Package 'whapi'

September 29, 2025

Title R Client for 'whapi.cloud'

Version 0.0.2

Date 2025-09-10

Description Provides an 'R' interface to the 'Whapi' 'API' <https://whapi.cloud>, enabling sending and receiving 'WhatsApp' messages directly from 'R'. Functions include sending text, images, documents, stickers, geographic locations, and interactive messages (buttons and lists). Also includes 'webhook' parsing utilities and channel health checks.

License MIT + file LICENSE

URL https://github.com/StrategicProjects/whapi/

BugReports https://github.com/StrategicProjects/whapi/issues/

Depends R (>= 4.2.0)

Imports httr2 (>= 1.0.0), cli (>= 3.6.0), tibble (>= 3.2.0), purrr (>= 1.0.0), stringr (>= 1.5.0), stringi, dplyr, readr, lubridate (>= 1.9.0), jsonlite (>= 1.8.0), mime, openssl

Suggests testthat (>= 3.0.0), knitr, rmarkdown, pkgdown

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation no

Author Andre Leite [aut, cre],
Hugo Vaconcelos [aut],
Diogo Bezerra [aut]

Maintainer Andre Leite <leite@castlab.org>

Repository CRAN

Date/Publication 2025-09-29 17:00:02 UTC

Contents

`whapi_build_servicos_sections` *Build Whapi service sections from a tibble (with* `section` *column)*

Description

Converts a tibble of services (with columns `id`, `title`, `description`, and `section`) into a nested list of **sections/rows** in the format expected by Whapi interactive messages.

Usage

```
whapi_build_servicos_sections(tbl, section_order = NULL)
```

Arguments

`tbl`	A tibble/data.frame containing at least: • `section` (chr): section name • `id` (chr): unique identifier (e.g., `"ap1"`, `"r2"`, `"os3"`) • `title` (chr): display title (can include emoji) • `descricao` (chr): short description of the service
`section_order`	Character vector defining desired order (and subset) of sections. If `NULL`, all sections are included in alphabetical order.

Details

- If `section_order = NULL`, all sections are included, ordered alphabetically.
- If `section_order` is provided, it acts as both:
  - a **filter**: only sections listed will be included,
  - and an **order**: sections appear in the same order as in `section_order`.

Within each section, rows are ordered by the numeric part of `id` (via `readr::parse_number(id)`).

Value

A list of sections, where each section is a list with:

- `title`: section title
- `rows`: a list of rows, each being a list with `id`, `title`, and `description`

Examples

```
de_para_servicos <- tibble::tibble(
  section   = c("Outros Servicos","Renovacoes","Anuencia Previa"),
  id        = c("os2","r4","ap1"),
  title     = c("Consulta Previa",
                "Renovacao de Consulta Previa",
                "Desmembramento"),
```

```
  descricao = c("Initial analysis...","Renewal...","Technical authorization...")
)

# All sections (alphabetical)
whapi_build_servicos_sections(de_para_servicos)

# Custom order and filter
whapi_build_servicos_sections(
  de_para_servicos,
  section_order = c("Anuencia Previa","Outros Servicos")
)
```

`whapi_check_health` *Check Whapi.Cloud channel health and status*

Description

Calls `GET /health` to retrieve channel status, versions, uptime, device, IP, and user info.

Usage

```
whapi_check_health(
  wakeup = TRUE,
  channel_type = c("web", "mobile"),
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

`wakeup`	Logical(1). If TRUE, adds `wakeup=true` query param. Default: TRUE.
`channel_type`	Character(1). Channel type, either `"web"` or `"mobile"`. Default: `"web"`.
`token`	Character(1). Bearer token. Defaults to env var `WHAPI_TOKEN`.
`timeout`	Numeric(1). Request timeout (s). Default 30.
`verbose`	Logical(1). Print CLI logs? Default TRUE.

Value

A tibble with key health information:

- `channel_id`, `uptime`, `version`, `core_version`, `api_version`,
- `device_id`, `ip`,
- `status_code`, `status_text`,
- `user_id`, `user_name`, `user_pushname`, `is_business`,
- `profile_pic`, `profile_pic_full`, `user_status`, plus the raw response in `resp`.

Examples

```
## Not run:
Sys.setenv(WHAPI_TOKEN = "your_token_here")
# Default check (wakeup=TRUE, channel_type="web")
whapi_check_health()

# Check with channel_type = "mobile"
whapi_check_health(channel_type = "mobile")

## End(Not run)
```

`whapi_clip_text` *Clip long text with ellipsis*

Description

Shortens text to a maximum width (character length). If the text is longer, it is truncated and an ellipsis (default `"..."`) is appended.

Usage

```
whapi_clip_text(x, width = 420, ellipsis = "...")
```

Arguments

`x`	Character string to clip.
`width`	Maximum length of the output including ellipsis.
`ellipsis`	Character to indicate truncation.

Value

A clipped string with ellipsis if needed.

Examples

```
whapi_clip_text("This is a very long sentence that should be clipped.", width = 20)
#> "This is a very long..."
```

whapi_coerce_buttons_base *Coerce and normalize button specs for Whapi interactive messages*

Description

Internal helper that converts a `data.frame`/`tibble` or `list` of button definitions into a normalized **list-of-lists**, applying a few rules:

- Accepts aliases `label` / `name` and maps them to `title`;
- Requires a non-empty `title`;
- Auto-generates `id` when missing using slugification plus uniqueness enforcement (e.g., `"Buy Now"` -> `"buy_now"`, duplicates become `"buy_now_2"`, `"buy_now_3"`, ...).

This is useful before building payloads for Whapi interactive endpoints (e.g., buttons, mixed actions, etc.).

Usage

