# Package 'mRpostman'

December 17, 2023

**Type** Package

**Title** An IMAP Client for R

**Version** 1.1.2

**Date** 2023-12-17

**Description** An easy-to-use IMAP client that provides tools for message searching, selective fetching of message attributes, mailbox management, attachment extraction, and several other IMAP features, paving the way for e-mail data analysis in R.

**License** GPL-3

**Encoding** UTF-8

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

**Depends** R (>= 3.1.0)

**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

**VignetteBuilder** knitr

**RoxygenNote** 7.2.3

**NeedsCompilation** no

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

**Maintainer** Allan Quadros <allanvcq@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-12-17 07:50:07 UTC

# R topics documented:



---

`mRpostman-package` *An IMAP client for R*

---

## Description

**mRpostman** is an easy-to-use IMAP client that provides tools for message searching, selective fetching of message attributes, mailbox management, attachment extraction, and several other IMAP features, paving the way for e-mail data analysis in R.

## 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(), flag(), larger_than(), older_than(), on(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), younger_than()

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages SINCE "30-Ago-2019" AND SMALLER than 512KB.
res <- con$search(request = AND(sent_since(date_char = "30-Ago-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()`, `flag()`, `larger_than()`, `older_than()`, `on()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `younger_than()`

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages BEFORE "17-Apr-2019" AND NOT SMALLER than 512KB.
res <- con$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,
  use_ssl = TRUE,
  verbose = FALSE,
  buffersize = 16000,
  timeout_ms = 0,
  ...
)
```

## 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. |
| use_ssl | A logical indicating the use or not of Secure Sockets Layer encryption when connecting to the IMAP server. Default is TRUE. |
| 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. |
| ... | 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 Borentein, 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 runs 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"
decoded_string <- 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"
decoded_string <- decode_mime_header(string = qp_encoded)

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

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

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

## 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. |

## See Also

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

## Examples

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

## End(Not run)
```

---

# ImapCon *An IMAP Connection Class*

---

## Description

Configure an IMAP connection using the ImapCon R6 class.

## 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$reset_xoauth2_bearer()
- ImapCon$list_server_capabilities()
- ImapCon$list_mail_folders()
- ImapCon$select_folder()
- ImapCon$examine_folder()

- ImapCon$create_folder()
- ImapCon$rename_folder()
- ImapCon$list_flags()
- ImapCon$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_text()
- ImapCon$copy_msg()
- ImapCon$move_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$get_attachments()
- ImapCon$fetch_attachments_list()
- ImapCon$fetch_attachments()
- ImapCon$clone()

**Method** `new()`**:** Configure and create a new IMAP connection.

*Usage:*

```
ImapCon$new(
  url,
  username,
  password = NULL,
```

```
  xoauth2_bearer = NULL,
  use_ssl = TRUE,
  verbose = FALSE,
  buffersize = 16000,
  timeout_ms = 0,
  ...
)
```

*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.

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

`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.

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

*Returns:* A new 'ImapCon' object.

**Method** `reset_url()`**:** Reset the previously informed url

*Usage:*

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

*Arguments:*

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

**Method** `reset_username()`**:** Reset the previously informed username

*Usage:*

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

*Arguments:*

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

**Method** `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`.

**Method** `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.

**Method** `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.

**Method** `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.

**Method** `reset_password()`**:** Reset the previously informed password

*Usage:*

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

*Arguments:*

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

**Method** `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.

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

*Usage:*

```
ImapCon$list_server_capabilities(retries = 1)
```

*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:*

```
\dontrun{
cap <- con$list_server_capabilities()
cap
}
```

**Method** `list_mail_folders()`**:** List mail folders in a mailbox.

*Usage:*

```
ImapCon$list_mail_folders(retries = 1)
```

*Arguments:*

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

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

*Examples:*

```
\dontrun{
folders <- con$list_mail_folders()
folders
}
```

**Method** `select_folder()`**:** Select a mail folder.

*Usage:*

```
ImapCon$select_folder(name, mute = FALSE, retries = 1)
```

*Arguments:*

`name` 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`.

