Package: nectar (via r-universe)

May 29, 2026

Title A Framework for Web API Packages

Version 0.0.0.9007

Description An opinionated framework for use within api-wrapping R packages.

License MIT + file LICENSE

URL https://nectar.api2r.org, https://github.com/api2r/nectar

BugReports https://github.com/api2r/nectar/issues

Depends R (>= 4.1.0)

Imports curl, fs, glue, httr2 (>= 1.1.0), jsonlite, lifecycle, purrr, rlang, stbl (>= 0.3.0), stringr, vctrs

Suggests covr, knitr, rmarkdown, stringi, testthat (>= 3.0.0), tibble, tibblify

VignetteBuilder knitr

Remotes wranglezone/tibblify

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libssl-dev

Repository https://api2r.r-universe.dev

Date/Publication 2026-05-12 19:59:54 UTC

RemoteUrl https://github.com/api2r/nectar

RemoteRef HEAD

RemoteSha 9c6c19dd303a71d024d1dd85d3a79ba6cdcfb11d

Contents

auth_api_key . . . 2
auth_prepare . . . 3
choose_pagination_fn . . . 4
compact_nested_list . . . 5
do_if_fn_defined . . . 5
iterate_with_json_cursor . . . 6
req_auth_api_key . . . 7
req_init . . . 9
req_modify . . . 10
req_pagination_policy . . . 11
req_perform_opinionated . . . 12
req_pkg_user_agent . . . 13
req_prepare . . . 14
req_tidy_policy . . . 16
resp_body_auto . . . 17
resp_body_csv . . . 18
resp_body_separate . . . 18
resp_parse . . . 19
resp_tidy . . . 20
resp_tidy_json . . . 21
resp_tidy_unknown . . . 22
tidy_policy_body_auto . . . 23
tidy_policy_json . . . 24
tidy_policy_prepare . . . 25
tidy_policy_unknown . . . 25
url_normalize . . . 26
url_path_append . . . 27

Index **28**

`auth_api_key` *Prepare API key authentication independent of a request*

Description

This helper creates a reusable authentication object that can be passed to `req_prepare()` via `auth`.

Usage

```
auth_api_key(
  parameter_name,
  ...,
  api_key = NULL,
  location = c("header", "query", "cookie"),
  call = rlang::caller_env()
)
```

Arguments

parameter_name	(length-1 character) The name of the parameter to use in the header, query, or cookie.
...	These dots are for future extensions and must be empty.
api_key	(length-1 character or NULL) The API key to use. If this value is NULL, the key will be removed from the request. If this value is NA or an empty string, the request is returned unchanged when the prepared auth is applied.
location	(length-1 character) Where the API key should be passed. One of "header" (default), "query", or "cookie".
call	(environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A list with class "nectar_auth" and elements auth_fn and auth_args.

See Also

Other opinionated auth functions: auth_prepare(), req_auth_api_key()

Examples

```
auth_api_key("X-API-Key", api_key = "my-api-key")
```

---

auth_prepare *Prepare authentication independent of a request*

Description

This constructor stores an authentication function and arguments so the same authentication strategy can be reused across requests.

Usage

```
auth_prepare(auth_fn, ..., call = rlang::caller_env())
```

Arguments

auth_fn	(function) A function to use to authenticate a request.
...	(any) Arguments to pass to auth_fn.
call	(environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A list with class "nectar_auth" and elements auth_fn and auth_args.

See Also

Other opinionated auth functions: auth_api_key(), req_auth_api_key()

Examples

```
auth_prepare(req_auth_api_key, "X-API-Key", api_key = "my-api-key")
```

---

choose_pagination_fn *Extract a pagination policy from a request*

Description

If a request has a pagination policy defined by req_pagination_policy(), extract the pagination_fn from that policy. Otherwise return NULL.

Usage

```
choose_pagination_fn(req)
```

Arguments

req	(httr2_request) A httr2::request() object.

Value

The pagination function, or NULL.

Examples

```
req <- httr2::request("https://example.com")
req <- req_pagination_policy(req, httr2::iterate_with_offset("page"))
choose_pagination_fn(req)
```

compact_nested_list *Discard empty elements*

Description

Discard empty elements in nested lists.

Usage

```
compact_nested_list(lst)
```

Arguments

lst	A (nested) list to filter.

Value

The list, minus empty elements and branches.

Examples

```
x <- list(
  a = list(
    b = letters,
    c = NULL,
    d = 1:5
  ),
  e = NULL,
  f = 1:3
)
compact_nested_list(x)
```

do_if_fn_defined *Use a provided function*

Description

When constructing API calls programmatically, you may encounter situations where an upstream task should indicate which function to apply. For example, one endpoint might use a special auth function that isn't used by other endpoints. This function exists to make coding such situations easier.

