Package 'brapiR2'

October 31, 2026

Title A Tidyverse-Native Client for the 'BrAPI' v2 (Breeding API) Specification

Version 0.2.1

Description Provides pipe-friendly, stateless read access to the Breeding API ('BrAPI') v2.1 specification, an open community standard for plant breeding data interchange maintained by the BrAPI project <https://brapi.org>. Wraps 32 of the 37 'BrAPI' v2.1 entities across all four modules, Core, Germplasm, Phenotyping, and Genotyping, covering 49 of the specification's 138 retrieval ('GET' and search) endpoints and returning tidy tibbles ready for analysis. Write and update endpoints are out of scope by design. Features include automatic pagination, async search handling, response caching, parallel batch fetching, and convenience functions for genomic selection workflows (e.g. dosage matrix extraction). Designed for plant breeders and bioinformaticians who need programmatic access to plant breeding databases that implement the 'BrAPI' v2 specification.

License MIT + file LICENSE

URL https://github.com/ropensci/brapiR2, https://docs.ropensci.org/brapiR2/

BugReports https://github.com/ropensci/brapiR2/issues

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports cli (>= 3.6.0), dplyr (>= 1.1.0), glue (>= 1.6.0), httr2 (>= 1.0.0), jsonlite (>= 1.8.0), purrr (>= 1.0.0), rlang (>= 1.1.0), stats, tibble (>= 3.2.0), tidyr (>= 1.3.0), tidyselect (>= 1.2.0), utils

Suggests furrr (>= 0.3.0), future (>= 1.33.0), keyring, knitr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

VignetteBuilder knitr

Config/Needs/website rmarkdown, AlphaSimR, AGHmatrix, BGLR, rrBLUP, sommer, metan, lme4

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Joash Joshua Ayo [aut, cre, cph] (ORCID: <https://orcid.org/0009-0007-1642-0172>), David Waring [rev] (David reviewed the package (v. 0.1.0) for rOpenSci, see <https://github.com/ropensci/software-review/issues/792>), Jenna Hershberger [rev] (Jenna reviewed the package (v. 0.1.0) for rOpenSci, see <https://github.com/ropensci/software-review/issues/792>)

Maintainer Joash Joshua Ayo <joashjoshua789@gmail.com>

Repository CRAN

Date/Publication 2026-10-31 17:20:02 UTC

Contents

brapi_allele_matrix . . . 3
brapi_cache_clear . . . 4
brapi_cache_enable . . . 5
brapi_calls . . . 6
brapi_call_sets . . . 7
brapi_connection . . . 7
brapi_crosses . . . 9
brapi_crossing_projects . . . 9
brapi_endpoints . . . 10
brapi_events . . . 11
brapi_fetch_parallel . . . 12
brapi_germplasm . . . 13
brapi_germplasm_attributes . . . 14
brapi_germplasm_detail . . . 15
brapi_germplasm_pedigree . . . 15
brapi_germplasm_progeny . . . 16
brapi_get . . . 17
brapi_get_dosage_matrix . . . 18
brapi_get_marker_map . . . 19
brapi_images . . . 20
brapi_list . . . 21
brapi_lists . . . 22
brapi_location . . . 23
brapi_locations . . . 23
brapi_login . . . 24
brapi_login_oauth2 . . . 25
brapi_map . . . 26
brapi_maps . . . 27
brapi_map_linkage_groups . . . 28

brapi_marker_positions . . . 28
brapi_methods . . . 30
brapi_observations . . . 31
brapi_observation_units . . . 32
brapi_observation_variables . . . 33
brapi_ontologies . . . 34
brapi_ontology . . . 35
brapi_pedigree . . . 36
brapi_people . . . 37
brapi_ping . . . 38
brapi_post_search . . . 39
brapi_program . . . 40
brapi_programs . . . 41
brapi_references . . . 42
brapi_reference_sets . . . 42
brapi_samples . . . 43
brapi_scales . . . 44
brapi_search_calls . . . 45
brapi_search_germplasm . . . 45
brapi_search_marker_positions . . . 46
brapi_search_observations . . . 48
brapi_search_pedigree . . . 49
brapi_search_variables . . . 50
brapi_search_variants . . . 51
brapi_seasons . . . 51
brapi_seed_lots . . . 52
brapi_server_info . . . 53
brapi_set_token . . . 54
brapi_studies . . . 54
brapi_study . . . 55
brapi_study_data . . . 56
brapi_traits . . . 57
brapi_trial . . . 58
brapi_trials . . . 58
brapi_variants . . . 59
brapi_variant_sets . . . 60
print.brapi_con . . . 61

Index **62**

`brapi_allele_matrix` *Get Allele Matrix*

Description

Retrieves genotype calls from the `/allelematrix` endpoint and returns a tidy tibble with one row per (variant, callSet) combination. The `/allelematrix` response has a unique structure (2-D pagination, no `result$data` envelope) so it cannot use the generic `brapi_get()`.

Usage

```
brapi_allele_matrix(con, variantSetDbId = NULL, ...)
```

Arguments

con	A brapi_connection() object.
variantSetDbId	Character or NULL. Filter by variant set.
...	Additional query parameters (e.g. expandHomozygotes, unknownString, sepPhased, sepUnphased).

Value

A tibble with columns variantDbId, callSetDbId, genotype.

BrAPI endpoint

GET /allelematrix - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

dimensionVariantPage, dimensionVariantPageSize, dimensionCallSetPage, dimensionCallSetPageSize.

Both dimensions are paged automatically, con$page_size variants by con$page_size call sets at a time, unless a dimension*PageSize is given. Supplying dimensionVariantPage or dimensionCallSetPage fetches only that page of that dimension.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_allele_matrix(con, variantSetDbId = "variantset1")
}
```

---

brapi_cache_clear *Clear the Response Cache*

Description

Removes all cached responses from the cache directory. Only brapiR2's own cache files are removed; anything else in the directory is left alone.

Usage

```
brapi_cache_clear(con)
```

Arguments

con	A brapi_connection() object with caching enabled.

Value

Invisibly returns con.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
con <- brapi_cache_enable(con, dir = tempfile("brapi_cache_"))
brapi_cache_clear(con)
```

`brapi_cache_enable` *Enable Response Caching*

Description

Returns a new connection object with caching enabled. Cached responses are stored as JSON files in the specified directory, keyed by URL, query parameters and (hashed) token, so connections with different credentials never share an entry.

Usage

```
brapi_cache_enable(con, dir = NULL, ttl = 3600)
```

Arguments

con	A brapi_connection() object.
dir	Character. Directory to store cached responses. Defaults to R's per-user cache directory for brapiR2, tools::R_user_dir("brapiR2", "cache").
ttl	Numeric. Time-to-live for cached entries in seconds. Default 3600 (1 hour).

