Package: mRpostman (via r-universe)

October 9, 2026

Type Package

Title An IMAP Client for R

Version 3.0.0

Date 2026-09-09

Description A session-based IMAP client that implements the full command sets of the IMAP4rev2 (RFC 9051) and IMAP4rev1 (RFC 3501) protocols, along with the optional extensions registered with the Internet Assigned Numbers Authority, allowing virtually all e-mail operations to be performed from within R, paving the way for e-mail data analysis.

License GPL-3

Encoding UTF-8

Imports curl, lifecycle, R6, stringr, stringi, magrittr, assertthat, base64enc, utils, rvest, xml2

Depends R (>= 4.1.0)

URL https://allanvc.github.io/mRpostman/

BugReports https://github.com/allanvc/mRpostman/issues/

SystemRequirements libcurl: libcurl-devel (rpm) or libcurl4-openssl-dev (deb)

Suggests knitr, rmarkdown, testthat (>= 3.1.7), httr2, jsonlite

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation yes

Author Allan Quadros [aut, cre] (ORCID: <https://orcid.org/0000-0003-3250-5380>), Paul Smith [ctb], Kurt Hornik [ctb]

Maintainer Allan Quadros <allanvcq@gmail.com>

Config/pak/sysreqs libssl-dev

Repository https://cran.r-universe.dev

Date/Publication 2026-09-09 15:10:02 UTC

RemoteUrl https://github.com/cran/mRpostman

RemoteRef HEAD

RemoteSha 643735a6394c8e91acdbb69f86c0bfee00f10e7d

Contents

mRpostman-package . . . 3
AND . . . 4
before . . . 5
clean_msg_text . . . 6
configure_imap . . . 7
decode_mime_header . . . 9
decode_mime_text . . . 10
enron_sandbox . . . 11
extract_attachments . . . 13
filter_stored . . . 14
flag . . . 15
fuzzy . . . 16
imap_url . . . 17
imap_utf7_encode . . . 17
ImapCon . . . 18
ingest_maildir . . . 103
larger_than . . . 104
list_attachments . . . 105
metadata_options . . . 106
modseq . . . 107
older_than . . . 108
on . . . 109
Ops.imap_search . . . 110
OR . . . 110
parse_bodystructure . . . 111
parse_envelope . . . 112
populate_sandbox . . . 113
sandbox_corpus . . . 114
saved_before . . . 115
sent_before . . . 116
sent_on . . . 117
sent_since . . . 118
since . . . 119
smaller_than . . . 120
string . . . 121
verbatim . . . 122
younger_than . . . 123

Index **124**

mRpostman-package *An IMAP client for R*

Description

mRpostman is a session-based IMAP client that implements the full command sets of the IMAP4rev2 (RFC 9051) and IMAP4rev1 (RFC 3501) protocols, along with the optional extensions registered with IANA, allowing virtually all e-mail operations to be performed from within R, paving the way for e-mail data analysis.

Options

`mRpostman.raw_debug` If `TRUE`, prints the raw dialogue of the second (event) connection used by `idle()`, `append_msgs()`, and the other raw-socket commands, in the same spirit as `verbose = TRUE` for the main connection. Default is `FALSE`.

Author(s)

Author & Mantainer: Allan Quadros <allanvcq@gmail.com>

References

Crispin, M. (2003), *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*, RFC 3501, March 2003, https://www.rfc-editor.org/rfc/rfc3501.

Heinlein, P. and Hartleben, P. (2008). *The Book of IMAP: Building a Mail Server with Courier and Cyrus*. No Starch Press. ISBN 978-1-59327-177-0.

Ooms, J. (2020). *curl: A Modern and Flexible Web Client for R*. R package version 4.3, https://CRAN.R-project.org/package=curl.

Stenberg, D. *Libcurl - The Multiprotocol File Transfer Library*, https://curl.se/libcurl/.

See Also

Useful links:

- mRpostman official website: https://allanvc.github.io/mRpostman/

AND *Relational-operator-function to construct a custom search statement*

Description

Relational-operator-function to construct a custom search statement

Usage

```
AND(..., negate = FALSE)
```

Arguments

`...`	a combination of criteria constructor functions with its arguments.
`negate`	If `TRUE`, negates the search and seeks for "NOT search_criterion". Default is `FALSE`.

Value

A search string to be used as a `request` parameter in `ImapCon$search()` function.

See Also

Other custom search: `ImapCon`, `OR()`, `before()`, `filter_stored()`, `flag()`, `fuzzy()`, `larger_than()`, `modseq()`, `older_than()`, `on()`, `saved_before()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `verbatim()`, `younger_than()`

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SINCE "30-Aug-2019" AND SMALLER than 512KB.
res <- con$search(request = AND(sent_since(date_char = "30-Aug-2019"),
                                smaller_than(size = 512000)))

## End(Not run)
```

`before` *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
before(date_char, negate = FALSE)
```

Arguments

`date_char`	A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this unusual date format.
`negate`	If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERIA". Default is `FALSE`.

Value

A search string to be used as a `request` parameter in `ImapCon$search()` function.

See Also

Other custom search: `AND()`, `ImapCon`, `OR()`, `filter_stored()`, `flag()`, `fuzzy()`, `larger_than()`, `modseq()`, `older_than()`, `on()`, `saved_before()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `verbatim()`, `younger_than()`

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages BEFORE "17-Apr-2019" AND NOT SMALLER than 512KB.
res <- con$search(request = AND(before(date_char = "17-Apr-2019"),
                                smaller_than(size = 512000, negate = TRUE)))

## End(Not run)
```

clean_msg_text *Extract text from MIME level*

Description

Extract text from MIME level

Usage

```
clean_msg_text(msg_list)
```

Arguments

msg_list	A list with the MIME level 1 of the body or text content of the messages fetched with ImapCon$fetch_body() or ImapCon$fetch_text().

Value

A list containing the decoded messages if applicable.

References

Moore, K. (1996), MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, RFC 2047, November 1996, https://tools.ietf.org/html/rfc2047.

Freed, N., Borenstein, N. (1996), Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC 2045, November 1996, https://tools.ietf.org/html/rfc2045.

Internal parts of this object, regarding the quoted printable type, were borrowed from https://github.com/hrbrmstr/hrbrmisc/blo with slight modifications.

Examples

```
## Not run:
ids <- con$search_since(date_char = "01-Apr-2020", use_uid = TRUE)

fetch_res <- ids %>%
  con$fetch_body(use_uid = TRUE, mime_level = 1L)

clean_text_list <- clean_msg_text(msg_list = fetch_res)

## End(Not run)
```

configure_imap *IMAP Connection Configuration*

Description

Configure and create a new IMAP connection.

Usage

```
configure_imap(
  url,
  username,
  password = NULL,
  xoauth2_bearer = NULL,
  oauth_mechanism = c("XOAUTH2", "OAUTHBEARER"),
  use_ssl = TRUE,
  verbose = FALSE,
  buffersize = 16000,
  timeout_ms = 0,
  use_uid = TRUE,
  mute = FALSE,
  retries = 1,
  ...
)
```

Arguments

url	A character string containing the IMAP server address
username	A character string containing the username.
password	A character string containing the user's password.
xoauth2_bearer	A character string containing the oauth2 bearer token.
oauth_mechanism	The SASL mechanism used to send the OAuth 2.0 token: "XOAUTH2" (default; advertised by Gmail, Yahoo, and Microsoft 365) or "OAUTHBEARER" (RFC 7628; advertised by Gmail, but not by the Microsoft 365 IMAP server). Check list_server_capabilities() for the AUTH= tokens of your server. Ignored when authenticating with a password.
use_ssl	A logical. TRUE (default) requires TLS: an imaps:// URL is encrypted from the start, and an imap:// URL is upgraded with STARTTLS before any credential is sent; if the server does not offer STARTTLS, the connection fails (there is no silent fallback to plaintext). FALSE sends everything unencrypted, for plain test servers such as the Docker sandbox.
verbose	If FALSE, mutes the flow of information between the server and the client. Default is FALSE.

buffersize	The size in bytes for the receive buffer. Default is 16000 bytes or 16kb, which means it will use the libcurl's default value. According to the libcurl's documentation, the maximum buffersize is 512kb (or 512000 bytes), but any number passed to buffersize is treated as a request, not an order.
timeout_ms	Time in milliseconds (ms) to wait for the execution or re-execution of a command. Default is 0, which means that no timeout limit is set.
use_uid	Connection-level default for the use_uid argument of the methods: if TRUE, operations use the "UID" (unique identifier), stable during the life cycle of a message, instead of message sequence numbers. Each call can still override it. Since 3.0.0 the default is TRUE: UIDs are stable, sequence numbers renumber whenever messages are expunged.
mute	Connection-level default for the mute argument of the methods: if TRUE, confirmation messages are suppressed. Each call can still override it. Default is FALSE.
retries	Connection-level default for the retries argument of the methods (number of attempts to connect and execute a command). Each call can still override it. Default is 1.
...	Further curl parameters (see curl::curl_options) that can be used with the IMAP protocol. Only for advanced users.

Value

A new 'ImapCon' object.

Examples

```
## Not run:
# w/ Plain authentication
con <- configure_imap(
  url="imaps://outlook.office365.com",
  username="user@agency.gov.br",
  password=rstudioapi::askForPassword(),
  verbose = TRUE)

# w/ OAuth2.0 authentication
con <- configure_imap(
  url="imaps://outlook.office365.com",
  username="user@agency.gov.br",
  verbose = TRUE,
  xoauth2_bearer = "XX.Ya9...")

## End(Not run)
```

decode_mime_header *Decode RFC 2047 quoted-printable and base64 MIME headers and strings*

Description

Decode RFC 2047 quoted-printable and base64 MIME headers and strings

Usage

```
decode_mime_header(string)
```

Arguments

string	A character vector containing a string to be decoded.

Value

A decoded character vector if applicable.

Note

The RFC 2047 (Moore, 1996) presents an encoded-word syntax to be used by e-mail clients to display body text and header information in character sets other than ASCII. According to the manual, non-ASCII content is encoded as an ASCII text string as follows: =?<charset>?<encoding>?<encoded-text>?=. The encoding can be of two types: "B" for "BASE64", or "Q" for quoted- printable content (Freed and Borenstein, 1996). Besides the standard RFC 2047 decoding, this function also enables users to decode content that does not strictly follow the =?<charset>?<encoding>?<encoded-text>?= RFC 2047 syntax, i.e. cases where only the encoded text part is present, such as the quoted-printable pattern in the string "Estat=EDstica" (Estatística, which is the equivalent word, in Portuguese, for Statistics).

References

Moore, K. (1996), MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, RFC 2047, November 1996, https://tools.ietf.org/html/rfc2047.

Freed, N., Borenstein, N. (1996), Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC 2045, November 1996, https://tools.ietf.org/html/rfc2045.

Internal parts of this object, regarding the quoted printable type, were borrowed from https://github.com/hrbrmstr/hrbrmisc/blo with slight modifications.

Examples

```
## Not run:
# The examples below run smoothly on any computer. The 'dontrun' flag is just to skip CRAN checks.

# Simple quoted-printable string - Portuguese example
```

```
qp_encoded <- "Minist=E9rio_da_Educa=E7=E3o"
decode_mime_header(string = qp_encoded)

# Simple quoted-printable string - French example
qp_encoded <- "sur la route =C3=A0 suivre les voil=C3=A0 bient=C3=B4t qui te d=C3=A9gradent"
decode_mime_header(string = qp_encoded)

# Simple quoted-printable string - Norwegian example
qp_encoded <- "p=C3=A5 veien for =C3=A5 f=C3=B8lge, snart vil de forringe deg"
decode_mime_header(string = qp_encoded)

# Simple quoted-printable string - Turkish example
qp_encoded <- "yak=C4=B1nda seni k=C3=BC=C3=A7=C3=BCk d=C3=BC=C5=9F=C3=BCrecekler"
decode_mime_header(string = qp_encoded)

# RFC 2047 quoted-printable header - Portuguese example
qp_encoded <- "=?iso-8859-1?Q?DIDEC_Capacita=E7=E3o?="
decode_mime_header(string = qp_encoded)

# RFC 2047 quoted-printable - German example
qp_encoded <- "=?UTF-8?Q?stern=2Ede_-_t=C3=A4glich?="
decode_mime_header(string = qp_encoded)

# RFC 2047 base64 - Portuguese example
b64_encoded <- "=?utf-8?B?Sk9BTkEgRlVTQ08gTE9CTyBubyBUZWFtcw==?="
decode_mime_header(string = b64_encoded)

## End(Not run)
```

`decode_mime_text` *Decode RFC 2047 quoted-printable and base64 MIME encoded text*

Description

Decode RFC 2047 quoted-printable and base64 MIME encoded text

Usage

```
decode_mime_text(string, charset = NULL)
```

Arguments

`string`	A `character` vector containing a string to be decoded.
`charset`	A `character` string with the charset declared in the MIME `Content-Type`. When supplied it is honored (via `apply_charset()`) instead of the legacy heuristic detection.

Value

A decoded `character` vector if applicable.

Note

The RFC 2047 (Moore, 1996) presents an encoded-word syntax to be used by e-mail clients to display body text and header information in character sets other than ASCII. According to the manual, non-ASCII content is encoded as an ASCII text string as follows: `=?<charset>?<encoding>?<encoded-text>?=`. The encoding can be of two types: "B" for "BASE64", or "Q" for quoted- printable content (Freed and Borenstein, 1996). Besides the standard RFC 2047 decoding, this function also enables users to decode content that does not strictly follow the `=?<charset>?<encoding>?<encoded-text>?=` RFC 2047 syntax, i.e. cases where only the encoded text part is present, such as the quoted-printable pattern in the string `"Estat=EDstica"` (Estatística, which is the equivalent word, in Portuguese, for Statistics).

References

Moore, K. (1996), MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text, RFC 2047, November 1996, https://tools.ietf.org/html/rfc2047.

Freed, N., Borenstein, N. (1996), Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC 2045, November 1996, https://tools.ietf.org/html/rfc2045.

Internal parts of this object, regarding the quoted printable type, were borrowed from https://github.com/hrbrmstr/hrbrmisc/blo with slight modifications.

`enron_sandbox` *Populate a Mailbox with a Subset of the Enron Corpus*

Description

Downloads the public Enron e-mail corpus (CMU, May 7 2015 release; ~423 MB, one time, cached locally), selects a subset by custodian, folder, and date, and ingests it into the connected IMAP server through `ingest_maildir` – i.e., through the package's own `APPEND` implementation. One server folder is created per custodian. It is meant to be run against the disposable local Docker IMAP server shipped in `system.file("docker", package = "mRpostman")`, providing real data for demonstrations and teaching on top of the synthetic corpus of `sandbox_corpus`.

Usage

```
enron_sandbox(
  con,
  custodians = c("kaminski-v", "lay-k", "skilling-j"),
  folder_pattern = "^_?sent(_items|_mail)?$",
  sent_since = NULL,
  sent_before = NULL,
  max_msgs = Inf,
  cache_dir = tools::R_user_dir("mRpostman", "cache"),
```

```
  url = "https://www.cs.cmu.edu/~enron/enron_mail_20150507.tar.gz",
  ask = TRUE,
  mute = FALSE
)
```

Arguments

con	An ImapCon object, as returned by configure_imap.
custodians	A character vector of custodian directory names, e.g. c("kaminski-v", "lay-k", "skilling-j") (the default). The 150 available names are the directories of the corpus' maildir/ root.
folder_pattern	A regular expression matched against the custodians' folder names. The default, "^_?sent(_items|_mail)?$", selects the sent-mail folder variants. Use ".*" for all folders of the selected custodians.
sent_since, sent_before	Optional Date (or "YYYY-MM-DD" string) limits applied to each message's Date: header before ingestion. NULL (default) applies no limit.
max_msgs	Maximum number of messages per custodian, applied after the other filters. Default is Inf.
cache_dir	Directory where the corpus tarball and its extraction are cached across sessions. Default is tools::R_user_dir("mRpostman", "cache").
url	URL of the corpus tarball. Default is the CMU distribution point.
ask	A logical. If TRUE (default), asks for confirmation before the one-time ~423 MB download (~1.4 GB extracted). Set to FALSE for non-interactive use.
mute	A logical. Passed to ingest_maildir. Default is FALSE.

Details

The corpus was released into the public domain by the U.S. Federal Energy Regulatory Commission and is curated at Carnegie Mellon University (https://www.cs.cmu.edu/~enron/); it contains about 517,000 messages from 150 custodians in maildir layout, without attachments. Nothing is downloaded when the cache is already in place. If the download fails (offline use, moved URL), the function fails gracefully with an informative message and leaves the cache untouched. Date filtering parses each candidate file's Date: header locale-independently; messages without a parseable date are kept unless both limits are set.

Value

Invisibly, a data.frame with one row per ingested message: custodian, path (relative to the corpus root), size, date (parsed from the header; NA when unparseable), and appended.

See Also

Other sandbox: ingest_maildir(), populate_sandbox(), sandbox_corpus()

Examples

```
## Not run:
con <- configure_imap(url = "imap://localhost:1430",
                      username = "enron", password = "sandbox",
                      use_ssl = FALSE)
# sent mail of three central custodians, 2000-2002
manifest <- enron_sandbox(con,
                          custodians = c("kaminski-v", "lay-k",
                                         "skilling-j"),
                          sent_since  = "2000-01-01",
                          sent_before = "2002-07-01")
table(manifest$custodian, manifest$appended)

## End(Not run)
```

`extract_attachments` *Extract attachments from already-fetched messages*

Description

The offline half of the attachment family (since the 2026 refactoring): takes the full text of messages already fetched with `ImapCon$fetch_body()`, walks the MIME multipart tree by its declared boundaries, and either reports the attachments (`dest = NULL`) or writes them to disk. It replaces both `list_attachments()` (reporting) and `ImapCon$get_attachments()` (writing).

Usage

```
extract_attachments(
  msg_list,
  dest = NULL,
  content_disposition = "both",
  override = FALSE,
  as_is = FALSE,
  mute = FALSE
)
```

Arguments

`msg_list`	A `list` with the fetched messages, as returned by `ImapCon$fetch_body()` (the full message, whose headers declare the MIME boundaries). Text-only fetches (`fetch_text()`) lack those headers and cannot be walked reliably.
`dest`	`NULL` (default) to only report: the return value is a named `list` with one `data.frame` per message (filename, content_disposition, type, size; zero rows when a message has no attachments). A directory path writes each message's attachments to `dest/<message id>/` and returns the same manifest invisibly.
`content_disposition`	One of `"both"` (default), `"attachment"`, or `"inline"`.

override — A logical. If TRUE, overrides existing files with the same name. Default is FALSE.

as_is — If TRUE, writes payloads without decoding the transfer encoding. Default is FALSE.

mute — A logical. If TRUE, mutes the confirmation message when writing. Default is FALSE.

Value

A named list of data.frames (see dest).

See Also

Other attachments: ImapCon, list_attachments()

Examples

```
## Not run:
out <- con$fetch_body(con$query(size > 1e6, use_uid = TRUE), use_uid = TRUE)
extract_attachments(out)                      # report only
extract_attachments(out, dest = "~/attachments") # write files

## End(Not run)
```

---

filter_stored — *Criterion constructor referencing a filter stored on the server*

Description

Builds a FILTER <name> search criterion (RFC 5466), which stands for the search criteria saved on the server under that name through the ManageSieve protocol. Requires the server FILTERS capability, which is checked when the search is executed. Experimental: none of the reference servers used to validate this package advertises FILTERS, so the criterion follows the RFC grammar but has not been exercised against a live server.

Usage

```
filter_stored(name)
```

Arguments

name — A character string with the name of the filter stored on the server.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

See Also

Other custom search: AND(), ImapCon, OR(), before(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
con$select_folder(folder = "INBOX")
res <- con$search(request = AND(filter_stored("on-the-road"),
                                string(expr = "boss@example.com", where = "FROM")))

## End(Not run)
```

---

flag *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
flag(name, negate = FALSE)
```

Arguments

name	A string containing one or more flags to search for. Use ImapCon$list_flags() to list the flags in a selected mail folder.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages with Flag "UNSEEN" AND NOT Smaller Than  512KB.
res <- con$search(request = AND(flag("UNSEEN"),
                                smaller_than(size = 512000, negate = TRUE)))

## End(Not run)
```

`fuzzy` *Criterion modifier for fuzzy (approximate) matching in a custom search*

Description

Prefixes a search criterion with `FUZZY` (RFC 6203), asking the server to match the criterion approximately (typically through its full-text search index) instead of by exact substring. Requires the server `SEARCH=FUZZY` capability, which is checked when the search is executed. Experimental: none of the reference servers used to validate this package advertises `SEARCH=FUZZY`, so the modifier follows the RFC grammar but has not been exercised against a live server.

