews WorkflowManagement, ReportWriting

URL https://github.com/augere-bioinfo/augere.core

BugReports https://github.com/augere-bioinfo/augere.core/issues

git_url https://git.bioconductor.org/packages/augere.core

git_branch devel

git_last_commit 892e35c

git_last_commit_date 2026-05-14

Repository Bioconductor 3.24

Date/Publication 2026-06-04

Author Aaron Lun [cre, aut] (ORCID: <https://orcid.org/0000-0002-3564-4813>)

Maintainer Aaron Lun <infinite.monkeys.with.keyboards@gmail.com>

Contents

`augere.core-package` *augere.core: Core Utilities for Automatically Generated Reports*

Description

Provides utility functions for automatic generation of reports in the augere framework. This mostly involves parsing and dynamically editing Rmarkdown files, controlling inputs into the augere pipelines. Each pipeline generates and executes a self-contained Rmarkdown file for a common bioinformatics analysis; see downstream packages like augere.de and augere.screen for examples of some analysis pipelines.

Author(s)

Maintainer: Aaron Lun <infinite.monkeys.with.keyboards@gmail.com> (ORCID)

See Also

Useful links:

- https://github.com/augere-bioinfo/augere.core
- Report bugs at https://github.com/augere-bioinfo/augere.core/issues

---

`compileReport` *Compile an Rmarkdown report*

Description

Compile a Rmarkdown report, typically generated from a template in a pipeline function.

Usage

```
compileReport(
  file,
  env,
  skip.chunks = NULL,
  contents = NULL,
  suppress.plots = FALSE
)
```

Arguments

file	String containing a path to an Rmarkdown file.
env	Environment in which the R code is to be evaluated.
skip.chunks	Character vector of the names of Rmarkdown chunks to skip.
contents	List or character vector containing the contents of file. Note that file must still be supplied, see Details.
suppress.plots	Boolean indicating whether to suppress the generation of plots. If TRUE, all plots are redirected to the null device.

Details

This function leverages the input cache populated by processInputCommands. If a chunk contains code generated by processInputCommands, compileReport will attempt to load the corresponding object from cache instead of re-running the associated code. This avoids using more time/memory to create an object that is already available in the R session.

compileReport automatically changes the working directory to that of file. This is consistent with the behavior of **knitr** and ensures that any relative paths in file are respected. If contents is supplied, the actual file need not exist but its directory should be accessible.

Value

Code chunks from file are compiled, typically populating env with new variables. NULL is invisibly returned.

Author(s)

Aaron Lun

See Also

processInputCommands, to define the input objects in the Rmarkdown file.

extractChunks and evaluateChunks, to extract the R code chunks and evaluate them, respectively.

Examples

```
tmp <- tempfile()
dir.create(tmp)
path <- file.path(tmp, "test.Rmd")

write("```{r}
res <- data.frame(foo=1:5, bar=LETTERS[2:6])
write.csv(res, file='results.csv')
```", file=path)

env <- new.env()
compileReport(path, env)
env$res

list.files(dirname(path))
```
```

`deparseToString` *Deparse object to a string*

Description

Deparse an object to a single string by concatenation, possibly with indentation for multi-line output.

Usage

```
deparseToString(x, indent = NULL)
```

Arguments

`x`	R object to be deparsed.
`indent`	Integer scalar specifying the indenting for multi-line deparsed output. If `NULL`, the deparsed output will be put on a single line with no indenting.

Details

This function is based on `deparse` but with special accommodations when the deparsed output spans multiple lines. If `indent=NULL`, the multiple output lines are directly `pasted` together with no separator. If `indent` is an integer, a newline is added between lines, indented by the specified number of spaces.

Value

String containing the deparsed value of `x`.

Author(s)

Aaron Lun

Examples

```
x <- sample(100)
deparse(x)
deparseToString(x)
deparseToString(x, indent=4)
```

`evaluateChunks` *Evaluate R chunks*

Description

Evaluate R chunks, typically derived from an Rmarkdown report.

Usage

```
evaluateChunks(chunks, env)
```

Arguments

chunks	List of character vectors, typically from `extractChunks`. Each vector corresponds to a chunk of R code.
env	Environment in which to evaluate chunks.

Details

Each character vector may have some **knitr**-inspired attributes:

- `eval` is a string that, upon evaluation within `env`, specifies if the current chunk should be evaluated.
- `ref.label` is a string that, upon evaluation within `env`, contains the name of another character vector in `chunks`. The chunk in the named character vector will be evaluated instead of the current chunk.
- `error` is a string that, upon evaluation within `env`, specifies if errors should be ignored for this chunk.

Value

All `chunks` are evaluated within `env`.