Value

A new brapi_con object with caching configured.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
con <- brapi_cache_enable(con, dir = tempfile("brapi_cache_"), ttl = 7200)
con
```

`brapi_calls` *List Genotype Calls*

Description

List Genotype Calls

Usage

```
brapi_calls(con, variantSetDbId = NULL, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`variantSetDbId`	Character or NULL. Filter by variant set.
`...`	Additional query parameters.

Value

A tibble with one row per genotype call.

BrAPI endpoint

`GET /calls` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`callSetDbId`, `variantDbId`, `variantSetDbId`, `expandHomozygotes`, `unknownString`, `sepPhased`, `sepUnphased`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_calls(con, variantSetDbId = "variantset1")
}
```

brapi_call_sets *List Call Sets (Samples with Genotype Data)*

Description

List Call Sets (Samples with Genotype Data)

Usage

```
brapi_call_sets(con, ...)
```

Arguments

con	A `brapi_connection()` object.
...	Additional query parameters.

Value

A tibble with one row per call set.

BrAPI endpoint

`GET /callsets` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`callSetDbId`, `callSetName`, `variantSetDbId`, `sampleDbId`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_call_sets(con)
}
```

brapi_connection *Create a BrAPI Connection Object*

Description

Creates a connection object that holds server URL, authentication token, and configuration. This object is passed as the first argument to all `brapiR2` functions. No global state is used.

Usage

```
brapi_connection(
  url,
  token = NULL,
  version = "v2",
  path = "brapi",
  user_agent = NULL,
  page_size = 1000L,
  timeout = 120
)
```

Arguments

url	Character. Base URL of the BrAPI server (e.g. "https://test-server.brapi.org"). Trailing slashes are removed.
token	Character or NULL. An existing Bearer token for authentication. If NULL, you can authenticate later with brapi_login() or brapi_login_oauth2().
version	Character. BrAPI version path segment. Default "v2".
path	Character. URL path segment before the version, for servers that do not serve BrAPI at /brapi/. Default "brapi". GRIN-Global instances, for example, use "gringlobal/brapi".
user_agent	Character or NULL. Overrides the user agent brapiR2 sends with each request. The default identifies the package, its version, and the httr2 and R versions in use.
page_size	Integer. Number of records per page for paginated requests. Default 1000.
timeout	Numeric. Request timeout in seconds. Default 120.

Value

An S3 object of class "brapi_con" (a named list).

Examples

```
# Connect to the public BrAPI test server (no auth needed)
con <- brapi_connection("https://test-server.brapi.org")
con

# Connect with an existing token
con <- brapi_connection("https://my-breedbase.org", token = "my_token_here")

# Connect to a server that serves BrAPI under a different path
con <- brapi_connection("https://npgsweb.ars-grin.gov",
                        path = "gringlobal/brapi")
```

brapi_crosses *List Crosses*

Description

List Crosses

Usage

```
brapi_crosses(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per cross.

BrAPI endpoint

`GET /crosses` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`crossingProjectDbId`, `crossingProjectName`, `crossDbId`, `crossName`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_crosses(con)
}
```

brapi_crossing_projects *List Crossing Projects*

Description

List Crossing Projects

Usage

```
brapi_crossing_projects(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per crossing project.

BrAPI endpoint

GET /crossingprojects - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

crossingProjectDbId, crossingProjectName, includePotentialParents.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_crossing_projects(con)
}
```

---

brapi_endpoints *List Available Endpoints*

Description

Queries the /serverinfo endpoint to list which BrAPI calls the server supports, along with their HTTP methods and versions.

Usage

```
brapi_endpoints(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with columns for endpoint service, method(s), and version(s).

BrAPI endpoint

`GET /serverinfo` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`contentType`, `dataType`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_endpoints(con)
}
```

`brapi_events` *List Events*

Description

List Events

Usage

```
brapi_events(con, studyDbId = NULL, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`studyDbId`	Character or NULL. Filter by study.
`...`	Additional query parameters.

Value

A tibble with one row per event.

BrAPI endpoint

`GET /events` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`observationUnitDbId`, `eventDbId`, `eventType`, `dateRangeStart`, `dateRangeEnd`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_events(con, studyDbId = "study1")
}
```

`brapi_fetch_parallel` *Parallel Batch Fetching*

Description

Fetches data from multiple BrAPI endpoints or IDs in parallel using the `furrr` package. Useful for retrieving data across many studies, trials, or germplasm records simultaneously.

Usage

```
brapi_fetch_parallel(con, .fn, ids, .workers = NULL, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`.fn`	A brapiR2 function to call for each item (e.g. `brapi_study_data`).
`ids`	Character vector. A set of IDs to iterate over.
`.workers`	Deprecated. No longer used - the parallel backend is now the caller's choice, set via `future::plan()` before calling this function. Supplying a non-`NULL` value emits a deprecation warning and otherwise has no effect.
`...`	Additional arguments passed to `.fn`.

Details

This function uses whatever `future` plan is already active when it is called, and does not set or restore one itself. If you have not called `future::plan()`, `furrr::future_map_dfr()` falls back to `future::sequential`, so nothing runs in parallel until you set a plan yourself - call `future::plan(future::multisession, workers = N)` before this function to fetch in parallel, and `future::plan(future::sequential)` afterwards to shut the workers back down. Per the future package's best-practices vignette, choosing the parallel backend is the caller's decision: a package that sets and restores a plan on every call still mutates session-wide state the caller did not ask it to touch, and can silently replace a backend they configured deliberately.

Value

A tibble with results from all IDs combined. A column whose type differs between IDs becomes character, or a list-column. IDs whose call failed are listed, with their error messages, in a tibble in the `"failed"` attribute (zero rows if none failed), and a warning names them.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  study_ids <- c("study1", "study2", "study3")

  # Set the parallel backend yourself before calling; brapi_fetch_parallel()
  # uses whatever plan is active rather than setting one for you.
  future::plan(future::multisession, workers = 2)
  all_data <- brapi_fetch_parallel(con, brapi_study_data, study_ids)
  future::plan(future::sequential) # shut the workers back down when done
}
```

---

brapi_germplasm *List Germplasm*

Description

List Germplasm

Usage

```
brapi_germplasm(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters (e.g. commonCropName, germplasmName, studyDbId).

Value

A tibble with one row per germplasm accession.

BrAPI endpoint

GET /germplasm - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

accessionNumber, collection, binomialName, genus, species, synonym, parentDbId, progenyDbId.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_germplasm(con)
}
```

`brapi_germplasm_attributes` *List Germplasm Attributes*

Description

List Germplasm Attributes

Usage

```
brapi_germplasm_attributes(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per attribute definition.

BrAPI endpoint

`GET /attributes` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`attributeCategory`, `attributeDbId`, `attributeName`, `attributePUI`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_germplasm_attributes(con)
}
```

brapi_germplasm_detail

Get a Single Germplasm by ID

Description

Get a Single Germplasm by ID

Usage

```
brapi_germplasm_detail(con, germplasmDbId)
```

Arguments

con	A brapi_connection() object.
germplasmDbId	Character. The unique germplasm identifier.