Usage

```
fuzzy(criterion)
```

Arguments

`criterion`	A search criterion string, usually built with `string` (e.g. `string(expr = "jump", where = "SUBJECT")`).

Value

A search criterion of class `imap_search`, to be combined into a search statement (see `Ops.imap_search`).

See Also

Other custom search: `AND()`, `ImapCon`, `OR()`, `before()`, `filter_stored()`, `flag()`, `larger_than()`, `modseq()`, `older_than()`, `on()`, `saved_before()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `verbatim()`, `younger_than()`

Examples

```
## Not run:
con$select_folder(folder = "INBOX")
res <- con$search(request = fuzzy(string(expr = "jump", where = "SUBJECT")))

## End(Not run)
```

imap_url — *An IMAP URL naming a message, or a part of it, for CATENATE (RFC 5092)*

Description

Builds the relative URL that `ImapCon$append_catenate()` uses to refer to a message stored on the server: `/<folder>;UID=<uid>` with an optional `/;SECTION=<section>` (e.g. `"HEADER"`, `"TEXT"`, or a MIME part number such as `"2"`).

Usage

```
imap_url(folder, uid, section = NULL)
```

Arguments

`folder`	A `character` string with the folder name.
`uid`	The message UID.
`section`	`NULL` (the whole message) or a section specifier.

Value

An object of class `imap_url`.

Examples

```
imap_url("INBOX", uid = 12)
imap_url("INBOX", uid = 12, section = "HEADER")
```

imap_utf7_encode — *Encode and decode mailbox names in IMAP modified UTF-7*

Description

IMAP mailbox names are transmitted in the modified UTF-7 encoding of RFC 3501 (section 5.1.3): non-ASCII runs are written as `&<base64>-`, where the base64 alphabet uses , instead of / and carries no padding, and a literal & is written `&-`. `mRpostman` applies the encoding to every folder name it sends and decodes the names it receives, so folders can be referred to by their real (UTF-8) names; these helpers are exported for users who handle raw names.

Usage

```
imap_utf7_encode(x)

imap_utf7_decode(x)
```

Arguments

x	A `character` vector of mailbox names.

Value

A `character` vector of the same length.

Examples

```
imap_utf7_encode("École")   # "&AMk-cole"
imap_utf7_decode("&AMk-cole")    # "École"
imap_utf7_encode("Q&A")          # "Q&-A"
```

ImapCon *An IMAP Connection Class*

Description

Configure an IMAP connection using the `ImapCon` `R6` class.

Value

An `ImapCon` object holding one stateful IMAP session: the connection parameters, the transport handle, and the session caches (selected folder, server capabilities). All the package's operations are methods of this object.

Methods

Public methods:

- `ImapCon$new()`
- `ImapCon$reset_url()`
- `ImapCon$reset_username()`
- `ImapCon$reset_use_ssl()`
- `ImapCon$reset_verbose()`
- `ImapCon$reset_buffersize()`
- `ImapCon$reset_timeout_ms()`
- `ImapCon$reset_password()`
- `ImapCon$reset_xoauth2_bearer()`
- `ImapCon$print()`
- `ImapCon$has_capability()`
- `ImapCon$disconnect()`
- `ImapCon$idle()`
- `ImapCon$notify()`
- `ImapCon$fetch_binary()`
- `ImapCon$append_catenate()`

- ImapCon$append_msgs()
- ImapCon$esearch_partial()
- ImapCon$esort_partial()
- ImapCon$replace_msg()
- ImapCon$fetch_objectid()
- ImapCon$uid_batches()
- ImapCon$esearch_multi()
- ImapCon$unauthenticate()
- ImapCon$language()
- ImapCon$comparator()
- ImapCon$genurlauth()
- ImapCon$urlfetch()
- ImapCon$fetch_convert()
- ImapCon$fetch_annotation()
- ImapCon$store_annotation()
- ImapCon$query()
- ImapCon$list_server_capabilities()
- ImapCon$enable()
- ImapCon$namespace()
- ImapCon$id()
- ImapCon$get_quota_root()
- ImapCon$get_quota()
- ImapCon$set_quota()
- ImapCon$get_metadata()
- ImapCon$set_metadata()
- ImapCon$get_acl()
- ImapCon$set_acl()
- ImapCon$delete_acl()
- ImapCon$list_rights()
- ImapCon$my_rights()
- ImapCon$noop()
- ImapCon$check()
- ImapCon$list_mail_folders()
- ImapCon$list_subscribed_folders()
- ImapCon$list_folders_status()
- ImapCon$list_special_use_folders()
- ImapCon$select_folder()
- ImapCon$resync_folder()
- ImapCon$fetch_changes()
- ImapCon$close_folder()
- ImapCon$unselect_folder()
- ImapCon$examine_folder()

- ImapCon$status()
- ImapCon$create_folder()
- ImapCon$rename_folder()
- ImapCon$delete_folder()
- ImapCon$subscribe_folder()
- ImapCon$unsubscribe_folder()
- ImapCon$list_flags()
- ImapCon$sort()
- ImapCon$thread()
- ImapCon$search()
- ImapCon$search_larger_than()
- ImapCon$search_smaller_than()
- ImapCon$search_before()
- ImapCon$search_since()
- ImapCon$search_on()
- ImapCon$search_period()
- ImapCon$search_sent_before()
- ImapCon$search_sent_since()
- ImapCon$search_sent_on()
- ImapCon$search_sent_period()
- ImapCon$search_flag()
- ImapCon$search_older_than()
- ImapCon$search_younger_than()
- ImapCon$search_string()
- ImapCon$fetch_body()
- ImapCon$fetch_header()
- ImapCon$fetch_metadata()
- ImapCon$fetch_preview()
- ImapCon$fetch_envelope()
- ImapCon$fetch_bodystructure()
- ImapCon$fetch_text()
- ImapCon$copy_msg()
- ImapCon$move_msg()
- ImapCon$append_msg()
- ImapCon$esearch_count()
- ImapCon$delete_msg()
- ImapCon$expunge()
- ImapCon$esearch_min_id()
- ImapCon$esearch_max_id()
- ImapCon$add_flags()
- ImapCon$replace_flags()
- ImapCon$remove_flags()

- ImapCon$attachments()
- ImapCon$attachments_manifest()
- ImapCon$get_attachments()
- ImapCon$fetch_attachments_list()
- ImapCon$fetch_attachment_parts()
- ImapCon$fetch_attachments()
- ImapCon$clone()

`ImapCon$new()`: Configure and create a new IMAP connection.

Usage:

```
ImapCon$new(
  url,
  username,
  password = NULL,
  xoauth2_bearer = NULL,
  oauth_mechanism = c("XOAUTH2", "OAUTHBEARER"),
  use_ssl = TRUE,
  verbose = FALSE,
  buffersize = 16000,
  timeout_ms = 0,
  use_uid = TRUE,
  mute = FALSE,
  retries = 1,
  ...
)
```

Arguments:

`url` A character string containing the IMAP server address

`username` A character string containing the username.

`password` A character string containing the user's password.

`xoauth2_bearer` A character string containing the oauth2 bearer token.

`oauth_mechanism` The SASL mechanism used to send the OAuth 2.0 token: `"XOAUTH2"` (default; Gmail, Yahoo, Microsoft 365) or `"OAUTHBEARER"` (RFC 7628; Gmail). Ignored when authenticating with a password.

`use_ssl` A logical. `TRUE` (default) requires TLS: an `imaps://` URL is encrypted from the start, and an `imap://` URL is upgraded with `STARTTLS` before any credential is sent; if the server does not offer `STARTTLS`, the connection fails (there is no silent fallback to plaintext). `FALSE` sends everything unencrypted, for plain test servers such as the Docker sandbox.

`verbose` If `FALSE`, mutes the flow of information between the server and the client. Default is `FALSE`.

`buffersize` The size in bytes for the receive buffer. Default is 16000 bytes or 16kb, which means it will use the libcurl's default value. According to the libcurl's documentation, the maximum buffersize is 512kb (or 512000 bytes), but any number passed to `buffersize` is treated as a request, not an order.

`timeout_ms` Time in milliseconds (ms) to wait for the execution or re-execution of a command. Default is 0, which means that no timeout limit is set.

`use_uid` Connection-level default for the `use_uid` argument of the methods; each call can still override it. Since 3.0.0 the default is `TRUE` (UIDs are stable; sequence numbers renumber on expunge).

`mute` Connection-level default for the `mute` argument of the methods; each call can still override it. Default is `FALSE`.

`retries` Connection-level default for the `retries` argument of the methods; each call can still override it. Default is `1`.

`...` Further curl parameters (see `curl::curl_options`) that can be used with the IMAP protocol. Only for advanced users.

Returns: A new 'ImapCon' object.

`ImapCon$reset_url()`**:** Reset the previously informed url

Usage:

```
ImapCon$reset_url(x)
```

Arguments:

`x` A character string containing a new url to be set.

`ImapCon$reset_username()`**:** Reset the previously informed username

Usage:

```
ImapCon$reset_username(x)
```

Arguments:

`x` A character string containing a new username to be set.

`ImapCon$reset_use_ssl()`**:** Reset the previously informed use_ssl parameter

Usage:

```
ImapCon$reset_use_ssl(x)
```

Arguments:

`x` A logical indicating the use or not of Secure Sockets Layer encryption when connecting to the IMAP server. Default is `TRUE`.

`ImapCon$reset_verbose()`**:** Reset the previously informed verbose parameter

Usage:

```
ImapCon$reset_verbose(x)
```

Arguments:

`x` If `FALSE`, mutes the flow of information between the server and the client.

`ImapCon$reset_buffersize()`**:** Reset the previously informed buffersize parameter

Usage:

```
ImapCon$reset_buffersize(x)
```

Arguments:

`x` The size in bytes for the receive buffer. Default is 16000 bytes or 16kb, which means it will use the libcurl's default value. According to the libcurl's documentation, the maximum buffersize is 512kb (or 512000 bytes), but any number passed to `buffersize` is treated as a request, not an order.

`ImapCon$reset_timeout_ms()`: Reset the previously informed buffersize parameter

Usage:

```
ImapCon$reset_timeout_ms(x)
```

Arguments:

`x` Time in milliseconds (ms) to wait for the execution or re-execution of a command. Default is 0, which means that no timeout limit is set.

`ImapCon$reset_password()`: Reset the previously informed password

Usage:

```
ImapCon$reset_password(x)
```

Arguments:

`x` A character string containing the user's password.

`ImapCon$reset_xoauth2_bearer()`: Reset the previously informed oauth2 bearer token

Usage:

```
ImapCon$reset_xoauth2_bearer(x)
```

Arguments:

`x` A character string containing the oauth2 bearer token.

`ImapCon$print()`: Disconnect and release the connection handle. After calling this method the connection object can no longer be used to issue commands; a new one must be created with `configure_imap`. Dropping the handle reference lets 'libcurl' close the underlying connection when the handle is garbage-collected.

Print a compact summary of the connection: server, user, TLS, the selected folder, and whether the session has been used. Credentials are never printed.

Usage:

```
ImapCon$print(...)
```

Arguments:

`...` Ignored (matches the generic).

Returns: `TRUE`, invisibly.

Examples:

```
con$disconnect()
```

`ImapCon$has_capability()`: Check whether the server advertises one capability.

Usage:

```
ImapCon$has_capability(cap, retries = NULL)
```

Arguments:

`cap` The capability token, case-insensitive (e.g. `"ESEARCH"`, `"THREAD=REFERENCES"`).

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `logical`.

Examples:

```
if (con$has_capability("ESEARCH")) ids <- con$search_since("01-Jan-2020", esearch = TRUE)
```

`ImapCon$disconnect()`:

Usage:

```
ImapCon$disconnect()
```

`ImapCon$idle()`: Wait for new messages and other mailbox events (IMAP `IDLE`, RFC 2177). A second, dedicated connection is opened on a raw TLS socket (the main connection stays free), the folder is selected there, and the server's unsolicited notifications (`EXISTS`, `EXPUNGE`, `FETCH` flag changes, `RECENT`) are collected until `timeout` seconds elapse or `callback` returns `FALSE`. Requires the server `IDLE` capability. TLS is used as on the main connection: an `imaps://` URL, or `use_ssl = TRUE` on an `imap://` URL (`STARTTLS`).

Usage:

```
ImapCon$idle(
  timeout = 300,
  callback = NULL,
  folder = NULL,
  renew = 25 * 60,
  compress = FALSE
)
```

Arguments:

`timeout` Maximum number of seconds to wait. Default is `300`.

`callback` `NULL` (default) or a function called with a `data.frame` of events each time the server sends some; return `FALSE` from it to stop waiting.

`folder` The folder to watch. If `NULL` (default), the currently selected folder.

`renew` Seconds after which the `IDLE` command is renewed (servers may close connections idling for too long). Default is 25 minutes.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978; requires that server capability). Default is `FALSE`.

Returns: A `data.frame` with one row per event: `type` (`"EXISTS"`, `"EXPUNGE"`, `"FETCH"`, `"RECENT"`), `id` (the message count or sequence number reported), and `detail`.

Examples:

```
con$select_folder("INBOX")
# block until something arrives (or 10 minutes pass), then fetch it
ev <- con$idle(timeout = 600, callback = function(ev) !any(ev$type == "EXISTS"))
if (any(ev$type == "EXISTS")) con$fetch_envelope(max(ev$id[ev$type == "EXISTS"]))
```

`ImapCon$notify()`: Receive the server's notifications about one or several mailboxes without idling on each (IMAP `NOTIFY`, RFC 5465). On a dedicated second connection, `NOTIFY SET` registers the events of interest; the server then reports new and expunged messages and flag changes of the selected folder (`EXISTS`, `EXPUNGE`, `FETCH`), the same for other mailboxes through `STATUS` lines, and mailbox creations, renames, and deletions through `LIST` lines. Requires the server `NOTIFY` capability.

Usage:

```
ImapCon$notify(
  mailboxes = "personal",
  events = c("MessageNew", "MessageExpunge"),
  timeout = 300,
  callback = NULL,
  compress = FALSE,
  retries = NULL
)
```

Arguments:

`mailboxes` A `character` vector: `"selected"` (the currently selected folder), `"personal"` (every folder of the account), `"subscribed"`, `"inboxes"`, and/or folder names. Default is `"personal"`.

`events` A `character` vector with any of `"MessageNew"`, `"MessageExpunge"`, `"FlagChange"`, `"AnnotationChange"`, `"MailboxName"`, `"SubscriptionChange"`. Default is the first two.

`timeout` Maximum number of seconds to wait. Default is `300`.

`callback` `NULL` (default) or a function called with a `data.frame` of events each time some arrive; return `FALSE` from it to stop waiting.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978). Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with one row per event: `type` (`"EXISTS"`, `"EXPUNGE"`, `"FETCH"`, `"RECENT"`, `"STATUS"`, `"LIST"`), `id` (a message count or sequence number, `NA` for `STATUS`/`LIST`), and `detail` (the rest of the server line). The initial `STATUS` of each watched mailbox is included.

Examples:

```
# wait up to 10 minutes for mail in any folder of the account
ev <- con$notify(mailboxes = "personal", timeout = 600,
                 callback = function(ev) !any(ev$type == "STATUS"))
```

`ImapCon$fetch_binary()`**:** Fetch a message part with the transfer encoding reversed by the server (IMAP `FETCH ... (BINARY.PEEK[<part>])`, RFC 3516), over the raw socket layer, since the reply is a binary literal. The bytes of a base64 or quoted-printable attachment thus arrive already decoded. Requires the server `BINARY` capability.

Usage:

```
ImapCon$fetch_binary(
  msg_id,
  part,
  use_uid = NULL,
  folder = NULL,
  compress = FALSE,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`part` A `character` string with the section number, as reported by `fetch_bodystructure()` (e.g. `"2"`).

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). If TRUE, ids are UIDs.

folder The folder to read from. If NULL (default), the currently selected folder.

compress A logical. If TRUE, the second connection is compressed with COMPRESS DEFLATE (RFC 4978). Default is FALSE.

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A named list of raw vectors, one per message.

Examples:

```
parts <- con$fetch_bodystructure(msg_id = 3)
pdf <- con$fetch_binary(msg_id = 3, part = parts$part[parts$is_attachment][1])
writeBin(pdf[[1]], "attachment.pdf")
```

ImapCon$append_catenate(): Append a message assembled by the server from parts of messages it already stores and from text supplied by the client (IMAP APPEND ... CATENATE, RFC 4469), over the raw socket layer. Typical use: forwarding or archiving a message with a new header without downloading it. Requires the server CATENATE capability.

Usage:

```
ImapCon$append_catenate(
  parts,
  folder = NULL,
  flags = NULL,
  compress = FALSE,
  retries = NULL
)
```

Arguments:

parts A list whose elements are imap_url() objects (parts copied on the server), character strings, or raw vectors (sent as literals), concatenated in order.

folder A character string with the destination folder. If NULL, the previously selected folder is used.

flags NULL (default) or a character vector of flags stored with the message.

compress A logical. If TRUE, the second connection is compressed with COMPRESS DEFLATE (RFC 4978). Default is FALSE.

retries Number of attempts to connect and execute the command. Default is 1.

Returns: Invisibly, the UID assigned to the new message (NA when the server does not report it).

Examples:

```
# a new message whose body is the text of message UID 12, with a new header
con$append_catenate(parts = list(
  "From: me@example.com\r\nSubject: Fwd: report\r\n\r\n",
  imap_url("INBOX", uid = 12, section = "TEXT")), folder = "Archive")
```

ImapCon$append_msgs(): Append several messages to a mail folder in a single command (IMAP MULTIAPPEND, RFC 3502), sent over the raw socket layer as one literal per message. On servers without MULTIAPPEND, one append_msg() per message is issued instead.

Usage:

```
ImapCon$append_msgs(
  messages,
  folder = NULL,
  flags = NULL,
  mute = NULL,
  retries = NULL,
  compress = FALSE
)
```

Arguments:

`messages` A `character` vector, or a `list` of `character` strings or `raw` vectors, each a full RFC 822 message.

`folder` A `character` string with the destination folder. If `NULL`, the previously selected folder is used.

`flags` `NULL` (default) or a `character` vector of flags stored with every message (e.g. `"Seen"`).

`mute` A `logical`. If `TRUE`, mutes the confirmation message. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978). Default is `FALSE`.

Returns: Invisibly, an `integer` vector with the UIDs assigned to the messages (`APPENDUID`, UIDPLUS), or `NA`s when the server does not report them.

Examples:

```
msgs <- vapply(1:3, function(i) paste0("Subject: m", i, "\r\n\r\nbody\r\n"), "")
con$append_msgs(msgs, folder = "Archive", flags = "Seen")
```

`ImapCon$esearch_partial()`**:** Paged search: return only one slice of the result set (`SEARCH RETURN (PARTIAL m:n)`, RFC 9394; also accepted by servers with the older `CONTEXT=SEARCH` capability, RFC 5267). Negative positions count backwards from the most recent result. Requires the server `PARTIAL` or `CONTEXT=SEARCH` capability.

Usage:

```
ImapCon$esearch_partial(
  range,
  criteria = "ALL",
  use_uid = NULL,
  retries = NULL
)
```

Arguments:

`range` A `character` string `"m:n"` with the positions of the first and the last result wanted, e.g. `"1:100"`, or `"-1:-100"` for the hundred most recent matches.

`criteria` A string with the search criteria. Default is `"ALL"`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, the command is performed with UIDs and results are presented as UIDs.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: An `integer` vector with the ids of the requested slice; the requested range is kept in the `"range"` attribute.

Examples:

```
# the fifty most recent matches only
con$esearch_partial(range = "-1:-50", criteria = "UNSEEN", use_uid = TRUE)
```

`ImapCon$esort_partial()`: Paged sort: return only one slice of the sorted result set (`SORT RETURN (PARTIAL m:n)`, `CONTEXT=SORT`, RFC 5267). Requires the server `SORT` and `CONTEXT=SORT` capabilities. Experimental: no server available for this package's validation advertises `CONTEXT=SORT`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$esort_partial(
  range,
  by = "DATE",
  reverse = FALSE,
  criteria = "ALL",
  use_uid = NULL,
  char_set = "UTF-8",
  retries = NULL
)
```

Arguments:

`range` A `character` string `"m:n"` with the positions of the first and the last result wanted, e.g. `"1:100"`.

