Package 'blastula'

October 12, 2022

Type Package

Title Easily Send HTML Email Messages

Version 0.3.2

Description Compose and send out responsive HTML email messages that render perfectly across a range of email clients and device sizes. Helper functions let the user insert embedded images, web link buttons, and 'ggplot2' plot objects into the message body. Messages can be sent through an 'SMTP' server, through the 'RStudio Connect' service, or through the 'Mailgun' API service <http://mailgun.com/>.

License MIT + file LICENSE

URL https://github.com/rich-iannone/blastula

BugReports https://github.com/rich-iannone/blastula/issues

Depends R (>= 3.2.1)

Imports base64enc (>= 0.1-3), commonmark (>= 1.7), curl (>= 4.3), digest, dplyr (>= 0.8.3), fs (>= 1.3.1), getPass (>= 0.2-2), here (>= 0.1), htmltools (>= 0.4.0), httr (>= 1.4.0), jsonlite (>= 1.6), magrittr (>= 1.5), mime (>= 0.6), rlang (>= 0.4.1), rmarkdown, stringr (>= 1.4.0), uuid (>= 0.1-2)

Suggests covr, ggplot2, glue, testthat (>= 2.1.0), keyring, knitr, spelling, xml2

SystemRequirements pandoc (>= 1.12.3) - http://pandoc.org

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Richard Iannone [aut, cre] (<https://orcid.org/0000-0003-3925-190X>), Joe Cheng [aut], Jeroen Ooms [ctb] (<https://orcid.org/0000-0002-4035-0289>), Ted Goas [cph] (cerberus-meta.html)

Maintainer Richard Iannone <riannone@me.com>

Repository CRAN

Date/Publication 2020-05-19 06:30:09 UTC

R topics documented:

add_attachment . . . 3
add_cta_button . . . 3
add_ggplot . . . 4
add_image . . . 6
add_imgur_image . . . 7
add_readable_time . . . 8
article . . . 9
attach_connect_email . . . 10
blastula_email . . . 11
blastula_template . . . 12
blocks . . . 13
block_articles . . . 14
block_social_links . . . 15
block_spacer . . . 17
block_text . . . 18
block_title . . . 19
compose_email . . . 20
create_smtp_creds_file . . . 22
create_smtp_creds_key . . . 23
credential_helpers . . . 24
delete_all_credential_keys . . . 26
delete_credential_key . . . 27
get_html_str . . . 27
md . . . 28
prepare_rsc_example_files . . . 28
prepare_test_message . . . 29
render_email . . . 30
send_by_mailgun . . . 31
smtp_send . . . 32
social_link . . . 35
suppress_scheduled_email . . . 38
view_credential_keys . . . 38
%>% . . . 39

Index **40**

add_attachment *Add a file attachment to an email message*

Description

This gives us a simple interface for attaching a file to the email object. When it comes time to send the email through `smtp_send()`, all attachments (specified by individual calls to `add_attachment()`) will be faithfully transmitted along with the message.

Usage

```
add_attachment(
  email,
  file,
  content_type = mime::guess_type(file),
  filename = basename(file)
)
```

Arguments

`email`	The email message object, as created by the `compose_email()` function. The object's class is `email_message`.
`file`	The filename for the file to be attached.
`content_type`	The MIME type for the attachment. By default, this is guessed by the `mime::guess_type()` function on the basis of the `file`'s extension. The available MIME types that can be guessed are available in the `mime::mimemap` named character vector.
`filename`	the filename for the attachment. This can be different than the basename provided to `file` for the purpose of customization. By default, the basename of `file` is taken to be the attachment's filename.

Details

There are options available to specify the attachment's MIME type, its disposition, and customize the attachment's recipient-facing filename.

add_cta_button *Create an HTML fragment for a CTA button*

Description

Create the HTML fragment for a call to action button. This can be used as part of the email body but, since this HTML, it must be contained within `md()`. There are options to specify the button text, the URL, and the button's alignment.

Usage

```
add_cta_button(url, text, align = "center")
```

Arguments

`url`	A URL for the button.
`text`	The text that is placed atop the CTA button.
`align`	The alignment of the button inside the main content area. Options are `"center"` (the default), `"left"`, and `"right"`.

Value

A character object with an HTML fragment that can be placed inside the message body wherever the CTA button should appear.

Examples

```
# Create the button as an HTML fragment
cta_button <-
  add_cta_button(
    url = "http://www.website.net",
    text = "Press This Button"
  )

# Include the button in the email
# message body by using it as part of
# a vector inside of `md()`
email <-
  compose_email(
    body = md(
      c(
  "Pressing the button will take
  you to an example website",
  cta_button
      )
    )
  )

if (interactive()) email
```

---

add_ggplot *Create an HTML fragment for an embedded ggplot image*

Description

Add an ggplot plot inside the body of the email with this helper function.

Usage

```
add_ggplot(
  plot_object,
  width = 5,
  height = 5,
  alt = NULL,
  align = c("center", "left", "right", "inline"),
  float = c("none", "left", "right")
)
```

Arguments

plot_object	The ggplot2 plot object.
width	The width of the output plot in inches.
height	The height of the output plot in inches.
alt	Text description of image passed to the alt attribute inside of the image (<img>) tag for use when image loading is disabled and on screen readers. Defaults to the ggplot2 plot object's title, if exists. Override by passing a custom character string or "" for no text.
align	The alignment to be used for the image. If not "inline", the image will appear in its own block, i.e. there will not be text to the left or right of it.
float	The float value to be used for the image. If not "none", text will flow around the image, and the align argument will be ignored.

Value

An HTML fragment that can be placed inside the message body wherever the plot image should appear.

Examples

```
library(ggplot2)

# Create a ggplot plot
plot <-
  ggplot(
    data = mtcars,
    aes(x = disp, y = hp,
        color = wt, size = mpg)) +
  geom_point()

# Create an HTML fragment that
# contains an the ggplot as an
# embedded plot
plot_html <-
  add_ggplot(plot_object = plot)

# Include the plot in the email
# message body by simply referencing
```

```
# the `plot_html` object
email <-
  compose_email(
    body = md(
      c(
"Hello!

Here is a plot that will change
the way you look at cars forever.\n",
plot_html,
"Let me know what you think
about it!"
      )
    )
  )

if (interactive()) email
```

---

add_image *Create an HTML fragment for an embedded image*

Description

Add a local image inside the body of the email with this helper function.

