Package 'emayili'

August 30, 2023

Type Package

Title Send Email Messages

Version 0.7.18

Description A light, simple tool for sending emails with minimal dependencies.

URL https://datawookie.github.io/emayili/

BugReports https://github.com/datawookie/emayili/issues

License GPL-3

Language en-GB

Imports base64enc, commonmark, curl (>= 4.0), digest, dplyr, glue, htmltools, httr, logger, magrittr (>= 2.0.1), mime, purrr, rmarkdown, rvest, stringi, stringr, tidyr, urltools, xfun, xml2

Suggests cld2, cld3, gpg, here, jinjar, lintr, memoise, testthat (>= 2.1.0), roxygen2, showtext, Microsoft365R

SystemRequirements The function render() requires Pandoc (http://pandoc.org). To use PGP/GnuPG encryption requires gpg.

Encoding UTF-8

RoxygenNote 7.2.3

KeepSource true

NeedsCompilation no

Author Andrew B. Collier [aut, cre, cph],
Matt Dennis [ctb],
Antoine Bichat [ctb] (<https://orcid.org/0000-0001-6599-7081>),
Daniel Fahey [ctb],
Johann R. Kleinbub [ctb],
Panagiotis Moulos [ctb],
Swechhya Bista [ctb],
Colin Fay [ctb] (<https://orcid.org/0000-0001-7343-1846>)

Maintainer Andrew B. Collier <andrew.b.collier@gmail.com>

Repository CRAN

Date/Publication 2023-08-30 16:50:33 UTC

R topics documented:

address . . . 3
addresses . . . 4
after.envelope . . . 5
as.address . . . 6
as.character.address . . . 6
as.character.envelope . . . 7
as.character.header . . . 7
as.character.MIME . . . 8
attachment . . . 8
c.address . . . 9
cleave . . . 10
comments . . . 10
compare . . . 11
compliant . . . 11
cutoff . . . 12
display . . . 13
domain . . . 13
encrypt . . . 14
envelope . . . 15
format.address . . . 16
html . . . 17
keywords . . . 18
length.address . . . 19
local . . . 20
mime-parameters . . . 20
normalise . . . 21
parties . . . 22
precedence . . . 22
print.address . . . 23
print.envelope . . . 24
qp . . . 24
raw . . . 25
receipt . . . 26
render . . . 26
response . . . 28
sensitivity . . . 29
server . . . 30
subject . . . 34
template . . . 36
text . . . 37
validate . . . 38

Index **40**

address *Email Address*

Description

Create an address object which represents an email address.

Usage

```
address(
  email = NA,
  display = NA,
  local = NA,
  domain = NA,
  normalise = TRUE,
  validate = FALSE
)
```

Arguments

email	Email address.
display	Display name.
local	Local part of email address.
domain	Domain part of email address.
normalise	Whether to try to normalise address to RFC-5321 requirements.
validate	Whether to validate the address.

Value

An address object, representing an email address.

Examples

```
address("gerry@gmail.com")
address("gerry@gmail.com", "Gerald")
address("gerry@gmail.com", "Gerald Durrell")
# Display name in "Last, First" format.
address("gerry@gmail.com", "Durrell, Gerald")
# Display name contains non-ASCII characters.
address("hans@gmail.com", "Hansjörg Müller")
```

addresses *Add address fields to message*

Description

Add address fields to message

Usage

```
to(msg, ..., append = TRUE)

cc(msg, ..., append = TRUE)

bcc(msg, ..., append = TRUE)

from(msg, addr = NULL)

reply(msg, addr = NULL)

return_path(msg, addr = NULL)

sender(msg, addr = NULL)
```

Arguments

msg	A message object.
...	Addresses.
append	Whether to append or replace addresses.
addr	Single address.

Value

A message object.

Examples

```
# Populating the To field.
msg <- envelope()
msg %>% to("bob@gmail.com, alice@yahoo.com")
msg %>% to("bob@gmail.com", "alice@yahoo.com")
msg %>% to(c("bob@gmail.com", "alice@yahoo.com"))

# Populating the Cc field.
msg <- envelope()
msg %>% cc("bob@gmail.com, alice@yahoo.com")
msg %>% cc("bob@gmail.com", "alice@yahoo.com")
msg %>% cc(c("bob@gmail.com", "alice@yahoo.com"))
```

```
# Populating the Bcc field.
msg <- envelope()
msg %>% bcc("bob@gmail.com, alice@yahoo.com")
msg %>% bcc("bob@gmail.com", "alice@yahoo.com")
msg %>% bcc(c("bob@gmail.com", "alice@yahoo.com"))

msg <- envelope()

# Populating the From field.
msg %>% from("craig@gmail.com")

# Populating the Reply-To field.
msg <- envelope()
msg %>% reply("gerry@gmail.com")

# Populating the Return-Path field.
msg <- envelope()
msg %>% return_path("bounced-mail@devnull.org")

# Populating the Sender field.
msg <- envelope()
msg %>% sender("on_behalf_of@gmail.com")
```

`after.envelope` *Append children to message*

Description

Append children to message

Usage

```
## S3 method for class 'envelope'
after(x, child)
```

Arguments

`x`	Message object
`child`	A child to be appended

as.address *Create an address object*

Description

Create an address object

Usage

```
as.address(addr, validate = FALSE)
```

Arguments

addr	An email address.
validate	Whether to validate the address.