`by` A character vector of sort keys, a subset of `"ARRIVAL"`, `"CC"`, `"DATE"`, `"FROM"`, `"SIZE"`, `"SUBJECT"`, `"TO"`, `"DISPLAYFROM"`, `"DISPLAYTO"`. Default is `"DATE"`.

`reverse` A logical. If `TRUE`, each sort key is applied in descending order. Default is `FALSE`.

`criteria` A string with the search criteria restricting the set to be sorted. Default is `"ALL"`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, the command is performed with UIDs and results are presented as UIDs.

`char_set` A string with the charset of the search criteria. Default is `"UTF-8"`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: An `integer` vector with the ids of the requested slice, in sort order; the requested range is kept in the `"range"` attribute.

Examples:

```
con$esort_partial(range = "1:20", by = "SIZE", reverse = TRUE)
```

`ImapCon$replace_msg()`: Replace a message by a new version in a single command (IMAP `REPLACE`, RFC 8508), over the raw socket layer, since the new message is sent as a literal. The replacement is atomic: no moment exists in which both, or neither, of the versions are present (the usual APPEND + EXPUNGE dance). Typical use: updating a draft. Requires the server `REPLACE` capability. Experimental: no server available for this package's validation advertises `REPLACE`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$replace_msg(
  msg_id,
  message,
  folder = NULL,
```

```
  flags = NULL,
  use_uid = NULL,
  mute = NULL,
  compress = FALSE,
  retries = NULL
)
```

Arguments:

`msg_id` A single message id.

`message` A `character` string or `raw` vector with the full RFC 822 message that takes the place of message `msg_id`.

`folder` The folder holding the message. If `NULL` (default), the currently selected folder.

`flags` `NULL` (default) or a `character` vector of flags stored with the new version.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, `msg_id` is a UID.

`mute` A `logical`. If `TRUE`, mutes the confirmation message. Default is `FALSE`.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978). Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: Invisibly, the UID assigned to the new version (`NA` when the server does not report it).

Examples:

```
con$replace_msg(msg_id = 4, message = new_draft, folder = "Drafts",
                flags = c("Seen", "Draft"), use_uid = TRUE)
```

`ImapCon$fetch_objectid()`**:** Fetch the unique, immutable object identifiers of messages (`EMAILID` and `THREADID`, OBJECTID, RFC 8474). `EMAILID` survives moves and copies (unlike UIDs), and `THREADID` names the conversation the server files the message under. Requires the server `OBJECTID` capability. Experimental: no server available for this package's validation advertises `OBJECTID`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$fetch_objectid(msg_id, use_uid = NULL, retries = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, ids are UIDs.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with the columns `id`, `emailid`, and `threadid` (`NA` when the server has no thread for the message).

Examples:

```
con$fetch_objectid(msg_id = 1:5)
```

`ImapCon$uid_batches()`**:** Partition the UIDs of the selected folder into batches of a given size (IMAP `UIDBATCHES`, RFC 10022), from the most recent messages to the oldest, so that a large mailbox can be processed in fixed-size pages. Requires the server `UIDBATCHES` capability. Experimental: no server available for this package's validation advertises `UIDBATCHES`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$uid_batches(batch_size, retries = NULL)
```

Arguments:

`batch_size` A single positive number: how many messages each batch should contain.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with the columns `from` and `to`: the first (highest) and last (lowest) UID of each batch, in server order (most recent batch first).

Examples:

```
batches <- con$uid_batches(batch_size = 500)
```

`ImapCon$esearch_multi()`**:** Search several mailboxes with one command (IMAP `ESEARCH IN`, MULTISEARCH, RFC 7377), over the raw socket layer, since the reply carries one untagged response per matching mailbox. Requires the server `MULTISEARCH` capability. Experimental: no server available for this package's validation advertises `MULTISEARCH`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$esearch_multi(
  mailboxes = "personal",
  criteria = "ALL",
  compress = FALSE,
  retries = NULL
)
```

Arguments:

`mailboxes` `"personal"` (every folder of the account, the default), `"subscribed"`, `"inboxes"`, `"selected"`, or a `character` vector of folder names.

`criteria` A string with the search criteria. Default is `"ALL"`.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978). Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with one row per matching message: `mailbox`, `uidvalidity`, and `uid`.

Examples:

```
con$esearch_multi(mailboxes = "personal", criteria = "UNSEEN")
```

`ImapCon$unauthenticate()`**:** Return the connection to the not-authenticated state (IMAP `UNAUTHENTICATE`, RFC 8437). Since libcurl's request model has no use for an unauthenticated connection, the method then opens a fresh connection (authenticating again), so the object remains usable; the practical effect is a clean session: no selected folder, no enabled extensions. Requires the server `UNAUTHENTICATE` capability. Experimental: no server available for this package's validation advertises `UNAUTHENTICATE`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$unauthenticate(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: Invisibly, `TRUE`.

Examples:

```
con$unauthenticate()
```

`ImapCon$language()`: List or choose the language of the server's human-readable responses (IMAP `LANGUAGE`, RFC 5255). Requires the server `LANGUAGE` capability. Experimental: no server available for this package's validation advertises `LANGUAGE`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$language(language = NULL, retries = NULL)
```

Arguments:

`language` `NULL` (default) to list the languages the server supports, or a `character` vector of RFC 4646 language tags in order of preference (e.g. `c("pt-BR", "en")`).

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `character` vector: the supported languages (when listing) or the language the server switched to.

Examples:

```
con$language()
con$language("pt-BR")
```

`ImapCon$comparator()`: List or choose the comparator that collates search and sort results (IMAP `COMPARATOR`, RFC 5255). Requires the server `I18NLEVEL=2` capability. Experimental: no server available for this package's validation advertises `I18NLEVEL=2`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$comparator(order = NULL, retries = NULL)
```

Arguments:

`order` `NULL` (default) to ask which comparator is active, or a `character` vector of comparator names in order of preference (e.g. `"i;basic"`).

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `character` string with the comparator in use.

Examples:

```
con$comparator()
```

`ImapCon$genurlauth()`: Generate an authorized IMAP URL that grants access to a message, or a part of it, without sharing credentials (IMAP `GENURLAUTH`, URLAUTH, RFC 4467), e.g. for forward-without- download submission (BURL). Requires the server `URLAUTH` capability. Experimental: no server available for this package's validation advertises `URLAUTH`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$genurlauth(
  url,
  access = "anonymous",
  mechanism = "INTERNAL",
  expire = NULL,
  retries = NULL
)
```

Arguments:

`url` An `imap_url()` object or an IMAP URL string naming the message (or message part).

`access` Who may use the URL: `"anonymous"` (default), `"authuser"`, `"submit+<user>"`, or `"user+<user>"`.

`mechanism` The authorization mechanism. Default is `"INTERNAL"`.

`expire` `NULL` (default) or an RFC 3339 date-time string after which the URL stops working.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `character` vector with the authorized URL(s) returned by the server.

Examples:

```
con$genurlauth(imap_url("INBOX", uid = 20, section = "1.2"),
               access = "submit+fred")
```

`ImapCon$urlfetch()`**:** Fetch the content named by URLAUTH-authorized IMAP URLs (IMAP `URLFETCH`, RFC 4467), over the raw socket layer, since the reply is a literal. Requires the server `URLAUTH` capability. Experimental: no server available for this package's validation advertises `URLAUTH`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$urlfetch(urls, compress = FALSE, retries = NULL)
```

Arguments:

`urls` A `character` vector of authorized IMAP URLs, as returned by `genurlauth()`.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978). Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A named `list` of `raw` vectors, one per URL.

Examples:

```
u <- con$genurlauth(imap_url("INBOX", uid = 20), access = "anonymous")
con$urlfetch(u)
```

`ImapCon$fetch_convert()`**:** Fetch a body part converted by the server to another MIME type (IMAP `CONVERT`, RFC 5259), over the raw socket layer, since the reply is a binary literal. Requires the server `CONVERT` capability. Experimental: no known server deploys `CONVERT`, so the method follows the RFC grammar but has not been exercised against a live server.

Usage:

```
ImapCon$fetch_convert(
  msg_id,
  mimetype,
  part = "1",
```

```
  params = NULL,
  use_uid = NULL,
  folder = NULL,
  compress = FALSE,
  retries = NULL
)
```

Arguments:

`msg_id` A single message id.

`mimetype` A string with the target MIME type, e.g. `"application/pdf"`.

`part` A `character` string with the section number of the part to convert. Default is `"1"`.

`params` `NULL` (default) or a named `list`/vector of conversion parameters (e.g. `c("pix-x" = "320")`).

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, `msg_id` is a UID.

`folder` The folder to read from. If `NULL` (default), the currently selected folder.

`compress` A `logical`. If `TRUE`, the second connection is compressed with `COMPRESS DEFLATE` (RFC 4978). Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `raw` vector with the converted content.

Examples:

```
pdf <- con$fetch_convert(msg_id = 2, mimetype = "application/pdf",
                         part = "3")
```

`ImapCon$fetch_annotation()`: Fetch per-message annotations (IMAP `FETCH ANNOTATION`, ANNOTATE, RFC 5257). Requires the server `ANNOTATE-EXPERIMENT-1` capability. Experimental, as the capability name itself declares: the extension never left experimental status and no server available for this package's validation advertises it.

Usage:

```
ImapCon$fetch_annotation(
  msg_id,
  entries = "/*",
  attributes = "value",
  use_uid = NULL,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`entries` A `character` vector of annotation entries. Default is `"/*"` (every entry); `"/comment"` and `"/flags/..."` are typical.

`attributes` A `character` vector of attributes: `"value"` (default; both private and shared), `"value.priv"`, `"value.shared"`, or the corresponding `"size"` forms.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, ids are UIDs.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with one row per annotation: `id`, `entry`, `attribute`, and `value`.

Examples:

```
con$fetch_annotation(msg_id = 1)
```

`ImapCon$store_annotation()`: Store a per-message annotation (IMAP `STORE ANNOTATION`, ANNOTATE, RFC 5257). Requires the server `ANNOTATE-EXPERIMENT-1` capability. Experimental, as the capability name itself declares: the extension never left experimental status and no server available for this package's validation advertises it.

Usage:

```
ImapCon$store_annotation(
  msg_id,
  entry,
  values,
  use_uid = NULL,
  mute = NULL,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`entry` A single annotation entry, e.g. `"/comment"`.

`values` A named `character` vector whose names are `"value.priv"` and/or `"value.shared"`; use `NA` as a value to delete that annotation.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, ids are UIDs.

`mute` A `logical`. If `TRUE`, mutes the confirmation message. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: Invisibly, `TRUE`.

Examples:

```
con$store_annotation(msg_id = 1, entry = "/comment",
                     values = c("value.priv" = "check this one"))
```

`ImapCon$query()`: Search with an ordinary R expression (the query language). The expression is captured unevaluated, translated into an RFC 3501 search string by the pure function `translate_query()`, and executed exactly like `search()`. Fields: `subject`, `from`, `to`, `cc`, `bcc`, `body`, `text` (with `==` meaning contains), `flag`, `size` (bytes), `age` (seconds), the date fields `sent`, `date`, and `saved`, and `header("Name")`. Comparisons combine with `&`, `|`, `!`, `%in%`, and parentheses; other calls and variables are evaluated in the caller's environment. Raw protocol fragments outside the field table (vendor extensions such as Gmail's `X-GM-RAW`, sequence sets, `FUZZY`) can be embedded verbatim with `verbatim()`.

Usage:

```
ImapCon$query(
  expr,
  negate = FALSE,
  use_uid = NULL,
  esearch = FALSE,
```

```
  save = FALSE,
  retries = NULL
)
```

Arguments:

`expr` An unquoted expression, e.g. `(subject == "budget" | "budget 3") & flag != "SEEN"`.

`negate` If `TRUE`, negates the whole search. Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, results are presented as UIDs.

`esearch` A logical. If `TRUE` and the server advertises `ESEARCH`, condenses the result transmission. Default is `FALSE`.

`save` A logical. If `TRUE` and the server advertises `SEARCHRES`, keeps the result on the server for use as `msg_id = "$"`. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Details: The full field reference:

field	compares with	protocol key
`subject`, `from`, `to`, `cc`, `bcc`, `body`, `text`	`==` (contains), `!=`	`SUBJECT`, `FROM`, ...
`header("Name")`	`==`, `!=`	`HEADER Name`
`flag`	`==`, `!=`	`SEEN`/`UNSEEN`, ...; custom keywords as `KEYWORD`/`UNK`
`size` (bytes)	`>`, `>=`, `<`, `<=`, `==`	`LARGER`, `SMALLER`
`age` (seconds)	`<`, `<=`, `>`, `>=`	`YOUNGER`, `OLDER` (WITHIN servers)
`sent` (the `Date:` header)	`>=`, `>`, `<`, `<=`, `==`	`SENTSINCE`, `SENTBEFORE`, `SENTON`
`date` (the internal date)	idem	`SINCE`, `BEFORE`, `ON`
`saved`	idem	`SAVEDSINCE`, ... (SAVEDATE servers)
`modseq`	`>=`, `>`, `<`	`MODSEQ` (CONDSTORE servers)
any field, with `%in%`	`field %in% c(v1, v2, ...)`	an `OR` chain over the field's key

Dates accept `"YYYY-MM-DD"`, `"DD-Mon-YYYY"`, or `Date` values, exact at the protocol's day granularity.

Returns: A `numeric vector` with the matching message ids, as in `search()`.

Examples:

```
con$select_folder("INBOX")
con$query((subject == "budget" | "budget 3") & flag != "SEEN")
con$query(sent >= "2001-10-01" & size > 5e6, use_uid = TRUE)
con$query(verbatim('X-GM-RAW "has:attachment"') & flag != "SEEN")
```

`ImapCon$list_server_capabilities()`**:** List the server's IMAP capabilities.

Usage:

```
ImapCon$list_server_capabilities(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A character `vector` containing the server's IMAP capabilities.

Examples:

```
cap <- con$list_server_capabilities()
cap
```

`ImapCon$enable()`: Enable server extensions for the current session (IMAP `ENABLE`, RFC 5161). Some extensions, such as `CONDSTORE` or `UTF8=ACCEPT`, only take effect after the client enables them. Requires the server `ENABLE` capability. The command is only accepted before a folder is selected (RFC 5161), and what it enables lasts for the current connection; the package handles `UTF8=ACCEPT` itself for non-ASCII searches.

Usage:

```
ImapCon$enable(capabilities, retries = NULL)
```

Arguments:

`capabilities` A `character` vector with the names of the extensions to enable.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `character` vector with the extensions the server confirmed as enabled (possibly empty).

Examples:

```
con$enable("CONDSTORE")
```

`ImapCon$namespace()`: Request the server's namespaces (IMAP `NAMESPACE`, RFC 2342): the personal, other users', and shared namespace prefixes and their hierarchy delimiters. Requires the server `NAMESPACE` capability.

Usage:

```
ImapCon$namespace(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A named `list` with elements `personal`, `other_users` and `shared`, each a `data.frame` with `prefix` and `delimiter` columns, or `NULL` when the server returns `NIL` for that component.

Examples:

```
con$namespace()
```

`ImapCon$id()`: Exchange client/server identification (IMAP `ID`, RFC 2971). Optionally sends the client's id fields and returns the server's id. Requires the server `ID` capability.

Usage:

```
ImapCon$id(fields = NULL, retries = NULL)
```

Arguments:

`fields` A named `character` vector with the client id fields to send, e.g. `c(name = "mRpostman", version = "1.2.1")`. If `NULL` (default), sends `ID NIL` (asks for the server id without disclosing the client id).

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A named `character` vector with the server's id fields (empty when the server returns `NIL`).

Examples:

```
con$id()
con$id(fields = c(name = "mRpostman", version = "1.2.1"))
```

`ImapCon$get_quota_root()`: Get the quota root(s) and quota usage/limits of a mail folder (IMAP `GETQUOTAROOT`, RFC 9208). Requires the server `QUOTA` capability.

Usage:

```
ImapCon$get_quota_root(folder = NULL, retries = NULL, name = NULL)
```

Arguments:

`folder` A `character` string with the mail folder name. If no name is passed, the command uses the previously selected folder.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: A `data.frame` with columns `quota_root`, `resource`, `usage` and `limit` (one row per resource; `STORAGE` is reported by the server in kibibytes).

Examples:

```
con$get_quota_root(name = "INBOX")
```

`ImapCon$get_quota()`: Get the quota usage/limits of a quota root (IMAP `GETQUOTA`, RFC 9208). Requires the server `QUOTA` capability.

Usage:

```
ImapCon$get_quota(quota_root = "", retries = NULL)
```

Arguments:

`quota_root` A `character` string with the quota root name. Default is `""` (the default root). Use `get_quota_root()` to discover the root(s) of a folder.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with columns `quota_root`, `resource`, `usage` and `limit`.

Examples:

```
con$get_quota(quota_root = "")
```

`ImapCon$set_quota()`: Set the resource limits of a quota root (IMAP `SETQUOTA`, RFC 9208). Most servers restrict this command to administrators. Requires the server `QUOTA` capability.

Usage:

```
ImapCon$set_quota(quota_root, storage = NULL, message = NULL, retries = NULL)
```

Arguments:

`quota_root` A `character` string with the quota root name, as returned by `get_quota_root()`.

`storage` `NULL` or the new `STORAGE` limit, in kibibytes.

`message` `NULL` or the new `MESSAGE` limit (number of messages). At least one of the two limits must be given.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with the quota as confirmed by the server (columns `quota_root`, `resource`, `usage`, `limit`).

Examples:

```
con$set_quota(quota_root = "User quota", storage = 2 * 1024^2)
```

`ImapCon$get_metadata()`: Get metadata entries (annotations) of a mail folder or of the server (IMAP `GETMETADATA`, RFC 5464). Requires the server `METADATA` (or `METADATA-SERVER`) capability.

Usage:

```
ImapCon$get_metadata(
  folder = NULL,
  entries,
  depth = NULL,
  max_size = NULL,
  retries = NULL,
  name = NULL
)
```

Arguments:

`folder` A `character` string with the mail folder name, or `NULL` for server-level entries.

`entries` A `character` vector of entry names, e.g. `"/private/comment"` or `"/shared/vendor/..."`.

`depth` `NULL` (default), `"0"`, `"1"`, or `"infinity"`: how many levels below each entry to return.

`max_size` `NULL` (default) or the maximum size, in bytes, of a value to return.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: A `data.frame` with columns `mailbox`, `entry`, and `value` (`NA` when the entry has no value).

Examples:

```
con$get_metadata(folder = "INBOX", entries = "/private/comment")
con$get_metadata(folder = NULL, entries = "/shared/comment")
```

`ImapCon$set_metadata()`: Set (or remove) metadata entries of a mail folder or of the server (IMAP `SETMETADATA`, RFC 5464). Requires the server `METADATA` (or `METADATA-SERVER`) capability.

Usage:

```
ImapCon$set_metadata(folder = NULL, entries, retries = NULL, name = NULL)
```

Arguments:

`folder` A `character` string with the mail folder name, or `NULL` for server-level entries.

`entries` A named `character` vector: the names are the entries, the values the new values; `NA` removes an entry. Values cannot contain line breaks.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$set_metadata(folder = "INBOX", entries = c("/private/comment" = "reviewed"))
con$set_metadata(folder = "INBOX", entries = c("/private/comment" = NA))
```

`ImapCon$get_acl()`: Get the access control list of a mail folder (IMAP `GETACL`, RFC 4314). Requires the server `ACL` capability.