Usage

```
add_image(
  file,
  alt = "",
  width = 520,
  align = c("center", "left", "right", "inline"),
  float = c("none", "left", "right")
)
```

Arguments

file	A path to an image file.
alt	Text description of image passed to the alt attribute inside of the image (<img>) tag for use when image loading is disabled and on screen readers. NULL default produces blank ("") alt text.
width	The width to be used for the image, in pixels.
align	The alignment to be used for the image. If not "inline", the image will appear in its own block, i.e. there will not be text to the left or right of it.
float	The float value to be used for the image. If not "none", text will flow around the image, and the align argument will be ignored.

Value

A character object with an HTML fragment that can be placed inside the message body wherever the image should appear.

Examples

```
# Create an HTML fragment that
# contains an image
img_file_path <-
  system.file(
    "example_files",
    "test_image.png",
    package = "blastula"
  )

img_file_html <-
  add_image(file = img_file_path)

# Include the image in the email
# message body by simply referencing
# the `img_file_html` object
email <-
  compose_email(
    body = md(
      c(
"Hello,

Here is an image:\n",
img_file_html
      )
    )
  )

if (interactive()) email
```

`add_imgur_image` *Deploy a local image to Imgur and create an image tag*

Description

Getting images into email message bodies (and expecting them to appear for the recipient) can be a harrowing experience. External images (i.e., available at public URLs) work exceedingly well and most email clients will faithfully display these images. With the `imgur_image()` function, we can take a local image file or a `ggplot2` plot object and send it to the Imgur service, and finally receive an image (`<img>`) tag that can be directly inserted into an email message using `compose_email()`.

Usage

```
add_imgur_image(
  image,
  client_id = NULL,
  alt = NULL,
  width = 520,
  align = c("center", "left", "right", "inline"),
  float = c("none", "left", "right")
)
```

Arguments

image	The path to the local image we would like to deploy to Imgur and for which we'd like an image tag.
client_id	The Imgur Client ID value.
alt	Text description of image passed to the alt attribute inside of the image (<img>) tag for use when image loading is disabled and on screen readers. NULL default produces blank ("") alt text.
width	The width to be used for the image, in pixels.
align	The alignment to be used for the image. If not "inline", the image will appear in its own block, i.e. there will not be text to the left or right of it.
float	The float value to be used for the image. If not "none", text will flow around the image, and the align argument will be ignored.

Details

To take advantage of this, we need to first have an account with Imgur and then obtain a Client-ID key for the Imgur API. This can be easily done by going to https://api.imgur.com/oauth2/addclient and registering an application. Be sure to select the OAuth 2 authorization type without a callback URL.

Value

An HTML fragment that can be placed inside the message body wherever the image should appear.

---

add_readable_time *Create a string with a more readable date/time*

Description

Add a nicely-formatted date/time string inside the body of the email with this helper function. We can provide a POSIXct date-time object or use the current date/time/tz (based on the user's locale information at the time of the function call). There are options to specify whether the date, time, and time zone parts are to be included.

Usage

```
add_readable_time(time = NULL, use_date = TRUE, use_time = TRUE, use_tz = TRUE)
```

Arguments

time	The POSIXct time to use, and to make more readable for email recipients. If a time is not provided (the default), the current system time will be used.
use_date, use_time, use_tz	Logical value that indicate whether to include the date, time, or time zone components.

Value

A character object that can be placed inside any message component message wherever the function is called.

---

article *Specify the components of an article*

Description

The article() function is used exclusively within block_articles(), and having one, two, or three calls will arrange the articles in a row (or as a column of articles at lower screen widths).

Usage

```
article(image = NULL, title = NULL, content = NULL, link = NULL)
```

Arguments

image	An optional URL pointing to an image resource.
title	An optional title for the article.
content	An optional paragraph of text for the article.
link	An optional link to apply to the content elements.

Examples

```
# We can define an article with a link
# to an image, title text, some content,
# and a link to relevant content
article <-
  article(
    image = "https://i.imgur.com/dxSXzGb.jpg",
    title = "Hong Kong",
    content =
      "Once home to fishermen and farmers, \\
      modern Hong Kong is a teeming, \\
```

```
      commercially-vibrant metropolis where \\
      Chinese and Western influences fuse.",
    link = "http://www.discoverhongkong.com"
  )

if (interactive()) article
```

`attach_connect_email` *Associate an email when publishing an R Markdown document to RStudio Connect*

Description

This function is used to customize emails sent by RStudio Connect in conjunction with publishing an R Markdown document. It associates a custom email message with the main R Markdown document, which Connect can send to selected recipients. The main input is a rendered email message, which can be produced by either the `render_email()` or `render_connect_email()` function.

Usage

```
attach_connect_email(
  email = NULL,
  subject = NULL,
  attachments = NULL,
  attach_output = FALSE,
  text = NULL,
  preview = TRUE
)
```

Arguments

`email`	A rendered email message. Normally, we'd want to use an associated .Rmd file with the `blastula::blastula_email` R Markdown output format in `render_connect_email()` call (where its `input` is the email .Rmd file).
`subject`	An option to specify the the email subject while attaching the email object.
`attachments`	A vector of attachments for the Connect email. These files can be any of those deployed when publishing to RStudio Connect, and, any generated files (via R Markdown rendering).
`attach_output`	Should the rendered output of the main R Markdown document be included as an email attachment? By default, this is `FALSE`. If `TRUE` the main R Markdown document will be attached first (before any files specified in `attachments`) and the filename will be preserved.
`text`	Instead of using a rendered email document through the `email` option, we can use plain text here. However, if any `email` object is supplied then input to `text` is ignored.
`preview`	Should the email message display it's own preview window? If `TRUE` (the default), the rendered email message will be shown in the default browser.

Details

Since this function needs to be invoked within an R Markdown document, the chunk option `echo=FALSE` is useful here (so that viewers of the rendered document don't have to unnecessarily read code related to email inclusion). While the output is invisible, any errors related to rendering will be visible to the author.

---

blastula_email *The R Markdown* `blastula_email` *output format*

Description

The R Markdown `blastula_email` output format

Usage

```
blastula_email(
  content_width = "1000px",
  toc = FALSE,
  toc_depth = 3,
  toc_float = FALSE,
  number_sections = FALSE,
  section_divs = TRUE,
  fig_width = 5.35,
  fig_height = 5,
  fig_retina = 2,
  fig_caption = TRUE,
  dev = "png",
  smart = TRUE,
  self_contained = TRUE,
  template = "blastula",
  includes = NULL,
  keep_md = FALSE,
  md_extensions = NULL,
  connect_footer = FALSE,
  ...
)
```

Arguments

`content_width`	The width of the rendered HTML content. By default, this is set to `1000px`. Using widths less than `600px` is generally not advised but, if necessary, be sure to test such HTML emails with a wide range of email clients before sending to the intended recipients. The Outlook mail client (Windows, Desktop) does not respect `content_width`.
`toc`	If you would like an automatically-generated table of contents in the output email, choose `TRUE`. By default, this is `FALSE` where no table of contents will be generated.

