Package 'cliapp'

April 24, 2024

Title Create Rich Command Line Applications

Version 0.1.2

Description Create rich command line applications, with colors, headings, lists, alerts, progress bars, etc. It uses CSS for custom themes. This package is now superseded by the 'cli' package. Please use 'cli' instead in new projects.

License MIT + file LICENSE

URL https://github.com/r-lib/cliapp#readme

BugReports https://github.com/r-lib/cliapp/issues

Depends R (>= 3.6)

Imports cli, crayon, fansi, glue (>= 1.3.0), prettycode, progress (>= 1.2.0), R6, selectr, utils, withr, xml2

Suggests callr, covr, rstudioapi, testthat

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Gábor Csárdi [aut, cre], Posit Software, PBC [cph, fnd]

Maintainer Gábor Csárdi <csardi.gabor@gmail.com>

Repository CRAN

Date/Publication 2024-04-24 20:50:02 UTC

R topics documented:

`builtin_theme` *The built-in CLI theme*

Description

This theme is always active, and it is at the bottom of the theme stack. See themes.

Usage

```
builtin_theme()
```

Value

A named list, a CLI theme.

See Also

themes, `simple_theme()`.

`cli_alert` *CLI alerts*

Description

Alerts are typically short status messages.

Usage

```
cli_alert(text, id = NULL, class = NULL, wrap = FALSE, .envir = parent.frame())

cli_alert_success(
  text,
  id = NULL,
  class = NULL,
  wrap = FALSE,
  .envir = parent.frame()
)

cli_alert_danger(
  text,
  id = NULL,
  class = NULL,
  wrap = FALSE,
  .envir = parent.frame()
)

cli_alert_warning(
  text,
  id = NULL,
  class = NULL,
  wrap = FALSE,
  .envir = parent.frame()
)

cli_alert_info(
  text,
  id = NULL,
  class = NULL,
  wrap = FALSE,
  .envir = parent.frame()
)
```

Arguments

text	Text of the alert.
id	Id of the alert element. Can be used in themes.
class	Class of the alert element. Can be used in themes.
wrap	Whether to auto-wrap the text of the alert.
.envir	Environment to evaluate the glue expressions in.

Examples

```
cli_alert("Cannot lock package library.")
```

```
cli_alert_success("Package {pkg cliapp} installed successfully.")
cli_alert_danger("Could not download {pkg cliapp}.")
cli_alert_warning("Internet seems to be unreacheable.")
cli_alert_info("Downloaded 1.45MiB of data")
```

`cli_div` *Generic CLI container*

Description

See containers. A `cli_div` container is special, because it may add new themes, that are valid within the container.

Usage

```
cli_div(
  id = NULL,
  class = NULL,
  theme = NULL,
  .auto_close = TRUE,
  .envir = parent.frame()
)
```

Arguments

`id`	Element id, a string. If `NULL`, then a new id is generated and returned.
`class`	Class name, sting. Can be used in themes.
`theme`	A custom theme for the container. See themes.
`.auto_close`	Whether to close the container, when the calling function finishes (or `.envir` is removed, if specified).
`.envir`	Environment to evaluate the glue expressions in. It is also used to auto-close the container if `.auto_close` is `TRUE`.

Value

The id of the new container element, invisibly.

Examples

```
## div with custom theme
d <- cli_div(theme = list(h1 = list(color = "blue",
                                    "font-weight" = "bold")))
cli_h1("Custom title")
cli_end(d)

## Close automatically
div <- function() {
```

```
  cli_div(class = "tmp", theme = list(.tmp = list(color = "yellow")))
  cli_text("This is yellow")
}
div()
cli_text("This is not yellow any more")
```

`cli_dl` *Definition list*

Description

A definition list is a container, see containers.

Usage

```
cli_dl(
  items = NULL,
  id = NULL,
  class = NULL,
  .close = TRUE,
  .auto_close = TRUE,
  .envir = parent.frame()
)
```

Arguments

`items`	Named character vector, or `NULL`. If not `NULL`, they are used as list items.
`id`	Id of the list container. Can be used for closing it with `cli_end()` or in themes. If `NULL`, then an id is generated and retuned invisibly.
`class`	Class of the list container. Can be used in themes.
`.close`	Whether to close the list container if the `items` were specified. If `FALSE` then new items can be added to the list.
`.auto_close`	Whether to close the container, when the calling function finishes (or `.envir` is removed, if specified).
`.envir`	Environment to evaluate the glue expressions in. It is also used to auto-close the container if `.auto_close` is `TRUE`.