Value

A single-row tibble with germplasm details.

BrAPI endpoint

GET /germplasm/{germplasmDbId} - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_germplasm_detail(con, "germplasm1")
}
```

brapi_germplasm_pedigree

Get Germplasm Pedigree

Description

The /germplasm/{germplasmDbId}/pedigree endpoint this function originally called was deprecated in BrAPI v2.1. It now queries /pedigree?germplasmDbId= instead, which returns a richer record.

Usage

```
brapi_germplasm_pedigree(con, germplasmDbId, ...)
```

Arguments

con	A brapi_connection() object.
germplasmDbId	Character. The unique germplasm identifier.
...	Additional query parameters.

Value

A single-row tibble of the germplasm's pedigree node, with parents, siblings and progeny as list-columns of tidy tibbles. See brapi_pedigree(), which this function calls.

BrAPI endpoint

GET /pedigree - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

accessionNumber, collection, familyCode, binomialName, genus, species, synonym, includeParents, includeSiblings, includeProgeny, includeFullTree, pedigreeDepth, progenyDepth.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_germplasm_pedigree(con, "germplasm1")
}
```

---

brapi_germplasm_progeny

Get Germplasm Progeny

Description

Get Germplasm Progeny

Usage

```
brapi_germplasm_progeny(con, germplasmDbId, ...)
```

Arguments

con	A brapi_connection() object.
germplasmDbId	Character. The unique germplasm identifier.
...	Additional query parameters.

Value

A single-row tibble of the germplasm's pedigree node, with `progeny` as a list-column of a tidy tibble of descendants. See `brapi_pedigree()`, which this function calls.

BrAPI endpoint

`GET /pedigree` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`accessionNumber`, `collection`, `familyCode`, `binomialName`, `genus`, `species`, `synonym`, `includeParents`, `includeSiblings`, `includeProgeny`, `includeFullTree`, `pedigreeDepth`, `progenyDepth`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_germplasm_progeny(con, "germplasm1")
}
```

---

brapi_get *Call Any BrAPI GET Endpoint*

Description

The named functions in brapiR2 cover 32 of the 37 BrAPI v2.1 entities. This is the layer beneath them, for endpoints brapiR2 does not wrap, for servers with non-standard extensions, and for query parameters a named function does not expose. Pagination, caching, authentication and error reporting work exactly as they do for the named functions.

Usage

```
brapi_get(con, endpoint, query = list(), max_pages = Inf)
```

Arguments

`con`	A `brapi_connection()` object.
`endpoint`	Character. The endpoint path, with or without a leading slash (for example `"/programs"` or `"commoncropnames"`). The base URL, BrAPI path and version come from `con`.
`query`	Named list. Query parameters to append to the URL. `pageSize` defaults to the connection's page size; `page` is managed by the pagination loop and should not be set here.
`max_pages`	Numeric. Stop after this many pages instead of fetching all of them. `Inf`, the default, fetches everything. A production server may hold hundreds of thousands of records, so a small value is useful for looking at what an unfamiliar server holds. A truncated result is never cached.

Value

A tibble of results, or an empty tibble if the endpoint returned no data.

Return shape

The response passes through the same parser the named functions use, so a well-formed BrAPI collection returns one row per record. An endpoint returning something the parser does not recognise may come back with list-columns or a shape you need to reshape yourself. The named functions are the better choice wherever one exists.

See Also

`brapi_post_search()` for the POST search endpoints.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {

  # An endpoint brapiR2 does not wrap
  brapi_get(con, "/commoncropnames")

  # A query parameter no named function exposes
  brapi_get(con, "/studies", query = list(active = "true"))
}
```

`brapi_get_dosage_matrix`

Get Dosage Matrix for Genomic Selection

Description

Fetches genotype data from a BrAPI server and converts it into a numeric dosage matrix (samples × markers) compatible with genomic selection packages like `rrBLUP`, `BGLR`, and `sommer`.

Usage

```
brapi_get_dosage_matrix(con, variantSetDbId, sep = "/", unknown_string = ".")
```

Arguments

`con`	A `brapi_connection()` object.
`variantSetDbId`	Character. The variant set to retrieve.
`sep`	Character. Unphased allele separator. Default `"/"` (e.g. `"0/1"`). Phased calls using `"|"` are also handled automatically.
`unknown_string`	Character. String representing missing data. Default `"."`.

Details

The server is asked for expanded homozygotes, sep as the unphased separator and unknown_string for missing data. Allele dosage is then computed by splitting each genotype string on sep (or "|" for phased calls) and counting how many alleles are non-reference (i.e. not "0"). A call with any missing allele (unknown_string, "." or ""), such as "./1", becomes NA. A server that still collapses a homozygote to a single allele ("1") gets it expanded to the ploidy seen in the rest of the data, or to 2 if every call is collapsed. Alleles must be numeric indices ("0", "1", ...); nucleotide calls such as "A/T" are an error.

Value

A numeric (double) matrix: rows = samples (callSetDbIds), columns = markers (variantDbIds). Values are whole-number dosages (0, 1, 2 for diploids; 0–N for polyploids). Missing calls are NA.

BrAPI endpoint

GET /allelematrix - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  dosage <- brapi_get_dosage_matrix(con, "variantset1")
  dim(dosage)
  # Use with rrBLUP:
  # library(rrBLUP)
  # result <- mixed.solve(y = pheno$yield, Z = dosage)
}
```

---

brapi_get_marker_map *Get Marker Map*

Description

Convenience function that retrieves marker positions on a genome map as a tidy tibble. Positions come from the Genome Maps entity (brapi_marker_positions() / /markerpositions), which places a marker on a named brapi_map() - genetic (cM) or physical (bp), per the map's type and unit - not from brapi_variants()'s start/referenceName, which places a variant on a reference assembly instead. A server may populate either, both, or neither; the two are independent coordinate systems, not duplicates of each other.

Usage

