Package 'bioclients'

September 15, 2026

Title Clients for Biological Database Web Services

Version 0.1.1

Description Look up genes, variants and proteins from R, without writing a client for every biological web service. Each service gets one client that makes the request and returns a table. Parsing is a separate function that needs no network, so it can run on a saved response and be tested offline. Transport, retries, caching and error handling are left to the 'biohttp' package. Dependencies for single services are optional, so you do not install what you will not use. The services covered include 'Ensembl', described in Dyer et al. (2025) <doi:10.1093/nar/gkae1071>, 'UniProt', in The UniProt Consortium (2025) <doi:10.1093/nar/gkae1010>, 'gnomAD', in Chen et al. (2024) <doi:10.1038/s41586-023-06045-0>, 'Open Targets', in Buniello et al. (2025) <doi:10.1093/nar/gkae1128>, and the 'AlphaFold' Protein Structure Database, in Varadi et al. (2024) <doi:10.1093/nar/gkad1011>. Each client's help page cites the service it calls.

License MIT + file LICENSE

URL https://github.com/samuelbharti/bioclients, https://www.samuelbharti.com/bioclients/

BugReports https://github.com/samuelbharti/bioclients/issues

Encoding UTF-8

Depends R (>= 4.0)

Imports biohttp (>= 0.1.2), httr2, tibble

Suggests jsonlite, knitr, rmarkdown, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Samuel Bharti [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-4190-7058>), Barret Schloerke [ths] (ORCID: <https://orcid.org/0000-0001-9986-114X>), Carson Sievert [ths] (ORCID: <https://orcid.org/0000-0002-4958-2844>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Samuel Bharti <samuelbharti.io@gmail.com>

Repository CRAN

Date/Publication 2026-09-15 13:00:08 UTC

Contents

alphafold_model *The predicted structure for a UniProt accession*

Description

Metadata only. No coordinates are downloaded; use `pdb_url` from the result for that, and only when something is actually going to render it.

Usage

```
alphafold_model(accession, ...)
```

Arguments

`accession`	A UniProt accession, for example `"P15056"`.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a one-row tibble. See `alphafold_parse_model()`.

References

Varadi et al. (2024). AlphaFold Protein Structure Database in 2024: providing structure coverage for over 214 million protein sequences. Nucleic Acids Research 52(D1), D368-D375. doi:10.1093/nar/gkad1011

Service documentation: https://alphafold.ebi.ac.uk/

Examples

```
biohttp::body_or_null(alphafold_model("P15056"))
```

alphafold_parse_model *Turn an AlphaFold prediction response into a table*

Description

Pure.

Usage

```
alphafold_parse_model(body, accession = NA_character_)
```

Arguments

body	A parsed AlphaFold prediction response.
accession	The UniProt accession that was queried.

Value

A one-row tibble of accession, pdb_url, cif_url, version, mean_plddt, and source_url. NULL when there is no model.

The one-element array

The API returns a JSON **array** of model records, not an object, even for a single accession. Reading it as an object yields NULL for every field while looking like a successful parse. This handles either shape.

References

Varadi et al. (2024). AlphaFold Protein Structure Database in 2024: providing structure coverage for over 214 million protein sequences. Nucleic Acids Research 52(D1), D368-D375. doi:10.1093/nar/gkad1011

Service documentation: https://alphafold.ebi.ac.uk/

Examples

```
body <- list(list(
  pdbUrl = "https://alphafold.ebi.ac.uk/files/AF-P15056-F1-model_v6.pdb",
  latestVersion = 6, globalMetricValue = 66.38
))
alphafold_parse_model(body, "P15056")
```

---

civic_gene *Curated clinical evidence counts for a gene*

Description

Curated clinical evidence counts for a gene

Usage

```
civic_gene(symbol, ...)
```

Arguments

symbol	A HUGO gene symbol.
...	Passed to biohttp::post_json(), for example throttle.

Value

A biohttp envelope whose `data` is a one-row tibble. See `civic_parse_gene()`.

On the interpolated query

CIViC's gene query takes the symbol as an inline string rather than a GraphQL variable, so the symbol is substituted into the query text. It is run through this package's internal `clean_symbol()` first, which strips everything outside `[A-Za-z0-9._-]`. That is what stops a crafted symbol closing the string and appending its own query. Do not remove it, and do not switch to `paste0()` without it.

References

Griffith et al. (2017). CIViC is a community knowledgebase for expert crowdsourcing the clinical interpretation of variants in cancer. Nature Genetics 49(2), 170-174. doi:10.1038/ng.3774

Service documentation: https://civicdb.org/

Examples

```
biohttp::body_or_null(civic_gene("NF1"))
```

`civic_parse_gene` *Turn a CIViC gene response into a table*

Description

Pure.

Usage

```
civic_parse_gene(body)
```

Arguments

`body`	A parsed CIViC GraphQL response body.

Value

A one-row tibble of `id`, `symbol`, `entrez_id`, `evidence_items`, `assertions`, `variants`, and `source_url`. `NULL` when CIViC does not track the gene, which it reports as `data.gene = null`.

References

Griffith et al. (2017). CIViC is a community knowledgebase for expert crowdsourcing the clinical interpretation of variants in cancer. Nature Genetics 49(2), 170-174. doi:10.1038/ng.3774

Service documentation: https://civicdb.org/

Examples

```
body <- list(data = list(gene = list(
  id = 3867, name = "NF1", entrezId = 4763, link = "/features/3867",
  stats = list(evidenceItemCount = 56, assertionCount = 0, variantCount = 37)
)))
civic_parse_gene(body)
```

`clingen_alleles` *Resolve HGVS to canonical allele ids*

Description

One batched POST. The response comes back in input order.

Usage

```
clingen_alleles(hgvs, throttle = CLINGEN_THROTTLE, ...)
```

Arguments

`hgvs`	HGVS strings.
`throttle`	A throttle spec. Defaults to the documented 3 per second.
`...`	Passed to `biohttp::perform()` indirectly through the request.

Value

A biohttp envelope whose `data` is the tibble described in `clingen_parse_allele()`.

References

Pawliczek et al. (2018). ClinGen Allele Registry links information about genetic variants. Human Mutation 39(11), 1690-1701. doi:10.1002/humu.23637

Service documentation: https://reg.clinicalgenome.org/

Examples

```
biohttp::body_or_null(clingen_alleles("NM_000546.6:c.215C>G"))
```

clingen_gene_validity *The ClinGen gene-disease validity table*

Description

Fetches the whole published CSV. See the note in the file header on why this is one download rather than a request per gene, and use `clingen_validity_for()` to narrow the result.

Usage

```
clingen_gene_validity(...)
```

Arguments

`...`	Passed to `biohttp::get_text()`, for example `timeout`.

Value

A biohttp envelope whose `data` is the tibble described in `clingen_parse_validity()`.

References

Strande et al. (2017). Evaluating the clinical validity of gene-disease associations: an evidence-based framework developed by the Clinical Genome Resource. The American Journal of Human Genetics 100(6), 895-906. doi:10.1016/j.ajhg.2017.04.015

Service documentation: https://search.clinicalgenome.org/

Examples

```
table <- biohttp::body_or_null(clingen_gene_validity())
clingen_validity_for(table, c("NF1", "TP53"))
```

clingen_parse_allele *Turn one Allele Registry element into a table row*

Description

Pure.

Usage

```
clingen_parse_allele(element)
```

Arguments

element	One parsed Allele Registry element.

Value

A one-row tibble.

An unresolved input is a row, not a gap

ClinGen answers an input it could not resolve with an object carrying no @id, and an errorType/message instead. That becomes a row with resolved = FALSE and the reason, rather than being dropped. Dropping it would shift every later row onto the wrong variant, and would lose the reason the input was rejected.

References

Pawliczek et al. (2018). ClinGen Allele Registry links information about genetic variants. Human Mutation 39(11), 1690-1701. doi:10.1002/humu.23637

Service documentation: https://reg.clinicalgenome.org/

Examples

```
clingen_parse_allele(list(errorType = "IncorrectHgvsPosition"))
```

---

clingen_parse_batch *Turn an Allele Registry batch response into a table*

Description

Pure. One row per element, in the order returned, which is the order asked.

Usage

```
clingen_parse_batch(body)
```

Arguments

body	A parsed Allele Registry response, an array.

Value

A tibble with one row per element.

References

Pawliczek et al. (2018). ClinGen Allele Registry links information about genetic variants. Human Mutation 39(11), 1690-1701. doi:10.1002/humu.23637

Service documentation: https://reg.clinicalgenome.org/

`clingen_parse_validity` *Parse the ClinGen gene-validity CSV*

Description

Pure. Takes the file as a single string, the way `biohttp::get_text()` returns it.

Usage

```
clingen_parse_validity(text)
```

Arguments

`text`	The CSV file contents.

Value

A tibble of `gene`, `hgnc`, `disease`, `mondo`, `moi`, `classification`, `sop`, `panel`, `date`, and `source_url`, one row per curation. A gene appears once per disease it has been curated against. `NULL` when the file is empty or carries no header.