Value

The id of the new container element, invisibly.

Examples

```
## Specifying the items at the beginning
cli_dl(c(foo = "one", bar = "two", baz = "three"))

## Adding items one by one
cli_dl()
cli_it(c(foo = "one"))
cli_it(c(bar = "two"))
cli_it(c(baz = "three"))
cli_end()
```

`cli_end` *Close a CLI container*

Description

Close a CLI container

Usage

```
cli_end(id = NULL)
```

Arguments

`id`	Id of the container to close. If missing, the current container is closed, if any.

Examples

```
## If id is omitted
cli_par()
cli_text("First paragraph")
cli_end()
cli_par()
cli_text("Second paragraph")
cli_end()
```

`cli_h1` *CLI headers*

Description

CLI headers

Usage

```
cli_h1(text, id = NULL, class = NULL, .envir = parent.frame())

cli_h2(text, id = NULL, class = NULL, .envir = parent.frame())

cli_h3(text, id = NULL, class = NULL, .envir = parent.frame())
```

Arguments

text	Text of the header. It can contain inline markup.
id	Id of the header element, string. It can be used in themes.
class	Class of the header element, string. It can be used in themes.
.envir	Environment to evaluate the glue expressions in.

Examples

```
cli_h1("Main title")
cli_h2("Subtitle")
cli_text("And some regular text....")
```

---

cli_it *CLI list item(s)*

Description

A list item is a container, see containers.

Usage

```
cli_it(
  items = NULL,
  id = NULL,
  class = NULL,
  .auto_close = TRUE,
  .envir = parent.frame()
)
```

Arguments

items	Character vector of items, or NULL.
id	Id of the new container. Can be used for closing it with cli_end() or in themes. If NULL, then an id is generated and retuned invisibly.
class	Class of the item container. Can be used in themes.
.auto_close	Whether to close the container, when the calling function finishes (or .envir is removed, if specified).
.envir	Environment to evaluate the glue expressions in. It is also used to auto-close the container if .auto_close is TRUE.

Value

The id of the new container element, invisibly.

Examples

```
## Adding items one by one
cli_ul()
cli_it("one")
cli_it("two")
cli_it("three")
cli_end()

## Complex item, added gradually.
cli_ul()
cli_it()
cli_verbatim("Beginning of the {emph first} item")
cli_text("Still the first item")
cli_end()
cli_it("Second item")
cli_end()
```

---

cli_ol *Ordered CLI list*

Description

An ordered list is a container, see containers.

Usage

```
cli_ol(
  items = NULL,
  id = NULL,
  class = NULL,
  .close = TRUE,
  .auto_close = TRUE,
  .envir = parent.frame()
)
```

Arguments

items	If not NULL, then a character vector. Each element of the vector will be one list item, and the list container will be closed by default (see the .close argument).
id	Id of the list container. Can be used for closing it with cli_end() or in themes. If NULL, then an id is generated and retuned invisibly.
class	Class of the list container. Can be used in themes.
.close	Whether to close the list container if the items were specified. If FALSE then new items can be added to the list.

.auto_close	Whether to close the container, when the calling function finishes (or .envir is removed, if specified).
.envir	Environment to evaluate the glue expressions in. It is also used to auto-close the container if .auto_close is TRUE.

Value

The id of the new container element, invisibly.

Examples

```
## Specifying the items at the beginning
cli_ol(c("one", "two", "three"))

## Adding items one by one
cli_ol()
cli_it("one")
cli_it("two")
cli_it("three")
cli_end()

## Nested lists
cli_div(theme = list(ol = list("margin-left" = 2)))
cli_ul()
cli_it("one")
cli_ol(c("foo", "bar", "foobar"))
cli_it("two")
cli_end()
cli_end()
```

---

cli_par *CLI paragraph*

Description

See containers.