`toc_depth` The depth of headers to include in the table of contents (should `toc` be set to `TRUE`).

`toc_float` An option to float the table of contents to the left of the main document content. By default, this is `FALSE`.

`number_sections`
Sections can be sequentially numbered if this is set to `TRUE`. By default, this is `FALSE`.

`section_divs` This wraps sections in `<section>` tags and attaches identifiers to the enclosing `<section>`s. This is set to `TRUE`.

`fig_width, fig_height`
The figure width and height in units of inches.

`fig_retina` The scaling factor for retina displays. The default value is `2`, which is the preferred choice for most retina displays. This can be set to `NULL` to prevent retina scaling. Note that this will always be `NULL` if `keep_md` is set to `TRUE`.

`fig_caption` An option to render figures with captions. By default, this is set to `TRUE`.

`dev` The R graphics device for figures. By default, this is the `png` device.

`smart` An option to produce typographically correct output. This will convert straight quotes to curly quotes, `---` to em dashes, `--` to en dashes, and instances of `...` to ellipses. By default, this is `TRUE`.

`self_contained` Should a self-contained output file be generated. By default, this is `TRUE`. The standalone HTML file will have no external dependencies, it will use URIs to incorporate the contents of linked scripts, stylesheets, images, and videos.

`template` The Pandoc template to use for rendering. This is the `"blastula"` template by default.

`includes` A named list of additional content to include within the document. This is typically created using the `rmarkdown::includes()` function. By default, this is set to `NULL`.

`keep_md` Should you need the keep the intermediate Markdown (.md) file, set this to `TRUE`. By default, the .md file is not kept.

`md_extensions` Markdown extensions to be added or removed from the default definition or R Markdown.

`connect_footer` Should a prepared footer message with links be included in the rendered email?

`...` Specify other options in `rmarkdown::html_document()`.

`blastula_template` *Default template for* `compose_email()`

Description

A template function that is suitable for using as the `template` argument of `compose_email()`. Template functions should generally not be called directly. When implementing your own template function, you must include parameters for `html_body`, `html_header`, `html_footer`, and `title`; you may also optionally add your own parameters, which callers to `compose_email()` can provide through the `...` argument.

Usage

```
blastula_template(
  html_body,
  html_header,
  html_footer,
  title,
  content_width = "1000px",
  font_family = "Helvetica, sans-serif"
)
```

Arguments

`html_body`, `html_header`, `html_footer`	htmltools tag objects (e.g. `htmltools::tags()` or `htmltools::HTML()`), or `NULL` to omit.
`title`	Plain text title to be used for the `<title>` element; may be displayed in mobile phone notifications.
`content_width`	The width that should be used for the content area. By default, this is set to `1000px`. Using widths less than `600px` is generally not advised but, if necessary, be sure to test such HTML emails with a wide range of email clients before sending to the intended recipients.
`font_family`	The CSS value to use for `font-family`.

Value

A string containing a complete HTML document.

`blocks` *An enclosure for all HTML block functions*

Description

To contain all of the block-based HTML `block_*()` calls, we should use the `blocks()` function. We can pass the resulting `blocks` object to either of the `body`, `header`, and `footer` arguments of `compose_email()`.

Usage

```
blocks(...)
```

Arguments

`...`	One or more `block_*()` calls.

Examples

```
# This is an example of how a
# title and text looks in each of
# the three content areas
email <-
  compose_email(
    header =
      blocks(
        block_title("This is a Title in the **Header**"),
        block_text("This is text in the **Header**.")
      ),
    body =
      blocks(
        block_title("This is a Title in the **Body**"),
        block_text("This is text in the **Body**.")
      ),
    footer =
      blocks(
        block_title("This is a Title in the **Footer**"),
        block_text("This is text in the **Footer**.")
      )
  )

if (interactive()) email
```

`block_articles` *A block of one, two, or three articles with a multicolumn layout*

Description

With `block_articles()`, we can create a single- or multi-column layout of articles. The articles are responsive to the screen width, so side-by-side articles will collapse and any of the optional images will resize accordingly. The function can accept one to three `article()` calls, each with varying amounts of text and imagery. Like all `block_*()` functions, `block_articles()` must be placed inside of `blocks()` and the resultant `blocks` object can be provided to the `body`, `header`, or `footer` arguments of `compose_email()`.

Usage

```
block_articles(...)
```

Arguments

`...`	One, two, or three calls to `article()`.

Examples

```
# Create a block of three, side-by-side
# articles with three `article()`
# calls inside of `block_articles()`,
# itself placed in `blocks()`
email <-
  compose_email(
    body =
      blocks(
        block_articles(
          article(
            image = "https://i.imgur.com/XMU8yJa.jpg",
            title = "Taiwan",
            content =
              "It is a thriving mosaic of tradition,
              culture, and high-tech development,
              merging Eastern and Western influences."
          ),
          article(
            image = "https://i.imgur.com/aYOm3Tk.jpg",
            title = "Japan",
            content =
              "Japan is an archipelago consisting
              of 6,852 islands along East Asia's
              Pacific Coast."
          ),
          article(
             image = "https://i.imgur.com/ekjFVOL.jpg",
             title = "Singapore",
             content =
               "Singapore is an island city-state
               in Southeast Asia. It's lies at the
               southern tip of the Malay Peninsula."
          )
        )
      )
    )

if (interactive()) email
```

---

block_social_links *A block of social sharing icons with links*

Description

With `block_social_links()`, we can create a block of social sharing links and links to websites, email, or RSS feeds. The function can accept as many `social_link()` calls as seen fit to email. Like all `block_*()` functions, `block_social_links()` must be placed inside of `blocks()` and the resultant `blocks` object can be provided to the `body`, `header`, or `footer` arguments of `compose_email()`.

Usage

```
block_social_links(...)
```

Arguments

...	One or more calls to `social_link()`.