Author(s)

Aaron Lun

Examples

```
chunks <- list("a <- 1", "a <- a + 2", "", "a <- NULL")
names(chunks) <- c("", "foo1", "", "foo3")
attr(chunks[[3]], "ref.label") <- "paste0('foo', 1)"
attr(chunks[[4]], "eval") <- "a < 0"

env <- new.env()
evaluateChunks(chunks, env)
env$a
```

`extractChunks` *Extract R chunks*

Description

Extract R chunks from an Rmarkdown document, along with their names and **knitr** properties.

Usage

```
extractChunks(contents)
```

Arguments

`contents`	A character vector or list (of lists) of character vectors containing an Rmarkdown document, typically obtained via `parseRmdTemplate`. Each string should correspond to a separate line.

Value

List of character vectors, each of which contains the R code for a single chunk. Each vector is named according to the name of the chunk. **knitr** properties are stored in the attributes for each vector.

Author(s)

Aaron Lun

Examples

```
contents <- "
```{r alpha}
alpha <- 1
bravo <- 2
```

Some random text here

```{r, echo=FALSE, eval=FALSE}
charlie <- 3
delta <- 4
```

- point 1 with some `r inline <- TRUE`.
- point 2.

  ```{r i-am-indented, error=TRUE}
  echo <- 5
  foxtrot <- 6
  ```

even more text here with `r 'some'` `r 'more'` `r 'inlining'`.
"

extractChunks(strsplit(contents, "\n"))
```

`parseRmdTemplate` *Parse Rmarkdown template*

Description

Parse an Rmarkdown report template to extract blocks of text or code.

Usage

```
parseRmdTemplate(contents)
```

Arguments

`contents`	String or character vector containing an Rmarkdown report template.

Details

An Rmarkdown report template may contain any number of paired `:BEGIN` and `:END` tags. These tags enclose blocks of text that may be dynamically removed or duplicated by the pipeline functions.

Tag pairs should follow these rules:

- Each tag should start on its own line.
- The `:BEGIN` tag should be followed by a name, e.g., `:BEGIN my-name-here`.
- The corresponding `:END` tag does not need to be followed by a name, but if it does, the name should be the same as that of the matching `:BEGIN` tag.
- Tag pairs may be nested within another tag pair.
- Names should be unique for any given level of nesting. For nested tag pairs, there should be no other tag pair with the same name within the parent tag pair. For unnested tag pairs, there should be no other unnested tag pair with the same name.

In the output of this function, each entry of the top-level list may be either:

- A list, corresponding to a tag pair. This list contains all text in the Rmarkdown template enclosed by the tag pair, which may be character vectors or further lists corresponding to nested tag pairs. If the tag pair is named, the same name will be used for this list entry.
- A character vector, to store newline-separated text within or between tag pairs.

Calling `unlist` on the output of this function will return a newline-separated version of `contents` without the tags.

Value

List of character vectors or nested lists, see Details.

Author(s)

Aaron Lun

Examples

```
template <- "
Ochite iku sunadokei bakari miteru yo
Sakasama ni sureba hora mata hajimaru yo
Kizanda dake susumu jikan ni
Itsuka boku mo haireru kana

```{r}
nagisa <- 'furukawa'
```

:BEGIN clannad

Kimi dake ga sugisatta saka no tochuu wa
Atataka na hidamari ga ikutsu mo dekiteta
Boku hitori ga koko de yasashii
Atatakasa wo omoikaeshiteru

:BEGIN after-story

```{r}
tomoya <- 'Okazaki'
```

:END

Kimi dake wo kimi dake wo
Suki de ita yo
Kaze de me ga nijinde
Tooku naru yo

:BEGIN toki-wa-kizamu-uta

Itsumademo oboeteru
Kono machi ga kawatte mo
Dore dake no kanashimi to deau koto ni natte mo
Misete yaru hontou wa tsuyokatta doki no koto
Saa iku yo arukidasu saka no michi wo

```{r}
ushio <- TRUE
```

:END

:END"

parseRmdTemplate(template)
```

---

processInputCommands *Process commands for a pipeline input*

Description

Convert an input object into commands that can be stored in an Rmarkdown report for a reproducible analysis.

Usage

```
processInputCommands(x, name)

resetInputCache()

wrapInput(object, commands)
```

Arguments

x	An arbitrary R object. Alternatively, a list with the augere.input class, created by createInput.
name	String containing the name of the object in the Rmarkdown report.
object	An arbitrary R object. Alternatively NULL, in which case the object will be generated from commands.
commands	Character vector of R commands to be used to generate object. If missing, this is obtained by deparsing the expression supplied as object. Alternatively NULL, if no commands are available.

Details

Pipeline developers should call processInputCommands to define the generation of input objects in the Rmarkdown report. Each call to processInputCommands will store the input R object in an internal cache. Upon compiling the report with compileReport, the cached object will be used instead of rerunning the code returned by processInputCommands. This saves time and avoids errors for inputs that have no associated commands. (The exception is if object=NULL, in which case the commands must be run to generate the object.)