References

Strande et al. (2017). Evaluating the clinical validity of gene-disease associations: an evidence-based framework developed by the Clinical Genome Resource. The American Journal of Human Genetics 100(6), 895-906. doi:10.1016/j.ajhg.2017.04.015

Service documentation: https://search.clinicalgenome.org/

Examples

```
text <- paste(
  '"CLINGEN GENE DISEASE VALIDITY CURATIONS","",""',
  '"+++","+++","+++"',
  '"GENE SYMBOL","DISEASE LABEL","CLASSIFICATION"',
  '"+++","+++","+++"',
  '"NF1","neurofibromatosis type 1","Definitive"',
  sep = "\n"
)
clingen_parse_validity(text)
```

clingen_validity_for *Filter a parsed validity table to one or more genes*

Description

Pure. Matching is case-insensitive on the symbol.

Usage

```
clingen_validity_for(table, symbols)
```

Arguments

table	A tibble from clingen_parse_validity().
symbols	Gene symbols.

Value

The matching rows, or NULL when none match.

References

Strande et al. (2017). Evaluating the clinical validity of gene-disease associations: an evidence-based framework developed by the Clinical Genome Resource. The American Journal of Human Genetics 100(6), 895-906. doi:10.1016/j.ajhg.2017.04.015

Service documentation: https://search.clinicalgenome.org/

Examples

```
table <- tibble::tibble(
  gene = c("NF1", "TP53"),
  classification = c("Definitive", "Definitive")
)
clingen_validity_for(table, "nf1")
```

clinvar_category *Bucket a ClinVar significance string into a coarse category*

Description

Pure. For colouring and grouping, where the full free-text significance is too granular to plot.

Usage

```
clinvar_category(significance)
```

Arguments

`significance`	A ClinVar clinical significance string.

Details

The order of the checks matters. A conflicting record usually contains the word "pathogenic" too, so conflicting has to be tested first or every conflicting record reads as pathogenic.

Value

One of `"Conflicting"`, `"Pathogenic / likely"`, `"Benign / likely"`, `"Uncertain"`, or `"Other"`.

References

Landrum et al. (2018). ClinVar: improving access to variant interpretations and supporting evidence. Nucleic Acids Research 46(D1), D1062-D1067. doi:10.1093/nar/gkx1153

Service documentation: https://www.ncbi.nlm.nih.gov/clinvar/

Examples

```
clinvar_category("Pathogenic")
clinvar_category("Conflicting interpretations of pathogenicity")
```

`clinvar_classification`

Look up the ClinVar classification for a variant

Description

Resolves `term` to a ClinVar UID, then fetches that record. An rsID is the term that works best.

Usage

```
clinvar_classification(term, throttle = clinvar_throttle(), ...)
```

Arguments

`term`	A search term, usually an rsID or an accession.
`throttle`	A throttle spec, see `biohttp::req_defaults()`. Defaults to the documented E-utilities rate for the key in use.
`...`	Passed to `biohttp::get_json()`.

Value

A biohttp envelope whose `data` is a one-row tibble. See `clinvar_parse_record()`.

An rsID does not identify an allele

The same caveat as `gnomad_frequency()`. `7:g.140753336A>T` and `7:g.140753336A>C` share `rs113488022`, so a term that is only an rsID can resolve to a record for the other allele. ClinVar returns the first matching UID, and this function returns that record.

The NCBI API key

Set `NCBI_API_KEY` and both requests carry it, which raises the rate limit from 3 to 10 requests a second. It is passed as a `secret_query`, so it stays out of the cache key and out of every message. Without one the client works at the lower limit, and the default `throttle` follows: 3 a second without a key, 10 with one.

Identifying the caller

NCBI asks every client to send `tool` and `email`. They are read from `BIOHTTP_CALLER_IDENTITY` and `BIOHTTP_CONTACT_EMAIL`, the same variables biohttp builds its User-Agent from, and a blank one is omitted. They are ordinary query parameters, so they are part of the cache key.

References

Landrum et al. (2018). ClinVar: improving access to variant interpretations and supporting evidence. Nucleic Acids Research 46(D1), D1062-D1067. doi:10.1093/nar/gkx1153

Service documentation: `https://www.ncbi.nlm.nih.gov/clinvar/`

Examples

```
biohttp::body_or_null(clinvar_classification("rs113488022"))
```

`clinvar_conditions` *Collapse a ClinVar trait set into one condition string*

Description

Pure.

Usage

```
clinvar_conditions(germline)
```

Arguments

`germline`	The `germline_classification` block of an esummary record.

Value

A single semicolon-separated string, or `NA_character_`.

References

Landrum et al. (2018). ClinVar: improving access to variant interpretations and supporting evidence. Nucleic Acids Research 46(D1), D1062-D1067. doi:10.1093/nar/gkx1153

Service documentation: https://www.ncbi.nlm.nih.gov/clinvar/

Examples

```
clinvar_conditions(list(trait_set = list(list(trait_name = "RASopathy"))))
```

`clinvar_parse_record` *Turn a ClinVar esummary record into a table*

Description

Pure. Takes one record from the `result` block of an esummary response, not the whole response, because the record is keyed by UID and the caller already knows which UID it asked for.

Usage

```
clinvar_parse_record(record, uid = NA_character_)
```

Arguments

`record`	A parsed ClinVar esummary record.
`uid`	The ClinVar UID the record came from.

Value

A one-row tibble with `uid`, `accession`, `title`, `significance`, `review_status`, `last_evaluated`, and `conditions`. `NULL` when `record` is absent.

References

Landrum et al. (2018). ClinVar: improving access to variant interpretations and supporting evidence. Nucleic Acids Research 46(D1), D1062-D1067. doi:10.1093/nar/gkx1153

Service documentation: https://www.ncbi.nlm.nih.gov/clinvar/

Examples

```
record <- list(
  accession = "VCV000040389",
  title = "NM_004333.6(BRAF):c.1799T>G (p.Val600Gly)",
  germline_classification = list(
    description = "Pathogenic",
    review_status = "reviewed by expert panel"
  )
```

```
)
clinvar_parse_record(record, "40389")
```

`dgidb_gene` *Drug-gene interaction count for one gene*

Description

A thin wrapper over `dgidb_genes()`.

Usage

```
dgidb_gene(symbol, ...)
```

Arguments

`symbol`	A gene symbol.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a one-row tibble.

References

Cannon et al. (2024). DGIdb 5.0: rebuilding the drug-gene interaction database for precision medicine and drug discovery platforms. Nucleic Acids Research 52(D1), D1227-D1235. doi:10.1093/nar/gkad1040

Service documentation: https://dgidb.org/

Examples

```
biohttp::body_or_null(dgidb_gene("NF1"))
```

dgidb_genes *Drug-gene interaction counts for many genes*

Description

One request for the whole list, because DGIdb's genes query already takes an array.

Usage

```
dgidb_genes(symbols, ...)
```

Arguments

symbols	Gene symbols.
...	Passed to biohttp::post_json(), for example throttle.

Value

A biohttp envelope whose data is a tibble with one row per entry in symbols, in the same order. See dgidb_parse_genes() for the columns and for why an unknown gene is NA rather than 0.

References

Cannon et al. (2024). DGIdb 5.0: rebuilding the drug-gene interaction database for precision medicine and drug discovery platforms. Nucleic Acids Research 52(D1), D1227-D1235. doi:10.1093/nar/gkad1040

Service documentation: https://dgidb.org/

Examples

```
biohttp::body_or_null(dgidb_genes(c("NF1", "BRAF")))
```

dgidb_parse_genes *Turn a DGIdb genes response into a table*

Description

Pure. Rows come back in symbols order, one per input.

Usage

```
dgidb_parse_genes(body, symbols)
```

Arguments

body	A parsed DGIdb GraphQL response body.
symbols	The gene symbols that were queried, in the order asked.

Value

A tibble of symbol, concept_id, interaction_count, and source_url, one row per entry in symbols.

A real zero is not a miss

The distinction this parser exists to preserve. A gene DGIdb knows about with no recorded interactions has interaction_count = 0: that is an answer, and it means the gene is not currently druggable. A gene DGIdb has never heard of has interaction_count = NA: that is an absence of evidence.

Collapsing the two would tell a caller that an unknown gene is known not to be druggable, which is a different and much stronger claim than the data supports.

References

Cannon et al. (2024). DGIdb 5.0: rebuilding the drug-gene interaction database for precision medicine and drug discovery platforms. Nucleic Acids Research 52(D1), D1227-D1235. doi:10.1093/nar/gkad1040

Service documentation: https://dgidb.org/

Examples

```
body <- list(data = list(genes = list(nodes = list(
  list(name = "NF1", conceptId = "hgnc:7765", interactions = list(list(), list()))
))))
dgidb_parse_genes(body, "NF1")
```

---

diseases_channel *Genes DISEASES associates with a disease, from one channel*

Description

Genes DISEASES associates with a disease, from one channel

Usage

```
diseases_channel(
  doid,
  channel = c("Knowledge", "Textmining"),
  limit = 300,
  ...
)
```

Arguments

doid	A Disease Ontology id, for example "DOID:0060293".
channel	"Knowledge" for curated associations or "Textmining" for mined ones.
limit	Maximum genes to ask for.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in diseases_parse_channel().