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

*Examples:*

```
\dontrun{
con$select_mail_folder(name = "INBOX")
}
```

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

*Usage:*

```
ImapCon$examine_folder(name = NULL, retries = 1)
```

*Arguments:*

`name` 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`.

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

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
con$examine_folder()

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

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

*Usage:*

```
ImapCon$create_folder(name, mute = FALSE, retries = 1)
```

*Arguments:*

`name` 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`.

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

*Examples:*

```
\dontrun{
con$create_folder(name = "New Folder Name")
}
```

**Method** `rename_folder()`**:** Rename a mail folder.

*Usage:*

```
ImapCon$rename_folder(
  name = NULL,
  new_name,
  reselect = TRUE,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

`name` A string containing the name of the new 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(name = 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`.

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

*Examples:*

```
\dontrun{
con$select_folder(name = "Folder A")
con$rename_folder(new_name = "Folder B")
# or directly:
con$rename_folder(name = "Folder A", new_name = "Folder B")
}
```

**Method** `list_flags()`**:** List flags in a selected mail folder

*Usage:*

```
ImapCon$list_flags(retries = 1)
```

*Arguments:*

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

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

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
con$list_flags()
}
```

**Method** `search()`**:** Execute a custom search

*Usage:*

```
ImapCon$search(
  request,
  negate = FALSE,
  use_uid = FALSE,
  esearch = FALSE,
  retries = 1
)
```

*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 `younger_than`.

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

`use_uid` Default is `FALSE`. 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 `list` containing the flags (`character vector`), the permanent flags (`character vector`), and an indication if custom flags are allowed by the server (`logical vector`).

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# ex1
con$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")))
}
```

**Method** `search_larger_than()`**:** Search by size (LARGER)

*Usage:*

```
ImapCon$search_larger_than(
  size,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

**Method** `search_smaller_than()`**:** Search by size (SMALLER)

*Usage:*

```
ImapCon$search_smaller_than(
  size,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# search for messages with size smaller than 512Kb
con$search_smaller_than(size = 512000))
}
```

**Method** `search_before()`**:** Search by internal date (BEFORE)

*Usage:*

```
ImapCon$search_before(
  date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

**Method** `search_since()`**:** Search by internal date (SINCE)

*Usage:*

```
ImapCon$search_since(
  date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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. `POSIX*` like objects, since IMAP servers use this uncommon date format. `POSIX*` like, since IMAP servers like this not so common date format.

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

`use_uid` Default is `FALSE`. 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:*

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

**Method** `search_on()`**:** Search by internal date (ON)

*Usage:*

```
ImapCon$search_on(
  date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 FALSE. 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:*

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

**Method** `search_period()`**:** Search by internal date (Period)

*Usage:*

```
ImapCon$search_period(
  since_date_char,
  before_date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 FALSE. 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:*

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

**Method** search_sent_before(): Search by origination date (RFC 2822 Header - SENT BEFORE)

*Usage:*

```
ImapCon$search_sent_before(
  date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 FALSE. 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:*

```
\dontrun{
# 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)
}
```

**Method** `search_sent_since()`**:** Search by origination date (RFC 2822 Header - SENT SINCE)

*Usage:*

```
ImapCon$search_sent_since(
  date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

```
\dontrun{
# 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)
}
```

**Method** `search_sent_on()`**:** Search by origination date (RFC 2822 Header - SENT ON)

*Usage:*

```
ImapCon$search_sent_on(
  date_char,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

**Method** `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 = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

**Method** `search_flag()`**:** Search by flag(s)

*Usage:*

```
ImapCon$search_flag(
  name,
  negate = FALSE,
  use_uid = FALSE,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

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

*Usage:*

```
ImapCon$search_older_than(
  seconds,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

**Method** `search_younger_than()`**:** Search WITHIN a specific time (YOUNGER)

*Usage:*

```
ImapCon$search_younger_than(
  seconds,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

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

**Method** `search_string()`**:** Search by string or expression

*Usage:*

```
ImapCon$search_string(
  expr,
  where,
  negate = FALSE,
  use_uid = FALSE,
  flag = NULL,
  esearch = FALSE,
  retries = 1
)
```

*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 `FALSE`. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# search for all messages received in the last hour (younger than 3600 seconds)
con$search_string(expr = "@k-state.edu", where = "FROM")
}
```

**Method** `fetch_body()`**:** Fetch message body (message's full content)

*Usage:*

```
ImapCon$fetch_body(
  msg_id,
  use_uid = FALSE,
  mime_level = NULL,
  peek = TRUE,
  partial = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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`, keeps a copy of each fetch result while the operation is being performed with `write_to_disk = TRUE`. Default is `FALSE`, and it can only be set `TRUE` when `write_to_disk = TRUE`.