Usage:

```
ImapCon$get_acl(folder = NULL, retries = NULL, name = NULL)
```

Arguments:

`folder` A `character` string with the mail folder name. If no name is passed, the command uses the previously selected folder.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: A `data.frame` with columns `identifier` (a user name, or a group such as `anyone`) and `rights` (a string of right letters, e.g. `"lrwstipekxa"`).

Examples:

```
con$get_acl(folder = "INBOX")
```

`ImapCon$set_acl()`: Set or modify the rights of an identifier on a mail folder (IMAP `SETACL`, RFC 4314). Requires the server `ACL` capability and the `a` (administer) right on the folder.

Usage:

```
ImapCon$set_acl(folder = NULL, identifier, rights, retries = NULL, name = NULL)
```

Arguments:

`folder` A `character` string with the mail folder name. If no name is passed, the command uses the previously selected folder.

`identifier` A `character` string with the user name (or group, e.g. `"anyone"`) whose rights are set.

`rights` A `character` string of right letters. Without a prefix it replaces the current rights (e.g. `"lrs"`); prefixed with `"+"` or `"-"` it adds or removes rights (e.g. `"+w"`, `"-d"`).

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$set_acl(name = "Shared", identifier = "anyone", rights = "lrs")
con$set_acl(name = "Shared", identifier = "anyone", rights = "+w")
```

`ImapCon$delete_acl()`: Remove all rights of an identifier on a mail folder (IMAP `DELETEACL`, RFC 4314). Requires the server `ACL` capability.

Usage:

```
ImapCon$delete_acl(folder = NULL, identifier, retries = NULL, name = NULL)
```

Arguments:

`folder` A `character` string with the mail folder name. If no name is passed, the command uses the previously selected folder.

`identifier` A `character` string with the user name (or group) whose rights are removed.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: TRUE in case the operation is successful.

Examples:

```
con$delete_acl(name = "Shared", identifier = "anyone")
```

ImapCon$list_rights(): List the rights that may be granted to an identifier on a mail folder (IMAP LISTRIGHTS, RFC 4314). Requires the server ACL capability.

Usage:

```
ImapCon$list_rights(folder = NULL, identifier, retries = NULL, name = NULL)
```

Arguments:

folder A character string with the mail folder name. If no name is passed, the command uses the previously selected folder.

identifier A character string with the user name (or group).

retries Number of attempts to connect and execute the command. Default is 1.

name Deprecated alias of folder.

Returns: A list with required (the rights the identifier always has) and optional (a character vector with the sets of rights that may be granted).

Examples:

```
con$list_rights(name = "INBOX", identifier = "anyone")
```

ImapCon$my_rights(): Get the rights of the current user on a mail folder (IMAP MYRIGHTS, RFC 4314). Requires the server ACL capability.

Usage:

```
ImapCon$my_rights(folder = NULL, retries = NULL, name = NULL)
```

Arguments:

folder A character string with the mail folder name. If no name is passed, the command uses the previously selected folder.

retries Number of attempts to connect and execute the command. Default is 1.

name Deprecated alias of folder.

Returns: A character string of right letters.

Examples:

```
con$my_rights(folder = "INBOX")
```

ImapCon$noop(): Issue a NOOP command. It does nothing on the server other than resetting the inactivity autologout timer, which makes it useful as a keep-alive during long idle periods and as a way to keep the connection handle alive between operations.

Usage:

```
ImapCon$noop(retries = NULL)
```

Arguments:

retries Number of attempts to connect and execute the command. Default is 1.

Returns: TRUE in case the operation is successful.

Examples:

```
con$noop()
```

`ImapCon$check()`: Request a checkpoint of the selected mail folder (IMAP `CHECK`). The server performs any implementation-dependent housekeeping of the mailbox, such as flushing its state to disk. The command has no client-observable effect; use `noop()` as a keep-alive.

Usage:

```
ImapCon$check(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$select_folder(folder = "INBOX")
con$check()
```

`ImapCon$list_mail_folders()`: List mail folders in a mailbox.

Usage:

```
ImapCon$list_mail_folders(retries = NULL, detailed = FALSE)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

`detailed` A `logical`. If `TRUE`, issues `LIST ... RETURN (CHILDREN SUBSCRIBED SPECIAL-USE)` (LIST-EXTENDED, RFC 5258) and returns a `data.frame` with one row per folder and its attributes instead of the root/children list. Requires the server `LIST-EXTENDED` capability. Default is `FALSE`.

Returns: A `list` containing the mail folder names and their inherent structure or, with `detailed = TRUE`, a `data.frame` with columns `folder`, `delimiter`, `attributes`, `selectable`, `has_children`, `subscribed`, and `special_use`.

Examples:

```
folders <- con$list_mail_folders()
folders
```

`ImapCon$list_subscribed_folders()`: List the subscribed mail folders in a mailbox (IMAP `LSUB`). Unlike `list_mail_folders()` (which issues `LIST` and returns every folder), this returns only the folders the user is subscribed to.

Usage:

```
ImapCon$list_subscribed_folders(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `list` containing the subscribed mail folder names and their inherent structure.

Examples:

```
subscribed <- con$list_subscribed_folders()
subscribed
```

`ImapCon$list_folders_status()`: List the mail folders together with their status counts in a single round trip (IMAP `LIST ... RETURN (STATUS ...)`, RFC 5819). Equivalent to `list_mail_folders()` followed by `status()` on every folder, but issued as one command. Requires the server `LIST-STATUS` capability.

Usage:

```
ImapCon$list_folders_status(items = c("MESSAGES", "UNSEEN"), retries = NULL)
```

Arguments:

`items` A `character` vector with the status data items to request. Must be a subset of `"MESSAGES"`, `"RECENT"`, `"UIDNEXT"`, `"UIDVALIDITY"`, and `"UNSEEN"`, plus the extension items `"SIZE"` (STATUS=SIZE, RFC 8438) and `"HIGHESTMODSEQ"` (CONDSTORE, RFC 7162), which require the corresponding capability. Default is `c("MESSAGES", "UNSEEN")`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with the column `folder` followed by one numeric column per requested item (`NA` for folders that cannot be selected).

Examples:

```
con$list_folders_status()
con$list_folders_status(items = c("MESSAGES", "UNSEEN", "UIDNEXT"))
```

`ImapCon$list_special_use_folders()`: List the special-use mail folders (IMAP `LIST (SPECIAL-USE)`, RFC 6154), i.e. the folders the server has tagged with a role such as `\Sent`, `\Drafts`, `\Junk`, `\Trash`, `\Archive`, `\All`, or `\Flagged`. Requires the server `SPECIAL-USE` capability.

Usage:

```
ImapCon$list_special_use_folders(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with columns `folder` and `special_use` (one row per folder/attribute).

Examples:

```
con$list_special_use_folders()
```

`ImapCon$select_folder()`: Select a mail folder.

Usage:

```
ImapCon$select_folder(
  folder = NULL,
  mute = NULL,
  retries = NULL,
  condstore = FALSE,
  name = NULL
)
```

Arguments:

`folder` A string containing the name of an existing mail folder on the user's mailbox.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

condstore A logical. If TRUE, issues SELECT ... (CONDSTORE) (RFC 7162), so that the server reports modification sequences in this session. The folder's HIGHESTMODSEQ, when reported, is kept in concon_paramshighestmodseq. Default is FALSE.

name Deprecated alias of folder.

Returns: A list containing the mail folder names and their inherent structure.

Examples:

```
con$select_folder(folder = "INBOX")
```

ImapCon$resync_folder(): Select a mail folder with QRESYNC (RFC 7162) and report what changed since a known state: the UIDs expunged since the given modification sequence and the current flags of the messages modified since then. Requires the server QRESYNC capability (and UNSELECT if a folder is currently selected, since the extension must be enabled with no folder selected).

Usage:

```
ImapCon$resync_folder(
  folder = NULL,
  uidvalidity,
  modseq,
  retries = NULL,
  name = NULL
)
```

Arguments:

folder A character string with the mail folder name.

uidvalidity The folder's UIDVALIDITY at the time of the known state (from status() or a previous resync_folder()).

modseq The modification sequence of the known state (e.g. the HIGHESTMODSEQ recorded then).

retries Number of attempts to connect and execute the command. Default is 1.

name Deprecated alias of folder.

Returns: A list with vanished (an integer vector of expunged UIDs), changed (a data.frame with seq, uid, flags, modseq), highestmodseq, uidvalidity, uidnext, and exists. The folder is left selected.

Examples:

```
st <- con$status("INBOX", items = c("UIDVALIDITY", "HIGHESTMODSEQ"))
# ... later:
delta <- con$resync_folder("INBOX", uidvalidity = st[["UIDVALIDITY"]],
                           modseq = st[["HIGHESTMODSEQ"]])
delta$vanished; delta$changed
```

ImapCon$fetch_changes(): Fetch the flag changes (and, with QRESYNC, the expunges) in the selected folder since a modification sequence (UID FETCH 1:* (FLAGS MODSEQ) (CHANGEDSINCE ... VANISHED), RFC 7162). Requires the server CONDSTORE capability, and QRESYNC for vanished = TRUE.

Usage:

```
ImapCon$fetch_changes(modseq, vanished = TRUE, retries = NULL)
```

Arguments:

`modseq` The modification sequence to compare with.

`vanished` A `logical`. If `TRUE` (default), the UIDs expunged since `modseq` are reported as well.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `list` with `vanished` (an integer vector of UIDs, empty unless `vanished = TRUE`) and `changed` (a `data.frame` with `seq`, `uid`, `flags`, `modseq`).

Examples:

```
con$select_folder("INBOX", condstore = TRUE)
last <- con$con_params$highestmodseq
# ... later in the session:
con$fetch_changes(modseq = last)
```

`ImapCon$close_folder()`**:** Close the currently selected mail folder (IMAP `CLOSE`), permanently removing the messages flagged `\Deleted`. After this, no folder is selected.

Usage:

```
ImapCon$close_folder(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$select_folder("INBOX")
con$close_folder()
```

`ImapCon$unselect_folder()`**:** Close the currently selected mail folder **without** expunging (IMAP `UNSELECT`, RFC 3691). Requires the server `UNSELECT` capability. After this, no folder is selected.

Usage:

```
ImapCon$unselect_folder(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$select_folder("INBOX")
con$unselect_folder()
```

`ImapCon$examine_folder()`**:** Examine the number of messages in a mail folder.

Usage:

```
ImapCon$examine_folder(folder = NULL, retries = NULL, name = NULL)
```

Arguments:

`folder` A `character` string containing the name of an existing mail folder on the user's mailbox. If no name is passed, the command will be executed using the previously selected mail folder name.

`retries` Number of attempts to connect and execute the command. Default is 1.

`name` Deprecated alias of `folder`.

Returns: A `vector` (with names `"EXISTS"` and `"RECENT"`) containing the number of messages in each category.

Examples:

```
con$select_folder(folder = "INBOX")
con$examine_folder()

# or directly:
con$examine_folder("Sent")
```

`ImapCon$status()`**:** Request the status of a mail folder without selecting it. Unlike `examine_folder()`, this does not change the currently selected folder.

Usage:

```
ImapCon$status(
  folder = NULL,
  items = c("MESSAGES", "RECENT", "UIDNEXT", "UIDVALIDITY", "UNSEEN"),
  retries = NULL,
  name = NULL
)
```

Arguments:

`folder` A `character` string containing the name of an existing mail folder on the user's mailbox. If no name is passed, the command will be executed using the previously selected mail folder name.

`items` A `character` vector with the status data items to request. Must be a subset of `"MESSAGES"`, `"RECENT"`, `"UIDNEXT"`, `"UIDVALIDITY"`, and `"UNSEEN"`, plus the extension items `"SIZE"` (STATUS=SIZE, RFC 8438) and `"HIGHESTMODSEQ"` (CONDSTORE, RFC 7162), which require the corresponding capability. Default is all of them.

`retries` Number of attempts to connect and execute the command. Default is 1.

`name` Deprecated alias of `folder`.

Returns: A named `numeric` vector with the requested status counts.

Examples:

```
con$status(folder = "INBOX")

# or, for the selected folder and specific items only:
con$select_folder("INBOX")
con$status(items = c("MESSAGES", "UNSEEN"))
```

`ImapCon$create_folder()`**:** Create a new mail folder.

Usage:

```
ImapCon$create_folder(
  folder = NULL,
  mute = NULL,
  retries = NULL,
  special_use = NULL,
  name = NULL
)
```

Arguments:

`folder` A string containing the name of the new mail folder to be created.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`special_use` `NULL` (default) or a `character` vector of special-use attributes to assign to the new folder, e.g. `"\Archive"` (CREATE-SPECIAL-USE, RFC 6154; requires that server capability).

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$create_folder(folder = "New Folder Name")
```

`ImapCon$rename_folder()`**:** Rename a mail folder.

Usage:

```
ImapCon$rename_folder(
  folder = NULL,
  new_name,
  reselect = TRUE,
  mute = NULL,
  retries = NULL,
  name = NULL
)
```

Arguments:

`folder` A string containing the name of the mail folder to be renamed. If no name is passed, the command will be executed using the previously selected mail folder name.

`new_name` A string containing the new name to be assigned.

`reselect` A logical. If `TRUE`, calls `select_folder(folder = to_folder)` under the hood before returning the output. Default is `TRUE`.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$select_folder(folder = "Folder A")
con$rename_folder(new_name = "Folder B")
# or directly:
con$rename_folder(folder = "Folder A", new_name = "Folder B")
```

`ImapCon$delete_folder()`**:** Delete a mail folder.

Usage:

```
ImapCon$delete_folder(folder = NULL, mute = NULL, retries = NULL, name = NULL)
```

Arguments:

`folder` A string containing the name of the mail folder to be deleted.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$delete_folder(folder = "Folder to remove")
```

`ImapCon$subscribe_folder()`**:** Subscribe to a mail folder (IMAP `SUBSCRIBE`), adding it to the set returned by `list_subscribed_folders()`.

Usage:

```
ImapCon$subscribe_folder(
  folder = NULL,
  mute = NULL,
  retries = NULL,
  name = NULL
)
```

Arguments:

`folder` A string containing the name of the mail folder to subscribe to.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$subscribe_folder(folder = "INBOX")
```

`ImapCon$unsubscribe_folder()`**:** Unsubscribe from a mail folder (IMAP `UNSUBSCRIBE`), removing it from the set returned by `list_subscribed_folders()`.

Usage:

```
ImapCon$unsubscribe_folder(
  folder = NULL,
  mute = NULL,
  retries = NULL,
  name = NULL
)
```

Arguments:

`folder` A string containing the name of the mail folder to unsubscribe from.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`name` Deprecated alias of `folder`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$unsubscribe_folder(folder = "INBOX")
```

`ImapCon$list_flags()`: List flags in a selected mail folder

Usage:

```
ImapCon$list_flags(retries = NULL)
```

Arguments:

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: `TRUE` in case the operation is successful.

Examples:

```
con$select_folder(folder = "INBOX")
con$list_flags()
```

`ImapCon$sort()`: Sort messages on the server (IMAP `SORT`, RFC 5256). Returns the message ids ordered by the server according to the sort keys. Requires the server to advertise the `SORT` capability (check with `list_server_capabilities()`).

Usage:

```
ImapCon$sort(
  by = "DATE",
  reverse = FALSE,
  criteria = "ALL",
  use_uid = NULL,
  char_set = "UTF-8",
  return = NULL,
  retries = NULL
)
```

Arguments:

`by` A `character` vector of sort keys, a subset of `"ARRIVAL"`, `"CC"`, `"DATE"`, `"FROM"`, `"SIZE"`, `"SUBJECT"`, and `"TO"`. Default is `"DATE"`.

`reverse` A `logical`. If `TRUE`, each sort key is prefixed with `REVERSE` (descending order). Default is `FALSE`.

`criteria` A `character` string with the search criteria that restricts the set to be sorted. Default is `"ALL"`.

`use_uid` A `logical`. If `TRUE`, issues `UID SORT` and returns UIDs instead of sequence numbers. Default is `FALSE`.

`char_set` A `character` string with the charset of the search criteria. Default is `"UTF-8"`.

`return` `NULL` (default) or a `character` vector with any of `"COUNT"`, `"MIN"`, `"MAX"`, and `"ALL"`. When given, issues `SORT RETURN (...)` (ESORT, RFC 5267) and returns only the requested items, computed by the server in sort order, as a named `list`. Requires the server `ESORT` capability.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: An `integer` vector of message ids in the server-provided (sorted) order.

Examples:

```
con$select_folder("INBOX")
con$sort(by = "DATE", reverse = TRUE)
```

`ImapCon$thread()`: Thread messages on the server (IMAP `THREAD`, RFC 5256). Returns the messages grouped into threads. Requires the server to advertise a `THREAD=` capability (check with `list_server_capabilities()`).

Usage:

```
ImapCon$thread(
  algorithm = "REFERENCES",
  criteria = "ALL",
  use_uid = NULL,
  char_set = "UTF-8",
  retries = NULL
)
```

Arguments:

`algorithm` A `character` string with the threading algorithm, either `"REFERENCES"` or `"ORDEREDSUBJECT"`. Default is `"REFERENCES"`.

`criteria` A `character` string with the search criteria that restricts the set to be threaded. Default is `"ALL"`.

`use_uid` A `logical`. If `TRUE`, issues `UID THREAD` and returns UIDs instead of sequence numbers. Default is `FALSE`.

`char_set` A `character` string with the charset of the search criteria. Default is `"UTF-8"`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `list` of `integer` vectors, one per top-level thread.

Examples:

```
con$select_folder("INBOX")
con$thread(algorithm = "REFERENCES")
```

`ImapCon$search()`: Execute a custom search

Usage:

```
ImapCon$search(
  request,
  negate = FALSE,
  use_uid = NULL,
  esearch = FALSE,
  save = FALSE,
  retries = NULL
)
```

Arguments:

`request` A string directly specifying what to search or constructed by a combination of relational-operator-helper-functions `OR` and `AND`, and criteria helper functions such as `before`, `since`, `on`, `sent_before`, `sent_since`, `sent_on`, `flag`, `string`, `smaller_than`, `larger_than`, `younger_than`, or `older_than`.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERIA". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`save` A logical. Default is `FALSE`. If `TRUE`, the result is saved on the server (`SEARCH RETURN (SAVE)`, SEARCHRES, RFC 5182) instead of being returned, and the method returns the `"$"` reference, which the fetch, flag, copy, move, and delete methods accept as `msg_id`. Requires the server `SEARCHRES` capability.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `numeric vector` with the ids of the messages that match the search, or `integer(0)` when none does. With `save = TRUE` the result set is also kept on the server (SEARCHRES) and returned invisibly.

Examples:

```
con$select_folder(folder = "INBOX")
# ex1
con$search(OR(before(date_char = "17-Apr-2015"),
              string(expr = "John", where = "FROM")))

# ex2
con$search(AND(smaller_than(size = "512000"),
               string(expr = "John", where = "FROM"),
               string(expr = "@ksu.edu", where = "CC")))
```

`ImapCon$search_larger_than()`: Search by size (LARGER)

Usage:

```
ImapCon$search_larger_than(
  size,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`size` An integer specifying the size in bytes to be used as the search criterion.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

flag An optional argument that sets one or more flags as an additional filter to the search. Use ImapCon$list_flags() to list the flags in a selected mail folder. Default is NULL.

esearch A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with ImapCon$list_server_capabilities().