References

Pletscher-Frankild et al. (2015). DISEASES: text mining and data integration of disease-gene associations. Methods 74, 83-89. doi:10.1016/j.ymeth.2014.11.020

Service documentation: https://diseases.jensenlab.org/

Examples

```
biohttp::body_or_null(diseases_channel("DOID:0060293", "Knowledge"))
```

---

diseases_gene_associations

Genes DISEASES associates with a disease, across both channels

---

Description

Queries Knowledge and Textmining together and keeps the strongest score per gene. Read the note on diseases_merge_channels() before relying on the combined score.

Usage

```
diseases_gene_associations(doid, limit = 300, ...)
```

Arguments

doid	A Disease Ontology id, for example "DOID:0060293".
limit	Maximum genes to ask for.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in diseases_merge_channels(), with source_url added.

A failed channel fails the call

If either channel errors, its envelope is returned rather than a combined score built from the other one. A result that silently dropped the curated channel would look like a weaker association rather than a partial answer, and the envelope has no status that says "half of this is missing". Call `diseases_channel()` per channel to handle the halves separately.

References

Pletscher-Frankild et al. (2015). DISEASES: text mining and data integration of disease-gene associations. Methods 74, 83-89. doi:10.1016/j.ymeth.2014.11.020

Service documentation: https://diseases.jensenlab.org/

Examples

```
biohttp::body_or_null(diseases_gene_associations("DOID:0060293"))
```

---

diseases_merge_channels

Combine DISEASES channels, keeping the strongest score per gene

---

Description

Pure.

Usage

```
diseases_merge_channels(tables)
```

Arguments

`tables`	A list of tibbles from `diseases_parse_channel()`. `NULL` entries are ignored.

Value

A tibble of `symbol`, `score`, and `channel`, ordered by score, highest first. `channel` names the channel the winning score came from. `NULL` when there is nothing to combine.

This is a combining convention, not the source's own answer

DISEASES scores each channel separately and does not publish a combined figure. Taking the maximum treats curated and mined evidence as interchangeable, which suits a screen looking for any support at all and suits nothing that cares where the support came from. Use `diseases_channel()` and keep the channels apart if that distinction matters.

References

Pletscher-Frankild et al. (2015). DISEASES: text mining and data integration of disease-gene associations. Methods 74, 83-89. doi:10.1016/j.ymeth.2014.11.020

Service documentation: https://diseases.jensenlab.org/

Examples

```
diseases_merge_channels(list(
  tibble::tibble(symbol = "NF1", protein = NA, score = 5, channel = "Knowledge"),
  tibble::tibble(symbol = "NF1", protein = NA, score = 2, channel = "Textmining")
))
```

`diseases_parse_channel` *Turn one DISEASES channel response into a table*

Description

Pure.

Usage

```
diseases_parse_channel(body, channel = NA_character_)
```

Arguments

`body`	A parsed response from one DISEASES channel.
`channel`	The channel the body came from, recorded in the result.

Value

A tibble of `symbol`, `protein`, `score`, and `channel`, one row per associated gene. `NULL` when the channel has no associations. Rows with no symbol or a non-finite score are dropped, because neither is usable.

References

Pletscher-Frankild et al. (2015). DISEASES: text mining and data integration of disease-gene associations. Methods 74, 83-89. doi:10.1016/j.ymeth.2014.11.020

Service documentation: https://diseases.jensenlab.org/

Examples

```
body <- list(
  list(ENSP00000351015 = list(name = "NF1", score = 5)),
  list()
)
diseases_parse_channel(body, "Knowledge")
```

`ensembl_gene_model` *The exon model for a gene*

Description

The exon model for a gene

Usage

```
ensembl_gene_model(gene_id, ...)
```

Arguments

`gene_id`	An Ensembl gene id, for example `"ENSG00000157764"`.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the list described in `ensembl_parse_gene_model()`.

References

Dyer et al. (2025). Ensembl 2025. Nucleic Acids Research 53(D1), D948-D957. doi:10.1093/nar/gkae1071

Service documentation: https://rest.ensembl.org/

Examples

```
biohttp::body_or_null(ensembl_gene_model("ENSG00000157764"))$exons
```

`ensembl_parse_consequences` *Turn Ensembl transcript consequences into a table*

Description

Pure.

Usage

```
ensembl_parse_consequences(consequences)
```

Arguments

`consequences`	The `transcript_consequences` array from a VEP record.

Details

Every consequence is returned, with its `biotype`, rather than only the protein-coding ones. Which biotypes are worth showing is the caller's call, and a filter here would hide the non-coding rows from a caller that wanted them.

Value

A tibble of `gene`, `gene_id`, `transcript`, `biotype`, `consequence`, `impact`, `sift`, and `polyphen`. `NULL` when there are none.

References

Dyer et al. (2025). Ensembl 2025. Nucleic Acids Research 53(D1), D948-D957. doi:10.1093/nar/gkae1071

Service documentation: https://rest.ensembl.org/

Examples

```
ensembl_parse_consequences(list(list(
  gene_symbol = "BRAF",
  transcript_id = "ENST00000646891",
  biotype = "protein_coding",
  consequence_terms = list("missense_variant"),
  impact = "MODERATE"
)))
```

`ensembl_parse_gene_model` *Turn an Ensembl gene lookup into a gene model*

Description

Pure.

Usage

```
ensembl_parse_gene_model(record)
```

Arguments

`record`	A parsed `lookup/id` response fetched with `expand=1`.

Value

A list of `transcript`, `strand`, `region`, `gene_start`, `gene_end`, and `exons`, a tibble of `start`, `end`, and `number` sorted by `start`. `NULL` when the record carries no transcript with exons.

Exon numbering follows the strand

Exons are sorted by genomic coordinate and then numbered in transcription order. On the minus strand that means the highest-coordinate exon is exon 1. Numbering by coordinate alone would put exon 1 at the wrong end of every minus-strand gene.

References

Dyer et al. (2025). Ensembl 2025. Nucleic Acids Research 53(D1), D948-D957. doi:10.1093/nar/gkae1071

Service documentation: https://rest.ensembl.org/

Examples

```
record <- list(
  seq_region_name = "17", start = 1, end = 900,
  Transcript = list(list(
    id = "ENST1", is_canonical = 1, strand = -1,
    Exon = list(list(start = 800, end = 900), list(start = 1, end = 100))
  ))
)
ensembl_parse_gene_model(record)$exons
```

ensembl_parse_vep *Turn an Ensembl VEP record into a result*

Description

Pure.

Usage

```
ensembl_parse_vep(record)
```

Arguments

record	One element of a VEP response.

Value

A list of most_severe, assembly, and consequences (the tibble from ensembl_parse_consequences()). NULL when the record is empty.

References

Dyer et al. (2025). Ensembl 2025. Nucleic Acids Research 53(D1), D948-D957. doi:10.1093/nar/gkae1071

Service documentation: https://rest.ensembl.org/

Examples

```
ensembl_parse_vep(list(
  most_severe_consequence = "missense_variant",
  assembly_name = "GRCh38"
))
```

ensembl_vep_id *Run VEP for a variant id*

Description

Run VEP for a variant id

Usage

```
ensembl_vep_id(rsid, ...)
```

Arguments

rsid	A variant id, for example "rs113488022".
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the list described in ensembl_parse_vep().

An rsID does not identify an allele

The same caveat as gnomad_frequency() and clinvar_classification(). 7:g.140753336A>T and 7:g.140753336A>C share rs113488022, so Ensembl may answer for more than one allele. The first record is used. Post coordinates with vep_variants() when the allele matters.

References

Dyer et al. (2025). Ensembl 2025. Nucleic Acids Research 53(D1), D948-D957. doi:10.1093/nar/gkae1071

Service documentation: https://rest.ensembl.org/

Examples

```
biohttp::body_or_null(ensembl_vep_id("rs113488022"))$consequences
```

---

europepmc_count *How many publications Europe PMC has for a query*

Description

Asks for one row with resultType = "idlist", because only the count is wanted. Read the note in the file header: a count of 0 comes back as ok, not as no_data.

Usage

```
europepmc_count(query, ...)
```

Arguments

query	A query string. Build one with europepmc_query() to get the quoting right.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a list of count, query, and source_url.

References

Ferguson et al. (2021). Europe PMC in 2020. Nucleic Acids Research 49(D1), D1507-D1514. doi:10.1093/nar/gkaa994

Service documentation: https://europepmc.org/

Examples

```
biohttp::body_or_null(europepmc_count(europepmc_query("NF1")))$count
```

`europepmc_parse_count` *Read the hit count off a Europe PMC response*

Description

Pure. `0` is a real count and comes back as `0L`; only an absent `hitCount` is `NA`.