Usage

```
do_if_fn_defined(x, fn = NULL, ..., call = rlang::caller_env())
```

Arguments

x	An object to potentially modify, such as a `httr2::request()` object.
fn	A function to apply to `x`. If `fn` is `NULL`, `x` is returned unchanged.
...	Additional arguments to pass to `fn`.
call	(`environment`) The environment from which a function was called, e.g. `rlang::caller_env()` (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

The object, potentially modified.

Examples

```
build_api_req <- function(endpoint, auth_fn = NULL, ...) {
  req <- httr2::request("https://example.com")
  req <- httr2::req_url_path_append(req, endpoint)
  do_if_fn_defined(req, auth_fn, ...)
}

# Most endpoints of this API do not require authentication.
unsecure_req <- build_api_req("unsecure_endpoint")
unsecure_req$headers

# But one endpoint requires authentication.
secure_req <- build_api_req(
  "secure_endpoint", httr2::req_auth_bearer_token, "secret-token"
)
secure_req$headers$Authorization
```

`iterate_with_json_cursor` *Iteration helper for json cursors*

Description

This function is intended as a replacement for `httr2::iterate_with_cursor()` for the common situation where the response body is json, and the cursor can be "empty" in various ways. Even within a single API, some endpoints might return a `NULL` `next_cursor` to indicate that there are no more pages of results, while other endpoints might return `""`. This function normalizes all such results to `NULL`.

Usage

```
iterate_with_json_cursor(param_name = "cursor", next_cursor_path)
```

Arguments

param_name (length-1 character) The name of the cursor parameter in the request.

next_cursor_path (character) A vector indicating the path to the next_cursor element in the body of the response. For example, for the Slack API, this value is c("response_metadata", "next_cursor"), while for the Crossref Unified Resource API, this value is "next-cursor".

Value

A function that takes the response and the previous request, and returns the next request if there are more results.

Examples

```
# Create a cursor iterator for the Slack API response structure
iterate_with_json_cursor("cursor", c("response_metadata", "next_cursor"))

# Create a cursor iterator for the Crossref API
iterate_xref <- iterate_with_json_cursor("cursor", c("message", "next-cursor"))

## Not run:
# Use the iterator to paginate through Crossref API results. The cursor must
# be set to "*" for the initial request to trigger the api to return the next
# cursor.
req <- httr2::request("https://api.crossref.org/works") |>
  httr2::req_url_query(rows = 5, cursor = "*", select = "DOI")

resps <- req_perform_opinionated(
  req, next_req_fn = iterate_xref, max_reqs = 2
)
resps

## End(Not run)
```

---

req_auth_api_key *Authenticate with an API key*

Description

Many APIs provide API keys that can be used to authenticate requests (or, often, provide other information about the user). This function helps to apply those keys to requests.

Usage

```
req_auth_api_key(
  req,
  parameter_name,
```

```
  ...,
  api_key = NULL,
  location = c("header", "query", "cookie"),
  call = rlang::caller_env()
)
```

Arguments

req	(`httr2_request`) A `httr2::request()` object.
parameter_name	(`length-1 character`) The name of the parameter to use in the header, query, or cookie.
...	These dots are for future extensions and must be empty.
api_key	(`length-1 character` or `NULL`) The API key to use. If this value is `NULL`, the key will be removed from the request. If this value is `NA` or an empty string, the request is returned unchanged when the prepared auth is applied.
location	(`length-1 character`) Where the API key should be passed. One of `"header"` (default), `"query"`, or `"cookie"`.
call	(`environment`) The environment from which a function was called, e.g. `rlang::caller_env()` (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A `httr2::request()` object with additional class `nectar_request`.

See Also

Other opinionated auth functions: `auth_api_key()`, `auth_prepare()`

Other opinionated request functions: `req_init()`, `req_modify()`, `req_pagination_policy()`, `req_prepare()`, `req_tidy_policy()`

Examples

```
req <- httr2::request("https://example.com")

# Add an API key named `"X-API-Key"` as a header (default)
req_auth_api_key(req, "X-API-Key", api_key = "my-api-key")

# Add an API key named `"api_key"` as a query parameter
req_auth_api_key(req, "api_key", api_key = "my-api-key", location = "query")

# If `api_key` is NULL, the key is removed from the request
req_auth_api_key(req, "X-API-Key", api_key = NULL)
```

`req_init` *Setup a basic API request*

Description

For a given API, the `base_url` and user agent will generally be the same for every call to that API. Use this function to prepare that piece of the request once for easy reuse.