```
whapi_coerce_buttons_base(buttons, verbose = TRUE)
```

Arguments

buttons	A `data.frame`/`tibble` with columns per button (e.g. `title`, `id`, etc.) **or** a list of named lists. Can be `NULL` (returns empty list).
verbose	Logical (default `TRUE`). If `TRUE`, prints progress messages via **cli** (how many buttons, how many ids auto-generated, etc.).

Value

A **list of named lists** (one per button), each guaranteed to have at least `title` (non-empty). If a button had no `id`, a slugified, unique `id` is created.

See Also

`whapi_slugify()`, `whapi_make_unique()`

Examples

```
# From tibble (title only -> ids auto-generated)
 tibble::tribble(~title, "Buy Now", "Buy Now", "Learn More") |>
   whapi_coerce_buttons_base()

# From list (mix with/without id)
 whapi_coerce_buttons_base(list(
   list(title = "Website", url = "https://example.com"),
   list(title = "Website")  # will get an auto id too
 ))
```

whapi_coerce_buttons_mixed *Coerce and validate MIXED buttons (url / call / copy) for Whapi*

Description

Internal helper that prepares **mixed action** buttons for Whapi interactive messages. It first normalizes input via `whapi_coerce_buttons_base()` (accepts `data.frame`/`tibble` or `list`, maps aliases to `title`, auto-creates `id` with slug + uniqueness) and then validates each button according to its declared `type`:

- `url` -> requires fields: `title`, `id`, `url`
- `call` -> requires fields: `title`, `id`, `phone_number`
- `copy` -> requires fields: `title`, `id`, `copy_code`

Enforces WhatsApp constraints: **1 to 3** buttons per message.

Usage

```
whapi_coerce_buttons_mixed(buttons, verbose = TRUE)
```

Arguments

buttons	A `data.frame`/`tibble` (one row per button) or a list of named lists. `title` is required (or via alias), `id` is auto-generated when missing by `whapi_coerce_buttons_base()`. Each button **must provide** a valid `type` among `{"url","call","copy"}`.
verbose	Logical (default `TRUE`). If `TRUE`, prints progress messages via **cli**.

Value

A list-of-lists of buttons, each validated to contain the fields required for its `type`.

See Also

`whapi_coerce_buttons_base()` for normalization; `whapi_coerce_buttons_quick()` for quick-reply buttons.

Examples

```
# Example with a tibble:
 tibble::tribble(
   ~title,        ~type, ~url, ~phone_number,
   "Website",     "url", "https://example.com", NA,
   "Call Support","call", NA, "5581999999999"
 ) |>
   whapi_coerce_buttons_base() |>
   whapi_coerce_buttons_mixed()
```

```
# Example with a list:
 whapi_coerce_buttons_mixed(list(
   list(type="url",  title="Website", url="https://example.com"),
   list(type="call", title="Call us", phone_number="5581999999999"),
   list(type="copy", title="Copy OTP", copy_code="123456")
 ))
```

`whapi_coerce_buttons_quick` *Coerce and validate QUICK REPLY buttons for Whapi*

Description

Internal helper that prepares **quick reply** buttons for Whapi interactive messages. It relies on `whapi_coerce_buttons_base()` to normalize input (accept `data.frame`/`list`, map aliases `label`/`name` -> `title`, auto-generate `id` via slug + uniqueness) and then:

- Enforces `type = "quick_reply"` for all buttons;
- Requires the fields `title` and `id`;
- Ensures the WhatsApp constraint of **1 to 3** buttons.

Usage

```
whapi_coerce_buttons_quick(buttons, verbose = TRUE)
```

Arguments

`buttons`	A `data.frame`/`tibble` (one row per button) or a list of named lists. `title` is required (or via alias), `id` will be auto-created when missing by `whapi_coerce_buttons_base()`.
`verbose`	Logical (default `TRUE`). If `TRUE`, prints progress messages via **cli**.

Value

A list-of-lists of buttons, each including at least `title`, `id`, and `type = "quick_reply"`.

See Also

`whapi_coerce_buttons_base()` for normalization; other coercers for mixed buttons (url/call/copy).

Examples

```
tibble::tribble(~title, "YES", "NO") |>
  whapi_coerce_buttons_base() |>
  whapi_coerce_buttons_quick()
```

whapi_common_blocks *Build common message blocks for Whapi interactive messages*

Description

Internal helper that constructs the standard structure shared by Whapi interactive messages (`button`, `list`, `mixed actions`, etc.).

It automatically normalizes the recipient (`to`) using `whapi_normalize_to()`, and creates `header`, `body`, and `footer` blocks only if the corresponding text is provided.

Usage

```
whapi_common_blocks(to, body_text, header_text = NULL, footer_text = NULL)
```

Arguments

`to`	Character(1). Recipient phone number in international format (digits only, no +), or a group/channel id.
`body_text`	Character(1). Main text of the interactive message body.
`header_text`	Character(1), optional. Optional header text.
`footer_text`	Character(1), optional. Optional footer text.

Details

Many Whapi interactive endpoints (e.g., `messages/interactive`) require the same basic structure:

- **to**: target number or chat id;
- **header**: optional text shown above the body;
- **body**: main message text (required);
- **footer**: optional small text shown below the body.

This helper ensures consistency and avoids repeating boilerplate code when building different interactive message payloads.

Value

A named list ready to be merged into a Whapi interactive message payload, containing elements: `to`, `header` (if provided), `body`, and `footer` (if provided).

See Also

`whapi_send_quick_reply()`, `whapi_send_list()`, `whapi_send_mixed_actions()`

Examples

```
## Not run:
# Minimal body only

whapi_common_blocks("5581999999999", body_text = "Choose an option below")

# With header and footer
whapi_common_blocks(
  to = "5581999999999",
  body_text   = "Do you confirm?",
  header_text = "Booking Confirmation",
  footer_text = "Reply now"
)

## End(Not run)
```

`whapi_date_diff_days` *Days between two dates*

Description

Calculates the difference in days between two dates (`end - start`). Returns `NA` if the start date is missing.

Usage

```
whapi_date_diff_days(start, end)
```

Arguments

`start`	Start date (`Date` or coercible).
`end`	End date (`Date` or coercible).

Value

Integer number of days, or `NA` if `start` is missing.

Examples