Examples

```
# Create an email message with some
# articles in the `body`; in the footer,
# add some social sharing icons linking
# to web content using `block_social_links()`
email <-
  compose_email(
    body =
      blocks(
        block_title("Exciting Travel Destinations"),
        block_articles(
          article(
            image = "https://i.imgur.com/dxSXzGb.jpg",
            title = "Hong Kong",
            content =
              "Once home to fishermen and farmers,
              modern Hong Kong is a teeming,
              commercially-vibrant metropolis where
              Chinese and Western influences fuse."
          ),
          article(
            image = "https://i.imgur.com/bJzVIrG.jpg",
            title = "Australia",
            content =
              "Australia ranks as one of the best
              places to live in the world by all
              indices of income, human development,
              healthcare, and civil rights."
          )
        )
      ),
    footer =
      blocks(
        block_text("Thanks for reading! Find us here:"),
        block_social_links(
          social_link(
            service = "pinterest",
            link = "https://www.pinterest.ca/TravelLeisure/",
            variant = "color"
          ),
          social_link(
            service = "tripadvisor",
            link = "https://www.tripadvisor.ca/TravelersChoice",
            variant = "color"
```

```
          )
        )
      )
  )

if (interactive()) email
```

`block_spacer` *A spacer block*

Description

With `block_spacer()` we can more easily define an area of whitespace in a block-based layout. This function is meant to be easily combined with other `block_*()` functions. Like all `block_*()` functions, `block_spacer()` must be placed inside of `blocks()` and the resultant `blocks` object can be provided to the `body`, `header`, or `footer` arguments of `compose_email()`.

Usage

```
block_spacer()
```

Examples

```
# Create a block of two, side-by-side
# articles with two `article()` calls
# inside of `block_articles()`, itself
# placed in `blocks()`; include some
# introductory text and place extra
# space around that text (with
# `block_spacer()`)
email <-
  compose_email(
    body =
      blocks(
        block_spacer(),
        block_text(
          "These are two of the cities I visited this year.
          I liked them a lot, so, I'll visit them again!"),
        block_spacer(),
        block_articles(
          article(
            image = "https://i.imgur.com/dig0HQ2.jpg",
            title = "Los Angeles",
            content =
              "I want to live in Los Angeles.
              Not the one in Los Angeles.
              No, not the one in South California.
              They got one in South Patagonia."
          ),
```

```
            article(
              image = "https://i.imgur.com/RUvqHV8.jpg",
              title = "New York",
              content =
                "Start spreading the news.
                I'm leaving today.
                I want to be a part of it.
                New York, New York."
            )
          )
        )
      )

if (interactive()) email
```

`block_text` *A block of text*

Description

With `block_text()` we can define a text area and this can be easily combined with other `block_*()` functions. The text will take the entire width of the block and will resize according to screen width. Like all `block_*()` functions, `block_text()` must be placed inside of `blocks()` and the resultant `blocks` object can be provided to the `body`, `header`, or `footer` arguments of `compose_email()`.

Usage

```
block_text(text, align = c("left", "center", "right", "justify"))
```

Arguments

`text`	Plain text or Markdown text (via `md()`).
`align`	The text alignment to be used for this block of text. The default is `"left"`.

Examples

```
# Create a block of two, side-by-side
# articles with two `article()` calls
# inside of `block_articles()`, itself
# placed in `blocks()`; also, include some
# text at the top with `block_text()`
email <-
  compose_email(
    body =
      blocks(
        block_text(
          "These are two of the cities I visited this year.
          I liked them a lot, so, I'll visit them again!"),
```

```
          block_articles(
            article(
              image = "https://i.imgur.com/dig0HQ2.jpg",
              title = "Los Angeles",
              content =
                "I want to live in Los Angeles.
                Not the one in Los Angeles.
                No, not the one in South California.
                They got one in South Patagonia."
            ),
            article(
              image = "https://i.imgur.com/RUvqHV8.jpg",
              title = "New York",
              content =
                "Start spreading the news.
                I'm leaving today.
                I want to be a part of it.
                New York, New York."
            )
          )
        )
      )

if (interactive()) email
```

`block_title` *A block with large title text*

Description

With `block_title()` we can define a title text area and this can be easily combined with other `block_*()` functions. The title will take the entire width of the block and will resize according to screen width. Like all `block_*()` functions, `block_title()` must be placed inside of `blocks()` and the resultant `blocks` object can be provided to the `body`, `header`, or `footer` arguments of `compose_email()`.

Usage

```
block_title(title)
```

Arguments

`title`	Plain text or Markdown text (via `md()`) for the title.

Examples

```
# Create a block of two, side-by-side
# articles with two `article()` calls
# inside of `block_articles()`, itself
```

```
# placed in `blocks()`; also, include a
# title at the top with `block_title()`
email <-
  compose_email(
    body =
      blocks(
        block_title("Two Cities I Visited Recently"),
        block_articles(
          article(
            image = "https://i.imgur.com/dig0HQ2.jpg",
            title = "Los Angeles",
            content =
              "I want to live in Los Angeles.
              Not the one in Los Angeles.
              No, not the one in South California.
              They got one in South Patagonia."
          ),
          article(
            image = "https://i.imgur.com/RUvqHV8.jpg",
            title = "New York",
            content =
              "Start spreading the news.
              I'm leaving today.
              I want to be a part of it.
              New York, New York."
          )
        )
      )
    )

if (interactive()) email
```

---

compose_email *Create the email message body*

Description

The compose_email() function allows us to easily create an email message. We can incorporate character vectors into the message body, the header, or the footer.

Usage

```
compose_email(
  body = NULL,
  header = NULL,
  footer = NULL,
  title = NULL,
  ...,
  template = blastula_template
)
```

Arguments

`header, body, footer`
: The three layout sections for an email message (ordered from top to bottom). Markdown text can be supplied to each of these by using the `md()` text helper function. Alternatively, we can supply a set of `block_*()` calls enclosed within the `blocks()` function to take advantage of precomposed HTML blocks.

`title`
: The title of the email message. This is not the subject but the HTML title text which may appear in limited circumstances.

`...`
: Additional arguments to pass to the `template` function. If you're using the default template, you can use `font_family` to control the base font, and `content_width` to control the width of the main content; see `blastula_template()`. By default, the `content_width` is set to `1000px`. Using widths less than `600px` is generally not advised but, if necessary, be sure to test such HTML emails with a wide range of email clients before sending to the intended recipients. The Outlook mail client (Windows, Desktop) does not respect `content_width`.

`template`
: An email template function to use. The default is `blastula_template()`.

Value

An `email_message` object.