Value

A list of address objects.

Examples

```
as.address("gerry@gmail.com")
as.address("Gerald <gerry@gmail.com>")
as.address(c("Gerald <gerry@gmail.com>", "alice@yahoo.com", "jim@aol.com"))
as.address("Gerald <gerry@gmail.com>, alice@yahoo.com, jim@aol.com")
as.address("Durrell, Gerald <gerry@gmail.com>")
```

as.character.address *Convert address object to character*

Description

If display name is specified as "Last, First" then the display name will be quoted.

Usage

```
## S3 method for class 'address'
as.character(x, ...)
```

Arguments

x	An address object.
...	Further arguments passed to or from other methods.

Value

A character vector.

as.character.envelope *Create formatted message.*

Description

Accepts a message object and formats it as a MIME document.

Usage

```
## S3 method for class 'envelope'
as.character(x, ..., details = TRUE, encode = FALSE)
```

Arguments

x	A message object.
...	Further arguments passed to or from other methods.
details	Whether or not to display full message content.
encode	Whether to encode headers.

Value

A formatted message object.

as.character.header *Create formatted header.*

Description

Accepts a header object and formats it as a header field.

Usage

```
## S3 method for class 'header'
as.character(x, width = 30, ...)
```

Arguments

x	A header object.
width	The width of the head name field.
...	Further arguments passed to or from other methods.

Value

A formatted header field.

as.character.MIME *Convert MIME object to character vector*

Description

Convert MIME object to character vector

Usage

```
## S3 method for class 'MIME'
as.character(x, ...)
```

Arguments

x	MIME object
...	Further arguments passed to or from other methods.

attachment *Add attachments to a message object*

Description

Add attachments to a message object

Usage

```
attachment(
  msg,
  path,
  name = NA,
  type = NA,
  cid = NA,
  disposition = "attachment"
)
```

Arguments

msg	A message object.
path	Path to file.
name	Name to be used for attachment (defaults to base name of path).
type	MIME type or *NA*, which will result in a guess based on file extension.
cid	Content-ID or NA.
disposition	How is attachment to be presented ("inline" or "attachment")?

Value

A message object.

Examples

```
path_mtcars <- tempfile(fileext = ".csv")
path_scatter <- tempfile(fileext = ".png")
path_cats <- system.file("cats.jpg", package = "emayili")

write.csv(mtcars, path_mtcars)

png(path_scatter)
plot(1:10)
dev.off()

msg <- envelope() %>%
  attachment(path_mtcars) %>%
  # This attachment will have file name "cats.jpg".
  attachment(path_cats, name = "cats.jpg", type = "image/jpeg") %>%
  attachment(path_scatter, cid = "scatter")

file.remove(path_scatter, path_mtcars)
```

---

c.address *Concatenate address objects*

Description

Concatenate address objects

Usage

```
## S3 method for class 'address'
c(...)
```

Arguments

...	Address objects to be concatenated.

Value

An address object.

Examples

```
gerry <- as.address("Gerald <gerry@gmail.com>")
alice <- address("alice@yahoo.com")
jim <- address("jim@aol.com", "Jim")
c(gerry, alice)
c(gerry, c(alice, jim))
```

cleave *Split a compound address object*

Description

Split a compound address object

Usage

```
cleave(addr)
```

Arguments

addr	An address object.

Value

A list of address objects, each of which contains only a single address.

Examples

```
cleave(as.address(c("foo@yahoo.com", "bar@yahoo.com")))
```

comments *Add or query comments of message.*

Description

Add or query comments of message.

Usage

```
comments(msg, comments = NULL)
```

Arguments

msg	A message object.
comments	Comments for the message.

Value

A message object or the comments of the message object (if comments is NULL).

See Also

subject

Examples

```
# Create a message and set the comments.
msg <- envelope() %>% comments("This is a comment")

# Retrieve the comments for a message.
comments(msg)
```

---

compare *Compare vectors*

Description

Returns TRUE wherever elements are the same (including NA), and FALSE everywhere else.

Usage

```
compare(lhs, rhs)
```

Arguments

lhs	LHS of operation.
rhs	RHS of operation.

Value

A Boolean value.

---

compliant *Tests whether an email address is syntactically correct*

Description

Checks whether an email address conforms to the syntax rules.

Usage

```
compliant(addr, error = FALSE)
```

Arguments

addr	An email address.
error	Whether to create an error if not compliant.

Details

An email address may take either of the following forms:

- `local@domain` or
- `Display Name <local@domain>`.

Value

A Boolean.

Examples

```
compliant("alice@example.com")
compliant("alice?example.com")
```

---

cutoff *Set or query message expiry or reply-by time*

Description

Functions to specify the time at which a message expires or by which a reply is requested.