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
# search for messages with size larger than 512Kb
con$search_larger_than(size = 512000)
```

ImapCon$search_smaller_than(): Search by size (SMALLER)

Usage:

```
ImapCon$search_smaller_than(
  size,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

size An integer specifying the size in bytes to be used as the search criterion.

negate If TRUE, negates the search and seeks for "NOT SEARCH CRITERION". Default is FALSE.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

flag An optional argument that sets one or more flags as an additional filter to the search. Use ImapCon$list_flags() to list the flags in a selected mail folder. Default is NULL.

esearch A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole

sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with ImapCon$list_server_capabilities().

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for messages with size smaller than 512Kb
con$search_smaller_than(size = 512000)
```

ImapCon$search_before(): Search by internal date (BEFORE)

Usage:

```
ImapCon$search_before(
  date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

date_char A character string with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use Date or POSIX* like objects, since IMAP servers use this uncommon date format.

negate If TRUE, negates the search and seeks for "NOT SEARCH CRITERION". Default is FALSE.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

flag An optional argument that sets one or more flags as an additional filter to the search. Use ImapCon$list_flags() to list the flags in a selected mail folder. Default is NULL.

esearch A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with ImapCon$list_server_capabilities().

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_before(date = "02-Jan-2020", use_uid = TRUE)
```

`ImapCon$search_since()`: Search by internal date (SINCE)

Usage:

```
ImapCon$search_since(
  date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is `NULL`.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for messages with date since "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_since(date = "02-Jan-2020", use_uid = TRUE)
```

`ImapCon$search_on()`: Search by internal date (ON)

Usage:

```
ImapCon$search_on(
  date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is `NULL`.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for messages received on date "02-Jan-2020", presenting the
#... results as unique identifiers (UID)
con$search_on(date = "02-Jan-2020", use_uid = TRUE)
```

`ImapCon$search_period()`**:** Search by internal date (Period)

Usage:

```
ImapCon$search_period(
  since_date_char,
  before_date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`since_date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`before_date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is `NULL`.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for all messages in the mail folder, EXCEPT (negate = TRUE) by
#... those received between the dates "02-Jan-2020" and "22-Mar-2020"
con$search_period(since_date_char = "02-Jan-2020",
                  before_date_char = "22-Mar-2020",
                  negate = TRUE)
```

`ImapCon$search_sent_before()`: Search by origination date (RFC 2822 Header - SENT BEFORE)

Usage:

```
ImapCon$search_sent_before(
  date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are

reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is NULL.

`esearch` A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_sent_before(date = "02-Jan-2020", use_uid = TRUE)
```

`ImapCon$search_sent_since()`: Search by origination date (RFC 2822 Header - SENT SINCE)

Usage:

```
ImapCon$search_sent_since(
  date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`date_char` A character string with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use Date or POSIX* like objects, since IMAP servers use this uncommon date format.

`negate` If TRUE, negates the search and seeks for "NOT SEARCH CRITERION". Default is FALSE.

`use_uid` Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is NULL.

`esearch` A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is 1.

Returns: A `numeric vector` containing the message ids.

Examples:

```
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_sent_since(date = "02-Jan-2020", use_uid = TRUE)
```

`ImapCon$search_sent_on()`: Search by origination date (RFC 2822 Header - SENT ON)

Usage:

```
ImapCon$search_sent_on(
  date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is `NULL`.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is 1.

Returns: A `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for messages received on date "02-Jan-2020", presenting the
#... results as unique identifiers (UID)
con$search_sent_on(date = "02-Jan-2020", use_uid = TRUE)
```

`ImapCon$search_sent_period()`: Search by origination date (RFC 2822 Header - SENT Period)

Usage:

```
ImapCon$search_sent_period(
  since_date_char,
  before_date_char,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`since_date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`before_date_char` A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this uncommon date format.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`flag` An optional argument that sets one or more flags as an additional filter to the search. Use `ImapCon$list_flags()` to list the flags in a selected mail folder. Default is `NULL`.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for all messages in the mail folder, EXCEPT (negate = TRUE) by
#... those received between the dates "02-Jan-2020" and "22-Mar-2020"
con$search_sent_period(since_date_char = "02-Jan-2020",
                  before_date_char = "22-Mar-2020",
                  negate = TRUE)
```

`ImapCon$search_flag()`**:** Search by flag(s)

Usage:

```
ImapCon$search_flag(
  name,
  negate = FALSE,
  use_uid = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`name` A string containing one or more flags to search for. Use `ImapCon$list_flags()` to list the flags in a selected mail folder.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`esearch` A logical. Default is `FALSE`. If the IMAP server has `ESEARCH` capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as `{1 2 3 4 5}`, it will be presented as `{1:5}`, which decreases transmission costs. This argument can be used along with `buffersize` to avoid results stripping. Check if your IMAP server supports `ESEARCH` with `ImapCon$list_server_capabilities()`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for all messages in the mail folder that are marked as "SEEN" AND
#.. "ANSWERED"
con$search_flag(name = c("SEEN", "ANSWERED"))
```

`ImapCon$search_older_than()`**:** Search WITHIN a specific time (OLDER)

Usage:

```
ImapCon$search_older_than(
  seconds,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

`seconds` An integer specifying the number of seconds to be used as the search criterion.

`negate` If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERION". Default is `FALSE`.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

flag An optional argument that sets one or more flags as an additional filter to the search. Use ImapCon$list_flags() to list the flags in a selected mail folder. Default is NULL.

esearch A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with ImapCon$list_server_capabilities().

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for all messages received in the last hour (not older than 3600 seconds)
con$search_older_than(seconds = 3600, negate = TRUE)
```

ImapCon$search_younger_than(): Search WITHIN a specific time (YOUNGER)

Usage:

```
ImapCon$search_younger_than(
  seconds,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

seconds An integer specifying the number of seconds to be used as the search criterion.

negate If TRUE, negates the search and seeks for "NOT SEARCH CRITERION". Default is FALSE.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

flag An optional argument that sets one or more flags as an additional filter to the search. Use ImapCon$list_flags() to list the flags in a selected mail folder. Default is NULL.

esearch A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole

sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with ImapCon$list_server_capabilities().

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for all messages received in the last hour (younger than 3600 seconds)
con$search_younger_than(seconds = 3600)
```

ImapCon$search_string(): Search by string or expression

Usage:

```
ImapCon$search_string(
  expr,
  where,
  negate = FALSE,
  use_uid = NULL,
  flag = NULL,
  esearch = FALSE,
  retries = NULL
)
```

Arguments:

expr A character string specifying the word or expression to search for in messages.

where A mandatory character string specifying in which message's Section or Header Field to search for the provided string.

negate If TRUE, negates the search and seeks for "NOT SEARCH CRITERION". Default is FALSE.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

flag An optional argument that sets one or more flags as an additional filter to the search. Use ImapCon$list_flags() to list the flags in a selected mail folder. Default is NULL.

esearch A logical. Default is FALSE. If the IMAP server has ESEARCH capability, it can be used to optimize search results. It will condense the results: instead of writing down the whole sequences of messages' ids, such as {1 2 3 4 5}, it will be presented as {1:5}, which decreases transmission costs. This argument can be used along with buffersize to avoid results stripping. Check if your IMAP server supports ESEARCH with ImapCon$list_server_capabilities().

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# search for messages with "@k-state.edu" in the FROM field
con$search_string(expr = "@k-state.edu", where = "FROM")
```

`ImapCon$fetch_body()`: Fetch message body (message's full content)

Usage:

```
ImapCon$fetch_body(
  msg_id,
  use_uid = NULL,
  mime_level = NULL,
  peek = TRUE,
  partial = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = NULL,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`mime_level` An `integer` specifying MIME multipart to fetch from the message's body. Default is `NULL`, which retrieves the full body content.

`peek` If `TRUE`, it does not mark messages as "read" after fetching. Default is `TRUE`.

`partial` `NULL` or a character string with format "startchar.endchar" indicating the size (in characters) of a message slice to fetch. Default is `NULL`, which will fetch the full specified content.

`write_to_disk` If `TRUE`, writes the fetched content of each message to a text file in a local folder inside the working directory, also returning the results with `invisible()`. Default is `FALSE`.

`keep_in_mem` If `TRUE` (default), keeps a copy of each fetch result in the returned list. It can only be set `FALSE` along with `write_to_disk = TRUE`, to write the results to disk without keeping them in memory.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. It is only effective when `write_to_disk = TRUE` and `keep_in_mem = FALSE`. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `list` with the fetch contents or a logical if `write_to_disk = TRUE` and `keep_in_mem = FALSE`.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and fetch the results (saving to disk) using the pipe
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_body(write_to_disk = TRUE, keep_in_mem = FALSE)

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")

con$fetch_body(msg = res, write_to_disk = TRUE, keep_in_mem = FALSE)
```

`ImapCon$fetch_header()`: Fetch message header

Usage:

```
ImapCon$fetch_header(
  msg_id,
  use_uid = NULL,
  fields = NULL,
  negate_fields = FALSE,
  peek = TRUE,
  partial = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = NULL,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`fields` An optional `character vector` specifying which field(s) will be fetched from the message's header. If none is specified, it will fetch the full header.

`negate_fields` If `TRUE`, negates the operation and seeks for "NOT in the field". Default is `FALSE`.

`peek` If `TRUE`, it does not mark messages as "read" after fetching. Default is `TRUE`.

`partial` `NULL` or a character string with format "startchar.endchar" indicating the size (in characters) of a message slice to fetch. Default is `NULL`, which will fetch the full specified content.

`write_to_disk` If `TRUE`, writes the fetched content of each message to a text file in a local folder inside the working directory, also returning the results with `invisible()`. Default is `FALSE`.

`keep_in_mem` If `TRUE` (default), keeps a copy of each fetch result in the returned list. It can only be set `FALSE` along with `write_to_disk = TRUE`, to write the results to disk without keeping them in memory.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. It is only effective when `write_to_disk = TRUE` and `keep_in_mem = FALSE`. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `list` with the fetch contents or a logical if `write_to_disk = TRUE` and `keep_in_mem = FALSE`.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and fetch the results (also saving to disk) using the pipe
out <- con$search_string(expr = "@k-state.edu", where = "CC") %>%
  con$fetch_header()

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "CC")
out <- con$fetch_header()
```

`ImapCon$fetch_metadata()`**:** Fetch message metadata

Usage:

```
ImapCon$fetch_metadata(
  msg_id,
  use_uid = NULL,
  attribute = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = NULL,
  retries = NULL,
  changed_since = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`attribute` An optional `character vector` specifying one or more attributes of the metadata of a message to fetch. See metadata_options. The extension attributes `"PREVIEW"` (RFC 8970), `"SAVEDATE"` (RFC 8514), and `"MODSEQ"` (CONDSTORE, RFC 7162) may also be requested when the server advertises the corresponding capability.

`write_to_disk` If `TRUE`, writes the fetched content of each message to a text file in a local folder inside the working directory, also returning the results with `invisible()`. Default is `FALSE`.

`keep_in_mem` If `TRUE` (default), keeps a copy of each fetch result in the returned list. It can only be set `FALSE` along with `write_to_disk = TRUE`, to write the results to disk without keeping them in memory.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. It is only effective when `write_to_disk = TRUE` and `keep_in_mem = FALSE`. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`changed_since` `NULL` (default) or a modification sequence: with it only the messages modified after that sequence are returned (`CHANGEDSINCE`, CONDSTORE, RFC 7162), each with its `MODSEQ`.

Returns: A `list` with the fetch contents or a logical if `write_to_disk = TRUE` and `keep_in_mem = FALSE`.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and fetch the results using the pipe
out <- con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_metadata()

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
out <- con$fetch_metadata(msg = res)
```

`ImapCon$fetch_preview()`**:** Fetch the server-generated preview of messages (IMAP `FETCH ... (PREVIEW)`, RFC 8970): a short text snippet of each message, produced by the server without transferring the message body. Requires the server `PREVIEW` capability.

Usage:

```
ImapCon$fetch_preview(msg_id, use_uid = NULL, retries = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids, or the `"$"` reference of a saved search.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, the command is performed with UIDs and the result is named by UID.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A named `character` vector with one preview per message (`NA` when the server has none).

Examples:

```
con$select_folder(folder = "INBOX")
con$search_flag("UNSEEN") %>% con$fetch_preview()
```

`ImapCon$fetch_envelope()`**:** Fetch the envelope of messages parsed into a data frame (IMAP `FETCH ... (ENVELOPE)`): date, subject, and the address lists, with RFC 2047 encoded words decoded. See `parse_envelope`.

Usage:

```
ImapCon$fetch_envelope(msg_id, use_uid = NULL, retries = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids, or the `"$"` reference of a saved search.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). If TRUE, the command is performed with UIDs and the first column is uid.

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A data.frame with one row per message: id (or uid), date, subject, from, sender, reply_to, to, cc, bcc, in_reply_to, and message_id.

Examples:

```
con$select_folder(folder = "INBOX")
con$search_since(date_char = "01-Jan-2026") %>% con$fetch_envelope()
```

ImapCon$fetch_bodystructure(): Fetch the MIME structure of messages parsed into a data frame of parts (IMAP FETCH ... (BODYSTRUCTURE)), one row per part with its section number, type, charset, filename, encoding, size, and disposition. See parse_bodystructure.

Usage:

```
ImapCon$fetch_bodystructure(msg_id, use_uid = NULL, retries = NULL)
```

Arguments:

msg_id A numeric vector containing one or more message ids, or the "$" reference of a saved search.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). If TRUE, the command is performed with UIDs and the first column is uid.

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A data.frame with one row per MIME part of each message.

Examples:

```
con$select_folder(folder = "INBOX")
parts <- con$fetch_bodystructure(msg_id = 1:10)
parts[parts$is_attachment, ]
```

ImapCon$fetch_text(): Fetch message text

Usage:

```
ImapCon$fetch_text(
  msg_id,
  use_uid = NULL,
  peek = TRUE,
  partial = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = NULL,
  base64_decode = FALSE,
  retries = NULL
)
```

Arguments:

msg_id A numeric vector containing one or more message ids.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It

may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

peek If TRUE, it does not mark messages as "read" after fetching. Default is TRUE.

partial NULL or a character string with format "startchar.endchar" indicating the size (in characters) of a message slice to fetch. Default is NULL, which will fetch the full specified content.

write_to_disk If TRUE, writes the fetched content of each message to a text file in a local folder inside the working directory, also returning the results with invisible(). Default is FALSE.

keep_in_mem If TRUE (default), keeps a copy of each fetch result in the returned list. It can only be set FALSE along with write_to_disk = TRUE, to write the results to disk without keeping them in memory.

mute A logical. If TRUE, mutes the confirmation message when the command is successfully executed. It is only effective when write_to_disk = TRUE and keep_in_mem = FALSE. Default is FALSE.

base64_decode If TRUE, tries to guess and decode the fetched text from base64 format to character. Default is FALSE.

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A list with the fetch contents or a logical if write_to_disk = TRUE and keep_in_mem = FALSE.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and partially fetch the results using the pipe
# first 200 characters, writing to disk, silence results in the console
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_text(partial = "0.200",
                 write_to_disk = TRUE,
                 keep_in_mem = FALSE)

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$fetch_text(msg = res,
               partial = "0.200",
               write_to_disk = TRUE,
               keep_in_mem = FALSE)
```

ImapCon$copy_msg(): Copy message(s) between the selected folder and another one

Usage:

```
ImapCon$copy_msg(
  msg_id,
  use_uid = NULL,
  folder = NULL,
  reselect = TRUE,
  mute = NULL,
```

```
  retries = NULL,
  to_folder = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`folder` A `character` string specifying the folder to which the messages will be copied.

`reselect` A logical. If `TRUE`, calls `ImapCon$select_folder(folder = to_folder)` under the hood before returning the output. Default is `TRUE`.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`to_folder` Deprecated alias of `folder`.

Returns: An invisible `numeric vector` containing the message ids. When the server advertises `UIDPLUS` (RFC 4315), the vector carries a `"copyuid"` attribute: a `data.frame` mapping each `source_uid` to the `dest_uid` assigned in the destination folder.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and copy the results to another folder
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$copy(folder = "Sent")

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$copy(msg = res, folder = "Sent")
```

`ImapCon$move_msg()`**:** Move message(s) between the selected folder and another one

Usage:

```
ImapCon$move_msg(
  msg_id,
  use_uid = NULL,
  folder = NULL,
  reselect = TRUE,
  mute = NULL,
  retries = NULL,
  to_folder = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`folder` A `character` string specifying the folder to which the messages will be copied.

`reselect` A logical. If `TRUE`, calls `ImapCon$select_folder(folder = to_folder)` under the hood before returning the output. Default is `TRUE`.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`to_folder` Deprecated alias of `folder`.

Returns: An invisible `numeric vector` containing the message ids. When the server advertises `UIDPLUS` (RFC 4315), the vector carries a `"copyuid"` attribute: a `data.frame` mapping each `source_uid` to the `dest_uid` assigned in the destination folder.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and copy the results to another folder
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$move(folder = "Sent")

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$move(msg = res, folder = "Sent")
```

`ImapCon$append_msg()`: Append a full RFC 822 message to a mail folder (IMAP `APPEND`). Useful to save a message to folders such as `Drafts` or `Sent`. Unlike the other operations this is performed by an upload to the folder. The message is stored with the flags given in `flags` (none by default). When the server advertises `UIDPLUS` (RFC 4315), the UID assigned to the message is returned.

Usage:

```
ImapCon$append_msg(
  message,
  folder = NULL,
  flags = NULL,
  mute = NULL,
  retries = NULL
)
```

Arguments:

`message` A `character` string or `raw` vector with the full RFC 822 message (headers and body).

`folder` A `character` string with the destination folder. If no folder is passed, the previously selected folder is used.

`flags` `NULL` (default) or a `character` vector with the flags to store with the message: any of `"Seen"`, `"Flagged"`, `"Answered"`, `"Draft"`, and `"Deleted"`. Requires libcurl >= 8.13; earlier versions ignore this argument and always store the message with `\Seen`.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: Invisibly, the UID assigned to the appended message when the server reports it (`APPENDUID` response code, UIDPLUS), or `NA` otherwise.

Examples:

```
msg <- paste("From: me@example.com", "To: you@example.com",
             "Subject: Hi", "", "Message body.", sep = "\r\n")
con$append_msg(message = msg, folder = "Drafts", flags = "Draft")
```

`ImapCon$esearch_count()`: Count the number of messages with a specific flag(s) in a folder (depends on ESEARCH capability)

Usage:

```
ImapCon$esearch_count(flag, use_uid = NULL, retries = NULL)
```

Arguments:

`flag` A mandatory parameter that specifies one or more flags as a filter to the counting operation. Use `ImapCon$list_flags()` to list the flags in a selected mail folder.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A numeric `vector` of length `1` containing the number of messages in the folder that meet the specified criteria.