Examples

```
# Create a simple email message using
# Markdown-formatted text in the `body`
# and `footer` sections with the `md()`
# text helper function
email <-
  compose_email(
    body = md(
"
## Hello!

This is an email message that was generated by the blastula package.

We can use **Markdown** formatting with the `md()` function.

Cheers,

The blastula team
"),
  footer = md(
"
sent via the [blastula](https://rich-iannone.github.io/blastula) R package
")
)

# The email message can always be
# previewed by calling the object
if (interactive()) email
```

`create_smtp_creds_file` *Store SMTP credentials in a file*

Description

We can create a file with SMTP configuration and access credentials for the purpose of more easily sending email messages through `smtp_send()`. With this file produced, the credentials helper `creds_file()` can be used in the `credentials` argument of `smtp_send()`.

Usage

```
create_smtp_creds_file(
  file,
  user = NULL,
  provider = NULL,
  host = NULL,
  port = NULL,
  use_ssl = NULL
)
```

Arguments

`file`	The output filename for the credentials file.
`user`	The username for the email account. Typically, this is the email address associated with the account.
`provider`	An optional email provider shortname for autocompleting SMTP configuration details (the `host`, `port`, `use_ssl` options). Options currently include `gmail`, `outlook`, and `office365`. If nothing is provided then values for `host`, `port`, and `use_ssl` are expected.
`host, port, use_ssl`	Configuration info for the SMTP server. The `host` and `port` parameters are the address and port for the SMTP server. `use_ssl` is an option as to whether to allow the use of STARTTLS, if available: it should be `TRUE` unless you have a specific reason to set it to `FALSE`.

Examples

```
# Create a credentials file to make it
# much easier to send email out through
# Gmail with `smtp_send()`; name the
# file "gmail_creds"

# create_smtp_creds_file(
#   file = "gmail_creds",
```

```
#   user = "user_name@gmail.com",
#   provider = "gmail"
#   )
```

`create_smtp_creds_key` *Store SMTP credentials in the system's key-value store*

Description

We can set SMTP access credentials in the system-wide key-value store for the purpose of more easily sending email messages through `smtp_send()`. With this key added, the credentials helper `creds_key()` can be used in the `credentials` argument of `smtp_send()` (the `id` value is used to unambiguously refer to each key).

Usage

```
create_smtp_creds_key(
  id,
  user = NULL,
  provider = NULL,
  host = NULL,
  port = NULL,
  use_ssl = NULL,
  overwrite = FALSE
)
```

Arguments

`id`	An identifying label for the keyname. The full key name is constructed in the following way: `blastula-v1-<id>`. This `id` value is what's needed later to either use the key with `creds_key()`, or, delete the key with `delete_credential_key()`. A single, non-`NA` character, numeric, or integer value can be supplied here; the `id` will be coerced to a character value. If the `id` is supplied as a single character value, it cannot be an empty string and it cannot include hyphen characters.
`user`	The username for the email account. Typically, this is the email address associated with the account.
`provider`	An optional email provider shortname for autocompleting SMTP configuration details (the `host`, `port`, `use_ssl` options). Options currently include `gmail`, `outlook`, and `office365`. If nothing is provided then values for `host`, `port`, and `use_ssl` are expected.
`host`	Configuration info for the SMTP server. The `host` and `port` parameters are the address and port for the SMTP server. `use_ssl` is an option as to whether to allow the use of STARTTLS, if available: it should be `TRUE` unless you have a specific reason to set it to `FALSE`.

port	Configuration info for the SMTP server. The host and port parameters are the address and port for the SMTP server. use_ssl is an option as to whether to allow the use of STARTTLS, if available: it should be TRUE unless you have a specific reason to set it to FALSE.
use_ssl	Configuration info for the SMTP server. The host and port parameters are the address and port for the SMTP server. use_ssl is an option as to whether to allow the use of STARTTLS, if available: it should be TRUE unless you have a specific reason to set it to FALSE.
overwrite	An option that controls the overwriting of existing keys with the same id value. By default, this is FALSE (where overwriting is prohibited).

Details

Support for setting keys through create_smtp_creds_key() is provided through the **keyring** package. This function cannot be used without that package being available on the system. We can use install.packages("keyring") to install **keyring**.

Examples

```
# Store SMTP credentials using the
# system's secure key-value store to
# make it much easier to send email
# out through Gmail with `smtp_send()`;
# provide the `id` of "gmail_creds"

# create_smtp_creds_key(
#   id = "gmail_creds",
#   provider = "gmail",
#   user = "user_name@gmail.com",
#   )
```

---

credential_helpers *Helpers for supplying SMTP credentials*

Description

These helper functions, the credential helpers, are used to supply SMTP configuration and authorization information for the smtp_send() function. The creds_file(), creds_anonymous(), creds_key(), and creds() functions are to be used expressly with the credentials argument of smtp_send().

Usage

```
creds(user = NULL, provider = NULL, host = NULL, port = NULL, use_ssl = TRUE)

creds_anonymous(provider = NULL, host = NULL, port = NULL, use_ssl = TRUE)
```

```
creds_envvar(
  user = NULL,
  pass_envvar = "SMTP_PASSWORD",
  provider = NULL,
  host = NULL,
  port = NULL,
  use_ssl = TRUE
)

creds_key(id)

creds_file(file)
```

Arguments

`user`	The username for the email account. Typically, this is the email address associated with the account.
`provider`	An optional email provider shortname for autocompleting SMTP configuration details (the `host`, `port`, `use_ssl` options). Options currently include `gmail`, `outlook`, and `office365`. If nothing is provided then values for `host`, `port`, and `use_ssl` are expected.
`host, port, use_ssl`	Configuration info for the SMTP server. The `host` and `port` parameters are the address and port for the SMTP server; `use_ssl` is an option as to whether to use SSL: supply a `TRUE` or `FALSE` value.
`pass_envvar`	The name of the environment variable that holds the value for an email account password. This is only used in the `creds_envvar()` credential helper function.
`id`	When using the `creds_key()` credential helper, the ID value of the key (in the system key-value store) needs to be given here. This was explicitly provided when using the `create_smtp_creds_key()` function (with its own `id` argument). To get an information table with all available **blastula** keys in the key-value store, we can use the `view_credential_keys()` function.
`file`	When using the `creds_file()` credential helper, we need to specify the location of the credential file, and, this is where that is done. The credential file was ideally generated by the `create_smtp_creds_file()` function.

Details

The `creds()` credential helper allows for manual specification of SMTP configuration and authentication.

The `creds_anonymous()` credential helper is similar to `creds()` but provides convenient defaults for authenticating anonymously with an SMTP server.

The `creds_key()` credential helper gets credentials stored in the system-wide key-value store. We can set that key and the credentials data using the `create_smtp_creds_key()` function.