Usage

```
expires(msg, datetime = NULL, tz = "")

replyby(msg, datetime = NULL, tz = "")
```

Arguments

`msg`	A message object.
`datetime`	Date and time.
`tz`	A character string specifying the time zone.

Details

Manipulate the `Expires` and `Reply-By` fields as specified in RFC 2156.

Value

A message object.

Examples

```
envelope() %>%
  expires("2030-01-01 13:25:00", "UTC")
envelope() %>%
  replyby("2021-12-25 06:00:00", "GMT")
```

display *Extract display name*

Description

Extracts the display name from an email address.

Usage

```
display(addr)
```

Arguments

addr	An address object.

Value

The display name or NA.

Examples

```
gerry <- as.address("Gerald <gerry@gmail.com>")
display(gerry)
```

domain *Extract domain of email address*

Description

Extract domain of email address

Usage

```
domain(addr)
```

Arguments

addr	An address object.

Value

A character vector.

Examples

```
domain("alice@example.com")
```

encrypt *Encrypt or sign a message*

Description

Specify whether the message should be encrypted, signed or have a public key attached.

Usage

```
encrypt(msg, encrypt = TRUE, sign = TRUE, public_key = TRUE)

signature(msg, public_key = TRUE)
```

Arguments

msg	A message object.
encrypt	Whether to encrypt the message. If TRUE then the entire message will be encrypted using the private key of the sender.
sign	Whether to sign the message. If TRUE then the entire message will be signed using the private key of the sender.
public_key	Whether to attach a public key. If TRUE then the public key of the sender will be attached.

Details

The signature() function will add a digital signature to a message. It will also optionally include a copy of the sender's public key.

The encrypt() function will encrypt the contents of a message using the public key(s) of the recipient(s). It can also add a digital signature to the message (this is the default behaviour) and include a copy of the sender's public key. Signing happens *before* encryption, so the digital signature will only be accessible once the message has been decrypted. If a recipient no longer has access to their private key or their email client is unable to decrypt the message then they will not be able to access the message contents.

Value

A message object.

Examples

```
## Not run:
msg <- envelope(
  from = "flotilla@kriegsmarine.gov",
  to = "schunk@u-boat.com",
  subject = "Top Secret Message",
  text = "Immediate readiness. There are indications that the invasion has begun."
)
```

```
# Encrypt and sign the message.
msg %>% encrypt()
# Only encrypt the message.
msg %>% encrypt(sign = FALSE)
# Only sign the message.
msg %>% signature()
msg %>% encrypt(encrypt = FALSE)

## End(Not run)
```

`envelope` *Create a message.*

Description

Create a message.

Usage

```
envelope(
  to = NULL,
  from = NULL,
  cc = NULL,
  bcc = NULL,
  reply = NULL,
  subject = NULL,
  importance = NULL,
  priority = NULL,
  text = NULL,
  html = NULL,
  encrypt = FALSE,
  sign = FALSE,
  public_key = FALSE
)
```

Arguments

`to`	See `to()`.
`from`	See `from()`.
`cc`	See `cc()`.
`bcc`	See `bcc()`.
`reply`	See `reply()`.
`subject`	See `subject()`.
`importance`	See `importance()`.
`priority`	See `priority()`.

text	See text().
html	See html().
encrypt	Whether to encrypt the message. If TRUE then the entire message will be encrypted using the private key of the sender.
sign	Whether to sign the message. If TRUE then the entire message will be signed using the private key of the sender.
public_key	Whether to attach a public key. If TRUE then the public key of the sender will be attached.

Value

A message object.

See Also

subject(), from(), to(), cc(), bcc(), reply() and encrypt().

Examples

```
# Create an (empty) message object.
#
msg <- envelope()

# Create a complete message object, specifying all available fields.
#
envelope(
  to = "bob@gmail.com",
  from = "craig@gmail.com",
  cc = "alex@gmail.com",
  bcc = "shannon@gmail.com",
  reply = "craig@yahoo.com",
  importance = "high",
  priority = "urgent",
  subject = "Hiya!",
  text = "Hi Bob, how are you?"
)
```

---

format.address *Encode email addresses in a common format*

Description

Encode email addresses in a common format

Usage

```
## S3 method for class 'address'
format(x, quote = TRUE, encode = FALSE, ...)
```

Arguments

x	An address object.
quote	Whether to quote display name (only relevant if display name is given in "Last, First" format).
encode	Whether to encode headers.
...	Further arguments passed to or from other methods.

Value

A character vector.

---

html *Add an HTML body to a message object.*

Description

Add an HTML body to a message object.

