Package 'buildr'

October 12, 2022

Title Organize & Run Build Scripts Comfortably

Version 0.1.1

Description Working with reproducible reports or any other similar projects often require to run the script that builds the output file in a specified way. 'buildr' can help you organize, modify and comfortably run those scripts. The package provides a set of functions that interactively guides you through the process and that are available as 'RStudio' Addin, meaning you can set up the keyboard shortcuts, enabling you to choose and run the desired build script with one keystroke anywhere anytime.

License GPL (>= 3)

URL https://netique.github.io/buildr/

BugReports https://github.com/netique/buildr/issues/

Imports rstudioapi, usethis, readr, glue, stringr, magrittr, tibble, utils

Encoding UTF-8

RoxygenNote 7.2.1

Suggests knitr, rmarkdown, roxygen2, testthat (>= 3.0.0), spelling, pkgdown

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Author Jan Netik [aut, cre] (<https://orcid.org/0000-0002-3888-3203>)

Maintainer Jan Netik <netikja@gmail.com>

Repository CRAN

Date/Publication 2022-08-13 12:40:02 UTC

R topics documented:

`aim` *Set Makefile Target*

Description

`aim()` looks for an existing `Makefile`, reads its content, and offers a list of discovered `Makefile` targets (denoting build scripts, in our case), all in an interactive way. When the session is not interactive, or you know the name of the desired target, you can declare it directly in the `target` argument.

Usage

```
aim(target = NULL)
```

Arguments

`target`	*Character*. The name of the Makefile target to set.

Value

No return value. Called for side effects.

Author(s)

Jan Netik

See Also

Other functions from buildr trinity: `build()`, `init()`

Examples

```
## Not run:
# We have several build scripts in our project root
# and we want to select script called "build_all.R":

aim(target = "all") # note that "build_" is stripped out by default

## End(Not run)
```

build *Run Selected Build Script*

Description

`build()` is the final function in the workflow, as it instructs 'RStudio' Build pane to take the first rule in the `Makefile` (set previously with `aim()`) and runs the respective recipe.

Usage

```
build()
```

Details

The 'Rstudio' Build pane is not allways visible and set to take `Makefiles`. However, the `build()` ensures that everything is set properly and if not, it offers you to automatically edit necessary settings automatically for you. Note that this action forces 'RStudio' user interface (UI) to reload and you have to call `build()` again afterwards.

Value

No return value. Called for side effects.

Author(s)

Jan Netik

See Also

Other functions from buildr trinity: `aim()`, `init()`

Examples

```
## Not run:
build()

## End(Not run)
```

`edit_makefile` *Edit Makefile*

Description

Opens `Makefile`, if present in the project root.

Usage

```
edit_makefile()
```

Value

No return value. Called for side effect.

See Also

The documentation for GNU Make.

`edit_shortcuts` *Show RStudio Keyboard Shortcuts Popup*

Description

Shows popup window with RStudio keyboard shortcuts. Uses `rstudioapi`. Applicable only in RStudio and in interactive session.

Usage

```
edit_shortcuts()
```

Details

You can quicky reach out solicited addin function by typing it in the `Filter...` box in the very top of the popup window. Then double click at the blank space just next to the addin function name and press down desired key or key combination. Apply the changes and from now on, just call the function with one keystroke.

Value

No return value. Called for side effect.

Examples

```
## Not run:
edit_schortcuts()

## End(Not run)
```

`init` *Discover Build Scripts & Create Makefile*

Description

`init()` looks for `.R` scripts in a project root (current working directory) that contain a specified prefix and separator. Then, it creates a `Makefile` with rules describing how to run discovered scripts.

Usage

```
init(
  prefix = "build",
  sep = "_",
  path = ".",
  ignore_case = TRUE,
  command_args = ""
)
```

Arguments

`prefix`	*Character*. Prefix that solicited build scripts have in common. It is trimmed and stripped in the list of `Makefile` targets because of redundancy. Default to "build".
`sep`	*Character*. Separator between `prefix` and "body" of a build script filename. It is also stripped in the list of `Makefile` targets because of redundancy. Default to underscore (i.e. "_").
`path`	*Character*. Path being searched. Default to the project root (i.e. ".", the current working directory, call `getwd()` to print it). See `list.files` for more details on the topic.
`ignore_case`	*Logical*. Should the search be case-sensitive? Default to FALSE.
`command_args`	*Single character*. Command argument(s) to include after the recipe call. Command argument can be picked up by your script with `commandArgs`. See `vignette("know_your_buildr"` for more details. Empty string by default (not in use).

Details

The build script names should all follow a common pattern that is both human and machine readable. Filename should incorporate a prefix ("build" by default) and the "body" describing what the given script builds. Those two essential parts are separated by underscore (i.e. "_") by default as it helps with the readibility. Both parts are configurable (see below), but we encourage you not to make any changes. Do not forget that build scripts are matched for a prefix and separator concatenated together, so the script named "build.R" won't be recognized, as it doesn't begin with "build_". Follow the example below on how to include "build.R".

Value

No return value. Called for side effects.

Author(s)

Jan Netik

See Also

Other functions from buildr trinity: `aim()`, `build()`

Examples

```
## Not run:
# if you stick with the defaults, run:
init()

# if you want to include "build.R",
# you have to tell {buildr} to
# use an empty separator, like:
init(sep = "")

## End(Not run)
```

Index