```
whapi_date_diff_days(Sys.Date() - 10, Sys.Date())
#> 10
```

whapi_extract_common_fields *Extract common fields from Whapi API responses*

Description

Helper function to standardize parsing of Whapi message responses. Whapi responses typically return a JSON object with a top-level element `sent` and a nested `message` object containing details such as `id`, `status`, `timestamp`, etc. This function consolidates those fields into a flat tibble, making it easier to work with message metadata in R.

Usage

```
whapi_extract_common_fields(out, fallback_to)
```

Arguments

`out`	A list (parsed JSON) as returned by `httr2::resp_body_json()` from a Whapi request.
`fallback_to`	Character(1). A fallback chat id (usually the `to` argument originally passed to the API) used when the response does not contain an explicit `chat_id` or `to`.

Details

The function safely looks up fields in multiple possible locations, since Whapi responses are not always consistent across endpoints:

- **id**: prefers `out$message$id`, then `out$id`, then `out$message_id`;
- **status**: `out$message$status` or `out$status`;
- **timestamp**: `out$message$timestamp` or `out$timestamp`;
- **chat_id**: from `out$message$chat_id`, `out$message$to`, or `fallback_to`;
- **type**: `out$message$type` or `out$type`;
- **sent**: top-level `out$sent` (boolean, TRUE if successfully sent).

The timestamp is returned both raw (numeric, seconds since epoch) and as a parsed `POSIXct` column (`timestamp_dt`, UTC).

Value

A tibble with one row and the following columns:

- `id`: message id;
- `to`: recipient chat id (phone or group);
- `status`: sending status (e.g., "pending", "sent");
- `timestamp`: numeric epoch timestamp (seconds);

- `timestamp_dt`: `POSIXct` parsed timestamp in UTC;
- `type`: message type (e.g., "text", "image", "location");
- `sent`: logical/boolean (TRUE if sent flag present);
- `resp`: the full raw response list for inspection.

See Also

Used internally in wrappers like `whapi_send_text()`, `whapi_send_image()`, `whapi_send_document()`, `whapi_send_location()`.

Examples

```
## Not run:
# Suppose `resp` is the parsed JSON returned from Whapi:
 out <- list(
   sent = TRUE,
   message = list(
     id = "abc123",
     chat_id = "558199999999@s.whatsapp.net",
     timestamp = 1756426418,
     type = "location",
     status = "pending"
   )
 )

 whapi_extract_common_fields(out, fallback_to = "558199999999")

## End(Not run)
```

`whapi_flatten_webhook` *Flatten Whapi webhook payload into tidy rows for persistence (robust)*

Description

Flatten Whapi webhook payload into tidy rows for persistence (robust)

Usage

```
whapi_flatten_webhook(payload, verbose = TRUE)
```

Arguments

`payload`	list parsed from JSON (e.g. plumber `req$body`)
`verbose`	logical, print logs with cli?

Value

tibble with normalized fields

`whapi_fmt_date` *Safe date formatting*

Description

Formats a date safely, returning a fallback value (`na`) when the input is `NULL` or `NA`.

Usage

```
whapi_fmt_date(x, fmt = "%d/%m/%Y", na = "-")
```

Arguments

`x`	Date or coercible to `Date`.
`fmt`	Date format passed to `base::format()`.
`na`	Fallback string if the input is missing.

Value

A formatted date string, or the `na` placeholder if missing.

Examples

```
whapi_fmt_date(Sys.Date())
#> "31/08/2025"
whapi_fmt_date(NA)
#> "-"
```

`whapi_get_contact_profile`
Get WhatsApp contact profile(s) via Whapi.Cloud

Description

Fetches profile information for one or more contacts using `GET /contacts/{ContactID}/profile` and returns a tidy tibble. This version assumes the response body contains:

- `name` (string, user name),
- `about` (string, user info in About section),
- `icon` (string, profile preview icon URL),
- `icon_full` (string, full avatar URL).

Usage

```
whapi_get_contact_profile(
  contacts,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  full = TRUE,
  timeout = 30,
  verbose = TRUE
)
```

Arguments

contacts	Character vector. Phones in E.164 digits (no "+") or chat IDs.
token	Bearer token. Defaults to env var WHAPI_TOKEN.
full	Logical. If TRUE, add _full=true query param. Default TRUE.
timeout	Numeric. Request timeout (seconds). Default 30.
verbose	Logical. Print progress with **cli**? Default TRUE.

Details

- Each contacts element may be a phone number (free text) or a JID (e.g., "1203...@g.us"). Phone numbers are normalized via whapi_normalize_to() (12 digits total); JIDs are kept as-is.
- When full = TRUE, _full=true is added to the querystring to request higher-resolution avatars (if supported).

Value

A tibble with one row per contact:

- contact_id (input id or JID as queried),
- name, about, icon, icon_full,
- resp with the raw response (list).

Examples

```
## Not run:

 Sys.setenv(WHAPI_TOKEN = "your_token_here")
# Single:
 whapi_get_contact_profile("5581999999999")

# Mixed (number + group JID):
 whapi_get_contact_profile(c("5581999999999", "1203630xxxxxx@g.us"))

## End(Not run)
```

whapi_get_message *Get a WhatsApp message by ID (Whapi.Cloud)*

Description

Get a WhatsApp message by ID (Whapi.Cloud)

Usage

```
whapi_get_message(
  message_id,
  resync = FALSE,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

message_id	Character(1). The message ID to fetch.
resync	Logical(1). Whether to resync from the device. Default FALSE.
token	Bearer token (default from WHAPI_TOKEN env var).
timeout	Timeout in seconds. Default 30.
verbose	Print CLI progress? Default TRUE.

Value

Tibble with fields: id, type, subtype, chat_id, chat_name, from, from_name, from_me, source, timestamp, timestamp_dt, device_id, status, and raw resp.