Examples:

```
con$select_folder(folder = "INBOX")
# count the number of messages marked as "Flagged" and "Answered"
con$esearch_count(flag = c("Flagged", "Answered"))
```

`ImapCon$delete_msg()`: Delete message(s) in the selected mail folder

Usage:

```
ImapCon$delete_msg(msg_id, use_uid = NULL, mute = NULL, retries = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: An invisible `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# delete messages 70 to 73
con$delete_msg(msg_id = 70:73)
```

`ImapCon$expunge()`**:** Permanently removes all or specific messages marked as deleted from the selected folder

Usage:

```
ImapCon$expunge(msg_id = NULL, mute = NULL, retries = NULL, msg_uid = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more messages UIDs. Only UIDs are allowed in this operation (note the "u" in msg_*u*id). Expunging specific messages (`UID EXPUNGE`) requires the server `UIDPLUS` capability (RFC 4315); a plain expunge does not.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`msg_uid` Deprecated alias of `msg_id`.

Returns: `TRUE` if the operation is successful.

Examples:

```
con$select_folder(folder = "INBOX")
# remove every message marked as deleted
con$expunge()
# or only a specific one (UIDPLUS servers)
con$expunge(msg_id = 71)
```

`ImapCon$esearch_min_id()`**:** Search the minimum message id in the selected mail folder (depends on ESEARCH capability)

Usage:

```
ImapCon$esearch_min_id(flag, use_uid = NULL, retries = NULL)
```

Arguments:

`flag` A mandatory parameter that specifies one or more flags as a filter to the searching operation. Use `ImapCon$list_flags()` to list the flags in a selected mail folder.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A numeric vector of length 1 containing the minimum message id in the folder.

Examples:

```
con$select_folder(folder = "INBOX")
# Search the minimum id of messages marked as "Answered"
con$esearch_min_id(flag = "Answered")
```

`ImapCon$esearch_max_id()`: Search the maximum message id in the selected mail folder (depends on ESEARCH capability)

Usage:

```
ImapCon$esearch_max_id(flag, use_uid = NULL, retries = NULL)
```

Arguments:

`flag` A mandatory parameter that specifies one or more flags as a filter to the searching operation. Use `ImapCon$list_flags()` to list the flags in a selected mail folder.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, results will be presented as message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier, and results are presented as such. UIDs are always the same during the life cycle of a message in a mail folder.

`retries` Number of attempts to connect and execute the command. Default is 1.

Returns: A numeric vector of length 1 containing the maximum message id in the folder.

Examples:

```
con$select_folder(folder = "INBOX")
# Search the minimum id of messages marked as "Seen"
con$esearch_max_id(flag = "Seen")
```

`ImapCon$add_flags()`: Add flags to one or more messages

Usage:

```
ImapCon$add_flags(
  msg_id,
  use_uid = NULL,
  flags_to_set,
  mute = NULL,
  retries = NULL,
  unchanged_since = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`flags_to_set` A `character vector` containing one or more flag names to add to the specified message ids. If the flag to be set is a system flag, such as `\SEEN`, `\ANSWERED`, the name should be preceded by two backslashes \.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`unchanged_since` `NULL` (default) or a modification sequence: with it the `STORE` is conditional (`UNCHANGEDSINCE`, CONDSTORE, RFC 7162) and only the messages not modified after that sequence are updated; the ids the server refused are returned in the `"modified"` attribute of the result.

Returns: An invisible `numeric vector` containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# Add the "\Seen" permanent flag to the messages received in the last hour
con$search_younger_than(seconds = 3600) %>% # depends on the WITHIN extension
  con$add_flags(flags_to_set = "\\Seen")
```

`ImapCon$replace_flags()`**:** Replace the current flags of one or more messages

Usage:

```
ImapCon$replace_flags(
  msg_id,
  use_uid = NULL,
  flags_to_set,
  mute = NULL,
  retries = NULL,
  unchanged_since = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`flags_to_set` A `character vector` containing one or more flag names that will replace the current ones. If the flag to be set is a system flag, such as `\SEEN`, `\ANSWERED`, the name should be preceded by two backslashes \.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

`unchanged_since` `NULL` (default) or a modification sequence: with it the `STORE` is conditional (`UNCHANGEDSINCE`, CONDSTORE, RFC 7162) and only the messages not modified after that sequence are updated; the ids the server refused are returned in the `"modified"` attribute of the result.

Returns: An invisible numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# Replace the current flags of the messages in the search results for the
#.. flags "\UNSEEN" and "\Flagged"
con$search_since(date_char = "20-Aug-2020") %>%
  con$replace_flags(flags_to_set = c("\\UNSEEN", "\\Flagged"))
```

`ImapCon$remove_flags()`**:** Remove flag(s) of one or more messages

Usage:

```
ImapCon$remove_flags(
  msg_id,
  use_uid = NULL,
  flags_to_unset,
  mute = NULL,
  retries = NULL,
  unchanged_since = NULL
)
```

Arguments:

`msg_id` A numeric vector containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (TRUE since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`flags_to_unset` A character vector containing one or more flag names that will be unset (removed). If the flag to be removed is a system flag, such as \SEEN, \ANSWERED, the name should be preceded by two backslashes \.

`mute` A logical. If TRUE, mutes the confirmation message when the command is successfully executed. Default is FALSE.

`retries` Number of attempts to connect and execute the command. Default is 1.

`unchanged_since` NULL (default) or a modification sequence: with it the STORE is conditional (UNCHANGEDSINCE, CONDSTORE, RFC 7162) and only the messages not modified after that sequence are updated; the ids the server refused are returned in the "modified" attribute of the result.

Returns: An invisible numeric vector containing the message ids.

Examples:

```
con$select_folder(folder = "INBOX")
# Remove the "\SEEN" flag from the messages in the search result
con$search_since(date_char = "20-Aug-2020") %>%
  con$remove_flags(flags_to_unset = "\\UNSEEN")
```

`ImapCon$attachments()`**:** Download the attachments of messages, guided by each message's BODYSTRUCTURE: exact MIME part numbers, nested multiparts included, one BODY.PEEK[part]

fetch per attachment, decoded according to the declared transfer encoding. This is the canonical attachment path since the 2026 refactoring; it replaces `fetch_attachments()` and `fetch_attachment_parts()`.

Usage:

```
ImapCon$attachments(
  msg_id,
  use_uid = NULL,
  parts = NULL,
  content_disposition = "both",
  dest = ".",
  override = FALSE,
  as_is = FALSE,
  mute = NULL,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, the operation uses the `"UID"` (unique identifier), stable during the life cycle of a message, instead of message sequence numbers.

`parts` `NULL` (default) to fetch every attachment part, or a `character/numeric vector` of MIME part numbers (e.g. `c("2", "3.1")`) to fetch specific ones.

`content_disposition` One of `"both"` (default), `"attachment"`, or `"inline"`.

`dest` `NULL` to keep the decoded payloads in memory (in the returned data.frame), or a directory path to write one folder per message with its attachment files.

`override` A `logical`. If `TRUE`, overrides existing files with the same name. Default is `FALSE`.

`as_is` If `TRUE`, writes the payloads without decoding the transfer encoding. Default is `FALSE`.

`mute` A `logical`. If `TRUE`, mutes the confirmation message. Default is `FALSE`.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `data.frame` with one row per attachment: `id` (or `uid`), `part`, `filename`, `type`, `size` (bytes), and `path` (or `content`).

Examples:

```
con$select_folder(folder = "INBOX")
res <- con$query(subject == "report" & flag != "SEEN", use_uid = TRUE)
manifest <- con$attachments(res, use_uid = TRUE, dest = "~/attachments")
```

`ImapCon$attachments_manifest()`**:** List the attachments of messages without downloading them: one server round trip over the `BODYSTRUCTURE` metadata. Replaces `fetch_attachments_list()`.

Usage:

```
ImapCon$attachments_manifest(msg_id, use_uid = NULL, retries = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). If `TRUE`, the operation uses the `"UID"` (unique identifier) instead of message sequence numbers.

`retries` Number of attempts to connect and execute the command. Default is 1.

Returns: A `list` with one `data.frame` per message (filename, type, encoding, size).

Examples:

```
con$select_folder(folder = "INBOX")
con$attachments_manifest(con$query(size > 1e6, use_uid = TRUE), use_uid = TRUE)
```

`ImapCon$get_attachments()`: Extract attached file(s) from fetched message(s)

Usage:

```
ImapCon$get_attachments(
  msg_list,
  content_disposition = "both",
  override = FALSE,
  mute = NULL,
  as_is = FALSE,
  local_dir = "."
)
```

Arguments:

`msg_list` A `list` with the body or text content of the messages fetched with `ImapCon$fetch_body()` or `ImapCon$fetch_text()`.

`content_disposition` A `string` indicating which type of "Content-Disposition" attachments should be retrieved. Default is `"both"`, which retrieves regular attachments ("Content-Disposition: attachment") and inline attachments ("Content-Disposition: inline").

`override` A `logical`. Provides a confirmation message if the command is successfully executed. Default is `FALSE`.

`mute` A `logical`. If `TRUE`, mutes the confirmation message when the command is successfully executed. Default is `FALSE`.

`as_is` If `TRUE`, writes the attachments out without reversing the transfer encoding. Default is `FALSE`.

`local_dir` A `character` string with the base directory where the attachments will be saved. A subfolder tree `<local_dir>/<username>/<mail folder>/<msg id>` is created inside it. Default is `"."` (the current working directory).

Returns: `TRUE` if the operation is successful. The files are saved locally.

Examples:

```
# example 1
con$select_folder(folder = "INBOX")
con$search_string(expr = "@gmail", where = "CC") %>%
 con$fetch_text(write_to_disk = TRUE) %>% # saving the message's content as txt files
  con$get_attachments()

# example 2
res <- con$search_string(expr = "@gmail", where = "CC")
out <- con$fetch_body(msg = res)
con$get_attachments(msg_list = out)
```

`ImapCon$fetch_attachments_list()`: Fetch attachments' list

Usage:

```
ImapCon$fetch_attachments_list(msg_id, use_uid = NULL, retries = NULL)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids.

`use_uid` Default is the connection-level setting of `configure_imap()` (`TRUE` since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If `TRUE`, the command will be performed using the `"UID"` or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

`retries` Number of attempts to connect and execute the command. Default is `1`.

Returns: A `list` with the fetch contents.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and fetch the attachments' list of the messages
out <- con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_attachments_list()
out

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
out <- con$fetch_attachments_list(msg = res)
out
```

`ImapCon$fetch_attachment_parts()`**:** Fetch attachments by MIME part, guided by the message's `BODYSTRUCTURE`. The structure of each message is retrieved first (see `fetch_bodystructure()`), the attachment parts are selected, each is fetched with `BODY.PEEK[<part>]`, decoded from its transfer encoding, and written to disk, or returned as raw vectors. Unlike `fetch_attachments()`, which parses MIME boundaries from the fetched body, this method relies on the parts as declared by the server and transfers nothing but the attachments.

Usage:

```
ImapCon$fetch_attachment_parts(
  msg_id,
  use_uid = NULL,
  parts = NULL,
  content_disposition = "both",
  local_dir = ".",
  override = FALSE,
  mute = NULL,
  retries = NULL
)
```

Arguments:

`msg_id` A `numeric vector` containing one or more message ids, or the `"$"` reference of a saved search.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). If TRUE, the command is performed with UIDs.

parts NULL (default: the parts selected by content_disposition) or a character vector of section numbers to fetch, e.g. c("2", "3.1").

content_disposition As in fetch_attachments(): "both" (default), "attachment", or "inline", selecting the parts by the Content-Disposition the server declares in the BODYSTRUCTURE. With "both", non-text parts that carry a filename but no disposition are included as well.

local_dir The base directory where the files are written, in a <username>/<folder>/<msg id> tree, as in fetch_attachments(). Default is ".". If NULL, nothing is written and the payloads are returned in a content list column.

override A logical. If TRUE, overwrites existing files; otherwise repeated filenames are numbered. Default is FALSE.

mute A logical. If TRUE, mutes the confirmation message. Default is FALSE.

retries Number of attempts to connect and execute the command. Default is 1.

Returns: A data.frame with one row per fetched part: id (or uid), part, filename, type, size (bytes), and path (or content).

Examples:

```
con$select_folder(folder = "INBOX")
con$search_string(expr = "report", where = "SUBJECT") %>%
  con$fetch_attachment_parts(local_dir = "~/attachments")
```

ImapCon$fetch_attachments(): Fetch message attachments

Usage:

```
ImapCon$fetch_attachments(
  msg_id,
  use_uid = NULL,
  content_disposition = "both",
  override = FALSE,
  mute = NULL,
  retries = NULL,
  as_is = FALSE,
  local_dir = "."
)
```

Arguments:

msg_id A numeric vector containing one or more message ids.

use_uid Default is the connection-level setting of configure_imap() (TRUE since 3.0.0). In this case, the operation will be performed using message sequence numbers. A message sequence number is a message's relative position to the oldest message in a mail folder. It may change after deleting or moving messages. If a message is deleted, sequence numbers are reordered to fill the gap. If TRUE, the command will be performed using the "UID" or unique identifier. UIDs are always the same during the life cycle of a message in a mail folder.

content_disposition A string indicating which type of "Content-Disposition" attachments should be retrieved. The options are both, attachment, and inline. Default is "both", which retrieves regular attachments ("Content-Disposition: attachment") and inline attachments ("Content-Disposition: inline").

override A logical. Provides a confirmation message if the command is successfully executed. Default is FALSE.

mute A logical. If TRUE, mutes the confirmation message when the command is successfully executed. Default is FALSE.

retries Number of attempts to connect and execute the command. Default is 1.

as_is If TRUE, writes the attachments out without reversing the transfer encoding. Default is FALSE.

local_dir A character string with the base directory where the attachments will be saved. A subfolder tree <local_dir>/<username>/<mail folder>/<msg id> is created inside it. Default is "." (the current working directory).

Returns: A list with the fetch contents.

Examples:

```
con$select_folder(folder = "INBOX")
# do a search and fetch the attachments' list of the messages
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_attachments() # the attachments will be downloaded to disk


# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$fetch_attachments(msg = res)
```

ImapCon$clone(): The objects of this class are cloneable with this method.

Usage:

```
ImapCon$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Note

ImapCon$new(): The configure_imap should be preferred instead of ImapCon$new().

ImapCon$search(): IMAP queries follow Polish notation, i.e. operators such as OR come before arguments, e.g. "OR argument1 argument2". Therefore, the relational-operator-helper-functions in this package should be used like the following examples: OR(before("17-Apr-2015"), string("FROM", "John")). Even though there is no "AND" operator in IMAP, this package adds a helper function AND to indicate multiple arguments that must be searched together, e.g. AND(since("01-Jul-2018"), smaller_than(16000)).

ImapCon$sent_before(): Search operations that use the origination/RFC-2822 Header date tend to be "slower" than those that use the internal date. Although the overhead is minimum, the difference is due to the fact that the internal date is kept on a database, while the origination date has to be retrieved from inside the message. Therefore, the server needs to access each message when executing this type of search. Despite this fact, both dates tend to be the same.

ImapCon$search_sent_since(): Search operations that use the origination/RFC-2822 Header date tend to be "slower" than those that use the internal date. Although the overhead is minimum, the difference is due to the fact that the internal date is kept on a database, while the origination date

has to be retrieved from inside the message. Therefore, the server needs to access each message when executing this type of search. Despite this fact, both dates tend to be the same.

`ImapCon$search_sent_on()`: Search operations that use the origination/RFC-2822 Header date tend to be "slower" than those that use the internal date. Although the overhead is minimum, the difference is due to the fact that the internal date is kept on a database, while the origination date has to be retrieved from inside the message. Therefore, the server needs to access each message when executing this type of search. Despite this fact, both dates tend to be the same.

`ImapCon$search_sent_period()`: Search operations that use the origination/RFC-2822 Header date tend to be "slower" than those that use the internal date. Although the overhead is minimum, the difference is due to the fact that the internal date is kept on a database, while the origination date has to be retrieved from inside the message. Therefore, the server needs to access each message when executing this type of search. Despite this fact, both dates tend to be the same.

`ImapCon$search_older_than()`: To be able to use this functionality, the server must support the `WITHIN` capability. You can check it by running `ImapCon$list_server_capabilities()`.

`ImapCon$search_older_than()`: To be able to use this functionality, the server must support the `WITHIN` capability. You can check it by running `ImapCon$list_server_capabilities()`.

`ImapCon$search_string()`: Using `where = "TEXT"`, may produce unexpected results since it will perform the search on raw data, i.e. the searched expression may be truncated by special formatting characters such as `\r\n` for example. It is recommended to perform this type of search using `where = "BODY"`, instead of `"TEXT"` (*Heinlein, P. and Hartleben, P. (2008)*).

`ImapCon$esearch_count()`: This operation depends on the `ESEARCH` extension.

`ImapCon$esearch_min_id()`: This operation depends on the `ESEARCH` extension.

`ImapCon$esearch_max_id()`: This operation depends on the `ESEARCH` extension.

`ImapCon$add_flags()`: Unlike the search operations, the add/replace/delete flags operations demand system flag names to be preceded by two backslashes `"\\"`.

`ImapCon$add_flags()`: `add_flags`, `remove_flags`, and `replace_flags` accept not only flags but also keywords (any word not beginning with two backslashes) which are custom flags defined by the user.

`ImapCon$replace_flags()`: Unlike the search operations, the add/replace/delete flags operations demand system flag names to be preceded by two backslashes `"\\"`.

`ImapCon$replace_flags()`: `add_flags`, `remove_flags`, and `replace_flags` accept not only flags but also keywords (any word not beginning with two backslashes) which are custom flags defined by the user.

`ImapCon$remove_flags()`: Unlike the search operations, the add/replace/delete flags operations demand system flag names to be preceded by two backslashes `"\\"`.

`ImapCon$remove_flags()`: `add_flags`, `remove_flags`, and `replace_flags` accept not only flags but also keywords (any word not beginning with two backslashes) which are custom flags defined by the user.

`ImapCon$get_attachments()`: This method is to be used after the body or the text part of one or more messages were fetched. This makes sense if the user is interested in keeping the message content (body or text) besides downloading the message attachments. Nonetheless, this is not the recommended approach if the user is only interested in downloading the files as the previous fetching operation will probably be costly. In this last case, the recommendation is to use `ImapCon$fetch_attachments()` as it will only fetch the attachment part.

`ImapCon$get_attachments()`: All attachments will be stored in a folder labeled with the message id inside the `working directory > servername > foldername`. This function currently handles only attachments encoded as `base64` text. It tries to guess all file extensions while decoding the text, but it may not be possible to do so in some circumstances. If it happens, you can try to change the file extension directly by renaming the file.

`ImapCon$get_attachments()`: The "Content-Disposition" header specifies if the multipart electronic messages will be presented as a main document with a list of separate attachments ("Content-Disposition: attachment") or as a single document with the various parts displayed inline. The first requires positive action on the part of the recipient (downloading the file, for example) whereas inline components are displayed automatically when the message is viewed (*Troost, R., Dorner, S., and K. Moore, Ed. (1997)*). You can choose to download `both`, or only one type of attachment, using the argument `content_disposition`.

`ImapCon$fetch_attachments()`: All attachments will be stored in a folder labeled with the message id inside the `working directory > servername > foldername`. This function currently handles only attachments encoded as `base64` text. It tries to guess all file extensions while decoding the text, but it may not be possible to do so in some circumstances. If it happens, you can try to change the file extension directly by renaming the file.

`ImapCon$fetch_attachments()`: The "Content-Disposition" header specifies if the multipart electronic messages will be presented as a main document with a list of separate attachments ("Content-Disposition: attachment") or as a single document with the various parts displayed inline. The first requires positive action on the part of the recipient (downloading the file, for example) whereas inline components are displayed automatically when the message is viewed (*Troost, R., Dorner, S., and K. Moore, Ed. (1997)*). You can choose to download `both`, or only one type of attachment, using the argument `content_disposition`.

References

`ImapCon$search_string()`: Heinlein, P. and Hartleben, P. (2008). The Book of IMAP: Building a Mail Server with Courier and Cyrus. No Starch Press. ISBN 978-1-59327-177-0.

`ImapCon$get_attachments()`: Troost, R., Dorner, S., and K. Moore (1997), Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field, RFC 2183, August 1997, https://www.rfc-editor.org/rfc/rfc2183.

`ImapCon$fetch_attachments()`: Troost, R., Dorner, S., and K. Moore (1997), Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field, RFC 2183, DOI 10.17487/RFC2183, August 1997, https://www.rfc-editor.org/rfc/rfc2183.