Usage

```
req_init(
  base_url,
  ...,
  additional_user_agent = NULL,
  call = rlang::caller_env()
)
```

Arguments

`base_url`	(`length-1 character`) The part of the url that is shared by all calls to the API. In some cases there may be a family of base URLs, from which you will need to choose one.
`...`	These dots are for future extensions and must be empty.
`additional_user_agent`	(`length-1 character`) A string to identify where a request is coming from. We automatically include information about your package and nectar, but use this to provide additional details. Default `NULL`.
`call`	(`environment`) The environment from which a function was called, e.g. `rlang::caller_env()` (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A `httr2::request()` object with additional class `nectar_request`.

See Also

Other opinionated request functions: `req_auth_api_key()`, `req_modify()`, `req_pagination_policy()`, `req_prepare()`, `req_tidy_policy()`

Examples

```
req_init("https://example.com")
req_init(
  "https://example.com",
  additional_user_agent = "my_api_client (https://my.api.client)"
)
```

`req_modify` *Modify an API request for a particular endpoint*

Description

Modify the basic request for an API by adding a path and any other path-specific properties.

Usage

```
req_modify(
  req,
  ...,
  path = NULL,
  query = NULL,
  body = NULL,
  mime_type = NULL,
  method = NULL,
  header = NULL,
  cookie = NULL,
  call = rlang::caller_env()
)
```

Arguments

`req`	(`httr2_request`) A `httr2::request()` object.
`...`	These dots are for future extensions and must be empty.
`path`	(`character` or `list`) The route to an API endpoint. Optionally, a list or character vector with the path as one or more unnamed arguments (which will be concatenated with "/") plus named arguments to `glue::glue()` into the path.
`query`	(`character` or `list`) An optional list or character vector of parameters to pass in the query portion of the request. Can also include a `.multi` argument to pass to `httr2::req_url_query()` to control how elements containing multiple values are handled.
`body`	(multiple types) An object to use as the body of the request. If any component of the body is a path, pass it through `fs::path()` or otherwise give it the class "fs_path" to indicate that it is a path.
`mime_type`	(`length-1 character`) The mime type of any files present in the body. Some APIs allow you to leave this as NULL for them to guess.
`method`	(`length-1 character`, optional) If the method is something other than `GET` or `POST`, supply it. Case is ignored.
`header`	(`list` or `NULL`) An optional list of headers to add to the request using `httr2::req_headers()`. A `NULL` value for an individual header will explicitly remove that header if it was previously set.
`cookie`	(`list` or `NULL`) An optional list of cookies to set on the request using `httr2::req_cookies_set()`. `NULL` elements are removed.

call	(environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A httr2::request() object with additional class nectar_request.

See Also

Other opinionated request functions: req_auth_api_key(), req_init(), req_pagination_policy(), req_prepare(), req_tidy_policy()

Examples

```
req_base <- req_init("https://example.com")
req_modify(req_base, path = c("specific/{path}", path = "endpoint"))
req_modify(req_base, query = c("param1" = "value1", "param2" = "value2"))
```

---

req_pagination_policy *Define a pagination policy for a request*

Description

APIs generally have a specified method for requesting multiple pages of results (or sometimes two or three methods). The methods are sometimes documented within a given endpoint, and sometimes documented at the "top" of the documentation. Use this function to attach a pagination policy to a request, so that req_perform_opinionated() can automatically handle pagination.

Usage

```
req_pagination_policy(req, pagination_fn, call = rlang::caller_env())
```

Arguments

req	(httr2_request) A httr2::request() object.
pagination_fn	(function) A function that takes the previous response (resp) to generate the next request in a call to httr2::req_perform_iterative(). This function can usually be generated using one of the iteration helpers described in httr2::iterate_with_offset(). This function will be extracted from the request by req_perform_opinionated() and passed on as next_req to httr2::req_perform_iterative().
call	(environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A `httr2::request()` object with additional class `nectar_request`.

See Also

Other opinionated request functions: `req_auth_api_key()`, `req_init()`, `req_modify()`, `req_prepare()`, `req_tidy_policy()`

Examples

```
req <- httr2::request("https://example.com")
req_pagination_policy(req, httr2::iterate_with_offset("page"))
```

`req_perform_opinionated`

Perform a request with opinionated defaults

Description

This function ensures that a request has `httr2::req_retry()` applied, and then performs the request, using either `httr2::req_perform_iterative()` (if a `next_req_fn` function is supplied) or `httr2::req_perform()` (if not).