Examples

```
## Not run:
# Needs API Key
whapi_get_message("PsobWy36679._7w-wKmB9tMeGQ")


## End(Not run)
```

`whapi_log_plumber_req` *Log details of a Plumber request object using the cli package*

Description

Prints a structured summary of a `req` object (Plumber request) to the console, with colored and formatted output using the **cli** package. This helper is useful for debugging APIs, inspecting request metadata, and logging incoming payloads in a readable way.

The function logs:

- HTTP method, path, query string, host, client IP/port, content type, and length.
- Parsed arguments: `argsQuery`, `argsBody`, `args`.
- Headers (with sensitive values redacted).
- Cookies (always redacted).
- Parsed request body (`req$body` or `req$postBody`).

Usage

```
whapi_log_plumber_req(
  req,
  show_headers = TRUE,
  show_cookies = TRUE,
  show_body = TRUE,
  max_chars = 2000L
)
```

Arguments

`req`	A Plumber request object, usually passed automatically inside an endpoint. Must contain fields such as `REQUEST_METHOD`, `PATH_INFO`, `HTTP_HOST`, etc.
`show_headers`	Logical. Whether to print request headers (default: `TRUE`). Sensitive values are redacted.
`show_cookies`	Logical. Whether to print cookies (default: `TRUE`). Values are always redacted.
`show_body`	Logical. Whether to print the parsed body or raw `postBody` (default: `TRUE`). Useful for debugging JSON payloads.
`max_chars`	Integer. Maximum number of characters to print for large JSON or raw bodies. Defaults to `2000`.

Value

Invisibly returns `NULL`. The function is called for its side-effect of printing formatted logs to the console.

See Also

`cli::cli_inform()`, `cli::cli_rule()`, `cli::cli_verbatim()`

Examples

```
## Not run:
# Inside a Plumber endpoint
#* @post /myendpoint
function(req, res) {
  whapi_log_plumber_req(req) # Prints nicely formatted info about the incoming request
  list(success = TRUE)
}

# Print only metadata, no headers/body
whapi_log_plumber_req(req, show_headers = FALSE, show_body = FALSE)

## End(Not run)
```

`whapi_log_pretty_json` *Convert R objects to pretty JSON for logging*

Description

Converts R objects to a JSON string for easier inspection in logs. Falls back to `utils::str()` output if `jsonlite` is not available or if JSON conversion fails. Long outputs are truncated with whapi_trunc.

Usage

```
whapi_log_pretty_json(x, max = 2000L)
```

Arguments

`x`	An R object (list, data frame, etc.).
`max`	Integer. Maximum number of characters to print (default: 2000).

Value

A character string containing JSON (pretty-printed if possible).

Examples

```
whapi_log_pretty_json(list(a = 1, b = "test"))
whapi_log_pretty_json(mtcars[1:2, ], max = 100)
```

whapi_make_unique *Make identifiers unique while preserving order*

Description

Ensures that a character vector of identifiers is unique by appending numeric suffixes (_2, _3, ...) when duplicates are found, while preserving the original order.

Usage

```
whapi_make_unique(x)
```

Arguments

x	A character vector of IDs (possibly with duplicates).

Details

This helper is particularly useful when generating button IDs for WhatsApp interactive messages via Whapi. Even after whapi_slugifying labels, duplicates may remain (e.g., two buttons both titled "Yes"). The function guarantees uniqueness by incrementally appending a suffix.

Algorithm:

- Iterates through x in order;
- Keeps a counter of how many times each ID has appeared;
- First occurrence is left unchanged;
- Subsequent duplicates get suffixed with _<n>.

Value

A character vector of the same length with unique IDs.

See Also

whapi_slugify() for slug-safe ID creation.

Examples

```
whapi_make_unique(c("yes", "no", "yes", "yes", "maybe", "no"))
# -> "yes", "no", "yes_2", "yes_3", "maybe", "no_2"

# Combined with whapi_slugify
titles <- c("Yes!", "Yes!", "No?")
ids <- whapi_make_unique(whapi_slugify(titles))
tibble::tibble(title = titles, id = ids)
```

whapi_mark_message_read *Mark a WhatsApp message as READ (Whapi.Cloud)*

Description

Marks a message as **read** using `PUT /messages/{MessageID}`. This endpoint returns only a minimalistic ACK in the body: `{"success": true | false}`.

Usage

```
whapi_mark_message_read(
  message_id,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

`message_id`	Character(1). Message ID (WAMID) to be marked as read.
`token`	Character(1). Bearer token. Default: `Sys.getenv("WHAPI_TOKEN")`.
`timeout`	Numeric(1). Timeout (s). Default: 30.
`verbose`	Logical(1). Show logs via **cli**? Default: `TRUE`.

Value

A tibble with the following columns:

- `id` - the provided `message_id`;
- `status` - `"read"` when `success=TRUE`, `"error"` otherwise;
- `success` - logical value returned by the endpoint;
- `resp` - raw response (list).

Examples

```
## Not run:
 Sys.setenv(WHAPI_TOKEN = "your_token_here")
 whapi_mark_message_read("PsqXn5SAD5v7HRA-wHqB9tMeGQ")

## End(Not run)
```

whapi_match_digits *Digit matching helper (with partial threshold)*

Description

Matches a digit-only `needle` inside a digit-only `haystack`. If `needle` has at least `min_partial` digits, partial matching is allowed. Otherwise, only exact matches are considered.

Usage

```
whapi_match_digits(haystack, needle, min_partial = 6)
```

Arguments

`haystack`	String with potential digits to search in.
`needle`	String with digits to search for.
`min_partial`	Minimum number of digits required to allow partial match.

Value

Logical (`TRUE`/`FALSE`) indicating whether a match was found.

Examples