The `creds_file()` credential helper is used to obtain credentials from a file stored on disk. We can create that file using the `create_smtp_creds_file()` function.

The creds_envvar() credential helper reads the password from the SMTP_PASSWORD environment variable (or an environment variable name that you specify). If using environment variables for other parameters, call Sys.getenv() manually (e.g. user = Sys.getenv("SMTP_USER")).

Value

A credentials list object.

`delete_all_credential_keys`

Delete all **blastula** *credential keys*

Description

The delete_all_credential_keys() function deletes all **blastula** credential keys, giving you a clean slate. Should specific keys need to be deleted, the delete_credential_key() could be used (one call per credential key to delete). Before using delete_all_credential_keys(), it may be useful to see which keys are available in the key-value store. For that, use the view_credential_keys() function.

Usage

```
delete_all_credential_keys()
```

Details

Support for using the delete_all_credential_keys() function (and for doing any credential key management) is provided through the **keyring** package. This function cannot be used without that package being available on the system. We can use install.packages("keyring") to install **keyring**.

Examples

```
# Delete all blastula credential keys
# in the system's key-value store

# delete_all_credential_keys()
```

delete_credential_key *Delete a single* **blastula** *credential key*

Description

It may be important to delete a credential key and the `delete_credential_key()` function makes this possible. To understand which keys are available in the key-value store (and to get their `id` values), use the `view_credential_keys()` function.

Usage

```
delete_credential_key(id)
```

Arguments

`id`	The identifying label for the credential key. Use the same `id` that was used to create the key with the `create_smtp_creds_key()` function.

Details

Support for using the `delete_credential_key()` function (and for doing any credential key management) is provided through the **keyring** package. This function cannot be used without that package being available on the system. We can use `install.packages("keyring")` to install **keyring**.

Examples

```
# Delete the credential key with
# the `id` value of "outlook"

# delete_credential_key("outlook")
```

get_html_str *Get the HTML content of an email message*

Description

Get the HTML content string from an `email_message` object as a single-length character vector.

Usage

```
get_html_str(message)
```

Arguments

message	The email message object, as created by the compose_email() function. The object's class is email_message

Value

A character object containing the email message's HTML content.

---

md *Interpret input text as Markdown-formatted text*

Description

Interpret input text as Markdown-formatted text

Usage

```
md(text)
```

Arguments

text	The text that is understood to contain Markdown formatting.

Value

A character object that is tagged for a Markdown-to-HTML transformation.

A rendered HTML object.

---

prepare_rsc_example_files *Prepare example files for RStudio Connect emailing with R Markdown*

Description

A set of example files relevant to emailing with R Markdown in RStudio Connect can be spawned in a specified location. There is a set of three files that work together to provide a full report, an emailable version of that report, and a file attachment; these files are:

Usage

```
prepare_rsc_example_files(path = NULL)
```

Arguments

path	The location to which the files (in a subdirectory named "connect_examples") will be written. The path needs to exist but the aforementioned subdirectory is not required to be present.

Details

- "connect-example-main.Rmd": The main R Markdown document. Contains a report template culminating in a final R code chunk that has calls to render_connect_email() and attach_connect_email().
- "connect-example-email.Rmd": An R Markdown document that contains the email message. It is associated with the main R Markdown document by incorporating some of its content (i.e., by reusing chunk names and extending assigned values). It uses the blastula::blastula_email output type in the YAML front matter.
- "austin_home_sales.csv": A CSV file that will be included as an attachment by way of the attachments argument in the attach_connect_email() function call within the main R Markdown document.

The main report and associated email can be published by opening "connect-example-main.Rmd" and pressing the Publish button at the top-right of the Editor pane (please ensure beforehand that you are set up work with RStudio Connect). If asked "What do you want to publish?", choose the first option where only the "connect-example-main" document is published. All three files should be checked in the final dialog box, press the Publish button to publish to RStudio Connect.

There is also the single "connect-example-text-only.Rmd" file that, when published, serves as a mechanism to send a text-only email. The content of the email is specified directly in the single attach_connect_email() function call and all other text in the R Markdown file is disregarded.

---

prepare_test_message *Prepare a email test message object*

Description

Create an email test message object, which is helpful for sending a test message with the smtp_send() function.

Usage

```
prepare_test_message(incl_ggplot = FALSE, incl_image = FALSE)
```

Arguments

incl_ggplot	An option to include a ggplot plot within the body of the test message. This requires that the **ggplot2** package is installed. By default, this is FALSE.
incl_image	An option to include a test image within the body of the test message. By default, this is FALSE.

Value

An email_message object.

Examples

```
# Create a credentials file to send
# a test message via Gmail's SMTP
# (this file is named "gmail_secret")

# create_smtp_creds_file(
#   file = "gmail_secret",
#   user = "sender@email.com",
#   provider = "gmail"
# )

# Send oneself a test message to
# test these new SMTP settings and
# to ensure that the message appears
# correctly in the email client

# prepare_test_message() %>%
#   smtp_send(
#     from = "sender@email.com",
#     to = "sender@email.com",
#     subject = "Test Message",
#     credentials = creds_file(
#       file = "gmail_secret"
#       )
#     )
```

---

render_email *R Markdown render functions for the* blastula_email *output format*

Description

The render_email() and render_connect_email() functions both allow for rendering an an email message. We can supply an R Markdown document (.Rmd) with the output specified as output: blastula::blastula_email. While the render_email() and render_connect_email() functions have similar arguments, the render_connect_email() is preferred when publishing to the RStudio Connect service. It allows for the inclusion of a predefined footer that contains useful links for email recipients.

Usage

```
render_email(
  input,
  envir = parent.frame(),
  quiet = TRUE,
```

```
  output_options = list(),
  render_options = list()
)

render_connect_email(
  input,
  connect_footer = TRUE,
  envir = parent.frame(),
  quiet = TRUE,
  output_options = list(),
  render_options = list()
)
```

Arguments

input	The input file to be rendered. This should be an R Markdown document (.Rmd) with the output specified as `output: blastula::blastula_email`.
envir	The environment in which the code chunks are to be evaluated during knitting.
quiet	An option to suppress printing of the command line output from Pandoc during rendering. By default, this is set to `TRUE`.
output_options, render_options	Lists of options can be used to augment the rendering of the email message. The `output_options` list will be passed as the `output_options` argument of `rmarkdown::render()`. The `render_options` list is for providing additional arguments to `rmarkdown::render()`. By default, both lists are empty.
connect_footer	Should a prepared footer message with links be included in the rendered email? This argument is only available in the `render_connect_email()` function and is set to `TRUE` by default.