Usage

```
html(
  msg,
  content,
  disposition = "inline",
  charset = "utf-8",
  encoding = NA,
  css_files = c(),
  language = FALSE,
  interpolate = TRUE,
  .open = "{{",
  .close = "}}",
  .envir = NULL
)
```

Arguments

msg	A message object.
content	A string of message content.
disposition	Should the content be displayed inline or as an attachment? Valid options are "inline" and "attachment". If set to NA then will guess appropriate value.
charset	What character set is used. Most often either "UTF-8" or "ISO-8859-1".
encoding	How content is transformed to ASCII. Options are "7bit", "quoted-printable" and "base64". Use NA or NULL for no (or "identity") encoding.
css_files	Extra CSS files.

language	Language of content. If FALSE then will not include language field. If TRUE then will attempt to auto-detect language. Otherwise will use the specified language.
interpolate	Whether or not to interpolate into input using glue.
.open	The opening delimiter.
.close	The closing delimiter.
.envir	Environment used for glue interpolation. Defaults to parent.frame().

Value

A message object.

See Also

text, render

Examples

```
# Inline HTML message.
envelope() %>% html("<b>Hello!</b>")

# Read HTML message from a file.
htmlfile <- tempfile(fileext = ".html")
cat("<p>Hello!</p>\n", file = htmlfile)
envelope() %>% html(htmlfile)

# You can pass a vector of character. Components will be separated by a
# "\n".
envelope() %>% html(c("<b>Hello</b>", "<p>World!</p>"))

# You can also pass a tagList from {htmltools}.
if (requireNamespace("htmltools", quietly = TRUE)) {
  library(htmltools)
  envelope() %>% html(tagList(h2("Hello"), p("World!")))
}
```

---

keywords *Add or query keywords of message.*

Description

Add or query keywords of message.

Usage

```
keywords(msg, ..., append = FALSE)
```

Arguments

msg	A message object.
...	Keywords.
append	Whether to append or replace keywords.

Value

A message object or the comments of the message object (if comments is NULL).

See Also

to, from, cc, bcc and reply

Examples

```
# Create a message and set the keywords.
envelope() %>% keywords("newsletter, marketing")
envelope() %>% keywords("newsletter", "marketing")
envelope() %>% keywords(c("newsletter", "marketing"))

# Retrieve the keywords for a message.
msg <- envelope() %>% keywords("newsletter, marketing")
keywords(msg)
```

---

length.address *Length of address object*

Description

Length of address object

Usage

```
## S3 method for class 'address'
length(x)
```

Arguments

x	An address object.

Value

A character vector.

local *Extract local part of email address*

Description

Extract local part of email address

Usage

```
local(addr)
```

Arguments

addr	An address object.

Value

A character vector.

Examples

```
local("alice@example.com")
```

mime-parameters *Parameters for MIME functions*

Description

These are parameters which occur commonly across functions for components of a MIME document.

Arguments

content	A string of message content.
disposition	Should the content be displayed inline or as an attachment? Valid options are "inline" and "attachment". If set to NA then will guess appropriate value.
charset	What character set is used. Most often either "UTF-8" or "ISO-8859-1".
encoding	How content is transformed to ASCII. Options are "7bit", "quoted-printable" and "base64". Use NA or NULL for no (or "identity") encoding.
language	Language of content. If FALSE then will not include language field. If TRUE then will attempt to auto-detect language. Otherwise will use the specified language.
description	Description of content.
name	Name used when downloading file.
filename	Path to a file.

boundary	Boundary string.
type	The MIME type of the content.
children	List of child MIME objects.
interpolate	Whether or not to interpolate into input using glue.
.open	The opening delimiter.
.close	The closing delimiter.
.envir	Environment used for glue interpolation. Defaults to parent.frame().

`normalise` *Normalise email address*

Description

Ensure that email address is in a standard format.

Usage

```
normalise(email)
```

Arguments

email	An email address.

Details

Performs the following transformations:

- lowercase the domain part
- replace some Unicode characters with compatible equivalents. See Unicode equivalence.

Value

An email address.

Examples

```
normalise("bob@GMAIL.COM")
```

parties *Extract sender and recipient(s)*

Description

Extract sender and recipient(s)

Usage

```
parties(msg)
```

Arguments

msg	A message object.

Value

A tibble.

Examples

```
msg <- envelope() %>%
  from("Gerald <gerald@gmail.com>") %>%
  to(c("bob@gmail.com", "alice@yahoo.com")) %>%
  cc("Craig     < craig@gmail.com>") %>%
  bcc("  Erin   <erin@yahoo.co.uk    >")

parties(msg)
```

precedence *Add fields for message importance and priority*

Description

Functions to influence message delivery speed and importance.

Usage

```
priority(msg, priority = NULL)

importance(msg, importance = NULL)
```

Arguments

msg	A message object.
priority	Priority level. One of "non-urgent", "normal", or "urgent".
importance	Importance level. One of "low", "normal", or "high".

Details

The `priority()` function adds the `Priority` header field which gives a hint to influence transmission speed and delivery. Valid values are `"non-urgent"`, `"normal"`, and `"urgent"`. The non-standard `X-Priority` header field is similar, for which valid values are `1` (Highest), `2` (High), `3` (Normal, the default), `4` (Low), and `5` (Lowest).

The `importance()` function adds the `Importance` header field, which gives a hint to the message recipient about how important the message is. Does not influence delivery speed.

Value

A message object.