Usage

```
europepmc_parse_count(body)
```

Arguments

`body`	A parsed Europe PMC `search` response.

Value

A single integer, or `NA_integer_`.

References

Ferguson et al. (2021). Europe PMC in 2020. Nucleic Acids Research 49(D1), D1507-D1514. doi:10.1093/nar/gkaa994

Service documentation: https://europepmc.org/

Examples

```
europepmc_parse_count(list(hitCount = 2530))
europepmc_parse_count(list(hitCount = 0))
```

`europepmc_parse_results` *Turn Europe PMC search results into a table*

Description

Pure.

Usage

```
europepmc_parse_results(body)
```

Arguments

body	A parsed Europe PMC `search` response.

Details

`source_id` is the citable identifier: `PMID:<n>` where there is a PMID, and `<source>:<id>` otherwise, because a preprint or a patent record has no PMID but is still groundable.

Value

A tibble of `title`, `authors`, `year`, `journal`, `pmid`, `doi`, `cited_by`, `source`, `source_id`, and `source_url`, newest first, which is the order Europe PMC returns. `NULL` when nothing matched.

References

Ferguson et al. (2021). Europe PMC in 2020. Nucleic Acids Research 49(D1), D1507-D1514. doi:10.1093/nar/gkaa994

Service documentation: https://europepmc.org/

Examples

```
body <- list(resultList = list(result = list(list(
  id = "36197410",
  source = "MED",
  pmid = "36197410",
  title = "TP53 or Not TP53",
  authorString = "Green SD.",
  journalTitle = "Clin Cancer Res",
  pubYear = "2022"
))))
europepmc_parse_results(body)
```

europepmc_query *Build a Europe PMC query from terms*

Description

Each term is quoted so a multi-word term matches as a phrase rather than as loose words, and the terms are ANDed. Quoting also stops a term containing a space or a colon from being read as query syntax.

Usage

```
europepmc_query(...)
```

Arguments

...	Terms, for example a gene symbol and an rsID. Blank terms are dropped.

Value

A single query string, or `NULL` when nothing usable was given.

References

Ferguson et al. (2021). Europe PMC in 2020. Nucleic Acids Research 49(D1), D1507-D1514. doi:10.1093/nar/gkaa994

Service documentation: https://europepmc.org/

Examples

```
europepmc_query("BRAF")
europepmc_query("BRAF", "V600E")
```

europepmc_search *Search Europe PMC*

Description

Search Europe PMC

Usage

```
europepmc_search(query, limit = 15, ...)
```

Arguments

query	A query string. Build one with europepmc_query() to get the quoting right.
limit	Maximum results to return.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a list of results (the tibble from europepmc_parse_results()), count (Europe PMC's total, not the page size), and query.

References

Ferguson et al. (2021). Europe PMC in 2020. Nucleic Acids Research 49(D1), D1507-D1514. doi:10.1093/nar/gkaa994

Service documentation: https://europepmc.org/

Examples

```
biohttp::body_or_null(europepmc_search(europepmc_query("BRAF", "V600E")))
```

---

gnomad_constraint *Gene constraint for one gene*

Description

How intolerant a gene is to variation. pli and loeuf summarize loss-of-function intolerance; the Z-scores summarize missense and synonymous depletion.

Usage

```
gnomad_constraint(symbol, reference_genome = "GRCh38", ...)
```

Arguments

symbol	A gene symbol.
reference_genome	"GRCh38" or "GRCh37".
...	Passed to biohttp::post_json(), for example throttle.

Value

A biohttp envelope whose data is a one-row tibble. See gnomad_parse_constraint().

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
biohttp::body_or_null(gnomad_constraint("BRAF"))
```

---

gnomad_constraints *Gene constraint for many genes*

Description

Batched through GraphQL aliases, so a gene list costs a handful of requests rather than one per gene. Chunked at `chunk_size` to stay under gnomAD's query cost cap of 25. See `GNOMAD_CHUNK`.

Usage

```
gnomad_constraints(
  symbols,
  reference_genome = "GRCh38",
  chunk_size = GNOMAD_CHUNK,
  ...
)
```

Arguments

`symbols`	Gene symbols.
`reference_genome`	`"GRCh38"` or `"GRCh37"`.
`chunk_size`	Genes per request.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a tibble with one row per entry in `symbols`, in the same order.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
biohttp::body_or_null(gnomad_constraints(c("BRAF", "TP53", "EGFR")))
```

`gnomad_frequencies` *Population allele frequency for many variants*

Description

Batched through GraphQL aliases and chunked at `chunk_size` to stay under gnomAD's query cost cap of 25, which was verified for this query. See `GNOMAD_CHUNK`. Dispatched through `biohttp::post_json_many()`, so only the chunks the cache is missing go over the wire.

Usage

```
gnomad_frequencies(
  variant_ids,
  dataset = GNOMAD_DATASET,
  reference_genome = "GRCh38",
  chunk_size = GNOMAD_CHUNK,
  ...
)
```

Arguments

`variant_ids`	gnomAD variant ids, see `gnomad_variant_id()`.
`dataset`	The gnomAD dataset. `gnomad_r4` is GRCh38; the `gnomad_r2` datasets are GRCh37.
`reference_genome`	The assembly the id is on. The variant query is keyed by dataset alone, so this is checked against `dataset` and a mismatch is refused rather than sent, because gnomAD would answer with whatever sits at those coordinates on the other assembly.
`chunk_size`	Variants per request.
`...`	Passed to `biohttp::post_json_many()`.

Details

A failed chunk yields a row of `NA` per variant rather than taking the whole call down, following `gnomad_constraints()`. A variant gnomAD has no record of is a row of `NA` too, because that is an answer.

Value

A biohttp envelope whose `data` is a tibble with one row per entry in `variant_ids`, in the same order. See `gnomad_parse_variant()`.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
biohttp::body_or_null(gnomad_frequencies(
  c("17-7676154-G-C", "7-117559590-ATCT-A")
))
```

---

gnomad_frequency *Population allele frequency for a variant*

Description

Looked up by rsID, which is what gnomAD's variant query takes.

Usage

```
gnomad_frequency(rsid, dataset = GNOMAD_DATASET, ...)
```

Arguments

`rsid`	A dbSNP rsID.
`dataset`	The gnomAD dataset.
`...`	Passed to `biohttp::post_json()`.

Value

A biohttp envelope whose `data` is the list described in `gnomad_parse_frequency()`.

An rsID does not identify an allele

`7:g.140753336A>T` and `7:g.140753336A>C` both map to `rs113488022`. Any lookup routed through an rsID is therefore lossy, and it fails **silently** while looking entirely plausible. The canonical key is `(assembly, chromosome, position, ref, alt)`.

This function is faithful to what gnomAD's variant query accepts, so the limitation is gnomAD's rather than this package's. Know about it before you rely on the answer for a multi-allelic site.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
biohttp::body_or_null(gnomad_frequency("rs113488022"))
```

`gnomad_frequency_by_id` *Population allele frequency for a variant, by id*

Description

Looked up by `chrom-pos-ref-alt`, which names one allele exactly. This is the lookup `gnomad_frequency()` cannot do, because an rsID is shared by every allele at a site.

Usage

```
gnomad_frequency_by_id(
  variant_id,
  dataset = GNOMAD_DATASET,
  reference_genome = "GRCh38",
  ...
)
```

Arguments

`variant_id`	A gnomAD variant id, see `gnomad_variant_id()`.
`dataset`	The gnomAD dataset. `gnomad_r4` is GRCh38; the `gnomad_r2` datasets are GRCh37.
`reference_genome`	The assembly the id is on. The variant query is keyed by dataset alone, so this is checked against `dataset` and a mismatch is refused rather than sent, because gnomAD would answer with whatever sits at those coordinates on the other assembly.
`...`	Passed to `biohttp::post_json()`.

Value

A biohttp envelope whose `data` is the one-row tibble described in `gnomad_parse_variant()`. `no_data` when gnomAD has no record of the variant.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
biohttp::body_or_null(gnomad_frequency_by_id("17-7676154-G-C"))
```

`gnomad_parse_constraint` *Turn a gnomAD constraint response into a table*

Description

Pure. Takes an already-parsed response body and never touches the network.

Usage

```
gnomad_parse_constraint(body, symbol = NA_character_)
```

Arguments

`body`	A parsed gnomAD GraphQL response body.
`symbol`	The gene symbol that was queried.

Details

`loeuf` is gnomAD's `oe_lof_upper`. The two names are the same number, and the field is called `oe_lof_upper` in the API but LOEUF everywhere else, including in the ranking models that consume it. Both names appear here so a reader of either can find it.

Value

A one-row tibble with `symbol`, `pli`, `loeuf`, `oe_lof`, `oe_mis`, `mis_z`, `syn_z`, and `lof_z`. `NULL` when the gene has no constraint block, which is common and is an answer rather than a fault.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
body <- list(data = list(gene = list(
  gnomad_constraint = list(pli = 1, oe_lof_upper = 0.23)
)))
gnomad_parse_constraint(body, "BRAF")
```

`gnomad_parse_constraints` *Turn an aliased gnomAD constraint response into a table*

Description

Pure. The batched query uses GraphQL aliases `g1`, `g2`, and so on, so the response is keyed by alias rather than by symbol. Rows are mapped back by alias index rather than by the echoed symbol, so the result stays aligned even for a gene gnomAD does not return a symbol for.