---

send_by_mailgun *Send an email message through the Mailgun API*

Description

Send an email message via the Mailgun API. This requires an account with Mailgun.

Usage

```
send_by_mailgun(message, subject = NULL, from, recipients, url, api_key)
```

Arguments

message	The email message object, as created by the `compose_email()` function. The object's class is `email_message`
subject	The subject of the email.

from	The email address of the sender. This does not have to be the same email that is associated with the account actually sending the message.
recipients	A vector of email addresses.
url	The URL for the sending domain.
api_key	The API key registered to the Mailgun service.

Examples

```
# Create a simple email message using
# Markdown formatting

# email <-
#   compose_email(
#   body = "
#   Hello!
#
#   ## This a section heading
#
#   We can use Markdown formatting \
#   to **embolden** text or to add \
#   *emphasis*. This is exciting, \
#   right?
#
#   Cheers")

# Generate a vector of recipients

# recipient_list <-
#   c("person_1@site.net",
#     "person_2@site.net")

# Send it to multiple people through
# the Mailgun API

# email %>%
#   send_by_mailgun(
#     subject = "Sent through Mailgun",
#     from = "The Sender <sender@send.org>",
#     recipients = recipient_list,
#     url = "<..mailgun_sending_domain..>",
#     api = "<..mailgun_api_key..>")
```

`smtp_send` *Send an email message through SMTP*

Description

Send an email message to one or more recipients via an SMTP server. The email message required as input to `smtp_send()` has to be created by using the `compose_email()` function. The `email_message` object can be previewed by printing the object, where the HTML preview will show how the message should appear in recipients' email clients. File attachments can be added to the email object by using the `add_attachment()` function (one call per attachment) prior to sending through this function.

Usage

```
smtp_send(
  email,
  to,
  from,
  subject = NULL,
  cc = NULL,
  bcc = NULL,
  credentials = NULL,
  creds_file = "deprecated",
  verbose = FALSE
)
```

Arguments

`email`	The email message object, as created by the `compose_email()` function. The object's class is `email_message`.
`to`	A vector of email addresses serving as primary recipients for the message. For secondary recipients, use the `cc` and `bcc` arguments. A named character vector can be used to specify the recipient names along with the their email address (e.g., `c("Jane Doe" = "jane_doe@example.com")`).
`from`	The email address of the sender. Often this needs to be the same email address that is associated with the account actually sending the message. As with `to`, `cc`, and `bcc`, we can either supply a single email address or use a named character vector with the sender name and email address (e.g., `c("John Doe" = "john_doe@example.com")`).
`subject`	The subject of the message, which is usually a brief summary of the topic of the message. If not provided, an empty string will be used (which is handled differently by email clients).
`cc, bcc`	A vector of email addresses for sending the message as a carbon copy or blind carbon copy. The CC list pertains to recipients that are to receive a copy of a message that is addressed primarily to others. The CC listing of recipients is visible to all other recipients of the message. The BCC list differs in that those recipients will be concealed from all other recipients (including those on the BCC list). A named character vector can be used to specify the recipient names along with the their email address (e.g., `c("Joe Public" = "joe_public@example.com")`).
`credentials`	One of three credential helper functions must be used here: (1) `creds()`, (2) `creds_key()`, or (3) `creds_file()`. The first, `creds()`, allows for a manual

specification of SMTP configuration and credentials within that helper function. This is the most secure method for supplying credentials as they aren't written to disk. The `creds_key()` function is used if credentials are stored in the system-wide key-value store, through use of the `create_smtp_creds_key()` function. The `creds_file()` helper function relies on a credentials file stored on disk. Such a file is created using the `create_smtp_creds_file()` function.

`creds_file` An option to specify a credentials file. As this argument is deprecated, please consider using `credentials = creds_file(<file>)` instead.

`verbose` Should verbose output from the internal curl `send_mail()` call be printed? While the username and password will likely be echoed during the exchange, such information is encoded and won't be stored on the user's system.

Details

We can avoid re-entering SMTP configuration and credentials information by retrieving this information either from disk (with the file generated by use of the `create_smtp_creds_file()` function), or, from the system's key-value store (with the key set by the `create_smtp_creds_key()` function).

Examples

```
# Before sending out an email through
# SMTP, we need an `email_message`
# object; for the purpose of a simple
# example, we can use the function
# `prepare_test_message()` to create
# a test version of an email (although
# we'd normally use `compose_email()`)
email <- prepare_test_message()

# The `email` message can be sent
# through the `smtp_send()` function
# so long as we supply the appropriate
# credentials; The following three
# examples provide scenarios for both
# the creation of credentials and their
# retrieval within the `credentials`
# argument of `smtp_send()`

# (1) Providing the credentials info
# directly via the `creds()` helper
# (the most secure means of supplying
# credentials information)

# email %>%
#   smtp_send(
#     from = "sender@email.com",
#     to = "recipient@email.com",
#     credentials = creds(
#       provider = "gmail",
#       user = "sender@email.com")
```

```
#   )

# (2) Using a credentials key (with
# the `create_smtp_creds_key()` and
# `creds_key()` functions)

# create_smtp_creds_key(
#  id = "gmail",
#  user = "sender@email.com",
#  provider = "gmail"
#  )

# email %>%
#   smtp_send(
#     from = "sender@email.com",
#     to = "recipient@email.com",
#     credentials = creds_key(
#       "gmail"
#       )
#   )

# (3) Using a credentials file (with
# the `create_smtp_creds_file()` and
# `creds_file()` functions)

# create_smtp_creds_file(
#  file = "gmail_secret",
#  user = "sender@email.com",
#  provider = "gmail"
#  )

# email %>%
#   smtp_send(
#     from = "sender@email.com",
#     to = "recipient@email.com",
#     credentials = creds_file(
#       "gmail_secret")
#   )
```

`social_link` *Specify the components of a social link*

Description

The `social_link()` function is used exclusively within `block_social_links()` with as many calls as the number of social sharing icons/links required. By providing a supported `service` name, a hosted icon image can be used. A `link` must be provided; it will be part of social sharing icon. All icons are rounded, transparent, and consist of a single color, or level of gray.

Usage

```
social_link(service, link, variant = NULL, alt = NULL)
```

Arguments

service	Either the name of a social sharing service or either of website, email, or rss.
link	The relevant link to content on the service.
variant	The variant of the icon to use. Options include bw (black and white, the default), color, dark_gray, gray, and light_gray.
alt	Text description of image passed to the alt attribute inside of the image (<img>) tag for use when image loading is disabled and on screen readers. If not supplied, then the name of the service will be used as alt text.