Usage

```
req_perform_opinionated(
  req,
  ...,
  next_req_fn = choose_pagination_fn(req),
  max_reqs = 2,
  max_tries_per_req = 3
)
```

Arguments

`req`	The first request to perform.
`...`	These dots are for future extensions and must be empty.
`next_req_fn`	(`function`) An optional function that takes the previous response (`resp`) and request (`req`), and returns a new request. This function is passed as `next_req` in a call to `httr2::req_perform_iterative()`. This function can usually be generated using one of the iteration helpers described in `httr2::iterate_with_offset()`. By default, `choose_pagination_fn()` is used to check for a pagination policy (see `req_pagination_policy()`), and returns `NULL` if no such policy is defined.
`max_reqs`	(`length-1 integer`) The maximum number of separate requests to perform. Passed to the max_reqs argument of `httr2::req_perform_iterative()` when `next_req` is supplied. You will mostly likely want to change the default value (`2`) to `Inf` after you validate that the request works.

max_tries_per_req
: (length-1 integer) The maximum number of times to attempt each individual request. Passed to the max_tries argument of httr2::req_retry().

Value

Always returns a list of httr2::response() objects, one for each request performed, to ensure that downstream operations are the same regardless of the number of responses. The list has additional class nectar_responses.

Examples

```
# Performs a single request and returns a list of responses
req <- httr2::request("https://jsonplaceholder.typicode.com/posts")
resps <- req_perform_opinionated(req)
httr2::resp_status(resps[[1]])
resp_parse(resps, response_parser = resp_tidy_json)
```

---

req_pkg_user_agent *Append package information to user agent*

Description

Add information about nectar and the calling package (if called from a package) to the user agent string.

Usage

```
req_pkg_user_agent(
  req,
  pkg_name = get_pkg_name(call),
  pkg_url = NULL,
  call = rlang::caller_env()
)
```

Arguments

req
: (httr2_request) A httr2::request() object.

pkg_name
: (length-1 character) The name of the calling package. This will usually be automatically determined based on the source of the call.

pkg_url
: (length-1 character) A url for information about the calling package (default NULL).

call
: (environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A `httr2::request()` object with additional class `nectar_request`.

Examples

```
req <- httr2::request("https://example.com")
req$options$useragent
req_pkg_user_agent(req)$options$useragent
req_pkg_user_agent(req, "stbl")$options$useragent
```

---

req_prepare *Prepare a request for an API*

Description

This function implements an opinionated framework for preparing an API request. It is intended to be used inside an API client package. It serves as a wrapper around the `req_` family of functions, such as `httr2::request()`.

Usage

```
req_prepare(
  base_url,
  ...,
  path = NULL,
  query = NULL,
  body = NULL,
  mime_type = NULL,
  method = NULL,
  header = NULL,
  cookie = NULL,
  additional_user_agent = NULL,
  auth = NULL,
  tidy_policy = tidy_policy_body_auto(),
  pagination_fn = NULL,
  call = rlang::caller_env()
)
```

Arguments

`base_url`	(`length-1 character`) The part of the url that is shared by all calls to the API. In some cases there may be a family of base URLs, from which you will need to choose one.
`...`	These dots are for future extensions and must be empty.
`path`	(`character` or `list`) The route to an API endpoint. Optionally, a list or character vector with the path as one or more unnamed arguments (which will be concatenated with "/") plus named arguments to `glue::glue()` into the path.

query (character or list) An optional list or character vector of parameters to pass in the query portion of the request. Can also include a .multi argument to pass to httr2::req_url_query() to control how elements containing multiple values are handled.

body (multiple types) An object to use as the body of the request. If any component of the body is a path, pass it through fs::path() or otherwise give it the class "fs_path" to indicate that it is a path.

mime_type (length-1 character) The mime type of any files present in the body. Some APIs allow you to leave this as NULL for them to guess.

method (length-1 character, optional) If the method is something other than GET or POST, supply it. Case is ignored.

header (list or NULL) An optional list of headers to add to the request using httr2::req_headers(). A NULL value for an individual header will explicitly remove that header if it was previously set.

cookie (list or NULL) An optional list of cookies to set on the request using httr2::req_cookies_set(). NULL elements are removed.

additional_user_agent
(length-1 character) A string to identify where a request is coming from. We automatically include information about your package and nectar, but use this to provide additional details. Default NULL.

auth (nectar_auth or NULL) Authentication prepared with auth_prepare(). By default (NULL), no authentication is performed.

tidy_policy (nectar_tidy_policy or NULL) A tidying policy prepared with tidy_policy_prepare(). By default, tidy_policy_body_auto() is used to automatically apply resp_body_auto() to responses.

pagination_fn (function) A function that takes the previous response (resp) to generate the next request in a call to httr2::req_perform_iterative(). This function can usually be generated using one of the iteration helpers described in httr2::iterate_with_offset(). This function will be extracted from the request by req_perform_opinionated() and passed on as next_req to httr2::req_perform_iterative().

call (environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A httr2::request() object with additional class nectar_request.