```
whapi_match_digits("tel: 81998765432", "987654")
#> TRUE
whapi_match_digits("12345", "123", min_partial = 4)
#> FALSE
```

whapi_normalize_to *Normalize WhatsApp contact IDs (phone numbers only)*

Description

Cleans and normalizes WhatsApp phone numbers by:

- Skipping normalization if the entry already contains a JID suffix (e.g., "@s.whatsapp.net" or "@g.us");
- Otherwise, removing all non-digit characters (+, spaces, parentheses, hyphens, etc.);
- Removing the **ninth digit** (when present) right after the country code (CC) and area code (SS), in order to standardize to 12 or 13 digits.

Usage

```
whapi_normalize_to(to)
```

Arguments

to	Character. A character vector of WhatsApp numbers in free text format or JIDs.

Details

This function is primarily designed for **Brazilian E.164 numbers** (e.g., `+55 (81) 9XXXX-YYYY`).

Value

A character vector of normalized IDs (phones -> digits only, JIDs kept as-is).

---

whapi_only_digits *Keep only digits from a string*

Description

Removes all non-digit characters from the input string. Useful for cleaning phone numbers, CNPJs/CPFs, or process codes.

Usage

```
whapi_only_digits(x)
```

Arguments

x	Character vector or string. If `NULL`, an empty string is used.

Value

A string containing only numeric digits.

Examples

```
whapi_only_digits("(81) 98765-4321")
#> "81987654321"
```

whapi_parse_body *Parse the raw body of a Plumber request*

Description

Attempts to parse the body of a Plumber `req` object. Supports both `req$body` (already parsed) and `req$bodyRaw` (raw binary). If the body looks like JSON (`{...}` or `[...]`), it tries to parse it with `jsonlite::fromJSON`. Otherwise, it returns the raw text.

Usage

```
whapi_parse_body(req)
```

Arguments

`req`	A Plumber request object (list-like).

Value

A list representing the parsed body, or `NULL` if no body is found.

whapi_parse_command *Parse a text message into command and arguments (Whapi helper)*

Description

Splits an incoming text (usually from a WhatsApp message) into a **command** (the first token, starting with `/`) and its associated **arguments**. Supports collapsing arguments into a single string or preserving them as a character vector.

Usage

```
whapi_parse_command(text, collapse_args = TRUE)
```

Arguments

`text`	A character string containing the full message (e.g. `"/groe ajuda"`).
`collapse_args`	Logical, default `TRUE`. If `TRUE`, all arguments are collapsed into a single space-separated string. If `FALSE`, arguments are returned as a character vector of tokens.

Value

A list with two elements:

`command` The command string (first token), trimmed (e.g. `"/groe"`).

`arguments` The arguments, either collapsed as a single string (default) or as a character vector, depending on `collapse_args`.

See Also

stringr::str_extract(), stringr::str_split()

Examples

```
whapi_parse_command("/groe ajuda")
#> $command
#> [1] "/groe"
#>
#> $arguments
#> [1] "ajuda"

whapi_parse_command("/groe nome empresa", collapse_args = FALSE)
#> $command
#> [1] "/groe"
#>
#> $arguments
#> [1] "nome" "empresa"
```

---

whapi_perform_request *Perform an HTTP request to Whapi.Cloud*

Description

Generic helper wrapping httr2 to call Whapi endpoints. Supports methods: "GET", "POST", "PUT". Handles JSON encoding, retries, errors, and CLI logging.

Usage

```
whapi_perform_request(
  endpoint,
  payload = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE,
  method = c("POST", "GET", "PUT")
)
```

Arguments

endpoint	Character(1). Endpoint path (e.g. "messages/text"). Full URL is constructed as "https://gate.whapi.cloud/{endpoint}".
payload	List. Request body (for POST/PUT) or query (for GET). Default: NULL.
token	Character(1). Bearer token. Defaults to env var WHAPI_TOKEN.
timeout	Numeric. Timeout in seconds. Default 30.
verbose	Logical. Print progress via cli? Default TRUE.
method	Character(1). HTTP method ("GET", "POST", "PUT"). Default "POST".

Value

Parsed JSON response as a list.

`whapi_react_to_message`

React to a WhatsApp message (Whapi.Cloud)

Description

Sends (or removes) an **emoji reaction** to a message via `PUT /messages/{MessageID}/reaction`. The endpoint returns only `{ "success": true | false }`.

Usage

```
whapi_react_to_message(
  message_id,
  emoji,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

`message_id`	Character(1). Target message ID (WAMID).
`emoji`	Character(1). Emoji to react with. To remove a reaction, pass an empty string `""`.
`token`	Character(1). Bearer token. Defaults to env var `WHAPI_TOKEN`.
`timeout`	Numeric(1). Request timeout in seconds. Default: 30.
`verbose`	Logical(1). Print CLI logs? Default: TRUE.

Value

A tibble with columns:

- `id` - message_id reacted to
- `emoji` - emoji used (or `""` if removed)
- `status` - `"ok"` if success, `"error"` otherwise
- `success` - logical flag from API
- `resp` - raw response (list)

whapi_redact *Redact sensitive values in headers/cookies*

Description

Replaces sensitive header or cookie values (e.g., `Authorization`, `Cookie`, `X-API-Key`) with the literal string `"<redacted>"`. Useful when logging HTTP requests while avoiding credential leaks.

Usage

```
whapi_redact(name, value)
```

Arguments

`name`	Header or cookie name (character).
`value`	Header or cookie value (character).

Value

The original value, or `"<redacted>"` if the header is considered sensitive.

Examples

```
whapi_redact("Authorization", "Bearer abc123")
whapi_redact("Content-Type", "application/json")
```

whapi_send_document *Send a DOCUMENT via Whapi.Cloud (file, url, or base64)*

Description

Sends a document using Whapi's `POST /messages/document`. Supports three input modes via `type`:

- `"file"` : local path -> reads bytes and builds a data URI (`data:<mime>;name=<file>;base64,<...>`);
- `"url"` : direct `http(s)` URL;
- `"base64"` : pre-built data URI (`data:application/...;name=...;base64,...`).