Examples

```
# How rapidly does the message need to be delivered?
#
envelope() %>%
  subject("Deliver this immediately!") %>%
  priority("urgent")

envelope(priority = "non-urgent") %>%
  subject("No rush with this.")

# How much attention should be paid by recipient?
#
envelope() %>%
  subject("Read this immediately!") %>%
  importance("high")

envelope(importance = "low") %>%
  subject("Not important at all. Just delete.")
```

`print.address` *Print an address object*

Description

If display name is specifed as "Last, First" then the display name will be quoted.

Usage

```
## S3 method for class 'address'
print(x, ...)
```

Arguments

`x`	An `address` object.
`...`	Further arguments passed to or from other methods.

Examples

```
gerry <- as.address("gerry@gmail.com")
print(gerry)
```

---

print.envelope *Print a message object*

Description

The message body will be printed if `details` is `TRUE` or if the `envelope_details` option is `TRUE`.

Usage

```
## S3 method for class 'envelope'
print(x, details = NA, ...)
```

Arguments

`x`	A message object.
`details`	Whether or not to display full message content.
`...`	Further arguments passed to or from other methods.

Examples

```
msg <- envelope() %>% text("Hello, World!")

print(msg)
print(msg, details = TRUE)

options(envelope_details = TRUE)
print(msg)
```

---

qp *Quoted-Printable encoding*

Description

Encode to and decode from Quoted-Printable encoding.

Usage

```
qp_encode(x, crlf = CRLF)

qp_decode(x)
```

Arguments

x	A string for encoding or decoding.
crlf	End-of-line characters.

Value

An encoded string for qp_encode() or a decoded string for qp_decode().

Examples

```
qp_encode("Mieux vaut être seul que mal accompagné.")
qp_decode("Mieux vaut =C3=AAtre seul que mal accompagn=C3=A9.")
```

---

raw *Extract raw email address*

Description

Strips the display name off an email address (if present).

Usage

```
raw(addr)
```

Arguments

addr	An address object.

Value

A raw email address.

Examples

```
gerry <- as.address("Gerald <gerry@gmail.com>")
raw(gerry)
```

receipt *Request read or delivery receipts*

Description

Request the recipient to acknowledge that they have read the message. Inserts MDN (Message Disposition Notification) header entries.

Usage

```
request_receipt_read(msg, addr = NULL)

request_receipt_delivery(msg, addr = NULL)
```

Arguments

msg	A message object.
addr	Single address (optional). If address is not specified then will use sender address.

Value

A message object.

render *Render Markdown into email*

Description

Render either Plain Markdown or R Markdown directly into the body of an email.

If input is a file then it will be interpreted as R Markdown it its extension is either "Rmd" or "Rmarkdown". Otherwise it will be processed as Plain Markdown.

Usage

```
render(
  msg,
  input,
  params = NULL,
  squish = TRUE,
  css_files = c(),
  include_css = c("rmd", "bootstrap"),
  language = FALSE,
  interpolate = TRUE,
  .open = "{{",
```

```
  .close = "}}",
  .envir = NULL
)
```

Arguments

msg	A message object.
input	The input Markdown file to be rendered or a character vector of Markdown text.
params	A list of named parameters that override custom parameters specified in the YAML front-matter.
squish	Whether to clean up whitespace in rendered document.
css_files	Extra CSS files.
include_css	Whether to include rendered CSS from various sources ("rmd" — native R Markdown CSS; "bootstrap" — Bootstrap CSS).
language	Language of content. If FALSE then will not include language field. If TRUE then will attempt to auto-detect language. Otherwise will use the specified language.
interpolate	Whether or not to interpolate into input using glue.
.open	The opening delimiter.
.close	The closing delimiter.
.envir	Environment used for glue interpolation. Defaults to parent.frame().

Value

A message object.

Plain Markdown

Plain Markdown is processed with commonmark::markdown_html().

R Markdown

R Markdown is processed with rmarkdown::render().

Regardless of what output type is specified in the input file, render() will always use the "html_document" output format.

Rending an R Markdown document can result in a lot of CSS. When all of the CSS is included in the HTML <head> and sent to GMail it can result in a message which is not correctly displayed inline in the Gmail web client. To get around this you can specify include_css = FALSE. This will mean that some styling will not be present in the resulting message, but that the message content will be correctly rendered inline.

See Also

text, html

Examples

```
# Plain Markdown

markdown <- "[This](https://www.google.com) is a link."
filename <- "message.md"

# Render from Markdown in character vector.
msg <- envelope() %>% render(markdown)

# Create a file containing Markdown
cat(markdown, file = filename)

# Render from Markdown in file.
msg <- envelope() %>% render(filename)

# Cleanup.
file.remove(filename)

# R Markdown

filename <- "gh-doc.Rmd"

# Create an Rmd document from template.
rmarkdown::draft(
  filename,
  template = "github_document",
  package = "rmarkdown",
  edit = FALSE
)

# Check for suitable version of Pandoc (https://pandoc.org/).
#
# Need to have version 2.0 or greater to support required --quiet option.
#
pandoc <- rmarkdown::find_pandoc()
suitable_pandoc <- !is.null(pandoc$dir) && grepl("^2", pandoc$version)

# Render from Rmd file.
if (suitable_pandoc) {
  msg <- envelope() %>%
    render(filename, include_css = c("rmd", "bootstrap"))
}

# Cleanup.
file.remove(filename)
```