Usage

```
cli_par(id = NULL, class = NULL, .auto_close = TRUE, .envir = parent.frame())
```

Arguments

id	Element id, a string. If NULL, then a new id is generated and returned.
class	Class name, sting. Can be used in themes.
.auto_close	Whether to close the container, when the calling function finishes (or .envir is removed, if specified).
.envir	Environment to evaluate the glue expressions in. It is also used to auto-close the container if .auto_close is TRUE.

Value

The id of the new container element, invisibly.

Examples

```
id <- cli_par()
cli_text("First paragraph")
cli_end(id)
id <- cli_par()
cli_text("Second paragraph")
cli_end(id)
```

`cli_progress_bar` *CLI progress bar*

Description

A progress bar using the progress package

Usage

```
cli_progress_bar(...)
```

Arguments

`...`	All arguments are passed to the constuctor of the progress::progress_bar class.

Value

A remote progress bar object that can be used the same way as progress::progress_bar, see examples below.

Examples

```
{
  p <- cli_progress_bar(total = 10)
  cli_alert_info("Starting computation")
  for (i in 1:10) { p$tick(); Sys.sleep(0.2) }
  cli_alert_success("Done")
}
```

cli_text *CLI text*

Description

It is wrapped to the screen width automatically. It may contain inline markup. (See inline-markup.)

Usage

```
cli_text(..., .envir = parent.frame())
```

Arguments

...	The text to show, in character vectors. They will be concatenated into a single string. Newlines are *not* preserved.
.envir	Environment to evaluate the glue expressions in.

Examples

```
cli_text("Hello world!")
cli_text(packageDescription("cliapp")$Description)

## Arguments are concatenated
cli_text("this", "that")

## Command substitution
greeting <- "Hello"
subject <- "world"
cli_text("{greeting} {subject}!")

## Inline theming
cli_text("The {fun cli_text} function in the {pkg cliapp} package")

## Use within container elements
ul <- cli_ul()
cli_it()
cli_text("{emph First} item")
cli_it()
cli_text("{emph Second} item")
cli_end(ul)
```

cli_ul *Unordered CLI list*

Description

An unordered list is a container, see containers.

Usage

```
cli_ul(
  items = NULL,
  id = NULL,
  class = NULL,
  .close = TRUE,
  .auto_close = TRUE,
  .envir = parent.frame()
)
```

Arguments

items	If not NULL, then a character vector. Each element of the vector will be one list item, and the list container will be closed by default (see the .close argument).
id	Id of the list container. Can be used for closing it with cli_end() or in themes. If NULL, then an id is generated and retuned invisibly.
class	Class of the list container. Can be used in themes.
.close	Whether to close the list container if the items were specified. If FALSE then new items can be added to the list.
.auto_close	Whether to close the container, when the calling function finishes (or .envir is removed, if specified).
.envir	Environment to evaluate the glue expressions in. It is also used to auto-close the container if .auto_close is TRUE.

Value

The id of the new container element, invisibly.

Examples

```
## Specifying the items at the beginning
cli_ul(c("one", "two", "three"))

## Adding items one by one
cli_ul()
cli_it("one")
cli_it("two")
cli_it("three")
cli_end()
```

```
## Complex item, added gradually.
cli_ul()
cli_it()
cli_verbatim("Beginning of the {emph first} item")
cli_text("Still the first item")
cli_end()
cli_it("Second item")
cli_end()
```

`cli_verbatim` *CLI verbatim text*

Description

It is not wrapped, but printed as is.

Usage

```
cli_verbatim(..., .envir = parent.frame())
```

Arguments

`...`	The text to show, in character vectors. Each element is printed on a new line.
`.envir`	Environment to evaluate the glue expressions in.

Examples

```
cli_verbatim("This has\nthree", "lines")
```

`console_width` *Determine the width of the console*

Description

It uses the `RSTUDIO_CONSOLE_WIDTH` environment variable, if set. Otherwise it uses the `width` option. If this is not set either, then 80 is used.

Usage

```
console_width()
```

Value

Integer scalar, the console with, in number of characters.

containers *CLI containers*

Description

Container elements may contain other elements. Currently the following commands create container elements: `cli_div()`, `cli_par()`, the list elements: `cli_ul()`, `cli_ol()`, `cli_dl()`, and list items are containers as well: `cli_it()`.