Usage

```
gnomad_parse_constraints(body, symbols)
```

Arguments

`body`	A parsed gnomAD GraphQL response body.
`symbols`	The gene symbols that were queried, in the order asked.

Value

A tibble with one row per entry in `symbols`, same order. A gene with no constraint block gets a row of `NA` rather than being dropped.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
body <- list(data = list(
  g1 = list(symbol = "BRAF", gnomad_constraint = list(oe_lof_upper = 0.23)),
  g2 = list(symbol = "TP53", gnomad_constraint = NULL)
))
gnomad_parse_constraints(body, c("BRAF", "TP53"))
```

gnomad_parse_frequency *Turn a gnomAD variant response into a frequency record*

Description

Pure.

Usage

```
gnomad_parse_frequency(body, dataset = GNOMAD_DATASET)
```

Arguments

body	A parsed gnomAD GraphQL response body.
dataset	The dataset that was queried, carried through to the result.

Value

A list of `variant_id`, `dataset`, `exome`, `genome`, and `populations`, or `NULL` when the body carries no variant.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
body <- list(data = list(variant = list(
  variant_id = "7-140753336-A-T",
  exome = list(af = 0.001, ac = 2, an = 2000)
)))
gnomad_parse_frequency(body)$variant_id
```

`gnomad_parse_populations` *Combine gnomAD per-ancestry counts into one frequency table*

Description

Pure. Sums the exome and genome sample sets per ancestry group.

Usage

```
gnomad_parse_populations(exome_pops, genome_pops)
```

Arguments

`exome_pops`, `genome_pops`
: The `populations` lists from each sample set.

Details

Sex-split ids such as `nfe_XX` and the bare `XX`/`XY` breakdowns are dropped, by keeping only the known ancestry codes, so the result is one row per ancestry rather than a mix of ancestries and sexes.

Value

A tibble of `pop`, `label`, `ac`, `an`, `af`, sorted by frequency descending. `NULL` when there is nothing to report.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
gnomad_parse_populations(
  list(list(id = "nfe", ac = 3, an = 1000)),
  list(list(id = "nfe", ac = 1, an = 500))
)
```

gnomad_parse_variant *Turn a gnomAD variant response into a frequency row*

Description

Pure. The flat, one-row form of a variant record, for a variant table that wants a frequency per row. `gnomad_parse_frequency()` is the nested form with the per-ancestry table.

Usage

```
gnomad_parse_variant(body, variant_id = NA_character_)
```

Arguments

body	A parsed gnomAD GraphQL response body.
variant_id	The id that was queried, carried through to the row.

Value

A one-row tibble of `variant_id`, `rsid`, `exome_af`, `exome_ac`, `exome_an`, `exome_nhomalt`, `genome_af`, `genome_ac`, `genome_an`, `genome_nhomalt`, `grpmax_af`, `grpmax_an`, `grpmax_id`, `faf95`, `faf95_pop`, and `filters`. `NULL` when the body carries no variant, which is how gnomAD answers for a variant it has never seen.

grpmax is derived

The API does not serve a group maximum. It is computed here from the per-group counts, exome and genome summed, leaving out the bottlenecked groups and the remaining bucket the same way gnomAD does. See `GNOMAD_GRPMAX_EXCLUDED`.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
body <- list(data = list(variant = list(
  variant_id = "17-7676154-G-C",
  rsids = list("rs1042522"),
  exome = list(af = 0.72, ac = 1046941, an = 1461558, homozygote_count = 380188)
)))
gnomad_parse_variant(body, "17-7676154-G-C")
```

gnomad_parse_variants *Turn an aliased gnomAD variant response into a table*

Description

Pure. The batched query uses aliases `v1`, `v2`, and so on, so rows are mapped back by alias index rather than by the echoed id. A variant gnomAD has not seen comes back as `null` under its alias, with a "Variant not found" entry in `errors`, and becomes a row of `NA`.

Usage

```
gnomad_parse_variants(body, variant_ids)
```

Arguments

`body`	A parsed gnomAD GraphQL response body.
`variant_ids`	The ids that were queried, in the order asked.

Value

A tibble with one row per entry in `variant_ids`, same order. See `gnomad_parse_variant()` for the columns.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
body <- list(data = list(
  v1 = list(variant_id = "17-7676154-G-C", exome = list(af = 0.72)),
  v2 = NULL
))
gnomad_parse_variants(body, c("17-7676154-G-C", "17-7676154-G-GTTTTT"))
```

gnomad_variant_id *Build a gnomAD variant id from variant components*

Description

Pure. gnomAD's variant query takes `chrom-pos-ref-alt`, with no `chr` prefix and the alleles in upper case. Vectorised over its arguments.

Usage

```
gnomad_variant_id(chrom, pos, ref, alt)
```

Arguments

`chrom`	A chromosome, with or without a `chr` prefix.
`pos`	A 1-based position.
`ref`, `alt`	Reference and alternate alleles.

Value

A character vector of ids such as `"1-55516888-G-GA"`.

References

Chen et al. (2024). A genomic mutational constraint map using variation in 76,156 human genomes. Nature 625(7993), 92-100. doi:10.1038/s41586023060450

Service documentation: https://gnomad.broadinstitute.org/

Examples

```
gnomad_variant_id("chr1", 55516888, "g", "ga")
```

gtex_gene_reference *Resolve a gene to GTEx's versioned GENCODE id*

Description

Resolve a gene to GTEx's versioned GENCODE id

Usage

```
gtex_gene_reference(gene, ...)
```

Arguments

gene	A gene symbol or an Ensembl gene id.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in gtex_parse_reference().

Why this call exists

GTEx expression queries need a **versioned** GENCODE id such as ENSG00000141510.16. MyGene and most other sources return the unversioned ENSG00000141510, and GTEx returns nothing for it. There is no way to derive the version, so it has to be looked up here first.

This is the reason a GTEx expression lookup costs two requests.

References

The GTEx Consortium (2020). The GTEx Consortium atlas of genetic regulatory effects across human tissues. Science 369(6509), 1318-1330. doi:10.1126/science.aaz1776

Service documentation: https://gtexportal.org/home/

Examples

```
biohttp::body_or_null(gtex_gene_reference("TP53"))$gencode_id
```

---

gtex_median_expression

Median expression across tissues

Description

Two requests: one to resolve the versioned GENCODE id, one for the expression. See gtex_gene_reference() for why the first is unavoidable.

Usage

```
gtex_median_expression(
  gene = NULL,
  gencode_id = NULL,
  dataset = GTEX_DATASET,
  ...
)
```

Arguments

gene	A gene symbol or an Ensembl gene id. Ignored when gencode_id is given.
gencode_id	A versioned GENCODE id, for example "ENSG00000141510.16".
dataset	The GTEx dataset. Must match the reference the id came from.
...	Passed to biohttp::get_json(), for example throttle.

Details

Pass gencode_id directly to skip the lookup when you already have a versioned id.

Value

A biohttp envelope whose data is the tibble described in gtex_parse_expression().

References

The GTEx Consortium (2020). The GTEx Consortium atlas of genetic regulatory effects across human tissues. Science 369(6509), 1318-1330. doi:10.1126/science.aaz1776

Service documentation: https://gtexportal.org/home/

Examples

```
biohttp::body_or_null(gtex_median_expression("TP53"))
```

---

gtex_parse_expression *Turn a GTEx median-expression response into a table*

Description

Pure. Tissue ids arrive underscore-separated (Adipose_Subcutaneous) and are returned both raw and as a readable label.

Usage

```
gtex_parse_expression(body)
```

Arguments

body	A parsed GTEx expression/medianGeneExpression response.

Value

A tibble of tissue_id, tissue, median_tpm, and gencode_id. Rows with no median are dropped. NULL when there is nothing usable.

References

The GTEx Consortium (2020). The GTEx Consortium atlas of genetic regulatory effects across human tissues. Science 369(6509), 1318-1330. doi:10.1126/science.aaz1776

Service documentation: https://gtexportal.org/home/

Examples

```
body <- list(data = list(list(
  median = 22.459, tissueSiteDetailId = "Adipose_Subcutaneous",
  gencodeId = "ENSG00000141510.16", unit = "TPM"
)))
gtex_parse_expression(body)
```

`gtex_parse_reference` *Turn a GTEx gene-reference response into a table*

Description

Pure.

Usage