At the start of a pipeline function, it is good practice to call resetInputCache. This clears any existing cached inputs from calls to other pipeline functions, e.g., in the pathological case where one pipeline function is called within another function. The function returned by resetInputCache should then be called once the pipeline function completes, e.g., in an on.exit block.

Value

processInputCommands will return a string containing R commands that can be inserted into an Rmarkdown code chunk. If x is an augere.input with non-NULL commands, those commands will be used directly. Otherwise, a stop command will be generated that instructs the user to manually insert commands to generate x.

wrapInput will return an augere.input list containing the object and commands fields.

resetInputCache will set the input cache to an empty list. It returns a function that restores the input cache to the state prior to the function's invocation.

Author(s)

Aaron Lun

See Also

compileReport, which uses the input cache.

replacePlaceholders, to insert the input commands into the Rmarkdown template.

Examples

```
fun <- resetInputCache()

df <- data.frame(foo=1:5, bar=LETTERS[2:6])
cat(processInputCommands(df, "my_df"))

wrapped <- wrapInput(df, c("y <- data.frame(foo = 1:5)", "y$bar <- LETTERS[2:6]", "y"))
cat(processInputCommands(wrapped, "my_df"))

fun()
```

---

replacePlaceholders *Substitute placeholders*

Description

Find all placeholders in an Rmarkdown template and replace them with actual text.

Usage

```
replacePlaceholders(contents, replacements, error = TRUE)
```

Arguments

contents	String, character vector or list containing an Rmarkdown report template. This is typically the output of parseRmdTemplate.
replacements	Named character vector of replacement text for placeholders. The name is the placeholder and the value is the replacement text. This may also be a named list of strings.
error	Boolean indicating whether to throw an error if a placeholder does not have a replacement in replacements. If FALSE, no error is thrown.

Details

A placeholder takes the form <%= LABEL %>, where LABEL can be any name without % or newlines. Pipeline functions are expected to replace these placeholders with actual text, usually based on user-supplied parameters.

Value

An object of the same type as contents, but with all of the placeholders replaced.

Author(s)

Aaron Lun

Examples

```
replacePlaceholders(
    "Hi my name is <%= NAME %> and I enjoy <%= FOOD %>.",
    replacements=c(NAME="Aaron", FOOD="pork")
)
```

`saveResult` *Save a DE result*

Description

Save result objects to disk using the **alabaster.base** framework.

Usage

```
saveResult(x, path, metadata, meta.name = "_metadata.json", requires = NULL)

readResult(path, meta.name = "_metadata.json")
```

Arguments

`x`	Result object to be saved.
`path`	String containing the path to the directory in which `x` is to be saved.
`metadata`	Named list of metadata to save alongside `x`. If `NULL`, no metadata is saved.
`meta.name`	String containing the name of the metadata file within `path`. Only used if `metadata` is not `NULL`.
`requires`	String containing the name of the **alabaster.*** package containing the relevant methods to save `x`. This may be necessary if the class of `x` is not known to `saveObject`.

Value

For `saveResult`, `x` and `metadata` (if supplied) are saved to disk at `path`. `NULL` is invisibly returned.

Author(s)

Aaron Lun

See Also

`saveObject` and `readObject`, for the underlying functionality.

Examples

```
df <- S4Vectors::DataFrame(A = 1:5, B=2:6)
meta <- list(
    title = "This is a very important result",
    author = "Myself",
    date = "2022-02-22"
)

tmp <- tempfile()
saveResult(df, tmp, metadata = meta)
list.files(tmp, recursive=TRUE)

readResult(tmp)
```

`writeRmd` *Write Rmarkdown file*

Description

Write Rmarkdown-formatted strings to file.

Usage

```
writeRmd(contents, file, append = FALSE, clean.empty = TRUE)
```

Arguments

`contents`	A character vector of a (nested) list of such vectors.
`file`	String containing a file path or a connection object to an opened file.
`append`	Logical scalar indicating whether to append `contents` to `file`.
`clean.empty`	Logical scalar indicating whether extraneous empty lines should be removed.

Details

Each element of each character vector in `contents` is placed on a new line in the output. Empty strings are respected and manifest as empty lines. If `clean.empty=TRUE`, repeated empty lines are removed.

Value

`contents` is written to `file` and `NULL` is invisibly returned.

Author(s)

Aaron Lun

Examples

```
contents <- list(
    c("# TITLE", ""),
    list(
        c("```{r}", "a <- 1", "```"),
        c("", "super foo bar")
    ),
    c("", "## section")
)
tmp <- tempfile(fileext=".Rmd")
writeRmd(contents, tmp)
cat(readLines(tmp), sep="\n") # having a look
```

Index