Usage

```
whapi_send_document(
  to,
  document,
  type = c("file", "url", "base64"),
  caption = NULL,
  filename = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

to	Character(1). WhatsApp target (E.164 digits only, no "+") or chat id.
document	Character(1). File path (when type="file"), URL ("url"), or data URI ("base64").
type	One of c("file","url","base64"). Default: "file".
caption	Optional caption text.
filename	Optional filename shown to the user. If omitted: • type="file" -> uses basename(document); • type="url" -> uses the URL basename (without query/fragment); • type="base64" -> tries the name= part inside the data URI.
token	Bearer token (defaults to env var WHAPI_TOKEN).
timeout	Numeric. Request timeout in seconds. Default 30.
verbose	Logical. Print CLI messages? Default TRUE.

Value

A tibble with id, to, status, timestamp, timestamp_dt, and raw resp.

Examples

```
## Not run:
 Sys.setenv(WHAPI_TOKEN = "your_token_here")
 whapi_send_document("5581999999999", "report.pdf", type="file", caption="Monthly report")
 whapi_send_document("5581999999999", "https://example.com/contract.docx", type="url")
 b <- openssl::base64_encode(readBin("memo.odt","raw",file.info("memo.odt")$size))
 du <- sprintf("data:application/vnd.oasis.opendocument.text;name=%s;base64,%s",
  basename("memo.odt"), b)
 whapi_send_document("5581999999999", du, type="base64")

## End(Not run)
```

whapi_send_image *Send an image via Whapi.Cloud (file, url, or base64)*

Description

Sends an image using Whapi's POST /messages/image. Supports three input modes through type:

- "file": local path -> reads bytes and builds a data:<mime>;name=<file>;base64,<...> URI
- "url": direct http(s) URL
- "base64": pre-built data URI (...)

Usage

```
whapi_send_image(
  to,
  image,
  type = c("file", "url", "base64"),
  caption = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

to	Character(1). WhatsApp target (E.164 digits only, no "+") or chat id.
image	Character(1). File path (for type="file"), URL (type="url"), or data URI (type="base64").
type	One of c("file","url","base64"). Default = "file".
caption	Optional caption text.
token	Bearer token (defaults to env var WHAPI_TOKEN if not given).
timeout	Numeric. Timeout in seconds. Default 30.
verbose	Logical. Show CLI messages? Default TRUE.

Value

A tibble with id, to, status, timestamp, timestamp_dt, and raw resp.

Examples

```
## Not run:
# Sys.setenv(WHAPI_TOKEN = "your_token_here")
 whapi_send_image("5581999999999", image = "card.png", type = "file", caption = "Card")
 whapi_send_image("5581999999999", image = "https://site.com/img.png", type = "url")
 b64 <- openssl::base64_encode(readBin("card.png","raw",file.info("card.png")$size))
```

```
data_uri <- sprintf("%s", basename("card.png"), b64)
whapi_send_image("5581999999999", image = data_uri, type = "base64")

## End(Not run)
```

---

whapi_send_list *Send a WhatsApp interactive LIST message (Whapi.Cloud)*

Description

Sends an interactive **LIST** message via Whapi. Sections/rows are validated by whapi_validate_list_sections(). The payload reuses whapi_common_blocks() to keep structure consistent across interactive message types.

Usage

```
whapi_send_list(
  to,
  body_text,
  list_sections,
  list_label = "Choose...",
  header_text = NULL,
  footer_text = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

to	Character(1). Phone in E.164 digits (without "+") or group id.
body_text	Character(1). Main body text.
list_sections	A list of sections. Each section is a named list: list(title = <chr>, rows = list(list(id=<chr>, ti
list_label	Character(1), optional. Button label that opens the list. Default: "Choose..."
header_text, footer_text	Character(1), optional. Header/footer texts.
token	Bearer token. Defaults to env var WHAPI_TOKEN.
timeout	Numeric. Request timeout in seconds. Default 30.
verbose	Logical. Print CLI messages? Default TRUE.

Value

A tibble with id, to, status, timestamp, and the raw response in resp.

See Also

whapi_validate_list_sections(), whapi_common_blocks(), whapi_perform_request(), whapi_extract_common_fi

Examples

```
## Not run:
 Sys.setenv(WHAPI_TOKEN = "your_token_here")
 sections <- list(
   list(
     title = "Burgers",
     rows = list(
       list(id="b1", title="Plain",  description="No cheese, no sauce"),
       list(id="b2", title="Cheese", description="With melted cheese")
     )
   ),
   list(
     title = "Drinks",
     rows = list(
       list(id="d1", title="Water"),
       list(id="d2", title="Soda", description="Assorted flavors")
     )
   )
 )
 whapi_send_list(
   to = "5581999999999",
   body_text = "Choose your order:",
   list_sections = sections,
   list_label = "Open menu",
   header_text = "Our Menu",
   footer_text = "Thanks!"
 )

## End(Not run)
```

---

whapi_send_location *Send a geographic location via Whapi.Cloud*

Description

Sends a location message using Whapi's `POST /messages/location`. Supports optional `name` (short title) and `address` (human-readable).

Usage

```
whapi_send_location(
  to,
  latitude,
  longitude,
  name = NULL,
  address = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

to	Character(1). WhatsApp target (E.164 digits only, no "+") or chat id.
latitude	Numeric(1). Latitude in decimal degrees (range: -90..90).
longitude	Numeric(1). Longitude in decimal degrees (range: -180..180).
name	Optional character(1). Short title for the location.
address	Optional character(1). Human-readable address/description.
token	Bearer token (defaults to env var WHAPI_TOKEN).
timeout	Numeric. Request timeout (seconds). Default 30.
verbose	Logical. Print CLI messages? Default TRUE.

Value

A tibble with id, to, status, timestamp, timestamp_dt, and raw resp.