Details

Container elements need to be closed with `cli_end()`. For convenience, they are have an `.auto_close` argument, which allows automatically closing a container element, when the function that created it terminates (either regularly, or with an error).

Examples

```
## div with custom theme
d <- cli_div(theme = list(h1 = list(color = "blue",
                                    "font-weight" = "bold")))
cli_h1("Custom title")
cli_end(d)

## Close automatically
div <- function() {
  cli_div(class = "tmp", theme = list(.tmp = list(color = "yellow")))
  cli_text("This is yellow")
}
div()
cli_text("This is not yellow any more")
```

inline-markup *CLI inline markup*

Description

CLI inline markup

Command substitution

All text emitted by cliapp supports glue interpolation. Expressions enclosed by braces will be evaluated as R code. See `glue::glue()` for details.

In addition to regular glue interpolation, cliapp can also add classes to parts of the text, and these classes can be used in themes. For example

```
cli_text("This is {emph important}.")
```

adds a class to the "important" word, class "emph". Note that in this cases the string within the braces is not a valid R expression. If you want to mix classes with interpolation, add another pair of braces:

```
adjective <- "great"
cli_text("This is {emph {adjective}}.")
```

An inline class will always create a `span` element internally. So in themes, you can use the `span.emph` CSS selector to change how inline text is emphasized:

```
cli_div(theme = list(span.emph = list(color = "red")))
adjective <- "nice and red"
cli_text("This is {emph {adjective}}.")
```

Classes

The default theme defines the following inline classes:

- `emph` for emphasized text.
- `strong` for strong importance.
- `code` for a piece of code.
- `pkg` for a package name.
- `fun` for a function name.
- `arg` for a function argument.
- `key` for a keyboard key.
- `file` for a file name.
- `path` for a path (essentially the same as `file`).
- `email` for an email address.
- `url` for a URL.
- `var` for a variable name.
- `envvar` for the name of an environment variable.

See examples below.

You can simply add new classes by defining them in the theme, and then using them, see the example below.

Examples

```
## Some inline markup examples
cli_ul()
cli_it("{emph Emphasized} text")
cli_it("{strong Strong} importance")
cli_it("A piece of code: {code sum(a) / length(a)}")
cli_it("A package name: {pkg cliapp}")
cli_it("A function name: {fun cli_text}")
cli_it("A function argument: {arg text}")
```

```
cli_it("A keyboard key: press {key ENTER}")
cli_it("A file name: {file /usr/bin/env}")
cli_it("An email address: {email bugs.bunny@acme.com}")
cli_it("A URL: {url https://acme.com}")
cli_it("A variable name: {var mtcars}")
cli_it("An environment variable: {envvar R_LIBS}")
cli_end()

## Adding a new class
cli_div(theme = list(
  span.myclass = list(color = "lightgrey"),
  "span.myclass::before" = list(content = "["),
  "span.myclass::after" = list(content = "]")))
cli_text("This is {myclass in brackets}.")
cli_end()
```

`simple_theme` *A simple CLI theme*

Description

Note that this is in addition to the builtin theme. To use this theme, you can set it as the `cli.theme` option:

Usage

```
simple_theme(dark = "auto")
```

Arguments

`dark`	Whether the theme should be optiomized for a dark background. If `"auto"`, then cliapp will try to detect this. Detection usually works in recent RStudio versions, and in iTerm on macOS, but not on other platforms.

Details

```
options(cli.theme = cliapp::simple_theme())
```

and then CLI apps started after this will use it as the default theme. You can also use it temporarily, in a div element:

```
cli_div(theme = cliapp::simple_theme())
```

See Also

themes, `builtin_theme()`.