`response` *Add In-Reply-To and References header fields*

Description

Add In-Reply-To and References header fields

Usage

```
inreplyto(msg, msgid, subject_prefix = "Re: ")

references(msg, msgid, subject_prefix = "Re: ")
```

Arguments

msg	A message object.
msgid	A message ID. This would be the contents of the Message-ID field from another message.
subject_prefix	Prefix to add to subject. If specified will be prepended onto the Subject field. Set to NULL if not required.

Value

A message object.

Examples

```
envelope() %>% inreplyto("<6163c08e.1c69fb81.65b78.183c@mx.google.com>")
# Now for German.
envelope() %>%
  inreplyto("6163c08e.1c69fb81.65b78.183c@mx.google.com", "AW: ")
# And also for Danish, Norwegian and Swedish (but not Finnish!).
envelope() %>%
  references("6163c08e.1c69fb81.65b78.183c@mx.google.com", "SV: ")
```

---

sensitivity *Set or query message sensitivity*

Description

Manipulate the Sensitivity field as specified in RFC 2156.

Usage

```
sensitivity(msg, sensitivity = NULL)
```

Arguments

msg	A message object.
sensitivity	Sensitivity level. One of "personal", "private", or "company-confidential".

Value

A message object.

Examples

```
# Not sensitive.
envelope() %>%
  subject("Your daily dose of spam")

# Sensitive personal message.
envelope() %>%
  subject("The results from your test") %>%
  sensitivity("personal")

# Sensitive private message.
envelope() %>%
  subject("Your OTP (don't show this to anybody!") %>%
  sensitivity("private")

# Sensitive business message.
envelope() %>%
  subject("Top Secret Strategy Document") %>%
  sensitivity("company-confidential")
```

---

server *Create a SMTP server object.*

Description

Create an object which can be used to send messages to an SMTP server.

Usage

```
server(
  host,
  port = 25,
  username = NULL,
  password = NULL,
  insecure = FALSE,
  reuse = TRUE,
  helo = NA,
  protocol = NA,
  use_ssl = NA,
  test = FALSE,
  pause_base = 1,
  max_times = 5,
  ...
```

```
)

gmail(username, password, ...)

sendgrid(password, ...)

mailgun(username, password, ...)

sendinblue(username, password, ...)

mailersend(username, password, ...)

mailfence(username, password, ...)

zeptomail(password, ...)

smtpbucket(...)
```

Arguments

host	DNS name or IP address of the SMTP server.
port	Port that the SMTP server is listening on.
username	Username for SMTP server.
password	Password for SMTP server or API key.
insecure	Whether to ignore SSL issues.
reuse	Whether the connection to the SMTP server should be left open for reuse.
helo	The HELO domain name of the sending host. If left as NA then will use local host name.
protocol	Which protocol (SMTP or SMTPS) to use for communicating with the server. Default will choose appropriate protocol based on port.
use_ssl	Whether to use SSL. If not specified then SSL will be used if the port is 465 or 587. This enables SSL on non-standard ports.
test	Test login to server.
pause_base	Base delay (in seconds) for exponential backoff. See rate_backoff.
max_times	Maximum number of times to retry.
...	Additional curl options. See curl::curl_options() for a list of supported options.

Details

These functions return a function that can then be called with a message object. This function mediates the interaction with the Simple Mail Transfer Protocol (SMTP) server.

SMTP is a plain text protocol, which means that it is not secure. The secure variant, SMTPS, comes in two flavours: TLS and StartTLS. With TLS (also called Implicit TLS) the connection with the server is initiated using an Secure Socket Layer (SSL) or Transport Layer Security (TLS) certificate.

Such a connection is secure from the start. By contract, a StartTLS connection is initiated in plain text and then upgraded to TLS if possible. By convention TLS operates on port 465 and StartTLS on port 587.

The specifications of an SMTP server are given in an SMTP URL, which takes one of the following forms:

- `mail.example.com` — hostname only
- `mail.example.com:587` — hostname and port
- `smtp://mail.example.com` — SMTP URL (default port)
- `smtps://mail.example.com` — SMTPS URL (default port)
- `smtp://mail.example.com:25` — SMTP URL (explicit port)
- `smtps://mail.example.com:587` — SMTPS URL (explicit port)

Value

A function which is used to send messages to the server.

Gmail

If you're having trouble authenticating with Gmail then you should try the following:

- enable 2-factor authentication and
- create an app password.

Then use the app password rather than your usual account password.

Sendgrid

To use SendGrid you'll need to first create an API key. # nolint Then use the API key as the password.

SendGrid will accept messages on ports 25, 587 and 2525 (using SMTP) as well as 465 (using SMTPS).

Mailgun

To use Mailgun you'll need to first register a sender domain. This will then be assigned a username and password.