See Also

Other opinionated request functions: req_auth_api_key(), req_init(), req_modify(), req_pagination_policy(), req_tidy_policy()

Examples

```
req_prepare("https://example.com")
req_prepare(
  "https://example.com",
  path = c("users/{user_id}", user_id = "42"),
  query = list(format = "json")
)
```

`req_tidy_policy` *Define a tidy policy for a request*

Description

API responses generally follow a structured format. Use this function to define a policy that will be used by `resp_tidy()` to extract the relevant portion of a response and wrangle it into a desired format.

Usage

```
req_tidy_policy(
  req,
  tidy_policy = tidy_policy_body_auto(),
  call = rlang::caller_env()
)
```

Arguments

`req`	(`httr2_request`) A `httr2::request()` object.
`tidy_policy`	(`nectar_tidy_policy` or `NULL`) A tidying policy prepared with `tidy_policy_prepare()`. By default, `tidy_policy_body_auto()` is used to automatically apply `resp_body_auto()` to responses.
`call`	(`environment`) The environment from which a function was called, e.g. `rlang::caller_env()` (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

A `httr2::request()` object with additional class `nectar_request`.

See Also

Other opinionated request functions: `req_auth_api_key()`, `req_init()`, `req_modify()`, `req_pagination_policy()`, `req_prepare()`

Other opinionated response parsers: `resp_tidy()`, `resp_tidy_json()`, `resp_tidy_unknown()`, `tidy_policy_body_auto()`, `tidy_policy_json()`, `tidy_policy_prepare()`, `tidy_policy_unknown()`

Examples

```
req <- httr2::request("https://example.com")
req_tidy_policy(
  req,
  tidy_policy_json()
)
```

`resp_body_auto` *Automatically choose a body parser*

Description

Use the `Content-Type` header (extracted using `httr2::resp_content_type()`) of a response to automatically choose and apply a body parser, such as `httr2::resp_body_json()` or `resp_body_csv()`.

Usage

```
resp_body_auto(resp)
```

Arguments

`resp`	(`httr2_response`) A single `httr2::response()` object (as returned by `httr2::req_perform()`).

Value

The parsed response body.

Examples

```
resp_json <- httr2::response_json(body = list(a = 1, b = "hello"))
resp_body_auto(resp_json)

resp_csv <- httr2::response(
  headers = list("Content-Type" = "text/csv"),
  body = charToRaw("a,b\n1,2\n3,4")
)
resp_body_auto(resp_csv)
```

resp_body_csv *Extract tabular data from response body*

Description

Extract tabular data in comma-separated or tab-separated format from a response body.

Usage

```
resp_body_csv(resp, check_type = TRUE)

resp_body_tsv(resp, check_type = TRUE)
```

Arguments

resp	(httr2_response) A single httr2::response() object (as returned by httr2::req_perform()).
check_type	(length-1 logical) Whether to check that the response has the expected content type. Set to FALSE if the response is not specifically tagged as the proper type.

Value

The parsed response body as a data frame.

Examples

```
resp_csv <- httr2::response(
  headers = list("Content-Type" = "text/csv"),
  body = charToRaw("a,b\n1,2\n3,4")
)
resp_body_csv(resp_csv)
resp_tsv <- httr2::response(
  headers = list("Content-Type" = "text/tab-separated-values"),
  body = charToRaw("a\tb\n1\t2\n3\t4")
)
resp_body_tsv(resp_tsv)
```

resp_body_separate *Extract response body into list*

Description

Wrap the parsed response body in a list(). Unlike resp_body_auto(), this function prevents individual response bodies from being concatenated when combining multiple responses, which is useful for raw or otherwise non-concatenatable types.

Usage

```
resp_body_separate(resp, resp_body_fn = resp_body_auto)
```

Arguments

resp	(httr2_response) A single httr2::response() object (as returned by httr2::req_perform()).
resp_body_fn	(function) A function to extract the body of the response. Default: resp_body_auto().

Value

The parsed response body wrapped in a list(). This is useful for things like raw vectors that you wish to parse with httr2::resps_data().

Examples

```
resp <- httr2::response_json(body = list(a = 1, b = "hello"))
resp_body_separate(resp)
```

---

resp_parse *Parse one or more responses*

Description

httr2 provides two methods for performing requests: httr2::req_perform(), which returns a single httr2::response() object, and httr2::req_perform_iterative(), which returns a list of httr2::response() objects. This function automatically determines whether a single response or multiple responses have been returned, and parses the responses appropriately.