`mute` A `logical`. It provides a confirmation message if 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# do a search and fetch the results (saving to disk) using the pipe
con$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)

}
```

**Method** `fetch_header()`**:** Fetch message header

*Usage:*

```
ImapCon$fetch_header(
  msg_id,
  use_uid = FALSE,
  fields = NULL,
  negate_fields = FALSE,
  peek = TRUE,
  partial = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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`, keeps a copy of each fetch result while the operation is being performed with `write_to_disk = TRUE`. Default is `FALSE`, and it can only be set `TRUE` when `write_to_disk = TRUE`.

`mute` A `logical`. It provides a confirmation message if 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# do a search and fetch the results (also saving to disk) using the pipe
out <- con$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()

}
```

**Method** `fetch_metadata()`**:** Fetch message metadata

*Usage:*

```
ImapCon$fetch_metadata(
  msg_id,
  use_uid = FALSE,
  attribute = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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.

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, keeps a copy of each fetch result while the operation is being performed with write_to_disk = TRUE. Default is FALSE, and it can only be set TRUE when write_to_disk = TRUE.

mute A logical. It provides a confirmation message if 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.

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.

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

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# do a search and fetch the results using the pipe
out <- con$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)

}
```

**Method** fetch_text()**:** Fetch message text

*Usage:*

```
ImapCon$fetch_text(
  msg_id,
  use_uid = FALSE,
  peek = TRUE,
  partial = NULL,
  write_to_disk = FALSE,
  keep_in_mem = TRUE,
  mute = FALSE,
  base64_decode = FALSE,
  retries = 1
)
```

*Arguments:*

msg_id A numeric vector containing one or more message ids.

use_uid Default is FALSE. 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, keeps a copy of each fetch result while the operation is being performed with write_to_disk = TRUE. Default is FALSE, and it can only be set TRUE when write_to_disk = TRUE.

mute A logical. It provides a confirmation message if 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# do a search and partially fetch the results using the pipe
# first 200 characters, writing to disk, silence results in the console
con$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)

}
```

**Method** copy_msg(): Copy message(s) between the selected folder and another one

*Usage:*

```
ImapCon$copy_msg(
  msg_id,
  use_uid = FALSE,
```

```
  to_folder,
  reselect = TRUE,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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.

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

`reselect` A logical. If `TRUE`, calls `ImapCon$select_folder(name = 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`.

*Returns:* An invisible `numeric vector` containing the message ids.

*Examples:*

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

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

}
```

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

*Usage:*

```
ImapCon$move_msg(
  msg_id,
  use_uid = FALSE,
  to_folder,
  reselect = TRUE,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

use_uid Default is FALSE. 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.

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

reselect A logical. If TRUE, calls ImapCon$select_folder(name = 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.

*Returns:* An invisible numeric vector containing the message ids.

*Examples:*

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

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

}
```

**Method** esearch_count()**:** Count the number of messages with a specific flag(s) in a folder (depend on ESEARCH capability)

*Usage:*

```
ImapCon$esearch_count(flag, use_uid = FALSE, retries = 1)
```

*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 FALSE. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
```

```
# count the number of messages marked as "Flagged" and "Answered"
con$esearch_count(flag = c("Flagged", "Answered"))
}
```

**Method** `delete_msg()`**:** Delete message(s) in the selected mail folder

*Usage:*

```
ImapCon$delete_msg(msg_id, use_uid = FALSE, mute = FALSE, retries = 1)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# delete
con$delete_msg(flag = c("Flagged", "Answered"))
}
```

**Method** `expunge()`**:** Permanently removes all or specific messages marked as deleted from the selected folder

*Usage:*