```
gtex_parse_reference(body)
```

Arguments

body	A parsed GTEx `reference/gene` response.

Value

A tibble of `symbol`, `gencode_id`, `entrez`, `chromosome`, and `gene_type`. `NULL` when the gene is not in the reference.

References

The GTEx Consortium (2020). The GTEx Consortium atlas of genetic regulatory effects across human tissues. Science 369(6509), 1318-1330. doi:10.1126/science.aaz1776

Service documentation: https://gtexportal.org/home/

Examples

```
body <- list(data = list(list(
  geneSymbol = "TP53", gencodeId = "ENSG00000141510.16", entrezGeneId = 7157
)))
gtex_parse_reference(body)
```

hpa_gene *The Human Protein Atlas record for a gene*

Description

Keyed by Ensembl gene id, which is what HPA's per-gene JSON path takes. Resolve a symbol first with `mygene_gene()`.

Usage

```
hpa_gene(ensembl, ...)
```

Arguments

`ensembl`	An Ensembl gene id, for example `"ENSG00000141510"`.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a one-row tibble. See `hpa_parse_gene()`.

References

Uhlén et al. (2015). Tissue-based map of the human proteome. Science 347(6220), 1260419. doi:10.1126/science.1260419

Service documentation: https://www.proteinatlas.org/

Examples

```
biohttp::body_or_null(hpa_gene("ENSG00000141510"))
```

hpa_parse_gene *Turn an HPA gene record into a table*

Description

Pure.

Usage

```
hpa_parse_gene(body, ensembl = NA_character_)
```

Arguments

body	A parsed HPA gene record.
ensembl	The Ensembl gene id that was queried.

Details

protein_class and disease_involvement are list columns, because a gene carries any number of tags and flattening them to one string would make them unusable without re-splitting.

Value

A one-row tibble of symbol, ensembl, uniprot, protein_class, disease_involvement, and source_url. NULL when the record is empty.

References

Uhlén et al. (2015). Tissue-based map of the human proteome. Science 347(6220), 1260419. doi:10.1126/science.1260419

Service documentation: https://www.proteinatlas.org/

Examples

```
body <- list(
  Gene = "TP53",
  Ensembl = "ENSG00000141510",
  Uniprot = list("P04637"),
  `Protein class` = list("Cancer-related genes", "Transcription factors"),
  `Disease involvement` = list("Tumor suppressor")
)
hpa_parse_gene(body, "ENSG00000141510")
```

---

hpo_gene_annotation *HPO's annotation for a gene*

Description

HPO's annotation for a gene

Usage

```
hpo_gene_annotation(entrez, ...)
```

Arguments

entrez	An NCBI Gene id, for example 7157. See the note on the lookup key in the file header: nothing else works here.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose `data` is a list of two tibbles, `diseases` and `phenotypes`, either of which may be `NULL`. Two tables rather than one, because they are independent arrays in the response and joining them would invent a disease-to-phenotype mapping the response does not carry. It also carries `source_url`. See `hpo_parse_diseases()` and `hpo_parse_phenotypes()`.

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
biohttp::body_or_null(hpo_gene_annotation(7157))$diseases
```

`hpo_parse_diseases` *Turn an HPO gene annotation into a table of diseases*

Description

Pure.

Usage

```
hpo_parse_diseases(body)
```

Arguments

`body`	A parsed HPO `network/annotation` response.

Details

Every row is grounded by an OMIM, ORPHA, or DECIPHER id in `id`, and by a MONDO id in `mondo` where HPO has one.

Value

A tibble of `id`, `name`, `mondo`, and `description`, one row per associated disease. `NULL` when there are none.

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
body <- list(diseases = list(
  list(
    id = "OMIM:151623",
    name = "Li-Fraumeni syndrome",
    mondoId = "MONDO:0018875",
    description = NULL
  )
))
hpo_parse_diseases(body)
```

`hpo_parse_phenotypes` *Turn an HPO gene annotation into a table of phenotypes*

Description

Pure.

Usage

```
hpo_parse_phenotypes(body)
```

Arguments

`body`	A parsed HPO `network/annotation` response.

Details

These are the HPO terms observed in the diseases the gene is associated with, not a separate assertion about the gene.

Value

A tibble of `id` and `name`, one row per phenotype term. `NULL` when there are none.

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
body <- list(phenotypes = list(list(id = "HP:0003002", name = "Breast carcinoma")))
hpo_parse_phenotypes(body)
```

hpo_parse_search *Turn an HPO search response into a table*

Description

Pure.

Usage

```
hpo_parse_search(body)
```

Arguments

body	A parsed HPO hp/search response.

Value

A tibble of id, name, definition, and descendant_count, best match first, which is the order JAX returns. NULL when nothing matched.

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
body <- list(terms = list(list(id = "HP:0001250", name = "Seizure")))
hpo_parse_search(body)
```

hpo_parse_term *Turn an HPO term response into a table*

Description

Pure.

Usage

```
hpo_parse_term(body)
```

Arguments

body	A parsed HPO hp/terms/<id> response.

Value

A one-row tibble of `id`, `name`, `definition`, `comment`, `descendant_count`, and the list columns `synonyms` and `xrefs`. `NULL` when the body carries no term.

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
body <- list(id = "HP:0001250", name = "Seizure", synonyms = list("Seizures"))
hpo_parse_term(body)
```

`hpo_search` *Search HPO terms by free text*

Description

Search HPO terms by free text

Usage

```
hpo_search(query, limit = 10, ...)
```

Arguments

`query`	Free text, for example `"seizure"`.
`limit`	Maximum terms to return.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `hpo_parse_search()`.

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
biohttp::body_or_null(hpo_search("seizure"))
```

hpo_term *Resolve one HP id to its term*

Description

Resolve one HP id to its term

Usage

```
hpo_term(id, ...)
```

Arguments

id	An HP id, for example "HP:0001250".
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the one-row tibble described in hpo_parse_term().

References

Gargano et al. (2024). The Human Phenotype Ontology in 2024: phenotypes around the world. Nucleic Acids Research 52(D1), D1333-D1346. doi:10.1093/nar/gkad1005

Service documentation: https://hpo.jax.org/

Examples

```
biohttp::body_or_null(hpo_term("HP:0001250"))
```

impc_gene_phenotypes *Significant knockout phenotypes IMPC records for a human gene*

Description

Resolves the mouse ortholog first, then queries the phenotype core by its MGI accession. See the note in the file header on why the second query cannot be made from the human symbol.

Usage

```
impc_gene_phenotypes(symbol, ...)
```

Arguments

symbol	A human gene symbol, for example "NF1".
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a list of mgi, marker_symbol, phenotypes (the tibble from impc_parse_phenotypes()), and source_url.

References

Groza et al. (2023). The International Mouse Phenotyping Consortium: comprehensive knockout phenotyping underpinning the study of human disease. Nucleic Acids Research 51(D1), D1038-D1045. doi:10.1093/nar/gkac972

Service documentation: https://www.mousephenotype.org/

Examples

```
biohttp::body_or_null(impc_gene_phenotypes("NF1"))$phenotypes
```

---

impc_mouse_ortholog *The mouse ortholog IMPC holds for a human gene*

Description

The mouse ortholog IMPC holds for a human gene

Usage

```
impc_mouse_ortholog(symbol, ...)
```

Arguments

symbol	A human gene symbol, for example "NF1".
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the one-row tibble described in impc_parse_ortholog().

References

Groza et al. (2023). The International Mouse Phenotyping Consortium: comprehensive knockout phenotyping underpinning the study of human disease. Nucleic Acids Research 51(D1), D1038-D1045. doi:10.1093/nar/gkac972

Service documentation: https://www.mousephenotype.org/

Examples

```
biohttp::body_or_null(impc_mouse_ortholog("NF1"))
```

`impc_parse_ortholog` *Read the mouse ortholog out of an IMPC gene-core response*

Description

Pure.

Usage

```
impc_parse_ortholog(body)
```

Arguments

`body`	A parsed IMPC `gene/select` response.

Value

A one-row tibble of `mgi` and `marker_symbol`. `NULL` when the gene has no IMPC mouse ortholog, which includes a document carrying no usable accession.

References

Groza et al. (2023). The International Mouse Phenotyping Consortium: comprehensive knockout phenotyping underpinning the study of human disease. Nucleic Acids Research 51(D1), D1038-D1045. doi:10.1093/nar/gkac972

Service documentation: https://www.mousephenotype.org/

Examples

```
body <- list(response = list(docs = list(
  list(mgi_accession_id = "MGI:97306", marker_symbol = "Nf1")
)))
impc_parse_ortholog(body)
```

`impc_parse_phenotypes` *Turn an IMPC phenotype response into a table*

Description

Pure.

Usage