Examples

```
## Not run:
 Sys.setenv(WHAPI_TOKEN = "your_token_here")
 whapi_send_location("5581999999999",
   latitude = -8.063169, longitude = -34.871139,
   name = "Marco Zero", address = "Recife, PE")

# Group id example
 whapi_send_location("1203630xxxxxxxx@g.us", -8.045, -34.91)

## End(Not run)
```

---

whapi_send_mixed_actions

Send a WhatsApp interactive message mixing URL/CALL/COPY buttons (Whapi.Cloud)

Description

Sends an interactive **buttons** message that mixes url, call, and/or copy actions. Input buttons are normalized/validated by whapi_coerce_buttons_mixed() (aliases mapped to title, auto id creation, required fields per type).

Usage

```
whapi_send_mixed_actions(
  to,
  body_text,
  buttons,
  header_text = NULL,
```

```
  footer_text = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

to	Character(1). Phone in E.164 digits (without "+") or group id.
body_text	Character(1). Main body text.
buttons	Data frame or list. Up to 3 items; each must define a type in {'url','call','copy'} and include the fields: • url: title, id, url • call: title, id, phone_number • copy: title, id, copy_code (id is auto-generated if missing; title required)
header_text, footer_text	Character (optional). Header/footer texts.
token	Bearer token. Defaults to env var WHAPI_TOKEN.
timeout	Numeric. Request timeout (seconds). Default 30.
verbose	Logical. Print CLI messages? Default TRUE.

Value

A tibble with fields id, to, status, timestamp, and the raw response in resp.

See Also

whapi_coerce_buttons_mixed(), whapi_common_blocks(), whapi_perform_request()

Examples

```
## Not run:
 Sys.setenv(WHAPI_TOKEN = "your_token_here")
 whapi_send_mixed_actions(
   to = "5581999999999",
   body_text = "Pick an option:",
   buttons = list(
     list(type="url",  title="Website",   url="https://example.com"),
     list(type="call", title="Call us",   phone_number="5581999999999"),
     list(type="copy", title="Copy OTP",  copy_code="123456")
   )
 )

## End(Not run)
```

`whapi_send_quick_reply` *Send a WhatsApp interactive message with QUICK REPLY buttons (Whapi.Cloud)*

Description

Sends an interactive message of type **QUICK REPLY** via Whapi. Each button is normalized/validated by `whapi_coerce_buttons_quick()` and automatically gets a unique slugified `id` if missing.

Usage

```
whapi_send_quick_reply(
  to,
  body_text,
  buttons,
  header_text = NULL,
  footer_text = NULL,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

`to`	Character (length 1). Phone in E.164 digits (without "+") or group id.
`body_text`	Character. Main body text.
`buttons`	Data frame or list. Up to 3 items; fields: `title`, `id` (id auto-generated if missing).
`header_text, footer_text`	Character (optional). Header/footer texts.
`token`	Bearer token. Defaults to env var WHAPI_TOKEN.
`timeout`	Numeric, request timeout in seconds. Default 30.
`verbose`	Logical, print CLI messages. Default TRUE.

Value

A tibble with fields `id`, `to`, `status`, `timestamp`, and raw response `resp`.

See Also

`whapi_coerce_buttons_quick()`, `whapi_common_blocks()`, `whapi_perform_request()`

Examples

```
## Not run:
 Sys.setenv(WHAPI_TOKEN = "your_token_here")
 whapi_send_quick_reply(
   to = "5581999999999",
   body_text = "Do you confirm?",
   buttons = tibble::tribble(~title, "YES", "NO")
 )

## End(Not run)
```

`whapi_send_sticker` *Send a STICKER via Whapi.Cloud (file, url, or base64)*

Description

Sends a WhatsApp sticker via `POST /messages/sticker`. The sticker must be in WebP format (`image/webp`).

Usage

```
whapi_send_sticker(
  to,
  sticker,
  type = c("file", "url", "base64"),
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  timeout = 30,
  verbose = TRUE
)
```

Arguments

`to`	Character(1). WhatsApp target (E.164 digits only, no "+") or chat id.
`sticker`	Character(1). • If type = "file": local `.webp` file path (will be read and encoded to Base64 data-URI). • If type = "url": direct http(s) URL to the `.webp` file. • If type = "base64": full data-URI string (e.g. "<...>").
`type`	One of c("file","url","base64"). Default = "file".
`token`	Bearer token (env var WHAPI_TOKEN if not provided).
`timeout`	Numeric. Request timeout (seconds). Default 30.
`verbose`	Logical. Print CLI messages? Default TRUE.

Value

tibble with id, to, status, timestamp, and raw response in `resp`.

Examples

```
## Not run:
Sys.setenv(WHAPI_TOKEN = "your_token_here")

# 1) Local file
 whapi_send_sticker("558191812121",
   sticker = "sticker.webp", type = "file")

# 2) Remote URL
 whapi_send_sticker("558191812121",
   sticker = "https://example.com/condepe.webp", type = "url")

# 3) Pre-encoded Base64
 b64 <- openssl::base64_encode(readBin("sticker.webp","raw",file.info("sticker.webp")$size))
 data_uri <- sprintf("%s", basename("sticker.webp"), b64)
 whapi_send_sticker("558191812121", sticker = data_uri, type = "base64")

## End(Not run)
```

---

whapi_send_text *Send a text message via Whapi.Cloud*

Description

Send a text message via Whapi.Cloud

Usage

```
whapi_send_text(
  to,
  body,
  token = Sys.getenv("WHAPI_TOKEN", unset = ""),
  quoted = NULL,
  edit = NULL,
  typing_time = NULL,
  no_link_preview = NULL,
  wide_link_preview = NULL,
  mentions = NULL,
  view_once = NULL,
  timeout = 30,
  verbose = TRUE
)
```

Arguments

to	Character. WhatsApp number in full international format WITHOUT "+" (digits only, e.g. "5581999999999"), or an existing group/channel id.
body	Character. Text of the message (UTF-8).

token	Character. Whapi Bearer token. By default, taken from the environment variable WHAPI_TOKEN if not provided.
quoted, edit	Character (optional). Message IDs to quote or edit.
typing_time	Numeric (optional). Seconds to simulate typing.
no_link_preview	Logical (optional). TRUE to disable link preview.
wide_link_preview	Logical (optional). TRUE to enable wide preview for links.
mentions	Character vector (optional). Numbers to mention (without "+").
view_once	Logical (optional). TRUE to mark message as "view once".
timeout	Numeric. Request timeout in seconds. Default: 30.
verbose	Logical. Print messages via cli? Default: TRUE.