Mailgun will accept messages on ports 25 and 587 (using SMTP) as well as 465 (using SMTPS).

Sendinblue

To use Sendinblue you'll need to first create an account. You'll find your SMTP username and password in the SMTP & API section of your account settings.

MailerSend

To use MailerSend you'll need to first create an account. You'll find your SMTP username and password under Domains. See How to send emails via SMTP with MailerSend.

Although this is not likely to be a problem in practice, MailerSend insists that all messages have at minimum a valid subject and either text or HTML content.

Mailfence

To use Mailfence you'll need to create a premium account.

ZeptoMail

SMTP Bucket is a fake SMTP server that captures all the messages it receives and makes them available through a website or REST API.

SMTP Bucket

SMTP Bucket is a fake SMTP server that captures all the messages it receives and makes them available through a website or REST API.

Examples

```
# Set parameters for SMTP server (with username and password).
smtp <- server(
  host = "smtp.gmail.com",
  port = 587,
  username = "bob@gmail.com",
  password = "bd40ef6d4a9413de9c1318a65cbae5d7"
)

# Set parameters for a (fake) testing SMTP server.
#
# More information about this service can be found at https://www.smtpbucket.com/.
#
smtp <- server(
  host = "mail.smtpbucket.com",
  port = 8025
)

# Create a message
msg <- envelope() %>%
  from("bob@gmail.com") %>%
  to("alice@yahoo.com")

# Send message (verbose output from interactions with server)
## Not run:
smtp(msg, verbose = TRUE)

## End(Not run)

# To confirm that the message was sent, go to https://www.smtpbucket.com/ then:
#
# - fill in "bob@gmail.com" for the Sender field and
# - fill in "alice@yahoo.com" for the Recipient field then
# - press the Search button.

# With explicit HELO domain.
#
smtp <- server(
```

```
  host = "mail.example.com",
  helo = "client.example.com"
)

# Set parameters for Gmail SMTP server. The host and port are implicit.
smtp <- gmail(
  username = "bob@gmail.com",
  password = "bd40ef6d4a9413de9c1318a65cbae5d7"
)

# Set API key for SendGrid SMTP server.
smtp <- sendgrid(
  password = "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"
)

# Set username and password for Mailgun SMTP server.
smtp <- mailgun(
  username = "postmaster@sandbox9ptce35fdf0b31338dec4284eb7aaa59.mailgun.org",
  password = "44d072e7g2b5f3bf23b2b642da0fe3a7-2ac825a1-a5be680a"
)

# Set username and password for Sendinblue SMTP server.
smtp <- sendinblue(
  username = "bob@gmail.com",
  password = "xsmtpsib-c75cf91323adc53a1747c005447cbc9a893c35888635bb7bef1a624bf773da33"
)

# Set username and password for MailerSend SMTP server.
smtp <- mailersend(
  username = "NS_Pf3ALM@gmail.com",
  password = "e5ATWLlTnWWDaKeE"
)

# Set username and password for Mailfence SMTP server.
smtp <- mailfence(
  username = "bob",
  password = "F!Uosd6xbhSjd%63"
)

# Set password for ZeptoMail SMTP server.
# nolint start
smtp <- zeptomail("yA6KbHsL4l2mmI8Ns0/fs9iSTj8yG0dYBgfIG0j6Fsv4P2uV32xh8ciEYNYlRkgCC7wRfkgWA==")
# nolint end

# SMTP Bucket server.
smtp <- smtpbucket()
```

`subject` *Add or query subject of message.*

Description

Add or query subject of message.

Usage

```
subject(
  msg,
  subject = NULL,
  prefix = NA,
  suffix = NA,
  interpolate = TRUE,
  .open = "{{",
  .close = "}}",
  .envir = NULL
)
```

Arguments

`msg`	A message object.
`subject`	A subject for the message.
`prefix`	A subject prefix.
`suffix`	A subject suffix.
`interpolate`	Whether or not to interpolate into input using glue.
`.open`	The opening delimiter.
`.close`	The closing delimiter.
`.envir`	Environment used for `glue` interpolation. Defaults to `parent.frame()`.

Details

The `prefix` and `suffix` can be used to add extra subject abbreviations.

Value

A message object or the subject of the message object (if `subject` is `NULL`).

See Also

`to`, `from`, `cc`, `bcc` and `reply`

Examples

```
# Create a message and set the subject
msg <- envelope() %>% subject("Updated report")

# Retrieve the subject for a message
subject(msg)
```

`template` *Add message body from template*

Description

Variables given as named arguments will override any variables in the environment with the same name.

Usage

```
template(msg, .name, ..., .envir = parent.frame())
```

Arguments

`msg`	A message object.
`.name`	A template name. This can be provided as either: (i) the name of a template that's baked into the package, (ii) a relative path or (iii) an absolute path. The paths must be for the directory containing the template files, not the files themselves.
`...`	Variables for substitution.
`.envir`	Environment for substitution.

Details

Will probably not get variables from environment if used as part of a pipeline. In this case might need to use the `%|>%` (nested pipe) operator.