```
ImapCon$expunge(msg_uid = NULL, mute = FALSE, retries = 1)
```

*Arguments:*

`msg_uid` A `numeric vector` containing one or more messages UIDs. Only UIDs are allowed in this operation (note the "u" in msg_*u*id).

`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:* `TRUE` if the operation is successful.

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# count the number of messages marked as "Flagged" and "Answered"
con$esearch_count(flag = c("Flagged", "Answered"))
}
```

**Method** `esearch_min_id()`**:** Search the minimum message id in the selected mail folder (depend on ESEARCH capability)

*Usage:*

```
ImapCon$esearch_min_id(flag, use_uid = FALSE, retries = 1)
```

*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 `FALSE`. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# Search the minimum id of messages marked as "Answered"
con$esearch_min_id(flag = "Answered")
}
```

**Method** `esearch_max_id()`**:** Search the maximum message id in the selected mail folder (depend on ESEARCH capability)

*Usage:*

```
ImapCon$esearch_max_id(flag, use_uid = FALSE, retries = 1)
```

*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 `FALSE`. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# Search the minimum id of messages marked as "Seen"
con$esearch_max_id(flag = "Seen")
}
```

**Method** `add_flags()`**:** Add flags to one or more messages

*Usage:*

```
ImapCon$add_flags(
  msg_id,
  use_uid = FALSE,
  flags_to_set,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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`.

*Returns:* An invisible `numeric vector` containing the message ids.

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# Add the "\Seen" permanent flag to the messages received in the last hour
con$search_younger_than(seconds = 3600) %>% # depends on the WITHIN extension
  con$add_flags(flags_to_set = "\\Seen")
}
```

**Method** `replace_flags()`**:** Replace the current flags of one or more messages

*Usage:*

```
ImapCon$replace_flags(
  msg_id,
  use_uid = FALSE,
  flags_to_set,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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.

*Returns:* An invisible numeric vector containing the message ids.

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# Replace the current flags of the messages in the search results for the
#.. flags "\UNSEEN" and "\Flagged"
con$search_since(date_char = "20-Aug-2020") %>%
  con$replace_flags(flags_to_set = c("\\UNSEEN", "\\Flagged")
}
```

**Method** `remove_flags()`**:** Remove flag(s) of one or more messages

*Usage:*

```
ImapCon$remove_flags(
  msg_id,
  use_uid = FALSE,
  flags_to_unset,
  mute = FALSE,
  retries = 1
)
```

*Arguments:*

msg_id A numeric vector containing one or more message ids.

use_uid Default is FALSE. 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.

*Returns:* An invisible numeric vector containing the message ids.

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# Remove the the "\SEEN" flag from the messages in the search result
con$search_since(date_char = "20-Aug-2020") %>%
  con$remove_flags(flags_to_unset = "\\UNSEEN")
}
```

**Method** `get_attachments()`**:** Extract attached file(s) from fetched message(s)

*Usage:*

```
ImapCon$get_attachments(
  msg_list,
  content_disposition = "both",
  override = FALSE,
  mute = FALSE,
  as_is = FALSE
)
```

*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` then write out attachments without base64 decoding. Default is `FALSE`.

*Returns:* `TRUE` if the operation is successful. The files are saved locally.

*Examples:*

```
\dontrun{
# example 1
con$select_folder(name = "INBOX")
con$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)
}
```

**Method** `fetch_attachments_list()`**:** Fetch attachments' list

*Usage:*

```
ImapCon$fetch_attachments_list(msg_id, use_uid = FALSE, retries = 1)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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:*

```
\dontrun{
con$select_folder(name = "INBOX")
# do a search and fetch the attachments' list of the messages
out < con$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

}
```

**Method** `fetch_attachments()`**:** Fetch message attachments

*Usage:*

```
ImapCon$fetch_attachments(
  msg_id,
  use_uid = FALSE,
  content_disposition = "both",
  override = FALSE,
  mute = FALSE,
  retries = 1,
  as_is = FALSE
)
```

*Arguments:*

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

`use_uid` Default is `FALSE`. 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 then write out attachments without base64 decoding. Default is FALSE.