See Also

Other custom search: `AND()`, `OR()`, `before()`, `filter_stored()`, `flag()`, `fuzzy()`, `larger_than()`, `modseq()`, `older_than()`, `on()`, `saved_before()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `verbatim()`, `younger_than()`

Other attachments: `extract_attachments()`, `list_attachments()`

Examples

```
## Not run:
# w/ Plain authentication
```

```
con <- configure_imap(
  url="imaps://outlook.office365.com",
  username="user@agency.gov.br",
  password=rstudioapi::askForPassword(),
  verbose = TRUE)

# OR
con <- ImapCon$new(
  url="imaps://outlook.office365.com",
  username="user@agency.gov.br",
  password=rstudioapi::askForPassword(),
  verbose = TRUE)

# w/ OAuth2.0 authentication
con <- configure_imap(
  url="imaps://outlook.office365.com",
  username="user@agency.gov.br",
  verbose = TRUE,
  xoauth2_bearer = "XX.Ya9...")

# OR
con <- ImapCon$new(
  url="imaps://outlook.office365.com",
  username="user@agency.gov.br",
  verbose = TRUE,
  xoauth2_bearer = "XX.Ya9...")


## End(Not run)



## ------------------------------------------------
## Method `ImapCon$print()`
## ------------------------------------------------

## Not run:
con$disconnect()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$has_capability()`
## ------------------------------------------------

## Not run:
if (con$has_capability("ESEARCH")) ids <- con$search_since("01-Jan-2020", esearch = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$idle()`
## ------------------------------------------------
```

```
## Not run:
con$select_folder("INBOX")
# block until something arrives (or 10 minutes pass), then fetch it
ev <- con$idle(timeout = 600, callback = function(ev) !any(ev$type == "EXISTS"))
if (any(ev$type == "EXISTS")) con$fetch_envelope(max(ev$id[ev$type == "EXISTS"]))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$notify()`
## ------------------------------------------------

## Not run:
# wait up to 10 minutes for mail in any folder of the account
ev <- con$notify(mailboxes = "personal", timeout = 600,
                 callback = function(ev) !any(ev$type == "STATUS"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_binary()`
## ------------------------------------------------

## Not run:
parts <- con$fetch_bodystructure(msg_id = 3)
pdf <- con$fetch_binary(msg_id = 3, part = parts$part[parts$is_attachment][1])
writeBin(pdf[[1]], "attachment.pdf")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$append_catenate()`
## ------------------------------------------------

## Not run:
# a new message whose body is the text of message UID 12, with a new header
con$append_catenate(parts = list(
  "From: me@example.com\r\nSubject: Fwd: report\r\n\r\n",
  imap_url("INBOX", uid = 12, section = "TEXT")), folder = "Archive")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$append_msgs()`
## ------------------------------------------------

## Not run:
msgs <- vapply(1:3, function(i) paste0("Subject: m", i, "\r\n\r\nbody\r\n"), "")
con$append_msgs(msgs, folder = "Archive", flags = "Seen")

## End(Not run)
```

```
## ------------------------------------------------
## Method `ImapCon$esearch_partial()`
## ------------------------------------------------

## Not run:
# the fifty most recent matches only
con$esearch_partial(range = "-1:-50", criteria = "UNSEEN", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esort_partial()`
## ------------------------------------------------

## Not run:
con$esort_partial(range = "1:20", by = "SIZE", reverse = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$replace_msg()`
## ------------------------------------------------

## Not run:
con$replace_msg(msg_id = 4, message = new_draft, folder = "Drafts",
                flags = c("Seen", "Draft"), use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_objectid()`
## ------------------------------------------------

## Not run:
con$fetch_objectid(msg_id = 1:5)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$uid_batches()`
## ------------------------------------------------

## Not run:
batches <- con$uid_batches(batch_size = 500)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esearch_multi()`
## ------------------------------------------------

## Not run:
con$esearch_multi(mailboxes = "personal", criteria = "UNSEEN")
```

```
## End(Not run)

## ------------------------------------------------
## Method `ImapCon$unauthenticate()`
## ------------------------------------------------

## Not run:
con$unauthenticate()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$language()`
## ------------------------------------------------

## Not run:
con$language()
con$language("pt-BR")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$comparator()`
## ------------------------------------------------

## Not run:
con$comparator()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$genurlauth()`
## ------------------------------------------------

## Not run:
con$genurlauth(imap_url("INBOX", uid = 20, section = "1.2"),
               access = "submit+fred")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$urlfetch()`
## ------------------------------------------------

## Not run:
u <- con$genurlauth(imap_url("INBOX", uid = 20), access = "anonymous")
con$urlfetch(u)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_convert()`
```

```
## ------------------------------------------------

## Not run:
pdf <- con$fetch_convert(msg_id = 2, mimetype = "application/pdf",
                         part = "3")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_annotation()`
## ------------------------------------------------

## Not run:
con$fetch_annotation(msg_id = 1)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$store_annotation()`
## ------------------------------------------------

## Not run:
con$store_annotation(msg_id = 1, entry = "/comment",
                     values = c("value.priv" = "check this one"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$query()`
## ------------------------------------------------

## Not run:
con$select_folder("INBOX")
con$query((subject == "budget" | "budget 3") & flag != "SEEN")
con$query(sent >= "2001-10-01" & size > 5e6, use_uid = TRUE)
con$query(verbatim('X-GM-RAW "has:attachment"') & flag != "SEEN")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_server_capabilities()`
## ------------------------------------------------

## Not run:
cap <- con$list_server_capabilities()
cap

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$enable()`
## ------------------------------------------------
```

```
## Not run:
con$enable("CONDSTORE")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$namespace()`
## ------------------------------------------------

## Not run:
con$namespace()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$id()`
## ------------------------------------------------

## Not run:
con$id()
con$id(fields = c(name = "mRpostman", version = "1.2.1"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$get_quota_root()`
## ------------------------------------------------

## Not run:
con$get_quota_root(name = "INBOX")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$get_quota()`
## ------------------------------------------------

## Not run:
con$get_quota(quota_root = "")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$set_quota()`
## ------------------------------------------------

## Not run:
con$set_quota(quota_root = "User quota", storage = 2 * 1024^2)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$get_metadata()`
```

```
## ------------------------------------------------

## Not run:
con$get_metadata(folder = "INBOX", entries = "/private/comment")
con$get_metadata(folder = NULL, entries = "/shared/comment")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$set_metadata()`
## ------------------------------------------------

## Not run:
con$set_metadata(folder = "INBOX", entries = c("/private/comment" = "reviewed"))
con$set_metadata(folder = "INBOX", entries = c("/private/comment" = NA))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$get_acl()`
## ------------------------------------------------

## Not run:
con$get_acl(folder = "INBOX")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$set_acl()`
## ------------------------------------------------

## Not run:
con$set_acl(name = "Shared", identifier = "anyone", rights = "lrs")
con$set_acl(name = "Shared", identifier = "anyone", rights = "+w")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$delete_acl()`
## ------------------------------------------------

## Not run:
con$delete_acl(name = "Shared", identifier = "anyone")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_rights()`
## ------------------------------------------------

## Not run:
con$list_rights(name = "INBOX", identifier = "anyone")
```

```
## End(Not run)

## ------------------------------------------------
## Method `ImapCon$my_rights()`
## ------------------------------------------------

## Not run:
con$my_rights(folder = "INBOX")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$noop()`
## ------------------------------------------------

## Not run:
con$noop()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$check()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$check()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_mail_folders()`
## ------------------------------------------------

## Not run:
folders <- con$list_mail_folders()
folders

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_subscribed_folders()`
## ------------------------------------------------

## Not run:
subscribed <- con$list_subscribed_folders()
subscribed

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_folders_status()`
## ------------------------------------------------
```

```
## Not run:
con$list_folders_status()
con$list_folders_status(items = c("MESSAGES", "UNSEEN", "UIDNEXT"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_special_use_folders()`
## ------------------------------------------------

## Not run:
con$list_special_use_folders()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$select_folder()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$resync_folder()`
## ------------------------------------------------

## Not run:
st <- con$status("INBOX", items = c("UIDVALIDITY", "HIGHESTMODSEQ"))
# ... later:
delta <- con$resync_folder("INBOX", uidvalidity = st[["UIDVALIDITY"]],
                           modseq = st[["HIGHESTMODSEQ"]])
delta$vanished; delta$changed

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_changes()`
## ------------------------------------------------

## Not run:
con$select_folder("INBOX", condstore = TRUE)
last <- con$con_params$highestmodseq
# ... later in the session:
con$fetch_changes(modseq = last)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$close_folder()`
## ------------------------------------------------
```

```
## Not run:
con$select_folder("INBOX")
con$close_folder()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$unselect_folder()`
## ------------------------------------------------

## Not run:
con$select_folder("INBOX")
con$unselect_folder()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$examine_folder()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$examine_folder()

# or directly:
con$examine_folder("Sent")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$status()`
## ------------------------------------------------

## Not run:
con$status(folder = "INBOX")

# or, for the selected folder and specific items only:
con$select_folder("INBOX")
con$status(items = c("MESSAGES", "UNSEEN"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$create_folder()`
## ------------------------------------------------

## Not run:
con$create_folder(folder = "New Folder Name")

## End(Not run)

## ------------------------------------------------
```

```
## Method `ImapCon$rename_folder()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "Folder A")
con$rename_folder(new_name = "Folder B")
# or directly:
con$rename_folder(folder = "Folder A", new_name = "Folder B")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$delete_folder()`
## ------------------------------------------------

## Not run:
con$delete_folder(folder = "Folder to remove")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$subscribe_folder()`
## ------------------------------------------------

## Not run:
con$subscribe_folder(folder = "INBOX")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$unsubscribe_folder()`
## ------------------------------------------------

## Not run:
con$unsubscribe_folder(folder = "INBOX")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_flags()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$list_flags()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$sort()`
## ------------------------------------------------

## Not run:
```

```
con$select_folder("INBOX")
con$sort(by = "DATE", reverse = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$thread()`
## ------------------------------------------------

## Not run:
con$select_folder("INBOX")
con$thread(algorithm = "REFERENCES")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# ex1
con$search(OR(before(date_char = "17-Apr-2015"),
              string(expr = "John", where = "FROM")))

# ex2
con$search(AND(smaller_than(size = "512000"),
               string(expr = "John", where = "FROM"),
               string(expr = "@ksu.edu", where = "CC")))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_larger_than()`
## ------------------------------------------------

## Not run:
# search for messages with size larger than 512Kb
con$search_larger_than(size = 512000)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_smaller_than()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for messages with size smaller than 512Kb
con$search_smaller_than(size = 512000)

## End(Not run)
```

```
## ------------------------------------------------
## Method `ImapCon$search_before()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_before(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_since()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for messages with date since "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_since(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_on()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for messages received on date "02-Jan-2020", presenting the
#... results as unique identifiers (UID)
con$search_on(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_period()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for all messages in the mail folder, EXCEPT (negate = TRUE) by
#... those received between the dates "02-Jan-2020" and "22-Mar-2020"
con$search_period(since_date_char = "02-Jan-2020",
                  before_date_char = "22-Mar-2020",
                  negate = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_sent_before()`
## ------------------------------------------------
```

```
## Not run:
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_sent_before(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_sent_since()`
## ------------------------------------------------

## Not run:
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_sent_since(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_sent_on()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for messages received on date "02-Jan-2020", presenting the
#... results as unique identifiers (UID)
con$search_sent_on(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_sent_period()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for all messages in the mail folder, EXCEPT (negate = TRUE) by
#... those received between the dates "02-Jan-2020" and "22-Mar-2020"
con$search_sent_period(since_date_char = "02-Jan-2020",
                  before_date_char = "22-Mar-2020",
                  negate = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_flag()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for all messages in the mail folder that are marked as "SEEN" AND
#.. "ANSWERED"
```

```
con$search_flag(name = c("SEEN", "ANSWERED"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_older_than()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for all messages received in the last hour (not older than 3600 seconds)
con$search_older_than(seconds = 3600, negate = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_younger_than()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for all messages received in the last hour (younger than 3600 seconds)
con$search_younger_than(seconds = 3600)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_string()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# search for messages with "@k-state.edu" in the FROM field
con$search_string(expr = "@k-state.edu", where = "FROM")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_body()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and fetch the results (saving to disk) using the pipe
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_body(write_to_disk = TRUE, keep_in_mem = FALSE)

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")

con$fetch_body(msg = res, write_to_disk = TRUE, keep_in_mem = FALSE)
```

```
## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_header()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and fetch the results (also saving to disk) using the pipe
out <- con$search_string(expr = "@k-state.edu", where = "CC") %>%
  con$fetch_header()

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "CC")
out <- con$fetch_header()


## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_metadata()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and fetch the results using the pipe
out <- con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_metadata()

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
out <- con$fetch_metadata(msg = res)


## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_preview()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$search_flag("UNSEEN") %>% con$fetch_preview()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_envelope()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$search_since(date_char = "01-Jan-2026") %>% con$fetch_envelope()
```

```
## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_bodystructure()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
parts <- con$fetch_bodystructure(msg_id = 1:10)
parts[parts$is_attachment, ]

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_text()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and partially fetch the results using the pipe
# first 200 characters, writing to disk, silence results in the console
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_text(partial = "0.200",
                 write_to_disk = TRUE,
                 keep_in_mem = FALSE)

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$fetch_text(msg = res,
               partial = "0.200",
               write_to_disk = TRUE,
               keep_in_mem = FALSE)


## End(Not run)

## ------------------------------------------------
## Method `ImapCon$copy_msg()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and copy the results to another folder
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$copy(folder = "Sent")

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$copy(msg = res, folder = "Sent")


## End(Not run)
```

```
## ------------------------------------------------
## Method `ImapCon$move_msg()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and copy the results to another folder
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$move(folder = "Sent")

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$move(msg = res, folder = "Sent")


## End(Not run)

## ------------------------------------------------
## Method `ImapCon$append_msg()`
## ------------------------------------------------

## Not run:
msg <- paste("From: me@example.com", "To: you@example.com",
             "Subject: Hi", "", "Message body.", sep = "\r\n")
con$append_msg(message = msg, folder = "Drafts", flags = "Draft")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esearch_count()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# count the number of messages marked as "Flagged" and "Answered"
con$esearch_count(flag = c("Flagged", "Answered"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$delete_msg()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# delete messages 70 to 73
con$delete_msg(msg_id = 70:73)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$expunge()`
```

```
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# remove every message marked as deleted
con$expunge()
# or only a specific one (UIDPLUS servers)
con$expunge(msg_id = 71)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esearch_min_id()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# Search the minimum id of messages marked as "Answered"
con$esearch_min_id(flag = "Answered")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esearch_max_id()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# Search the minimum id of messages marked as "Seen"
con$esearch_max_id(flag = "Seen")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$add_flags()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# Add the "\Seen" permanent flag to the messages received in the last hour
con$search_younger_than(seconds = 3600) %>% # depends on the WITHIN extension
  con$add_flags(flags_to_set = "\\Seen")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$replace_flags()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# Replace the current flags of the messages in the search results for the
#.. flags "\UNSEEN" and "\Flagged"
```

```
con$search_since(date_char = "20-Aug-2020") %>%
  con$replace_flags(flags_to_set = c("\\UNSEEN", "\\Flagged"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$remove_flags()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# Remove the "\SEEN" flag from the messages in the search result
con$search_since(date_char = "20-Aug-2020") %>%
  con$remove_flags(flags_to_unset = "\\UNSEEN")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$attachments()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
res <- con$query(subject == "report" & flag != "SEEN", use_uid = TRUE)
manifest <- con$attachments(res, use_uid = TRUE, dest = "~/attachments")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$attachments_manifest()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$attachments_manifest(con$query(size > 1e6, use_uid = TRUE), use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$get_attachments()`
## ------------------------------------------------

## Not run:
# example 1
con$select_folder(folder = "INBOX")
con$search_string(expr = "@gmail", where = "CC") %>%
  con$fetch_text(write_to_disk = TRUE) %>% # saving the message's content as txt files
  con$get_attachments()

# example 2
res <- con$search_string(expr = "@gmail", where = "CC")
out <- con$fetch_body(msg = res)
con$get_attachments(msg_list = out)
```

```
## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_attachments_list()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and fetch the attachments' list of the messages
out <- con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_attachments_list()
out

# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
out <- con$fetch_attachments_list(msg = res)
out


## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_attachment_parts()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
con$search_string(expr = "report", where = "SUBJECT") %>%
  con$fetch_attachment_parts(local_dir = "~/attachments")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_attachments()`
## ------------------------------------------------

## Not run:
con$select_folder(folder = "INBOX")
# do a search and fetch the attachments' list of the messages
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_attachments() # the attachments will be downloaded to disk


# or using a traditional approach
res <- con$search_string(expr = "@k-state.edu", where = "FROM")
con$fetch_attachments(msg = res)


## End(Not run)
```

ingest_maildir *Ingest a Local Maildir into an IMAP Folder*

Description

Uploads every message file found under a local directory to a folder on the connected IMAP server, using the package's own APPEND implementation (ImapCon$append_msg()). Message files are read as raw bytes and appended verbatim, so any RFC 5322-conformant maildir-style layout (one message per file) can be ingested – mail server backups, exported archives, or public corpora such as the Enron corpus (see enron_sandbox).

Usage

```
ingest_maildir(
  con,
  path = NULL,
  folder = basename(path),
  create = TRUE,
  recursive = TRUE,
  files = NULL,
  mute = FALSE,
  retries = 1
)
```

Arguments

con	An ImapCon object, as returned by configure_imap.
path	A string with the path to a local directory containing message files.
folder	A string with the name of the destination folder on the server. Default is basename(path).
create	A logical. If TRUE (default), the destination folder is created with CREATE when it does not exist yet.
recursive	A logical. If TRUE (default), message files are searched recursively under path.
files	An optional character vector of message file paths to ingest, overriding the directory listing. When provided, path and recursive are ignored for file discovery (a subset selected by the caller – see enron_sandbox – is uploaded as given). Default is NULL.
mute	A logical. If FALSE, prints progress every 500 messages and a final summary. Default is FALSE.
retries	Number of attempts for each APPEND. Default is 1.

Details

APPEND stores every message with the current time as its internal date (see populate_sandbox for the implications), so header-based SENT* searches are the appropriate way to query ingested corpora by date. Messages are stored without flags on libcurl >= 8.13 (earlier versions hardcode \Seen in

APPEND); use ImapCon$add_flags() to mark them as read if desired. Files that fail to upload (e.g., malformed messages rejected by the server) are skipped with a warning rather than aborting the ingestion.

Value

Invisibly, a data.frame with one row per message file: path, size, appended (a logical indicating whether the upload succeeded), and uid (the UID the server assigned to the message, when it advertises UIDPLUS; NA otherwise).

See Also

Other sandbox: enron_sandbox(), populate_sandbox(), sandbox_corpus()

Examples

```
## Not run:
con <- configure_imap(url = "imap://localhost:1430",
                      username = "testuser", password = "sandbox",
                      use_ssl = FALSE)
# upload a locally stored maildir folder to the server
manifest <- ingest_maildir(con, path = "~/backup/maildir/archive2020",
                           folder = "archive2020")
table(manifest$appended)

## End(Not run)
```

---

larger_than *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
larger_than(size, negate = FALSE)
```

Arguments

size	An integer specifying the number of bytes to be used as search criterion.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" field OR those that are LARGER than 512KB.
res <- con$search(request = OR(string(expr = "XYZ@k-state.edu",
                                      where = "FROM"),
                               larger_than(size = 512000)))

## End(Not run)
```

---

list_attachments *List attachments and content-disposition types*

Description

List attachments and content-disposition types

Usage

```
list_attachments(msg_list)
```

Arguments

msg_list	A list containing the messages (body or text) fetched from the server.

Value

A list of data.frames containing the filenames and their Content-Disposition types for each fetched message.

Note

Please, note that this is an independent function and not an R6 method that depends on the connection object. Therefore, it should be called alone without the ImapCon object.

See Also

Other attachments: ImapCon, extract_attachments()

Examples

```
## Not run:
con$select_folder(folder = "INBOX")
# do a search followed by a fetch operation, then extract the attachments' list
out <- con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_body()
att_list <- list_attachments(msg_list = out)

# or
att_list <- con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$fetch_body() %>%
  list_attachments()

## End(Not run)
```

---

metadata_options *Message Metadata Options*

Description

List Metadata fields used in messages.

Usage

```
metadata_options()
```

Value

A `vector` containing message metadata fields.