Value

A tibble with essential information (id, to, status, timestamp) and the full API response in column `resp` (as list).

Examples

```
## Not run:
# Make sure you set WHAPI_TOKEN in your environment or pass via argument
 Sys.setenv(WHAPI_TOKEN = "your_token_here")

# Simple example:
 whapi_send_text("5581999999999", "Hello! Test message via API")

# With extra options:
 whapi_send_text(
   to = "5581999999999",
   body = "Hello, @5581999999999 Ola",
   mentions = c("5581999999999"),
   typing_time = 2, no_link_preview = TRUE
 )

## End(Not run)
```

---

whapi_slugify *whapi_slugify strings for safe IDs (e.g., button IDs in Whapi)*

Description

Converts free-text labels into a safe "slug" format suitable for use as message button IDs or other identifiers in Whapi API requests. Ensures that IDs contain only lowercase letters, digits, and underscores, and are never empty (defaults to `"btn"` if the input is blank).

Usage

```
whapi_slugify(x)
```

Arguments

x	A character vector of labels.

Details

This function is particularly useful when creating interactive messages (buttons, lists) in WhatsApp via Whapi, where each button requires a valid id. By whapi_slugifying titles automatically, we can safely generate IDs even if users provide arbitrary labels with spaces, accents, or symbols.

Transformation steps:

1. Convert to lowercase;
2. Replace any sequence of non-alphanumeric characters with _;
3. Trim leading/trailing underscores;
4. Replace empty results with "btn".

Value

A character vector of the same length with slugified IDs.

See Also

Used internally in whapi_send_quick_reply() and other interactive message helpers.

Examples

```
whapi_slugify(c("Yes!", "Call Us", "Sale!", "###"))
# -> "yes", "call_us", "promocao_rapida", "btn"

# Use case in button creation:
titles <- c("Buy Now", "Learn More")
ids <- whapi_slugify(titles)
tibble::tibble(title = titles, id = ids)
```

---

whapi_to_ascii_lower *Convert text to lowercase ASCII (remove accents)*

Description

Converts input text to lowercase and strips diacritics (accents) by applying a latin-ascii transliteration. Useful for normalization before matching or slug generation.

Usage

```
whapi_to_ascii_lower(x)
```

Arguments

x	Character vector or string. If NULL, an empty string is used.

Value

A character vector in lowercase ASCII without accents.

Examples

```
whapi_to_ascii_lower("Sao Paulo")
#> "sao paulo"
```

---

whapi_to_posixct *Convert numeric timestamp to POSIXct (UTC)*

Description

Utility function to convert a Unix timestamp (seconds since 1970-01-01 UTC) into a POSIXct object. Uses lubridate::as_datetime() for readability and consistency with the tidyverse ecosystem.

Usage

```
whapi_to_posixct(x)
```

Arguments

x	A numeric or character vector representing a Unix timestamp (seconds since epoch). Can be NULL or NA.

Value

A POSIXct object (in UTC) or NA if x is NULL or NA.

See Also

lubridate::as_datetime()

Examples

```
# Single timestamp
whapi_to_posixct(1756426418)

# Vector of timestamps (with NA)
whapi_to_posixct(c(1756426418, NA))

# Character input
whapi_to_posixct("1756426418")
```

---

whapi_trunc *Truncate long strings for logging*

Description

Helper function to shorten long strings when printing to logs. If the input string exceeds `max` characters, it is truncated and suffixed with `"... (truncated)"`.

Usage

```
whapi_trunc(x, max = 2000L)
```

Arguments

`x`	A character string (or coercible to character).
`max`	Integer. Maximum number of characters to display (default: 2000).

Value

A character string, possibly truncated.

Examples

```
whapi_trunc("short text", max = 10)
whapi_trunc(paste(rep("a", 5000), collapse = ""), max = 20)
```

`whapi_validate_list_sections` *Validate* `list_sections` *for Whapi LIST interactive messages*

Description

Internal helper that validates the structure of `list_sections` used in Whapi **LIST** messages. Each section must provide a non-empty `title` and a non-empty `rows` list. Each row must provide non-empty `id` and `title`. The `description` field is optional.

Usage

```
whapi_validate_list_sections(list_sections, verbose = TRUE, trim = TRUE)
```

Arguments

`list_sections`	A list of sections; each section is a named list with `title` (character) and `rows` (list of row objects).
`verbose`	Logical (default `TRUE`). If `TRUE`, prints progress messages via **cli**.
`trim`	Logical (default `TRUE`). If `TRUE`, trims whitespace from `section$title`, `row$title`, and `row$id` before validating.

Details

Expected structure:

```
list_sections <- list(
  list(
    title = "Section title",
    rows = list(
      list(id = "r1", title = "Row title", description = "Optional"),
      ...
    )
  ),
  ...
)
```

This function performs **lightweight validation** and (optionally) trims whitespace from titles and ids to avoid subtle formatting issues.

Value

The (possibly trimmed) `list_sections` object, invisibly unchanged in shape.

See Also

Helpers for interactive payloads such as `whapi_coerce_buttons_base()`, `whapi_coerce_buttons_quick()`, and `whapi_coerce_buttons_mixed()`.

Examples

```
sections <- list(
  list(
    title = "Burgers",
    rows = list(
      list(id = "r1", title = "Plain",  description = "No cheese, no sauce"),
      list(id = "r2", title = "Cheese", description = "With melted cheese")
    )
  )
)
whapi_validate_list_sections(sections)
```

Index