```
impc_parse_phenotypes(body, mgi = NA_character_)
```

Arguments

`body`	A parsed IMPC `genotype-phenotype/select` response.
`mgi`	The MGI marker accession that was queried.

Value

A tibble of `mp_id`, `mp_name`, `allele`, `allele_symbol`, `zygosity`, and `source_url`, one row per distinct phenotype term. `NULL` when there are none.

One row per term, not per observation

IMPC reports a document per phenotype, sex, zygosity, and parameter combination, so the same term recurs many times over. Rows are collapsed to distinct `mp_id`, keeping the first occurrence. Counting the raw documents would report a gene's phenotype breadth several times over.

`mp_id` is an MP term for most phenotypes and an MPATH term for pathology findings. Both appear in the same field.

References

Groza et al. (2023). The International Mouse Phenotyping Consortium: comprehensive knockout phenotyping underpinning the study of human disease. Nucleic Acids Research 51(D1), D1038-D1045. doi:10.1093/nar/gkac972

Service documentation: https://www.mousephenotype.org/

Examples

```
body <- list(response = list(docs = list(
  list(
    mp_term_id = "MP:0011100",
    mp_term_name = "preweaning lethality, complete penetrance",
    allele_accession_id = "MGI:4364806",
    zygosity = "homozygote"
  )
)))
impc_parse_phenotypes(body, "MGI:97306")
```

`monarch_associations` *Associations with an entity on one end*

Description

Associations with an entity on one end

Usage

```
monarch_associations(entity, end = c("subject", "object"), limit = 200, ...)
```

Arguments

`entity`	An entity CURIE, for example `"MONDO:0007947"`.
`end`	`"subject"` or `"object"`.
`limit`	Maximum associations to return.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `monarch_parse_associations()`.

One direction per call

`end` picks which end of the association the entity sits on. Both directions are two calls, and concatenating them returns any association that has the entity on both ends **twice**. Collapsing those, and deciding what to do with the publications on each copy, is the caller's model rather than Monarch's answer, so this function does neither.

References

Putman et al. (2024). The Monarch Initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. Nucleic Acids Research 52(D1), D938-D949. doi:10.1093/nar/gkad1082

Service documentation: https://monarchinitiative.org/

Examples

```
biohttp::body_or_null(monarch_associations("MONDO:0007947"))
```

monarch_gene_phenotypes *HPO phenotypes Monarch associates with a gene*

Description

Keyed on the HGNC id, which mygene_gene() returns as hgnc. The rows come back in the shape monarch_parse_associations() describes, so the phenotype term is object and its label is object_label.

Usage

```
monarch_gene_phenotypes(hgnc, limit = 50, ...)
```

Arguments

hgnc	An HGNC id, bare or prefixed. See monarch_hgnc_id().
limit	Maximum associations to return.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a list of associations (the tibble) and total.

References

Putman et al. (2024). The Monarch Initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. Nucleic Acids Research 52(D1), D938-D949. doi:10.1093/nar/gkad1082

Service documentation: https://monarchinitiative.org/

Examples

```
biohttp::body_or_null(monarch_gene_phenotypes("11998"))$associations
```

monarch_hgnc_id *Normalise an HGNC id to the CURIE form Monarch expects*

Description

Monarch's entity route takes `HGNC:11998`. MyGene returns the bare digits. Exported because a caller assembling its own entity id needs the same rule.

Usage

```
monarch_hgnc_id(hgnc)
```

Arguments

hgnc	An HGNC id, bare (`"11998"`) or prefixed (`"HGNC:11998"`).

Value

A single string, or `NULL` when there is nothing usable.

References

Putman et al. (2024). The Monarch Initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. Nucleic Acids Research 52(D1), D938-D949. doi:10.1093/nar/gkad1082

Service documentation: https://monarchinitiative.org/

Examples

```
monarch_hgnc_id("11998")
monarch_hgnc_id("hgnc:11998")
```

monarch_parse_associations *Turn Monarch association records into a table*

Description

Pure.

Usage

```
monarch_parse_associations(body)
```

Arguments

body	A parsed Monarch association response, or the `items` list from one.

Details

One row per association, exactly as Monarch sent it. Nothing is dropped and nothing is merged. `publications` is a list column because an association carries any number of them, and `null` and an array both occur.

Value

A tibble of `subject`, `subject_label`, `subject_category`, `predicate`, `object`, `object_label`, `object_category`, `primary_knowledge_source`, `knowledge_level`, and `publications`. `NULL` when there are no associations.

References

Putman et al. (2024). The Monarch Initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. Nucleic Acids Research 52(D1), D938-D949. doi:10.1093/nar/gkad1082

Service documentation: https://monarchinitiative.org/

Examples

```
body <- list(items = list(list(
  subject = "MONDO:0017309",
  subject_label = "neonatal Marfan syndrome",
  predicate = "biolink:has_phenotype",
  object = "HP:0001653",
  object_label = "Mitral regurgitation",
  publications = NULL
)))
monarch_parse_associations(body)
```

---

monarch_parse_search *Turn a Monarch search response into a table*

Description

Pure.

Usage

```
monarch_parse_search(body)
```

Arguments

body	A parsed Monarch `search` response.

Value

A tibble of `id`, `name`, `category`, `description`, `taxon`, and `source_url`, one row per match. `NULL` when there are none.

Read the taxon

A gene symbol is not unique across species in Monarch, and a search returns the orthologs alongside the human gene under the same name. `taxon` is the only column that tells them apart. It is `NA` for a disease or a phenotype, which are not species-scoped.

References

Putman et al. (2024). The Monarch Initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. Nucleic Acids Research 52(D1), D938-D949. doi:10.1093/nar/gkad1082

Service documentation: https://monarchinitiative.org/

Examples

```
body <- list(items = list(list(
  id = "HGNC:3603",
  name = "FBN1",
  category = "biolink:Gene",
  full_name = "fibrillin 1",
  in_taxon_label = "Homo sapiens"
)))
monarch_parse_search(body)
```

---

monarch_search *Search Monarch for an entity*

Description

Turns typed text into the entity id the other calls need.

Usage

```
monarch_search(text, limit = 10, ...)
```

Arguments

`text`	Free text, for example a gene symbol or a disease name.
`limit`	Maximum matches to return.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a list of `matches` (the tibble from `monarch_parse_search()`) and `total`. `total` is Monarch's own count, not the page size, so a caller can say how much was left behind rather than presenting the first few as everything.

References

Putman et al. (2024). The Monarch Initiative in 2024: an analytic platform integrating phenotypes, genes and diseases across species. Nucleic Acids Research 52(D1), D938-D949. doi:10.1093/nar/gkad1082

Service documentation: https://monarchinitiative.org/

Examples

```
biohttp::body_or_null(monarch_search("Marfan syndrome"))$matches
```

`mygene_gene` *Look up one gene*

Description

Look up one gene

Usage

```
mygene_gene(symbol, species = "human", ...)
```

Arguments

`symbol`	A gene symbol, Ensembl gene id, or Entrez id.
`species`	Passed through to MyGene.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a one-row tibble. See `mygene_parse_hits()` for the columns.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://mygene.info/

Examples

```
res <- mygene_gene("TP53")
biohttp::body_or_null(res)
```

`mygene_genes` *Look up many genes in one request*

Description

The batch POST, which is the reason this client is worth installing. An N-symbol list is one round trip rather than N, and the round trip is where essentially all the time goes.

Usage

```
mygene_genes(symbols, species = "human", chunk_size = MYGENE_BATCH, ...)
```

Arguments

`symbols`	Gene symbols, Ensembl gene ids, or Entrez ids.
`species`	Passed through to MyGene.
`chunk_size`	Identifiers per request, at most `MYGENE_BATCH`.
`...`	Passed to `biohttp::post_json_many()`.

Details

MyGene takes at most 1000 identifiers per POST, see `MYGENE_BATCH`. A longer list is chunked, the chunks are dispatched through `biohttp::post_json_many()`, and the hits are merged back onto `symbols` in input order. A chunk that failed yields a row of `NA` per identifier rather than taking the whole call down, following `gnomad_constraints()`; only when every chunk failed is the failing envelope returned.

Value

A biohttp envelope whose `data` is a tibble with one row per entry in `symbols`, in the same order.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: `https://mygene.info/`

Examples

```
res <- mygene_genes(c("TP53", "BRCA1", "EGFR"))
biohttp::body_or_null(res)
```

`mygene_parse_batch` *Turn a MyGene batch response into a gene table*

Description

Pure. The batch `/query` POST returns a flat array where each element echoes its input `query`. An ambiguous query yields several elements, best `_score` first, and an unmatched one yields an element with `notfound = true`.

Usage

```
mygene_parse_batch(body, symbols)
```

Arguments

`body`	A parsed MyGene batch response, a flat list of hit records.
`symbols`	The identifiers that were queried, in the order asked.

Details

Hits are grouped by the echoed query and resolved through `mygene_pick_hit()`, then mapped back onto `symbols` **in input order**, so a caller zips the result onto its input by position. An unmatched or invalid token yields a row of `NA` rather than being dropped, because a shorter table would silently shift every row after it.

Value

A tibble with one row per entry in `symbols`, same order.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://mygene.info/

Examples

```
body <- list(
  list(query = "TP53", symbol = "TP53", entrezgene = "7157"),
  list(query = "NOPE", notfound = TRUE)
)
mygene_parse_batch(body, c("TP53", "NOPE"))
```

`mygene_parse_hits` *Turn MyGene hits into a gene table*

Description

Pure. Takes an already-parsed response body and never touches the network, so it is tested directly against a stored response.

Usage