```
brapi_get_marker_map(con, variantSetDbId = NULL, mapDbId = NULL)
```

Arguments

con	A brapi_connection() object.
variantSetDbId	Character or NULL. A variant set to retrieve marker positions for. Mutually exclusive with mapDbId.
mapDbId	Character or NULL. A single genome map to retrieve all marker positions from. Mutually exclusive with variantSetDbId.

Details

Supply exactly one of mapDbId (every marker placed on that one map) or variantSetDbId (positions for every variant in that set, wherever they have been placed). The variantSetDbId path looks variant IDs up first via brapi_variants(), then retrieves their positions in one call via brapi_search_marker_positions() rather than the GET /markerpositions filter, which only accepts a single variantDbId.

If a marker is placed on more than one map, it contributes one row per placement - the result is never collapsed to one row per marker.

Value

A tibble with columns variantDbId, variantName, mapDbId, mapName, type, unit, linkageGroupName, and position - one row per marker-map placement, so a marker on several maps appears more than once. type and unit are joined in from brapi_maps() so a caller can tell a genetic (cM) map from a physical (bp) one.

BrAPI endpoints

With mapDbId: GET /markerpositions. With variantSetDbId: GET /variants, then POST /search/markerpositions. Either way, GET /maps for the map metadata. See the v2.1 specification for /markerpositions, /variants, /search/markerpositions and /maps.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_get_marker_map(con, mapDbId = "genome_map1")
  brapi_get_marker_map(con, variantSetDbId = "variantset1")
}
```

---

brapi_images *List Images*

Description

List Images

Usage

```
brapi_images(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per image record.

BrAPI endpoint

GET /images - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

imageDbId, imageName, observationUnitDbId, observationDbId, descriptiveOntologyTerm.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_images(con)
}
```

---

brapi_list *Get a Single List by ID, With Its Contents*

Description

Unlike brapi_lists(), which returns only list metadata, this returns a single list together with its members in the data list-column. listType says what the members are (for example "germplasm").

Usage

```
brapi_list(con, listDbId)
```

Arguments

con	A brapi_connection() object.
listDbId	Character. The unique list identifier.

Value

A single-row tibble of list metadata, with the list's members as a character vector in the data list-column.

BrAPI endpoint

GET /lists/{listDbId} - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  lst <- brapi_list(con, "list1")
  lst$data[[1]]
}
```

---

brapi_lists *List Generic Lists*

Description

List Generic Lists

Usage

```
brapi_lists(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters (e.g. listType).

Value

A tibble with one row per list.

BrAPI endpoint

GET /lists - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

listType, listName, listDbId, listSource.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_lists(con)
}
```

brapi_location *Get a Single Location by ID*

Description

Get a Single Location by ID

Usage

```
brapi_location(con, locationDbId)
```

Arguments

con	A brapi_connection() object.
locationDbId	Character. The unique location identifier.

Value

A single-row tibble with location metadata, including coordinates where the server provides them.

BrAPI endpoint

GET /locations/{locationDbId} - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_location(con, "location_01")
}
```

brapi_locations *List Locations*

Description

List Locations

Usage

```
brapi_locations(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters (e.g. locationType).

Value

A tibble with one row per location.

BrAPI endpoint

GET /locations - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

locationType, locationDbId, locationName, parentLocationDbId, parentLocationName.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_locations(con)
}
```

---

brapi_login *Login to a BrAPI Server with Username and Password*

Description

Authenticates using the BrAPI /token endpoint and returns an updated connection object with the Bearer token set.

Usage

```
brapi_login(con, username, password)
```

Arguments

con	A brapi_connection() object.
username	Character. Your username.
password	Character. Your password.

Value

A new brapi_con object with the token populated.

BrAPI endpoint

`POST /token`. This is not part of the BrAPI v2.1 specification: it is a server extension, implemented by Breedbase and by the BrAPI test server. Servers without it need `brapi_login_oauth2()` or `brapi_set_token()` instead.

See Also

The "Handling Credentials Safely" section of `vignette("brapiR2")` for how to keep `username`/`password` out of your script, using `.Renviron` or the keyring package.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  con <- brapi_login(con, "brapi_reader", "brapi_reader")
  con
}
```

---

brapi_login_oauth2 *Login to a BrAPI Server with OAuth 2.0*

Description

Performs the OAuth 2.0 client-credentials grant: posts the client ID and secret, form-encoded, to the token endpoint and returns an updated connection object with the Bearer token set. Interactive (authorization-code) login is not supported.

Usage

```
brapi_login_oauth2(
  con,
  client_id,
  client_secret,
  authorize_url = NULL,
  access_url
)
```

Arguments

`con`	A `brapi_connection()` object.
`client_id`	Character. OAuth client ID.
`client_secret`	Character. OAuth client secret.
`authorize_url`	Deprecated and unused: the client-credentials grant only needs `access_url`. Supplying it gives a warning.
`access_url`	Character. The token endpoint URL.

Value

A new `brapi_con` object with the token populated.

BrAPI endpoint

None. The token request goes to `access_url`, your identity provider's OAuth 2.0 token endpoint, which is not part of the BrAPI specification.

Examples

```
## Not run:
# Needs an OAuth 2.0 token endpoint and client credentials of your own;
# the public BrAPI test server has no OAuth server to run this against.
con <- brapi_connection("https://my-brapi-server.org")
con <- brapi_login_oauth2(
  con,
  client_id = Sys.getenv("BRAPI_CLIENT_ID"),
  client_secret = Sys.getenv("BRAPI_CLIENT_SECRET"),
  access_url = "https://auth.my-brapi-server.org/token"
)
con

## End(Not run)
```

`brapi_map` *Get a Single Genome Map by ID*

Description

Get a Single Genome Map by ID

Usage

```
brapi_map(con, mapDbId)
```

Arguments

`con`	A `brapi_connection()` object.
`mapDbId`	Character. The unique genome map identifier.

Value

A single-row tibble with genome map details, including `type` (e.g. `"Genetic"` or `"Physical"`) and `unit` (e.g. `"cM"` or `"bp"`).

BrAPI endpoint

`GET /maps/{mapDbId}` - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_map(con, "genome_map1")
}
```

`brapi_maps` *List Genome Maps*

Description

List Genome Maps

Usage

```
brapi_maps(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per genome map.

BrAPI endpoint

`GET /maps` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`mapDbId`, `mapPUI`, `scientificName`, `type`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_maps(con)
}
```

brapi_map_linkage_groups

List the Linkage Groups of a Genome Map

Description

A linkage group is BrAPI's generic term for a named section of a map - it may represent a chromosome, a scaffold, or a generic linkage group.

Usage

```
brapi_map_linkage_groups(con, mapDbId)
```

Arguments

con	A brapi_connection() object.
mapDbId	Character. The unique genome map identifier.

Value

A tibble with one row per linkage group on the map.