Usage

```
resp_parse(resps, ...)

## Default S3 method:
resp_parse(
  resps,
  ...,
  arg = rlang::caller_arg(resps),
  call = rlang::caller_env()
)

## S3 method for class 'httr2_response'
resp_parse(resps, ..., response_parser = resp_tidy)
```

Arguments

resps	(httr2_response, nectar_responses, or list) A single httr2::response() object (as returned by httr2::req_perform()) or a list of such objects (as returned by req_perform_opinionated() or httr2::req_perform_iterative()).
...	Additional arguments passed on to the response_parser function (in addition to resps).
arg	(length-1 character) An argument name as a string. This argument will be mentioned in error messages as the input that is at the origin of a problem.
call	(environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.
response_parser	(function) A function to parse the server response (resp). Defaults to resp_tidy(), which applies a tidying policy from the request when available and otherwise uses resp_body_auto(). Set this to NULL to return the raw response from httr2::req_perform().

Value

The response parsed by the response_parser. If resps was a list, the parsed responses are concatenated when possible. Unlike httr2::resps_data, this function does not concatenate raw vector responses.

Examples

```
resp <- httr2::response_json(body = list(a = 1, b = "hello"))
resp_parse(resp, response_parser = httr2::resp_body_json)

resps <- list(
  httr2::response_json(body = list(list(id = 1), list(id = 2))),
  httr2::response_json(body = list(list(id = 3), list(id = 4)))
)
resp_parse(resps, response_parser = httr2::resp_body_json)
```

---

resp_tidy *Extract and clean an API response*

Description

API responses generally follow a structured format. Use this function to extract the relevant portion of a response, and wrangle it into a desired format. This function is most useful when the response was fetched with a request that includes a tidying policy defined via req_tidy_policy().

Usage

```
resp_tidy(resp)
```

Arguments

resp	(httr2_response) A single httr2::response() object (as returned by httr2::req_perform()).

Value

The extracted and cleaned response, or NULL if resp is NULL. By default, the response is processed with resp_body_auto(). If the request includes a resp_tidy policy (set via req_tidy_policy()), that policy's function and arguments are used instead.

See Also

Other opinionated response parsers: req_tidy_policy(), resp_tidy_json(), resp_tidy_unknown(), tidy_policy_body_auto(), tidy_policy_json(), tidy_policy_prepare(), tidy_policy_unknown()

Examples

```
# Without a tidy policy, resp_tidy() uses resp_body_auto()
resp <- httr2::response_json(body = list(a = 1, b = "hello"))
resp_tidy(resp)

# With a tidy policy, resp_tidy() uses the policy's tidy function.
req <- req_tidy_policy(
  httr2::request("https://example.com"),
  tidy_policy_prepare(httr2::resp_body_json)
)
# In practice, the request is attached automatically when the response is
# fetched with httr2::req_perform() or req_perform_opinionated().
resp$request <- req
resp_tidy(resp)
```

---

resp_tidy_json *Extract and clean a JSON API response*

Description

Parse the body of a response with httr2::resp_body_json(), extract a named subset of that body, and tidy the result with tibblify::tibblify().

Usage

```
resp_tidy_json(resp, spec = NULL, unspecified = "list", subset_path = NULL)
```

Arguments

resp	(httr2_response) A single httr2::response() object (as returned by httr2::req_perform()).
spec	(tspec or NULL) A specification used by tibblify::tibblify() to parse the extracted body of resp. When spec is NULL (the default), tibblify::tibblify() will attempt to guess a spec.

unspecified (length-1 character) A string that describes what happens if the extracted body of resp contains fields that are not specified in spec. While tibblify::tibblify() defaults to NULL for this value, we set it to list so that the body will still parse when resp contains extra data without throwing errors.

subset_path (character) An optional vector indicating the path to the "real" object within the body of resp. For example, many APIs return a body with information about the status of the response, cache information, perhaps pagination information, and then the actual data in a field such as data. If the desired part of the response body is in data$objects, the value of this argument should be c("data", "object").

Value

The tibblified response body.

See Also

Other opinionated response parsers: req_tidy_policy(), resp_tidy(), resp_tidy_unknown(), tidy_policy_body_auto(), tidy_policy_json(), tidy_policy_prepare(), tidy_policy_unknown()

Examples

```
resp <- httr2::response_json(
  body = list(list(id = 1, name = "Alice"), list(id = 2, name = "Bob"))
)
resp_tidy_json(resp)

# Extract a nested subset of the response body
resp_nested <- httr2::response_json(
  body = list(data = list(list(id = 1), list(id = 2)))
)
resp_tidy_json(resp_nested, subset_path = "data")
```

---

resp_tidy_unknown *Error informatively for unknown response types*

Description

If you have not defined a parser for a response type, use this function to return useful information to help construct a parser.