Examples

```
cli_div(theme = cliapp::simple_theme())

cli_h1("Header 1")
cli_h2("Header 2")
cli_h3("Header 3")

cli_alert_danger("Danger alert")
cli_alert_warning("Warning alert")
cli_alert_info("Info alert")
cli_alert_success("Success alert")
cli_alert("Alert for starting a process or computation",
  class = "alert-start")

cli_text("Packages and versions: {pkg cliapp} {version 1.0.0}.")
cli_text("Time intervals: {timestamp 3.4s}")

cli_text("{emph Emphasis} and  {strong strong emphasis}")

cli_text("This is a piece of code: {code sum(x) / length(x)}")
cli_text("Function names: {fun cliapp::simple_theme} and {arg arguments}.")

cli_text("Files: {file /usr/bin/env}")
cli_text("URLs: {url https://r-project.org}")

cli_h2("Longer code chunk")
cli_par(class = "r-code")
cli_verbatim(
  '# window functions are useful for grouped mutates',
  'mtcars %>%',
  '  group_by(cyl) %>%',
  '  mutate(rank = min_rank(desc(mpg)))')
cli_end()

cli_h2("Even longer code chunk")
cli_par(class = "r-code")
cli_verbatim(format(ls))
cli_end()

cli_end()
```

`start_app` *Start, stop, query the default cli application*

Description

`start_app` creates an app, and places it on the top of the app stack.

Usage

```
start_app(
  theme = getOption("cli.theme"),
  output = c("message", "stdout"),
  .auto_close = TRUE,
  .envir = parent.frame()
)

stop_app(app = NULL)

default_app()
```

Arguments

theme	Theme to use, passed to the cliapp initializer.
output	How to print the output, passed to cliapp initializer.
.auto_close	Whether to stop the app, when the calling frame is destroyed.
.envir	The environment to use, instead of the calling frame, to trigger the stop of the app.
app	App to stop. If NULL, the current default app is stopped. Otherwise we find the supplied app in the app stack, and remote it, together with all the apps above it.

Details

stop_app removes the top app, or multiple apps from the app stack.

default_app returns the default app, the one on the top of the stack.

Value

start_app returns the new app, default_app returns the default app. stop_app does not return anything.

---

themes *CLI themes*

Description

CLI elements can be styled via a CSS-like language of selectors and properties. Note that while most of the CSS3 language is supported, a lot visual properties cannot be implemented on a terminal, so these will be ignored.

Adding themes

The style of an element is calculated from themes from four sources. These form a stack, and the styles on the top of the stack take precedence, over styles in the bottom.

1. The cliapp package has a builtin theme. This is always active. See `builtin_theme()`.
2. When an app object is created via `start_app()`, the caller can specify a theme, that is added to theme stack. If no theme is specified for `start_app()`, the content of the `cli.theme` option is used. Removed when the corresponding app stops.
3. The user may speficy a theme in the `cli.user_theme` option. This is added to the stack *after* the app's theme (step 2.), so it can override its settings. Removed when the app that added it stops.
4. Themes specified explicitly in `cli_div()` elements. These are removed from the theme stack, when the corresponding `cli_div()` elements are closed.

Writing themes

A theme is a named list of lists. The name of each entry is a CSS selector. Most features of CSS selectors are supported here:, for a complete reference, see the selectr package.

The content of a theme list entry is another named list, where the names are CSS properties, e.g. `color`, or `font-weight` or `margin-left`, and the list entries themselves define the values of the properties. See `builtin_theme()` and `simple_theme()` for examples.

CSS pseudo elements

Currently only the `::before` and `::after` pseudo elements are supported.

Formatter callbacks

For flexibility, themes may also define formatter functions, with property name `fmt`. These will be called once the other styles are applied to an element. They are only called on elements that produce output, i.e. *not* on container elements.

Supported properties

Right now only a limited set of properties are supported. These include left, right, top and bottom margins, background and foreground colors, bold and italic fonts, underlined text. The `content` property is supported to insert text via `::before` and `::after` selectors.

More properties might be adder later.

Please see the example themes and the source code for now for the details.

Examples

Color of headers, that are only active in paragraphs with an 'output' class:

```
list(
  "par.output h1" = list("background-color" = "red", color = "#e0e0e0"),
  "par.output h2" = list("background-color" = "orange", color = "#e0e0e0"),
```

```
  "par.output h3" = list("background-color" = "blue", color = "#e0e0e0")
)
```

Create a custom alert type:

```
list(
  ".alert-start::before" = list(content = symbol$play),
  ".alert-stop::before"  = list(content = symbol$stop)
)
```

Index