BrAPI endpoint

GET /maps/{mapDbId}/linkagegroups - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_map_linkage_groups(con, "genome_map1")
}
```

brapi_marker_positions

List Marker Positions

Description

Retrieves marker placements on genome maps from /markerpositions. A position here is relative to a named brapi_map() (genetic, in cM, or physical, in bp) - a different coordinate system from brapi_variants()'s start/referenceName, which places a variant on a reference assembly instead. A server may populate either, both, or neither; one being empty does not imply the other is.

Usage

```
brapi_marker_positions(
  con,
  mapDbId = NULL,
  variantDbId = NULL,
  linkageGroupName = NULL,
  minPosition = NULL,
  maxPosition = NULL,
  ...
)
```

Arguments

con	A brapi_connection() object.
mapDbId	Character or NULL. Filter by genome map.
variantDbId	Character or NULL. Filter by a single marker/variant ID. For multiple IDs at once, use brapi_search_marker_positions() instead.
linkageGroupName	Character or NULL. Filter by linkage group (e.g. chromosome) name.
minPosition	Integer or NULL. Minimum position, inclusive.
maxPosition	Integer or NULL. Maximum position, inclusive.
...	Additional query parameters.

Value

A tibble with one row per marker placement.

BrAPI endpoint

GET /markerpositions - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

mapDbId, linkageGroupName, variantDbId, minPosition, maxPosition.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_marker_positions(con, mapDbId = "genome_map1")
}
```

`brapi_methods` *List Methods*

Description

List Methods

Usage

```
brapi_methods(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per measurement method.

BrAPI endpoint

`GET /methods` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`methodDbId`, `observationVariableDbId`.

See Also

`brapi_ontologies()` and `brapi_ontology()` to resolve the ontology a method's `ontologyDbId`/`ontologyReference` points to.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_methods(con)
}
```

`brapi_observations` *List Observations*

Description

List Observations

Usage

```
brapi_observations(con, studyDbId = NULL, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`studyDbId`	Character or NULL. Filter by study.
`...`	Additional query parameters.

Value

A tibble with one row per observation (trait measurement).

BrAPI endpoint

`GET /observations` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`observationDbId`, `observationUnitDbId`, `observationVariableDbId`, `locationDbId`, `seasonDbId`, `observationTimeStampRangeStart`, `observationTimeStampRangeEnd`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_observations(con, studyDbId = "study1")
}
```

`brapi_observation_units` *List Observation Units*

Description

List Observation Units

Usage

```
brapi_observation_units(con, studyDbId = NULL, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`studyDbId`	Character or NULL. Filter by study.
`...`	Additional query parameters.

Value

A tibble with one row per observation unit (plot/plant/sample).

BrAPI endpoint

`GET /observationunits` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`observationUnitDbId`, `observationUnitName`, `locationDbId`, `seasonDbId`, `includeObservations`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_observation_units(con, studyDbId = "study1")
}
```

`brapi_observation_variables` *List Observation Variables*

Description

Returns the ontology of observation variables (trait + method + scale).

Usage

```
brapi_observation_variables(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per variable definition.

BrAPI endpoint

`GET /variables` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`observationVariableDbId`, `observationVariableName`, `observationVariablePUI`, `traitClass`.

See Also

`brapi_ontologies()` and `brapi_ontology()` to resolve the ontology a variable's `ontologyDbId`/`ontologyReference` points to.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_observation_variables(con)
}
```

`brapi_ontologies` *List Ontologies*

Description

Retrieves the ontologies registered on the server: metadata about each ontology (name, version, authors, description, ...), not the trait terms that belong to it. `brapi_traits()`, `brapi_scales()`, `brapi_methods()`, and `brapi_observation_variables()` each carry an ontology reference back to one of these records.

Usage

```
brapi_ontologies(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per ontology.

BrAPI endpoint

`GET /ontologies` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`ontologyName`.

See Also

`brapi_ontology()` for a single ontology by ID; `brapi_traits()`, `brapi_scales()`, `brapi_methods()`, and `brapi_observation_variables()` for the records that reference these ontologies.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_ontologies(con)
}
```

brapi_ontology *Get a Single Ontology by ID*

Description

Get a Single Ontology by ID

Usage

```
brapi_ontology(con, ontologyDbId)
```

Arguments

con	A brapi_connection() object.
ontologyDbId	Character. The unique ontology identifier.

Value

A single-row tibble with ontology details.

BrAPI endpoint

GET /ontologies/{ontologyDbId} - see the v2.1 specification.

See Also

brapi_ontologies(); brapi_traits(), brapi_scales(), brapi_methods(), and brapi_observation_variables() for the records that reference ontologies.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_ontology(con, "O_001")
}
```

`brapi_pedigree` *List Pedigree Nodes*

Description

Retrieves a filtered subset of a pedigree tree via `/pedigree` - a batch endpoint for pulling pedigree records across many germplasm in one call. `brapi_germplasm_pedigree()` calls this endpoint for a single germplasm and returns just that germplasm's node, so it must be called once per germplasm. Use `brapi_pedigree()` (or `brapi_search_pedigree()`) to pull pedigree records for many germplasm at once - e.g. everything in a crop, program, or family - in one or a few requests; use `brapi_germplasm_pedigree()` when you already have a single germplasm ID in hand.

Usage

```
brapi_pedigree(
  con,
  germplasmDbId = NULL,
  includeParents = NULL,
  includeSiblings = NULL,
  includeProgeny = NULL,
  includeFullTree = NULL,
  pedigreeDepth = NULL,
  progenyDepth = NULL,
  ...
)
```

Arguments

`con`	A `brapi_connection()` object.
`germplasmDbId`	Character or NULL. Filter by germplasm.
`includeParents`	Logical or NULL. Include each node's parents.
`includeSiblings`	Logical or NULL. Include each node's siblings.
`includeProgeny`	Logical or NULL. Include each node's progeny.
`includeFullTree`	Logical or NULL. Recursively include every node reachable in the pedigree tree.
`pedigreeDepth`	Integer or NULL. Number of levels to include up the tree (parents, grandparents, ...).
`progenyDepth`	Integer or NULL. Number of levels to include down the tree (children, grandchildren, ...).
`...`	Additional query parameters.

Details

Each row is one pedigree node (one germplasm). The server only includes a node's relatives if asked: set `includeParents`, `includeSiblings`, and/or `includeProgeny` to `TRUE` to populate the `parents`, `siblings`, and `progeny` list-columns, each holding a small tibble of related germplasm (`germplasmDbId`, `germplasmName`, and `parentType` - `NA` for siblings, which have none) that you can `tidyr::unnest()` when you need one row per relationship rather than one row per node. Nodes are never collapsed or flattened by default: a pedigree is graph-shaped (each node has its own parents, siblings, and progeny edges), and the three relation types don't share a common row shape, so there is no lossless single flat table to fall back to.

Value

A tibble with one row per pedigree node. `parents`, `siblings`, and `progeny`, when requested, are list-columns of small tibbles (one row per relative) rather than raw nested lists or a flattened table.

BrAPI endpoint

`GET /pedigree` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`accessionNumber`, `collection`, `familyCode`, `binomialName`, `genus`, `species`, `synonym`, `includeParents`, `includeSiblings`, `includeProgeny`, `includeFullTree`, `pedigreeDepth`, `progenyDepth`.

See Also

`brapi_germplasm_pedigree()` for one germplasm's pedigree node; `brapi_search_pedigree()` for the same batch retrieval via `POST`, with a fuller set of filters.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_pedigree(con, includeParents = TRUE, includeProgeny = TRUE)
}
```

`brapi_people` *List People*

Description

List People

Usage

```
brapi_people(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per person.

BrAPI endpoint

GET /people - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

firstName, lastName, personDbId, userID.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_people(con)
}
```

---

brapi_ping *Ping a BrAPI Server*

Description

Tests whether the BrAPI server is reachable and responding.