Usage

```
resp_tidy_unknown(resp, call = rlang::caller_env())
```

Arguments

resp	(httr2_response) A single httr2::response() object (as returned by httr2::req_perform()).
call	(environment) The environment from which a function was called, e.g. rlang::caller_env() (the default). The environment will be mentioned in error messages as the source of the error. This argument is particularly useful for functions that are intended to be called as utilities inside other functions.

Value

This function always throws an error. The error lists the names of the response pieces after parsing with resp_body_auto().

See Also

Other opinionated response parsers: req_tidy_policy(), resp_tidy(), resp_tidy_json(), tidy_policy_body_auto(), tidy_policy_json(), tidy_policy_prepare(), tidy_policy_unknown()

Examples

```
resp <- httr2::response_json(body = list(status = "ok", data = list(id = 1)))
try(
  resp_tidy_unknown(resp)
)
```

---

tidy_policy_body_auto *A policy to automatically parse a response body*

Description

Create a reusable tidy policy that applies resp_body_auto().

Usage

```
tidy_policy_body_auto()
```

Value

A list with class "nectar_tidy_policy" and elements tidy_fn and tidy_args.

See Also

Other opinionated response parsers: req_tidy_policy(), resp_tidy(), resp_tidy_json(), resp_tidy_unknown(), tidy_policy_json(), tidy_policy_prepare(), tidy_policy_unknown()

Examples

```
tidy_policy_body_auto()
```

`tidy_policy_json` *A policy to parse a response body as JSON*

Description

Create a reusable tidy policy that applies `resp_tidy_json()`.

Usage

```
tidy_policy_json(spec = NULL, unspecified = "list", subset_path = NULL)
```

Arguments

`spec`	(`tspec` or `NULL`) A specification used by `tibblify::tibblify()` to parse the extracted body of `resp`. When `spec` is `NULL` (the default), `tibblify::tibblify()` will attempt to guess a spec.
`unspecified`	(`length-1 character`) A string that describes what happens if the extracted body of `resp` contains fields that are not specified in `spec`. While `tibblify::tibblify()` defaults to `NULL` for this value, we set it to `list` so that the body will still parse when `resp` contains extra data without throwing errors.
`subset_path`	(`character`) An optional vector indicating the path to the "real" object within the body of `resp`. For example, many APIs return a body with information about the status of the response, cache information, perhaps pagination information, and then the actual data in a field such as `data`. If the desired part of the response body is in `data$objects`, the value of this argument should be `c("data", "object")`.

Value

A list with class `"nectar_tidy_policy"` and elements `tidy_fn` and `tidy_args`.

See Also

Other opinionated response parsers: `req_tidy_policy()`, `resp_tidy()`, `resp_tidy_json()`, `resp_tidy_unknown()`, `tidy_policy_body_auto()`, `tidy_policy_prepare()`, `tidy_policy_unknown()`

Examples

```
tidy_policy_json(subset_path = "data")
```

tidy_policy_prepare *Prepare tidying independent of a request*

Description

This constructor stores a response tidying function and arguments so the same tidying strategy can be reused across requests.

Usage

```
tidy_policy_prepare(tidy_fn, ...)
```

Arguments

tidy_fn	(function) A function that will be invoked by resp_tidy() to tidy a response.
...	(any) Arguments to pass to tidy_fn.

Value

A list with class "nectar_tidy_policy" and elements tidy_fn and tidy_args.

See Also

Other opinionated response parsers: req_tidy_policy(), resp_tidy(), resp_tidy_json(), resp_tidy_unknown(), tidy_policy_body_auto(), tidy_policy_json(), tidy_policy_unknown()

Examples

```
tidy_policy_prepare(httr2::resp_body_json, simplifyVector = TRUE)
```

tidy_policy_unknown *A policy to error for unknown response bodies*

Description

Create a reusable tidy policy that applies resp_tidy_unknown(), signaling an informative error.

Usage

```
tidy_policy_unknown()
```

Value

A list with class "nectar_tidy_policy" and elements tidy_fn and tidy_args.

See Also

Other opinionated response parsers: `req_tidy_policy()`, `resp_tidy()`, `resp_tidy_json()`, `resp_tidy_unknown()`, `tidy_policy_body_auto()`, `tidy_policy_json()`, `tidy_policy_prepare()`

Examples

```
tidy_policy_unknown()
```

---

url_normalize *Normalize a URL*

Description

This function normalizes a URL by adding a trailing slash to the base if it is missing. It is mainly for testing and other comparisons.