Note

This function lists the message attributes that `ImapCon$fetch_metadata()` accepts. The last three (`PREVIEW`, `SAVEDATE`, `MODSEQ`) are capability-gated extensions, only sent when the server advertises them.

References

Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, doi:10.17487/RFC2060, December 1996, https://www.rfc-editor.org/info/rfc2060.

Examples

```
## Not run:

library(mRpostman)
metadata_options()


## End(Not run)
```

modseq *Criterion constructor function for the CONDSTORE extension (RFC 7162)*

Description

Builds the MODSEQ search key, which matches the messages whose modification sequence is equal to or greater than value, i.e. the messages that were added or whose flags changed since the folder had that HIGHESTMODSEQ (obtained with ImapCon$status(items = "HIGHESTMODSEQ")). It is to be combined with AND and OR and passed to ImapCon$search(). The server must advertise the CONDSTORE capability.

Usage

```
modseq(value, negate = FALSE)
```

Arguments

value	A single non-negative number, the modification sequence to compare with.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A character string with the search criterion.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
con$select_folder(folder = "INBOX")
last <- con$status(items = "HIGHESTMODSEQ")[["HIGHESTMODSEQ"]]
# ... later, in another run:
changed <- con$search(request = modseq(last + 1))

## End(Not run)
```

older_than *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
older_than(seconds, negate = FALSE)
```

Arguments

seconds	An integer specifying the number of seconds to be used as the search criterion.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

Note

To be able to use this functionality, the server must support the WITHIN capability.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" field AND those that are OLDER than 3600 seconds (1 hour).
res <- con$search(request = AND(string(expr = "XYZ@k-state.edu",
                                       where = "FROM"),
                                older_than(seconds = 3600)))

## End(Not run)
```

on *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
on(date_char, negate = FALSE)
```

Arguments

date_char	A character string with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use Date or POSIX* like objects, since IMAP servers use this unusual date format.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search string to be used as a request parameter in ImapCon$search() function.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SINCE "17-Apr-2019" AND SMALLER than 512KB.
res <- con$search(request = OR(on(date_char = "30-Jun-2019"),
                               on(date_char = "22-Mar-2018")))
# search for messages received ON "30-Jun-2019" OR ON "22-Mar-2018".


## End(Not run)
```

Ops.imap_search *Combine search criteria with R's own operators*

Description

Criteria built with the constructor functions (`string()`, `flag()`, `before()`, `sent_since()`, `larger_than()`, and their relatives) can be combined with the native operators `&` (AND), `|` (OR), and `!` (NOT), as an alternative to `AND` and `OR`. Precedence and grouping follow R's own rules, so parentheses work as usual. See also `ImapCon$query()` for the expression-based interface that does not require constructor calls at all.

Usage

```
## S3 method for class 'imap_search'
Ops(e1, e2)
```

Arguments

`e1, e2`	Search criteria built by the constructor functions.

Value

A search criterion string of class `"imap_search"`, to be passed to `ImapCon$search()`.

Examples

```
## Not run:
con$search(string("budget", where = "SUBJECT") &
             (sent_since(date_char = "01-Oct-2001") | !flag("SEEN")))

## End(Not run)
```

OR *Relational-operator-function to construct a custom search statement*

Description

Relational-operator-function to construct a custom search statement

Usage

```
OR(..., negate = FALSE)
```

Arguments

...	a combination of criteria constructor functions with its arguments.
negate	If TRUE, negates the search and seeks for "NOT search_criterion". Default is FALSE.

Value

A search string to be used as a request parameter in ImapCon$search() function.

See Also

Other custom search: AND(), ImapCon, before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SINCE "30-Aug-2019" OR SMALLER than 512KB.
res <- con$search(request = OR(sent_since(date_char = "30-Aug-2019"),
                               smaller_than(size = 512000)))

## End(Not run)
```

---

parse_bodystructure *Parse an IMAP BODYSTRUCTURE into a data frame of MIME parts*

Description

Reads the BODYSTRUCTURE (...) structure of a FETCH response (RFC 3501, section 7.4.2), as returned by ImapCon$fetch_metadata(attribute = "BODYSTRUCTURE"), into one row per MIME part, numbered as in FETCH BODY[<part>].

Usage

```
parse_bodystructure(x)
```

Arguments

x	A character string with a FETCH response (or the part of it starting at BODYSTRUCTURE or BODY).

Value

A data.frame with columns part (the section number, e.g. "1", "2.1"; multipart containers are listed with part = NA), type, subtype, charset, filename, encoding, size (bytes, NA for containers), disposition, and is_attachment.

Examples

```
x <- paste0('BODYSTRUCTURE (("text" "plain" ("charset" "utf-8") NIL NIL ',
            '"quoted-printable" 120 3 NIL NIL NIL NIL)',
            '("application" "pdf" ("name" "report.pdf") NIL NIL "base64" ',
            '4096 NIL ("attachment" ("filename" "report.pdf")) NIL NIL) ',
            '"mixed" ("boundary" "xyz") NIL NIL NIL)')
parse_bodystructure(x)
```

---

parse_envelope *Parse an IMAP ENVELOPE into a one-row data frame*

Description

Reads the ENVELOPE (...) structure of a FETCH response (RFC 3501, section 7.4.2), as returned by ImapCon$fetch_metadata(attribute = "ENVELOPE"), into analysis-ready columns. Address lists are formatted as Name <mailbox@host> and joined with commas; RFC 2047 encoded words in names and subjects are decoded.

Usage

```
parse_envelope(x)
```

Arguments

x	A character string with a FETCH response (or the part of it starting at ENVELOPE).

Value

A one-row data.frame with columns date, subject, from, sender, reply_to, to, cc, bcc, in_reply_to, and message_id (NA where the message has no value).

Examples

```
x <- paste0('ENVELOPE ("Mon, 7 May 2001 08:41:00 -0700" "A resume" ',
            '(("Vince Kaminski" NIL "vince.kaminski" "enron.com")) NIL NIL ',
            '((NIL NIL "stephen.stock" "enron.com")) NIL NIL NIL "<id@x>")')
parse_envelope(x)
```

`populate_sandbox` *Populate a Mailbox with the Synthetic Sandbox Corpus*

Description

Stores the deterministic corpus generated by `sandbox_corpus` in a mailbox, using the package's own IMAP operations: each message is uploaded with `APPEND` (`ImapCon$append_msg()`), a secondary folder is created with `CREATE`, and flags are set with `STORE`. It is meant to be run against the disposable local Docker IMAP server shipped in `system.file("docker", package = "mRpostman")` — see the *"A reproducible IMAP sandbox with Docker"* vignette — but works against any IMAP account (mind your provider's rate limits).

Usage

```
populate_sandbox(
  con,
  n = 200,
  seed = 3501,
  extra_folder = "ProjectAtlas",
  mute = FALSE
)
```

Arguments

`con`	An `ImapCon` object, as returned by `configure_imap`.
`n`	Number of messages to append to INBOX. Default is `200`.
`seed`	An integer used as the RNG seed for `sandbox_corpus`. Default is `3501`, after RFC 3501.
`extra_folder`	A string with the name of a secondary folder to be created and populated with every 10th message (for folder, COPY, and MOVE demonstrations), or `NULL` to skip it. Default is `"ProjectAtlas"`.
`mute`	A `logical`. If `FALSE`, prints progress and a final summary. Default is `FALSE`.

Details

`ImapCon$append_msg()` stores the messages without flags on libcurl >= 8.13, but earlier libcurl versions hardcode `\Seen` in `APPEND`, so `populate_sandbox()` sets the planned `\Seen` state explicitly in both directions after the upload. `APPEND` also stores the current time as the internal date of every message, which is why the date searches below use the header-based `SENT*` variants.

Value

Invisibly, the `info` data.frame of the generated corpus (see `sandbox_corpus`).

See Also

Other sandbox: `enron_sandbox()`, `ingest_maildir()`, `sandbox_corpus()`

Examples

```
## Not run:
# against the local Docker sandbox (see the vignette):
con <- configure_imap(url = "imap://localhost:1430",
                      username = "testuser",
                      password = "sandbox",
                      use_ssl = FALSE)
info <- populate_sandbox(con, n = 200)

## End(Not run)
```

---

sandbox_corpus *Generate a Deterministic Synthetic Mail Corpus*

Description

Generates a reproducible corpus of synthetic RFC 822 messages to be stored in a mailbox with populate_sandbox, so that the package's searching, fetching, and decoding features can be demonstrated and tested without a real mail account – typically against the local Docker IMAP sandbox shipped in `system.file("docker", package = "mRpostman")`.

Usage

```
sandbox_corpus(n = 200, seed = 3501)
```

Arguments

n	Number of messages to generate. Default is 200.
seed	An integer used as the RNG seed. The same n and seed always produce byte-identical messages. Default is 3501, after RFC 3501. The caller's RNG state is preserved.

Details

Message features are spread so that each capability of the package has matching messages to act upon: Date: headers spread over 2020 (SENTBEFORE/SENTSINCE/SENTON searches); a subset of large bodies (LARGER/SMALLER searches); accented subjects in MIME encoded-words and quoted-printable bodies (header and body decoding); CSV, PNG, and one-page PDF attachments – all generated deterministically in base R – some with repeated filenames (attachment listing, fetching, binary base64 decoding, and filename deduplication); and reply chains via In-Reply-To/References (SORT and THREAD).

Value

A list with two components: messages, a list of n character vectors (the lines of each RFC 822 message); and info, a data.frame with one row per message describing the features embedded in it (from, subject, date, is_utf8_body, is_large, has_attachment, attachment_type ("csv", "png", "pdf", or NA), is_reply, and the flags – seen, flagged – that populate_sandbox will set after appending).

See Also

Other sandbox: enron_sandbox(), ingest_maildir(), populate_sandbox()

Examples

```
corpus <- sandbox_corpus(n = 5)
# the first synthetic message
cat(corpus$messages[[1]], sep = "\n")
# features embedded in each message
corpus$info
```

---

saved_before *Criterion constructor functions for the SAVEDATE extension (RFC 8514)*

Description

Build the SAVEDBEFORE, SAVEDON, and SAVEDSINCE search keys, which compare the date a message was saved into the mailbox (the SAVEDATE attribute) rather than its internal date or its Date: header. They are to be combined with AND and OR and passed to ImapCon$search(). The server must advertise the SAVEDATE capability.

Usage

```
saved_before(date_char, negate = FALSE)

saved_since(date_char, negate = FALSE)

saved_on(date_char, negate = FALSE)
```

Arguments

date_char	A character string with a date in the IMAP format DD-Mon-YYYY, e.g. "17-Apr-2019".
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A character string with the search criterion.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
con$select_folder(folder = "INBOX")
# messages saved into the folder during the last ingestion day
res <- con$search(request = saved_since(date_char = "27-Aug-2026"))

## End(Not run)
```

---

sent_before *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
sent_before(date_char, negate = FALSE)
```

Arguments

date_char	A `character string` with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use `Date` or `POSIX*` like objects, since IMAP servers use this unusual date format.
negate	If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERIA". Default is `FALSE`.

Value

A search string to be used as a `request` parameter in `ImapCon$search()` function.

See Also

Other custom search: `AND()`, `ImapCon`, `OR()`, `before()`, `filter_stored()`, `flag()`, `fuzzy()`, `larger_than()`, `modseq()`, `older_than()`, `on()`, `saved_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `verbatim()`, `younger_than()`

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SINCE "30-Aug-2019" AND SMALLER than 512KB.
res <- con$search(request = AND(sent_since(date_char = "30-Aug-2019"),
                                smaller_than(size = 512000)))

## End(Not run)
```

sent_on *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
sent_on(date_char, negate = FALSE)
```

Arguments

date_char	A character string with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use Date or POSIX* like objects, since IMAP servers use this unusual date format.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search string to be used as a request parameter in ImapCon$search() function.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_since(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SINCE "30-Aug-2019" OR LARGER than 512KB.
res <- con$search(request = OR(sent_since(date_char = "30-Jun-2020"),
                               larger_than(size = 512000)))

## End(Not run)
```

sent_since *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
sent_since(date_char, negate = FALSE)
```

Arguments

date_char	A character string with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use Date or POSIX* like objects, since IMAP servers use this unusual date format.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search string to be used as a request parameter in ImapCon$search() function.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SENT SINCE "22-Mar-2020" OR containing the STRING
#  "congratulations" in the subject.
res <- con$search(request = AND(sent_since(date_char = "22-Mar-2020"),
                                string(expr = "congratulations",
                                       where = "SUBJECT")))

## End(Not run)
```

since *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
since(date_char, negate = FALSE)
```

Arguments

date_char	A character string with format "DD-Mon-YYYY", e.g. "01-Apr-2019". We opt not to use Date or POSIX* like objects, since IMAP servers use this unusual date format.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search string to be used as a request parameter in ImapCon$search() function.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), smaller_than(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages SINCE "17-Apr-2019" AND SMALLER than 512KB.
res <- con$search(request = AND(since(date_char = "17-Apr-2019"),
                                smaller_than(size = 512000)))

## End(Not run)
```

smaller_than *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
smaller_than(size, negate = FALSE)
```

Arguments

size	An integer specifying the number of bytes to be used as search criterion.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), string(), verbatim(), younger_than()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
# "FROM" field OR those that are SMALLER than 512KB.
res <- con$search(request = OR(string(expr = "XYZ@k-state.edu",
                                      where = "FROM"),
                               smaller_than(size = 512000)))

## End(Not run)
```

`string` *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
string(expr, where, negate = FALSE)
```

Arguments

`expr`	A character string specifying the word or expression to search for in messages.
`where`	A mandatory character string specifying in which message's Section or Header Field to search for the provided string.
`negate`	If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERIA". Default is `FALSE`.

Value

A search criterion of class `imap_search`, to be combined into a search statement (see `Ops.imap_search`).

See Also

Other custom search: `AND()`, `ImapCon`, `OR()`, `before()`, `filter_stored()`, `flag()`, `fuzzy()`, `larger_than()`, `modseq()`, `older_than()`, `on()`, `saved_before()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `verbatim()`, `younger_than()`

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" AND the string "@gmail.com" in the "CC" field.
res <- con$search(request = AND(string(expr = "XYZ@k-state.edu",
                                       where = "FROM"),
                                string(expr = "@gmail.com",
                                       where = "CC")))

## End(Not run)
```

verbatim *Criterion constructor function for a verbatim IMAP search fragment*

Description

Wraps any raw IMAP SEARCH fragment as a search criterion, so that protocol keys without a dedicated field in ImapCon$query() - vendor extensions such as Gmail's X-GM-RAW, sequence sets such as UID 100:200, or FUZZY matches (RFC 6203) - can be combined with the rest of the search language or with the other criterion constructors. The fragment is passed to the server exactly as provided (parenthesized if it is not already), and its keys are not validated locally: an unknown key results in a server-side BAD response.

Usage

```
verbatim(request)
```

Arguments

request	A character string with the raw search fragment.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), younger_than()

Examples

```
## Not run:
# a vendor extension combined with a regular field
res <- con$query(verbatim('X-GM-RAW "has:attachment smaller:25M"') &
                   flag != "SEEN")
# a UID range in a classic custom search
res <- con$search(request = AND(verbatim("UID 100:200"),
                                string(expr = "@gmail.com", where = "FROM")),
                  use_uid = TRUE)

## End(Not run)
```

younger_than *Criterion constructor function to be combined in a custom search statement*

Description

Criterion constructor function to be combined in a custom search statement

Usage

```
younger_than(seconds, negate = FALSE)
```

Arguments

seconds	An integer specifying the number of seconds to be used as the search criterion.
negate	If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE.

Value

A search criterion of class imap_search, to be combined into a search statement (see Ops.imap_search).

Note

To be able to use this functionality, the server must support the WITHIN capability.

See Also

Other custom search: AND(), ImapCon, OR(), before(), filter_stored(), flag(), fuzzy(), larger_than(), modseq(), older_than(), on(), saved_before(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), verbatim()

Examples

```
## Not run:
# select folder & search
con$select_folder(folder = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" field AND those that are YOUNGER than 3600 seconds (1 hour).
res <- con$search(request = AND(string(expr = "XYZ@k-state.edu",
                                       where = "FROM"),
                                younger_than(seconds = 3600)))

## End(Not run)
```

Index

∗ **attachments**
  extract_attachments, 13
  ImapCon, 18
  list_attachments, 105
∗ **complementary operations**
  ImapCon, 18
∗ **custom search**
  AND, 4
  before, 5
  filter_stored, 14
  flag, 15
  fuzzy, 16
  ImapCon, 18
  larger_than, 104
  modseq, 107
  older_than, 108
  on, 109
  OR, 110
  saved_before, 115
  sent_before, 116
  sent_on, 117
  sent_since, 118
  since, 119
  smaller_than, 120
  string, 121
  verbatim, 122
  younger_than, 123
∗ **decoders**
  decode_mime_text, 10
∗ **fetch**
  ImapCon, 18
∗ **options**
  metadata_options, 106
∗ **sandbox**
  enron_sandbox, 11
  ingest_maildir, 103
  populate_sandbox, 113
  sandbox_corpus, 114
∗ **search by date**
  ImapCon, 18
∗ **search by flag**
  ImapCon, 18
∗ **search by size**
  ImapCon, 18
∗ **search by string**
  ImapCon, 18
∗ **search within**
  ImapCon, 18

AND, 4, *49*, *79*, *107*, *110*, *115*
AND(), *5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–123*

before, 5, *49*
before(), *4*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–123*

clean_msg_text, 6
configure_imap, 7, *12*, *23*, *79*, *103*, *113*

decode_mime_header, 9
decode_mime_text, 10

enron_sandbox, 11, *103*
enron_sandbox(), *104*, *113*, *115*
extract_attachments, 13
extract_attachments(), *81*, *105*

filter_stored, 14
filter_stored(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–123*
flag, 15, *49*
flag(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–123*
fuzzy, 16
fuzzy(), *4, 5*, *15*, *81*, *105*, *107–109*, *111*, *115–123*

imap_url, 17, *26*, *32*
imap_utf7_decode *(*imap_utf7_encode*)*, 17

imap_utf7_encode, 17
ImapCon, *4, 5*, *14–16*, 18, *105*, *107–109*, *111*, *115–123*
ingest_maildir, *11, 12*, 103
ingest_maildir(), *12*, *113*, *115*

larger_than, *49*, 104
larger_than(), *4, 5*, *15, 16*, *81*, *107–109*, *111*, *115–123*
list_attachments, 105
list_attachments(), *14*, *81*

metadata_options, *64*, 106
modseq, 107
modseq(), *4, 5*, *15, 16*, *81*, *105*, *108, 109*, *111*, *115–123*
mRpostman *(*mRpostman-package*)*, 3
mRpostman-package, 3

older_than, *49*, 108
older_than(), *4, 5*, *15, 16*, *81*, *105*, *107*, *109*, *111*, *115–123*
on, *49*, 109
on(), *4, 5*, *15, 16*, *81*, *105*, *107, 108*, *111*, *115–123*
Ops.imap_search, 110
OR, *49*, *107*, 110, *110*, *115*
OR(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *115–123*

parse_bodystructure, *66*, 111
parse_envelope, *65*, 112
populate_sandbox, *103*, 113, *114*
populate_sandbox(), *12*, *104*, *115*

sandbox_corpus, *11*, *113*, 114
sandbox_corpus(), *12*, *104*, *113*
saved_before, 115
saved_before(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *116–123*
saved_on *(*saved_before*)*, 115
saved_since *(*saved_before*)*, 115
sent_before, *49*, 116
sent_before(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115*, *117–123*
sent_on, *49*, 117
sent_on(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115, 116*, *118–123*
sent_since, *49*, 118
sent_since(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–117*, *119–123*
since, *49*, 119
since(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–118*, *120–123*
smaller_than, *49*, 120
smaller_than(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–119*, *121–123*
string, *16*, *49*, 121
string(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–120*, *122, 123*

verbatim, 122
verbatim(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–121*, *123*

younger_than, *49*, 123
younger_than(), *4, 5*, *15, 16*, *81*, *105*, *107–109*, *111*, *115–122*