Usage

```
brapi_ping(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Details

A ping uses a timeout of at most 10 seconds and doesn't retry, so an unreachable server is reported quickly.

Value

Logical, invisibly. TRUE if /serverinfo answers with HTTP 200 and a JSON body, or with 401/403 (the server is up but needs authentication); FALSE otherwise. The HTTP status is attached as the "status" attribute (NA if the server could not be reached).

BrAPI endpoint

GET /serverinfo - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

contentType, dataType.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
brapi_ping(con)
```

---

brapi_post_search *Call Any BrAPI Search Endpoint*

Description

The companion to brapi_get() for the POST /search/{entity} endpoints, which take a filter body rather than query parameters and may run asynchronously. Use it for search endpoints brapiR2 does not wrap, or for filter fields a named search function does not expose.

Usage

```
brapi_post_search(
  con,
  endpoint,
  body = list(),
  poll_interval = 2,
  max_polls = 30L
)
```

Arguments

con	A brapi_connection() object.
endpoint	Character. The search endpoint, with or without a leading slash (for example "/search/germplasm").
body	Named list. The search request body. Filter fields are sent as JSON arrays, as BrAPI expects, even when you supply a single value.
poll_interval	Numeric. Seconds between polling attempts for an asynchronous search. Default 2.
max_polls	Integer. Maximum polling attempts before giving up. Default 30.

Value

A tibble of search results.

Asynchronous searches

A server may answer immediately with the results, or with HTTP 202 and a `searchResultsDbId` to be polled until the results are ready. Both are handled here; the polling happens inside the call and you get the finished results either way. Results spanning several pages are fetched page by page and returned together.

Return shape

As with `brapi_get()`, the response passes through the same parser the named functions use. A well-formed BrAPI result returns one row per record; an unusual one may need reshaping yourself.

See Also

`brapi_get()` for the GET endpoints.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_post_search(con, "/search/germplasm",
                    body = list(germplasmNames = "Tomatillo Fantastico"))
}
```

---

brapi_program *Get a Single Program by ID*

Description

Get a Single Program by ID

Usage

```
brapi_program(con, programDbId)
```

Arguments

`con`	A `brapi_connection()` object.
`programDbId`	Character. The unique program identifier.

Value

A single-row tibble with program details.

BrAPI endpoint

GET /programs/{programDbId} - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_program(con, "program1")
}
```

---

brapi_programs *List Breeding Programs*

Description

Retrieves a list of breeding programs from the BrAPI server.

Usage

```
brapi_programs(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters passed to the API (e.g. commonCropName = "rice", programName = "IRRI").

Value

A tibble with one row per program.

BrAPI endpoint

GET /programs - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

abbreviation, programType.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_programs(con)
  brapi_programs(con, commonCropName = "rice")
}
```

brapi_references *List References (Chromosomes/Contigs)*

Description

List References (Chromosomes/Contigs)

Usage

```
brapi_references(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per reference sequence.

BrAPI endpoint

GET /references - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

referenceDbId, referenceSetDbId, accession, md5checksum, isDerived, minLength, maxLength.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_references(con)
}
```

brapi_reference_sets *List Reference Sets (Genome Assemblies)*

Description

List Reference Sets (Genome Assemblies)

Usage

```
brapi_reference_sets(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per reference set.

BrAPI endpoint

GET /referencesets - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

referenceSetDbId, accession, assemblyPUI, md5checksum.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_reference_sets(con)
}
```

---

brapi_samples *List Samples*

Description

List Samples

Usage

```
brapi_samples(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters (e.g. studyDbId, germplasmDbId, observationUnitDbId).

Value

A tibble with one row per sample.

BrAPI endpoint

GET /samples - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

sampleDbId, sampleName, sampleGroupDbId, observationUnitDbId, plateDbId, plateName.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_samples(con)
}
```

`brapi_scales` *List Scales*

Description

List Scales

Usage

```
brapi_scales(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per scale definition.

BrAPI endpoint

`GET /scales` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`scaleDbId`, `observationVariableDbId`.

See Also

`brapi_ontologies()` and `brapi_ontology()` to resolve the ontology a scale's `ontologyDbId`/`ontologyReference` points to.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_scales(con)
}
```

brapi_search_calls *Search Genotype Calls*

Description

Search Genotype Calls

Usage

```
brapi_search_calls(con, variantSetDbIds = NULL, callSetDbIds = NULL, ...)
```

Arguments

con	A brapi_connection() object.
variantSetDbIds	Character vector. Filter by variant set IDs.
callSetDbIds	Character vector. Filter by call set IDs.
...	Additional search body parameters.

Value

A tibble of matching genotype calls.

BrAPI endpoint

POST /search/calls - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_calls(con, variantSetDbIds = "variantset1")
}
```

brapi_search_germplasm *Search Germplasm*

Description

Performs a BrAPI search for germplasm records matching the given criteria.

Usage

```
brapi_search_germplasm(
  con,
  germplasmNames = NULL,
  germplasmDbIds = NULL,
  commonCropNames = NULL,
  ...
)
```

Arguments

con	A brapi_connection() object.
germplasmNames	Character vector. Filter by germplasm names.
germplasmDbIds	Character vector. Filter by database IDs.
commonCropNames	Character vector. Filter by crop name.
...	Additional body parameters for the search request.

Value

A tibble of matching germplasm records.

BrAPI endpoint

POST /search/germplasm - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_germplasm(con, commonCropNames = "Tomatillo")
}
```

---

brapi_search_marker_positions

Search Marker Positions

Description

The /markerpositions GET filter (see brapi_marker_positions()) takes a single variantDbId; this search endpoint accepts many IDs at once, which is what brapi_get_marker_map() uses internally when looking up positions for an entire variant set.