Usage

```
url_normalize(url)
```

Arguments

`url`	A URL to normalize.

Value

A normalized URL

Examples

```
identical(
  url_normalize("https://example.com"),
  url_normalize("https://example.com/")
)
identical(
  url_normalize("https://example.com?param=value"),
  url_normalize("https://example.com/?param=value")
)
```

`url_path_append` *Add path elements to a URL*

Description

Append zero or more path elements to a URL without duplicating "/" characters. Based on `httr2::req_url_path_append()`

Usage

```
url_path_append(url, ...)
```

Arguments

`url`	A URL to modify.
`...`	Path elements to append, as strings.

Value

A modified URL.

Examples

```
url_path_append("https://example.com", "api", "v1", "users")
url_path_append("https://example.com/", "/api", "/v1", "/users")
url_path_append("https://example.com/", "/api/v1/users")
```

Index

∗ **opinionated auth functions**
  auth_api_key, 2
  auth_prepare, 3
  req_auth_api_key, 7
∗ **opinionated request functions**
  req_auth_api_key, 7
  req_init, 9
  req_modify, 10
  req_pagination_policy, 11
  req_prepare, 14
  req_tidy_policy, 16
∗ **opinionated response parsers**
  req_tidy_policy, 16
  resp_tidy, 20
  resp_tidy_json, 21
  resp_tidy_unknown, 22
  tidy_policy_body_auto, 23
  tidy_policy_json, 24
  tidy_policy_prepare, 25
  tidy_policy_unknown, 25

auth_api_key, 2
auth_api_key(), *4*, *8*
auth_prepare, 3
auth_prepare(), *3*, *8*, *15*

choose_pagination_fn, 4
choose_pagination_fn(), *12*
compact_nested_list, 5

do_if_fn_defined, 5

fs::path(), *10*, *15*

glue::glue(), *10*, *14*

httr2::iterate_with_cursor(), *6*
httr2::iterate_with_offset(), *11*, *12*, *15*
httr2::req_cookies_set(), *10*, *15*
httr2::req_headers(), *10*, *15*
httr2::req_perform(), *12*, *17–21*, *23*
httr2::req_perform_iterative(), *11*, *12*, *15*, *19*, *20*
httr2::req_retry(), *12*, *13*
httr2::req_url_path_append(), *27*
httr2::req_url_query(), *10*, *15*
httr2::request(), *4*, *6*, *8–16*
httr2::resp_body_json(), *17*, *21*
httr2::resp_content_type(), *17*
httr2::response(), *13*, *17–21*, *23*
httr2::resps_data, *20*
httr2::resps_data(), *19*

iterate_with_json_cursor, 6

list(), *18*, *19*

req_auth_api_key, 7
req_auth_api_key(), *3*, *4*, *9*, *11*, *12*, *15*, *16*
req_init, 9
req_init(), *8*, *11*, *12*, *15*, *16*
req_modify, 10
req_modify(), *8*, *9*, *12*, *15*, *16*
req_pagination_policy, 11
req_pagination_policy(), *4*, *8*, *9*, *11*, *12*, *15*, *16*
req_perform_opinionated, 12
req_perform_opinionated(), *11*, *15*, *20*
req_pkg_user_agent, 13
req_prepare, 14
req_prepare(), *2*, *8*, *9*, *11*, *12*, *16*
req_tidy_policy, 16
req_tidy_policy(), *8*, *9*, *11*, *12*, *15*, *20–26*
request, *12*
resp_body_auto, 17
resp_body_auto(), *15*, *16*, *18–21*, *23*
resp_body_csv, 18
resp_body_csv(), *17*
resp_body_separate, 18
resp_body_tsv *(*resp_body_csv*)*, 18
resp_parse, 19

resp_tidy, 20
resp_tidy(), *16*, *20*, *22–26*
resp_tidy_json, 21
resp_tidy_json(), *16*, *21*, *23–26*
resp_tidy_unknown, 22
resp_tidy_unknown(), *16*, *21–26*
rlang::caller_env(), *3*, *6*, *8, 9*, *11*, *13*, *15, 16*, *20*, *23*

tibblify::tibblify(), *21, 22*, *24*
tidy_policy_body_auto, 23
tidy_policy_body_auto(), *15, 16*, *21–26*
tidy_policy_json, 24
tidy_policy_json(), *16*, *21–23*, *25, 26*
tidy_policy_prepare, 25
tidy_policy_prepare(), *15, 16*, *21–24*, *26*
tidy_policy_unknown, 25
tidy_policy_unknown(), *16*, *21–25*

url_normalize, 26
url_path_append, 27