*Returns:* A list with the fetch contents.

*Examples:*

```
\dontrun{
con$select_folder(name = "INBOX")
# do a search and fetch the attachments' list of the messages
con$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)

}
```

**Method** `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 follows 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 multiples 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 formating 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(), flag(), larger_than(), older_than(), on(), sent_before(), sent_on(), sent_since(), since(), smaller_than(), string(), younger_than()

Other 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$list_server_capabilities`
## ------------------------------------------------

## Not run:
cap <- con$list_server_capabilities()
cap

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_mail_folders`
## ------------------------------------------------

## Not run:
folders <- con$list_mail_folders()
folders

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$select_folder`
## ------------------------------------------------

## Not run:
```

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

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$examine_folder`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
con$examine_folder()

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

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$create_folder`
## ------------------------------------------------

## Not run:
con$create_folder(name = "New Folder Name")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$rename_folder`
## ------------------------------------------------

## Not run:
con$select_folder(name = "Folder A")
con$rename_folder(new_name = "Folder B")
# or directly:
con$rename_folder(name = "Folder A", new_name = "Folder B")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$list_flags`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
con$list_flags()

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search`
## ------------------------------------------------

## Not run:
```

```
con$select_folder(name = "INBOX")
# ex1
con$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$select_folder(name = "INBOX")
# search for messages with size smaller than 512Kb
con$search_smaller_than(size = 512000))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_before`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# search for messages with date before "02-Jan-2020", presenting the
# .. results as unique identifiers (UID)
con$search_before(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_since`
## ------------------------------------------------

## Not run:
con$select_folder(name = "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$select_folder(name = "INBOX")
# search for messages received on date "02-Jan-2020", presenting the
#... results as unique identifiers (UID)
con$search_on(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_period`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# search for all messages in the mail folder, EXCEPT (negate = TRUE) by
#... those received between the dates "02-Jan-2020" and "22-Mar-2020"
con$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$select_folder(name = "INBOX")
# search for messages received on date "02-Jan-2020", presenting the
#... results as unique identifiers (UID)
con$search_sent_on(date = "02-Jan-2020", use_uid = TRUE)

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_sent_period`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# search for all messages in the mail folder, EXCEPT (negate = TRUE) by
#... those received between the dates "02-Jan-2020" and "22-Mar-2020"
con$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$select_folder(name = "INBOX")
# search for all messages in the mail folder that are marked as "SEEN" AND
#.. "ANSWERED"
con$search_flag(name = c("SEEN", "ANSWERED"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_older_than`
## ------------------------------------------------

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

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_younger_than`
## ------------------------------------------------

## Not run:
```

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

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$search_string`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# search for all messages received in the last hour (younger than 3600 seconds)
con$search_string(expr = "@k-state.edu", where = "FROM")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$fetch_body`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# do a search and fetch the results (saving to disk) using the pipe
con$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$select_folder(name = "INBOX")
# do a search and fetch the results (also saving to disk) using the pipe
out <- con$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$select_folder(name = "INBOX")
# do a search and fetch the results using the pipe
out <- con$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_text`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# do a search and partially fetch the results using the pipe
# first 200 characters, writing to disk, silence results in the console
con$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$select_folder(name = "INBOX")
# do a search and copy the results to another folder
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$copy(to_folder = "Sent")