Usage

```
brapi_search_marker_positions(
  con,
  mapDbIds = NULL,
  variantDbIds = NULL,
  linkageGroupNames = NULL,
  minPosition = NULL,
  maxPosition = NULL,
  ...
)
```

Arguments

con	A brapi_connection() object.
mapDbIds	Character vector. Filter by genome map IDs.
variantDbIds	Character vector. Filter by marker/variant IDs.
linkageGroupNames	Character vector. Filter by linkage group names.
minPosition	Integer. Minimum position, inclusive.
maxPosition	Integer. Maximum position, inclusive.
...	Additional search body parameters.

Value

A tibble of matching marker positions.

BrAPI endpoint

POST /search/markerpositions - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_marker_positions(
    con,
    variantDbIds = c("variant01", "variant02")
  )
}
```

`brapi_search_observations` *Search Observations*

Description

Search Observations

Usage

```
brapi_search_observations(
  con,
  studyDbIds = NULL,
  observationVariableDbIds = NULL,
  ...
)
```

Arguments

`con`	A `brapi_connection()` object.
`studyDbIds`	Character vector. Filter by study IDs.
`observationVariableDbIds`	Character vector. Filter by variable IDs.
`...`	Additional search body parameters.

Value

A tibble of matching observations.

BrAPI endpoint

`POST /search/observations` - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_observations(con, studyDbIds = "study1")
}
```

`brapi_search_pedigree` *Search Pedigree Nodes*

Description

The `POST` equivalent of `brapi_pedigree()`, taking the same tree-shaping parameters plus the fuller set of filters `/search/pedigree` accepts (crop, program, trial, study, accession number, collection, family code, genus/species, and more - pass any of these through `...`). See `brapi_pedigree()` for the shape of the returned tibble and its relationship to `brapi_germplasm_pedigree()`.

Usage

```
brapi_search_pedigree(
  con,
  germplasmDbIds = NULL,
  includeParents = NULL,
  includeSiblings = NULL,
  includeProgeny = NULL,
  includeFullTree = NULL,
  pedigreeDepth = NULL,
  progenyDepth = NULL,
  ...
)
```

Arguments

`con`	A `brapi_connection()` object.
`germplasmDbIds`	Character vector. Filter by germplasm IDs.
`includeParents`	Logical. Include each node's parents.
`includeSiblings`	Logical. Include each node's siblings.
`includeProgeny`	Logical. Include each node's progeny.
`includeFullTree`	Logical. Recursively include every node reachable in the pedigree tree.
`pedigreeDepth`	Integer. Number of levels to include up the tree.
`progenyDepth`	Integer. Number of levels to include down the tree.
`...`	Additional search body parameters.

Value

A tibble with one row per pedigree node; see `brapi_pedigree()` for column details.

BrAPI endpoint

`POST /search/pedigree` - see the v2.1 specification.

See Also

brapi_pedigree(), brapi_germplasm_pedigree()

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_pedigree(con, includeParents = TRUE)
}
```

---

brapi_search_variables

Search Observation Variables

Description

Search Observation Variables

Usage

```
brapi_search_variables(con, traitClasses = NULL, ...)
```

Arguments

con	A brapi_connection() object.
traitClasses	Character vector. Filter by trait class.
...	Additional search body parameters.

Value

A tibble of matching observation variables.

BrAPI endpoint

POST /search/variables - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_variables(con, traitClasses = "agronomic")
}
```

brapi_search_variants *Search Variants*

Description

Search Variants

Usage

```
brapi_search_variants(con, variantSetDbIds = NULL, ...)
```

Arguments

con	A brapi_connection() object.
variantSetDbIds	Character vector. Filter by variant set IDs.
...	Additional search body parameters.

Value

A tibble of matching variants.

BrAPI endpoint

POST /search/variants - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_search_variants(con, variantSetDbIds = "variantset1")
}
```

brapi_seasons *List Seasons*

Description

List Seasons

Usage

```
brapi_seasons(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters (e.g. year).

Value

A tibble with one row per season.

BrAPI endpoint

GET /seasons - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

seasonDbId, season, seasonName, year.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_seasons(con)
}
```

---

brapi_seed_lots *List Seed Lots*

Description

List Seed Lots

Usage

```
brapi_seed_lots(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble with one row per seed lot.

BrAPI endpoint

GET /seedlots - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

seedLotDbId, crossDbId, crossName.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_seed_lots(con)
}
```

---

brapi_server_info *Get Server Info*

Description

Returns a tibble of BrAPI calls supported by the server. Each row is one supported endpoint with columns for service name, HTTP methods, BrAPI versions, and content/data types.

Usage

```
brapi_server_info(con, ...)
```

Arguments

con	A brapi_connection() object.
...	Additional query parameters.

Value

A tibble of supported endpoints and their methods.

BrAPI endpoint

GET /serverinfo - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

contentType, dataType.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_server_info(con)
}
```

brapi_set_token *Manually Set an Authentication Token*

Description

If you already have a token (e.g. from a web browser session), you can set it directly without going through a login flow.

Usage

```
brapi_set_token(con, token)
```

Arguments

con	A brapi_connection() object.
token	Character. The Bearer token string.

Value

A new brapi_con object with the token populated.

See Also

The "Handling Credentials Safely" section of vignette("brapiR2") for how to keep token out of your script, using .Renviron or the keyring package.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
con <- brapi_set_token(con, "my_existing_token")
```

brapi_studies *List Studies*

Description

Retrieves studies (occurrences/environments), optionally filtered by trial.

Usage

```
brapi_studies(con, trialDbId = NULL, ...)
```

Arguments

con	A brapi_connection() object.
trialDbId	Character or NULL. Filter by trial.
...	Additional query parameters.

Value

A tibble with one row per study.

BrAPI endpoint

GET /studies - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

studyType, locationDbId, seasonDbId, studyCode, studyPUI, observationVariableDbId, active, sortBy, sortOrder.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_studies(con)
  brapi_studies(con, trialDbId = "trial1")
}
```

---

brapi_study *Get a Single Study by ID*

Description

Get a Single Study by ID

Usage

```
brapi_study(con, studyDbId)
```

Arguments

con	A brapi_connection() object.
studyDbId	Character. The unique study identifier.

Value

A single-row tibble with study metadata.