Value

A message object.

Examples

```
# Use a builtin template.
envelope() %>%
  template(
    "newsletter",
    title = "A Sample Newsletter",
    articles = list(
      list(
        "title" = "Article (with date)",
        "content" = as.list("Vivamus, justo quisque, sed."),
        "date" = "1 January 2022"
      ),
      list(
        "title" = "Another Article (without date)",
        "content" = as.list("Quam lorem sed metus egestas.")
      )
```

```
    )
  )
# Use a custom local template.
## Not run:
envelope() %>%
  template("./templates/custom-template")

## End(Not run)
```

`text` *Add a text body to a message.*

Description

Add `text/plain` content to a message.

Usage

```
text(
  msg,
  content,
  disposition = "inline",
  charset = "utf-8",
  encoding = "7bit",
  language = FALSE,
  interpolate = TRUE,
  .open = "{{",
  .close = "}}",
  .envir = NULL
)
```

Arguments

`msg`	A message object.
`content`	A string of message content.
`disposition`	Should the content be displayed inline or as an attachment? Valid options are `"inline"` and `"attachment"`. If set to `NA` then will guess appropriate value.
`charset`	What character set is used. Most often either `"UTF-8"` or `"ISO-8859-1"`.
`encoding`	How content is transformed to ASCII. Options are `"7bit"`, `"quoted-printable"` and `"base64"`. Use `NA` or `NULL` for no (or "identity") encoding.
`language`	Language of content. If `FALSE` then will not include language field. If `TRUE` then will attempt to auto-detect language. Otherwise will use the specified language.
`interpolate`	Whether or not to interpolate into input using glue.
`.open`	The opening delimiter.
`.close`	The closing delimiter.
`.envir`	Environment used for `glue` interpolation. Defaults to `parent.frame()`.

Details

The `text/plain` format is described in RFC 2646.

Uses `glue::glue()` to evaluate expressions enclosed in brackets as R code.

Value

A message object.

See Also

`html`, `render`

Examples

```
msg <- envelope() %>% text("Hello!")

# Using {glue} interpolation.
#
name <- "Alice"
msg <- envelope() %>% text("Hello {name}.")

print(msg, details = TRUE)

# Disable {glue} interpolation.
#
msg <- envelope() %>% text("This is a set: {1, 2, 3}.", interpolate = FALSE)
```

`validate` *Validate email address*

Description

Validate email address

Usage

```
validate(addr, deliverability = TRUE)
```

Arguments

`addr`	An email address.
`deliverability`	Whether to check for deliverability (valid domain).

Value

A `logical` indicating whether or not the address is valid.

Examples

```
# A valid address.
validate("cran-sysadmin@r-project.org")
# An invalid address.
validate("help@this-domain-does-not-exist.com")
```

Index

address, 3
addresses, 4
after.envelope, 5
as.address, 6
as.character.address, 6
as.character.envelope, 7
as.character.header, 7
as.character.MIME, 8
attachment, 8

bcc, *19*, *35*
bcc *(*addresses*)*, 4
bcc(), *15, 16*

c.address, 9
cc, *19*, *35*
cc *(*addresses*)*, 4
cc(), *15, 16*
cleave, 10
comments, 10
commonmark::markdown_html(), *27*
compare, 11
compliant, 11
cutoff, 12

display, 13
domain, 13

encrypt, 14
encrypt(), *16*
envelope, 15
expires *(*cutoff*)*, 12

format.address, 16
from, *19*, *35*
from *(*addresses*)*, 4
from(), *15, 16*

glue, *18*, *21*, *27*, *35*, *37*
gmail *(*server*)*, 30

html, 17, *27*, *38*
html(), *16*

importance *(*precedence*)*, 22
importance(), *15*
inreplyto *(*response*)*, 28

keywords, 18

length.address, 19
local, 20

mailersend *(*server*)*, 30
mailfence *(*server*)*, 30
mailgun *(*server*)*, 30
mime-parameters, 20

normalise, 21

parties, 22
precedence, 22
print.address, 23
print.envelope, 24
priority *(*precedence*)*, 22
priority(), *15*

qp, 24
qp_decode *(*qp*)*, 24
qp_encode *(*qp*)*, 24

rate_backoff, *31*
raw, 25
receipt, 26
references *(*response*)*, 28
render, *18*, 26, *38*
reply, *19*, *35*
reply *(*addresses*)*, 4
reply(), *15, 16*
replyby *(*cutoff*)*, 12
request_receipt_delivery *(*receipt*)*, 26
request_receipt_read *(*receipt*)*, 26

response, 28
return_path (addresses), 4
rmarkdown::render(), 27

sender (addresses), 4
sendgrid (server), 30
sendinblue (server), 30
sensitivity, 29
server, 30
signature (encrypt), 14
smtpbucket (server), 30
subject, 10, 34
subject(), 15, 16

template, 36
text, 18, 27, 37
text(), 16
to, 19, 35
to (addresses), 4
to(), 15, 16

validate, 38

zeptomail (server), 30