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

```
## End(Not run)

## ------------------------------------------------
## Method `ImapCon$move_msg`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# do a search and copy the results to another folder
con$search_string(expr = "@k-state.edu", where = "FROM") %>%
  con$move(to_folder = "Sent")

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


## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esearch_count`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# count the number of messages marked as "Flagged" and "Answered"
con$esearch_count(flag = c("Flagged", "Answered"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$delete_msg`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# delete
con$delete_msg(flag = c("Flagged", "Answered"))

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$expunge`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# count the number of messages marked as "Flagged" and "Answered"
con$esearch_count(flag = c("Flagged", "Answered"))

## End(Not run)
```

```
## ------------------------------------------------
## Method `ImapCon$esearch_min_id`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# Search the minimum id of messages marked as "Answered"
con$esearch_min_id(flag = "Answered")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$esearch_max_id`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# Search the minimum id of messages marked as "Seen"
con$esearch_max_id(flag = "Seen")

## End(Not run)

## ------------------------------------------------
## Method `ImapCon$add_flags`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# Add the "\Seen" permanent flag to the messages received in the last hour
con$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$select_folder(name = "INBOX")
# Replace the current flags of the messages in the search results for the
#.. flags "\UNSEEN" and "\Flagged"
con$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(name = "INBOX")
```

```
# Remove the 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$get_attachments`
## ------------------------------------------------

## Not run:
# example 1
con$select_folder(name = "INBOX")
con$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$select_folder(name = "INBOX")
# do a search and fetch the attachments' list of the messages
out < con$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_attachments`
## ------------------------------------------------

## Not run:
con$select_folder(name = "INBOX")
# do a search and fetch the attachments' list of the messages
con$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)
```

---

`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 seconds to be used as search criterion. |
| `negate` | If `TRUE`, negates the search and seeks for "NOT SEARCH CRITERIA". Default is `FALSE`. |

## See Also

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

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" field OR those that are LARGER than 512KB.
res <- con$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 its `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`

## Examples

```
## Not run:
con$select_folder(name = "INBOX")
# do a search followed by a fetch operation, then extract the attachments' list
out <- con$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 message metadata used by IMAP servers, according to the RFC 2060 (Crispin, 1996).

### 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)
```

## 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. |

### Note

To be able to use this functionality, the server must support the WITHIN capability.

### See Also

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

### Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" field AND those that are OLDER than 3600 seconds (1 hour).
res <- con$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()`, `flag()`, `larger_than()`, `older_than()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `younger_than()`

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages SINCE "17-Apr-2019" AND SMALLER than 512KB.
res <- con$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)
```

---

# 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()`, `flag()`, `larger_than()`, `older_than()`, `on()`, `sent_before()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `younger_than()`

## Examples

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

## 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()`, `flag()`, `larger_than()`, `older_than()`, `on()`, `sent_on()`, `sent_since()`, `since()`, `smaller_than()`, `string()`, `younger_than()`

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages SINCE "30-Ago-2019" AND SMALLER than 512KB.
res <- con$search(request = AND(sent_since(date_char = "30-Ago-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(), flag(), larger_than(), older_than(), on(), sent_before(), sent_since(), since(), smaller_than(), string(), younger_than()

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages SINCE "30-Ago-2019" OR LARGER than 512KB.
res <- con$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(), flag(), larger_than(), older_than(), on(), sent_before(), sent_on(), since(), smaller_than(), string(), younger_than()

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages SENT SINCE "22-Mar-2020" OR containing the STRING
#  "congratulations" in the subject.
res <- con$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(), flag(), larger_than(), older_than(), on(), sent_before(), sent_on(), sent_since(), smaller_than(), string(), younger_than()

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages SINCE "17-Apr-2019" AND SMALLER than 512KB.
res <- con$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 seconds to be used as search criterion. |
| negate | If TRUE, negates the search and seeks for "NOT SEARCH CRITERIA". Default is FALSE. |

### See Also

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

### Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
# "FROM" field OR those that are SMALLER than 512KB.
res <- con$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. |

## See Also

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

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" AND the string "@gmail.com" in the "CC" field.
res <- con$search(request = AND(string(expr = "XYZ@k-state.edu",
                                       where = "FROM"),
                                string(expr = "@gmail.com",
                                       where = "CC")))

## End(Not run)
```

---

# 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. |

## Note

To be able to use this functionality, the server must support the WITHIN capability.

## See Also

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

## Examples

```
## Not run:
# select folder & search
con$select_folder(name = "INBOX")
# search for messages containing the string "XYZ@k-state.edu" in the
#   "FROM" field AND those that are YOUNGER than 3600 seconds (1 hour).
res <- con$search(request = AND(string(expr = "XYZ@k-state.edu",
                                       where = "FROM"),
                                younger_than(seconds = 3600)))

## End(Not run)
```

# Index