BrAPI endpoint

GET /studies/{studyDbId} - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_study(con, "study1")
}
```

---

brapi_study_data *Get Study Data in Wide Format*

Description

A convenience function that fetches a study's observations and pivots them into a wide-format tibble with one row per observation unit and one column per trait — ready for analysis.

Usage

```
brapi_study_data(con, studyDbId, fallback = FALSE)
```

Arguments

con	A brapi_connection() object.
studyDbId	Character. The unique study identifier.
fallback	Logical. If the filtered request returns no observations, fetch all observations on the server and filter them to studyDbId locally. Default FALSE.

Details

Some servers ignore the studyDbId filter on /observations and return nothing for it; the public BrAPI test server (test-server.brapi.org) is one. Set fallback = TRUE to fetch every observation on such a server and filter them locally. That can mean a very large download on a production server, so it is off by default.

Value

A wide-format tibble with the observation unit, germplasm and study identifier columns the server provides, and one character column per observed trait holding its values. A cell with repeated measurements is a list of them.

BrAPI endpoint

GET /observations, filtered by studyDbId, and with no filter when fallback = TRUE - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  # The test server ignores the studyDbId filter, so opt in to the fallback
  data <- brapi_study_data(con, "study1", fallback = TRUE)
  head(data)
}
```

`brapi_traits` *List Traits*

Description

List Traits

Usage

```
brapi_traits(con, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`...`	Additional query parameters.

Value

A tibble with one row per trait.

BrAPI endpoint

`GET /traits` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`traitDbId`, `observationVariableDbId`.

See Also

`brapi_ontologies()` and `brapi_ontology()` to resolve the ontology a trait's `ontologyDbId`/`ontologyReference` points to.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_traits(con)
}
```

brapi_trial *Get a Single Trial by ID*

Description

Get a Single Trial by ID

Usage

```
brapi_trial(con, trialDbId)
```

Arguments

con	A brapi_connection() object.
trialDbId	Character. The unique trial identifier.

Value

A single-row tibble with trial details.

BrAPI endpoint

GET /trials/{trialDbId} - see the v2.1 specification.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_trial(con, "trial1")
}
```

brapi_trials *List Trials*

Description

Retrieves trials, optionally filtered by program.

Usage

```
brapi_trials(con, programDbId = NULL, ...)
```

Arguments

con	A brapi_connection() object.
programDbId	Character or NULL. Filter by program.
...	Additional query parameters.

Value

A tibble with one row per trial.

BrAPI endpoint

GET /trials - see the v2.1 specification.

Query parameters the specification defines, which may be passed through ...:

active, contactDbId, locationDbId, searchDateRangeStart, searchDateRangeEnd, trialPUI, sortBy, sortOrder.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_trials(con)
}
```

---

brapi_variants *List Variants (Markers/SNPs)*

Description

List Variants (Markers/SNPs)

Usage

```
brapi_variants(con, variantSetDbId = NULL, ...)
```

Arguments

con	A brapi_connection() object.
variantSetDbId	Character or NULL. Filter by variant set.
...	Additional query parameters.

Value

A tibble with one row per variant.

BrAPI endpoint

`GET /variants` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`variantDbId`, `variantSetDbId`, `referenceDbId`, `referenceSetDbId`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_variants(con, variantSetDbId = "variantset1")
}
```

`brapi_variant_sets` *List Variant Sets (Datasets)*

Description

List Variant Sets (Datasets)

Usage

```
brapi_variant_sets(con, studyDbId = NULL, ...)
```

Arguments

`con`	A `brapi_connection()` object.
`studyDbId`	Character or NULL. Filter by study.
`...`	Additional query parameters.

Value

A tibble with one row per variant set.

BrAPI endpoint

`GET /variantsets` - see the v2.1 specification.

Query parameters the specification defines, which may be passed through `...`:

`variantSetDbId`, `variantDbId`, `callSetDbId`, `referenceSetDbId`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
if (brapi_ping(con)) {
  brapi_variant_sets(con)
}
```

`print.brapi_con` *Print a BrAPI Connection Object*

Description

Print a BrAPI Connection Object

Usage

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

Arguments

`x`	A `brapi_con` object.
`...`	Additional arguments (ignored).

Value

Invisibly returns `x`.

Examples

```
con <- brapi_connection("https://test-server.brapi.org")
print(con)
```

Index

brapi_allele_matrix, 3
brapi_cache_clear, 4
brapi_cache_enable, 5
brapi_call_sets, 7
brapi_calls, 6
brapi_connection, 7
brapi_connection(), *4–7*, *9–18*, *20–36*, *38–60*
brapi_crosses, 9
brapi_crossing_projects, 9
brapi_endpoints, 10
brapi_events, 11
brapi_fetch_parallel, 12
brapi_germplasm, 13
brapi_germplasm_attributes, 14
brapi_germplasm_detail, 15
brapi_germplasm_pedigree, 15
brapi_germplasm_pedigree(), *36, 37*, *49, 50*
brapi_germplasm_progeny, 16
brapi_get, 17
brapi_get(), *39, 40*
brapi_get_dosage_matrix, 18
brapi_get_marker_map, 19
brapi_get_marker_map(), *46*
brapi_images, 20
brapi_list, 21
brapi_lists, 22
brapi_lists(), *21*
brapi_location, 23
brapi_locations, 23
brapi_login, 24
brapi_login(), *8*
brapi_login_oauth2, 25
brapi_login_oauth2(), *8*, *25*
brapi_map, 26
brapi_map(), *19*, *28*
brapi_map_linkage_groups, 28
brapi_maps, 27
brapi_maps(), *20*
brapi_marker_positions, 28
brapi_marker_positions(), *19*, *46*
brapi_methods, 30
brapi_methods(), *34, 35*
brapi_observation_units, 32
brapi_observation_variables, 33
brapi_observation_variables(), *34, 35*
brapi_observations, 31
brapi_ontologies, 34
brapi_ontologies(), *30*, *33*, *35*, *44*, *57*
brapi_ontology, 35
brapi_ontology(), *30*, *33, 34*, *44*, *57*
brapi_pedigree, 36
brapi_pedigree(), *16, 17*, *49, 50*
brapi_people, 37
brapi_ping, 38
brapi_post_search, 39
brapi_post_search(), *18*
brapi_program, 40
brapi_programs, 41
brapi_reference_sets, 42
brapi_references, 42
brapi_samples, 43
brapi_scales, 44
brapi_scales(), *34, 35*
brapi_search_calls, 45
brapi_search_germplasm, 45
brapi_search_marker_positions, 46
brapi_search_marker_positions(), *20*, *29*
brapi_search_observations, 48
brapi_search_pedigree, 49
brapi_search_pedigree(), *36, 37*
brapi_search_variables, 50
brapi_search_variants, 51
brapi_seasons, 51
brapi_seed_lots, 52
brapi_server_info, 53
brapi_set_token, 54

brapi_set_token(), *25*
brapi_studies, 54
brapi_study, 55
brapi_study_data, 56
brapi_traits, 57
brapi_traits(), *34, 35*
brapi_trial, 58
brapi_trials, 58
brapi_variant_sets, 60
brapi_variants, 59
brapi_variants(), *19, 20*, *28*

future::plan(), *12*

print.brapi_con, 61

tidyr::unnest(), *37*