Details

The following social sharing services have hosted icons available:

- Twitter - Micro-blogging internet service.
- GitHub - Web-based hosting service for software development projects using Git.
- Facebook - Global online social networking service.
- Instagram - Online photo-sharing and social networking service.
- LinkedIn - Social networking service for people in professional occupations.
- YouTube - Video-sharing service owned by Google.
- Vimeo - An ad-free open video platform.
- Behance - A site for self-promotion of design projects.
- Dribbble - Online community for showcasing user-made artwork.
- Pinterest - Photo-sharing and publishing website for discovering interesting things.
- 500px - Online platform for photographers to gain global exposure.
- Yelp - Local-search service powered by crowd-sourced reviews.
- TripAdvisor - Travel and restaurant website with reviews and accommodation bookings.
- WordPress - Blogging platform and content management system.
- Blogger - A blog-publishing service hosted by Google.
- Tumblr - Micro-blogging and social networking website.
- Deezer - Web-based music streaming service.
- SoundCloud - A music sharing website and publishing tool for music distribution.
- Meetup - A service used to organize online groups that host in-person events.
- Etsy - An e-commerce website focused on handmade or vintage items and supplies.
- Reddit - A social news aggregation, web content rating, and discussion website.
- Stack Overflow - Question and answer site for professional and enthusiast programmers.
- Youku - A video hosting service for user-made and professionally produced videos.
- Sina Weibo - Micro-blogging website and one of the biggest social media platforms in China.
- QQ - Instant messaging software service developed by Tencent.
- Douban - A Chinese social networking service with a reputation for high-quality content.

Examples

```
# Create an email message with some
# articles in the `body`; in the footer,
# add some social sharing icons linking
# to web content
email <-
  compose_email(
    body =
      blocks(
        block_title("Exciting Travel Destinations"),
        block_articles(
          article(
            image = "https://i.imgur.com/dxSXzGb.jpg",
            title = "Hong Kong",
            content =
              "Once home to fishermen and farmers,
              modern Hong Kong is a teeming,
              commercially-vibrant metropolis where
              Chinese and Western influences fuse."
          ),
          article(
            image = "https://i.imgur.com/bJzVIrG.jpg",
            title = "Australia",
            content =
              "Australia ranks as one of the best
              places to live in the world by all
              indices of income, human development,
              healthcare, and civil rights."
          )
        )
      ),
    footer =
      blocks(
        block_text("Thanks for reading! Find us here:"),
        block_social_links(
          social_link(
            service = "pinterest",
            link = "https://www.pinterest.ca/TravelLeisure/",
            variant = "color"
          ),
          social_link(
            service = "tripadvisor",
            link = "https://www.tripadvisor.ca/TravelersChoice",
            variant = "color"
          )
        )
      )
  )

if (interactive()) email
```

suppress_scheduled_email

Suppress any scheduled emailing in RStudio Connect

Description

This function is useful for suppressing the scheduled emailing of a published R Markdown document. It can be invoked anywhere in the R Markdown document and is useful in a conditional statement, where the result of the condition determines whether or not email suppression should occur.

Usage

```
suppress_scheduled_email(suppress = TRUE)
```

Arguments

suppress	A logical value for whether email suppression should occur after publication. By default, this is TRUE.

Details

Since this function needs to be invoked within an R Markdown document, the chunk option `echo=FALSE` is useful here (so that viewers of the rendered document don't have to unnecessarily read code related to email suppression). While the output is invisible, any errors related to the use of this function will be visible to the author.

view_credential_keys *View all available* **blastula** *credential keys*

Description

To understand which keys have been set using the `create_smtp_creds_key()` function (and how they are identified), we can use the `view_credential_keys()` function. What's provided is a tibble with three columns: `id`, `key_name`, and `username`.

Usage

```
view_credential_keys()
```

Details

Support for using the `view_credential_keys()` function (and for doing any credential key management) is provided through the **keyring** package. This function cannot be used without that package being available on the system. We can use `install.packages("keyring")` to install **keyring**.

Examples

```
# View the available SMTP credentials
# that are in the system's secure
# key-value store; the `id` values
# in the returned tibble provide what's
# necessary to send email through
# `smtp_send()` and the `creds_key()`
# credential helper function

# view_credential_keys()
```

`%>%` *The magrittr pipe*

Description

The `blastula` package uses the pipe function, `\%>\%`, to turn function composition into a series of imperative statements.

Index

%>%, 39

add_attachment, 3
add_attachment(), *33*
add_cta_button, 3
add_ggplot, 4
add_image, 6
add_imgur_image, 7
add_readable_time, 8
article, 9
attach_connect_email, 10
attach_connect_email(), *29*

blastula_email, 11
blastula_template, 12
blastula_template(), *21*
block_articles, 14
block_social_links, 15
block_spacer, 17
block_text, 18
block_title, 19
blocks, 13
blocks(), *21*

compose_email, 20
compose_email(), *3*, *12*, *33*
create_smtp_creds_file, 22
create_smtp_creds_file(), *25*, *34*
create_smtp_creds_key, 23
create_smtp_creds_key(), *25*, *27*, *34*, *38*
credential_helpers, 24
creds *(*credential_helpers*)*, 24
creds(), *24*, *25*, *33*
creds_anonymous *(*credential_helpers*)*, 24
creds_anonymous(), *24*, *25*
creds_envvar *(*credential_helpers*)*, 24
creds_envvar(), *25*, *26*
creds_file *(*credential_helpers*)*, 24
creds_file(), *22*, *24*, *25*, *33*, *34*
creds_key *(*credential_helpers*)*, 24
creds_key(), *23–25*, *33*, *34*

delete_all_credential_keys, 26
delete_credential_key, 27
delete_credential_key(), *23*, *26*

get_html_str, 27

htmltools::HTML(), *13*
htmltools::tags(), *13*

md, 28
md(), *3*, *18*, *19*, *21*

prepare_rsc_example_files, 28
prepare_test_message, 29

render_connect_email *(*render_email*)*, 30
render_connect_email(), *10*, *29*
render_email, 30
render_email(), *10*
rmarkdown::html_document(), *12*
rmarkdown::includes(), *12*

send_by_mailgun, 31
smtp_send, 32
smtp_send(), *3*, *22–24*
social_link, 35
suppress_scheduled_email, 38
Sys.getenv(), *26*

view_credential_keys, 38
view_credential_keys(), *25–27*