```
mygene_parse_hits(body, symbol = NA_character_)
```

Arguments

`body`	A parsed MyGene `/query` response, the whole body including `hits`.
`symbol`	The identifier that was queried, used to break the scoring tie described in `mygene_pick_hit()` and as the fallback symbol.

Value

A one-row tibble with `symbol`, `name`, `summary`, `entrez`, `ensembl_gene`, `uniprot`, `hgnc`, and `type_of_gene`. `NULL` when the body carries no usable hit.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://mygene.info/

Examples

```
body <- list(hits = list(list(
  symbol = "TP53",
  name = "tumor protein p53",
  entrezgene = "7157"
)))
mygene_parse_hits(body, "TP53")
```

`mygene_pick_hit` *Choose the best MyGene hit for a queried token*

Description

MyGene's `_score` can rank a fuzzy alias or retired match to a **different** gene above the exact symbol match. Querying `"TTN"` with the alias and retired scopes returns TTR (entrez 7276, score 19.07) ahead of TTN (7273, score 18.29). Taking the top-scored hit therefore returns the wrong gene, quietly.

Usage

```
mygene_pick_hit(hits, token)
```

Arguments

`hits`	A list of hit records from a MyGene response.
`token`	The identifier that was queried.

Details

So a hit whose official symbol equals the token wins, case-insensitively. Only when none does, which is the case for a deliberate alias such as `"p53"`, does MyGene's own best-scored hit get used.

This is the single most important thing in this file. It is exported so a caller assembling its own hits can apply the same rule.

Value

One hit record, or `NULL` when `hits` is empty.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://mygene.info/

Examples

```
hits <- list(
  list(symbol = "TTR", entrezgene = "7276"),
  list(symbol = "TTN", entrezgene = "7273")
)
mygene_pick_hit(hits, "TTN")$entrezgene
```

`myvariant_id` *Build a MyVariant identifier from variant components*

Description

Pure. MyVariant indexes variants by an HGVS `g.` string, so a caller holding chromosome, position, ref and alt needs this to ask about one.

Usage

```
myvariant_id(chrom, pos, ref, alt)
```

Arguments

`chrom`	A chromosome, with or without a `chr` prefix.
`pos`	A 1-based position.
`ref`, `alt`	Reference and alternate alleles.

Value

A single string.

This is not variant identity

This formats a variant the way one service wants it written. It is not a canonical variant key and it does not normalize anything. The canonical key is `(assembly, chromosome, position, ref, alt)`, and turning arbitrary input into one is `vcfcanon`'s job, not this package's.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: `https://myvariant.info/`

Examples

```
myvariant_id("17", 7676154, "G", "C")
myvariant_id("chr1", 100, "AT", "A")
```

`myvariant_parse_batch` *Turn a MyVariant batch response into a table*

Description

Pure. One row per requested id, in the order asked.

Usage

```
myvariant_parse_batch(body, ids)
```

Arguments

`body`	A parsed MyVariant batch response, a flat array.
`ids`	The MyVariant ids that were requested, in order.

Value

A tibble with one row per entry in `ids`, plus an `id` column.

notfound is an answer, not an error

MyVariant reports an unknown variant as HTTP 200 with `notfound: true`. That is the source saying it has nothing, which is different from the call failing. Such rows come back as `NA` rather than being dropped, so a caller zipping by position stays aligned.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://myvariant.info/

Examples

```
body <- list(list(query = "chr17:g.7676154G>C", dbnsfp = list(
  genename = list("TP53"), cadd = list(raw_rankscore = 0.17)
)))
myvariant_parse_batch(body, "chr17:g.7676154G>C")
```

`myvariant_parse_record` *Turn one MyVariant record into a table row*

Description

Pure.

Usage

```
myvariant_parse_record(record)
```

Arguments

`record`	A parsed MyVariant record.

Value

A one-row tibble of `gene`, `revel`, `cadd`, `clinpred`, `alphamissense`, and `clinvar_sig`.

CADD's key is not what you would guess

Every other dbNSFP predictor exposes `<name>.rankscore`. CADD does not: it is `cadd.raw_rankscore`, and there is no plain `cadd.rankscore`. Following the pattern gives `NA` for CADD on every variant with nothing to indicate it.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://myvariant.info/

Examples

```
record <- list(dbnsfp = list(
  genename = list("TP53"),
  cadd = list(raw_rankscore = 0.17018),
  revel = list(rankscore = 0.4)
))
myvariant_parse_record(record)
```

`myvariant_variants` *Annotate many variants in one request*

Description

Annotate many variants in one request

Usage

```
myvariant_variants(ids, assembly = "hg38", fields = MYVARIANT_FIELDS, ...)
```

Arguments

`ids`	MyVariant ids. Build them with `myvariant_id()`.
`assembly`	The genome assembly. Changing this from `"hg38"` only makes sense for genuinely GRCh37 coordinates.
`fields`	The MyVariant fields to request.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a tibble with one row per entry in `ids`, in the same order. See `myvariant_parse_batch()`.

assembly is not optional

`assembly = "hg38"` is always sent. Without it MyVariant answers 200 with `notfound` for every GRCh38 variant, so a whole cohort disappears with no error anywhere. There is no way to omit it through this function, and that is deliberate.

References

Xin et al. (2016). High-performance web services for querying gene and variant annotation. Genome Biology 17, 91. doi:10.1186/s1305901609539

Service documentation: https://myvariant.info/

Examples

```
ids <- myvariant_id("17", 7676154, "G", "C")
biohttp::body_or_null(myvariant_variants(ids))
```

opentargets_disease_targets *Genes associated with a disease*

Description

The discovery direction. `disease_id` is an Open Targets id in either separator form; `MONDO:0018975` and `MONDO_0018975` both work.

Usage

```
opentargets_disease_targets(disease_id, size = 1000, ...)
```

Arguments

`disease_id`	An EFO, MONDO, HP, or Orphanet id.
`size`	How many associations to ask for.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `opentargets_parse_targets()`.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: `https://platform.opentargets.org/`

Examples

```
biohttp::body_or_null(opentargets_disease_targets("MONDO_0018975"))
```

opentargets_drugs *Known drugs and clinical candidates for a gene*

Description

Takes an Ensembl gene id, the same as `opentargets_gene_diseases()`.

Usage

```
opentargets_drugs(ensembl_id, ...)
```

Arguments

ensembl_id	An Ensembl gene id, for example "ENSG00000141510".
...	Passed to biohttp::post_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in opentargets_parse_drugs().

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
biohttp::body_or_null(opentargets_drugs("ENSG00000157764"))
```

---

opentargets_gene_diseases

Diseases associated with a gene

Description

Takes an **Ensembl gene id**, not a symbol, because that is what the Open Targets target query accepts. Resolve a symbol first with mygene_gene() and read its ensembl_gene column.

Usage

```
opentargets_gene_diseases(ensembl_id, size = 20, ...)
```

Arguments

ensembl_id	An Ensembl gene id, for example "ENSG00000141510".
size	How many associations to ask for.
...	Passed to biohttp::post_json(), for example throttle.

Details

Keeping the resolution out of this function is deliberate. A client that silently made a second call to a different service would make one failure look like the other's, and a MyGene outage would read as an Open Targets outage.

Value

A biohttp envelope whose `data` is the tibble described in `opentargets_parse_diseases()`.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
biohttp::body_or_null(opentargets_gene_diseases("ENSG00000141510"))
```

---

opentargets_is_id *Is a term an ontology id rather than free text*

Description

Open Targets takes an exact id for a direct lookup and free text for a search, and the two are different queries. This is what tells them apart.

Usage

```
opentargets_is_id(term)
```

Arguments

`term`	A disease term.

Value

A single logical.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
opentargets_is_id("MONDO:0018975")
opentargets_is_id("neurofibromatosis type 1")
```

`opentargets_parse_diseases` *Turn target-to-disease rows into a table*

Description

Pure. Rows arrive already sorted by score, best first, and that order is kept.

Usage

```
opentargets_parse_diseases(body, ensembl_id = NA_character_)
```

Arguments

`body`	A parsed Open Targets GraphQL response body.
`ensembl_id`	The Ensembl gene id that was queried, used to build `source_url`.

Value

A tibble of `disease`, `disease_id`, `score`, and `source_url`, or `NULL` when the target is absent or has no associations.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
body <- list(data = list(target = list(
  approvedSymbol = "TP53",
  associatedDiseases = list(count = 1, rows = list(
    list(score = 0.88, disease = list(id = "MONDO_0018875", name = "LFS"))
  ))
)))
opentargets_parse_diseases(body, "ENSG00000141510")
```

`opentargets_parse_drugs` *Turn known-drug rows into a table*

Description

Pure.

Usage

```
opentargets_parse_drugs(body)
```

Arguments

`body`	A parsed Open Targets GraphQL response body.

Value

A tibble of `drug`, `drug_id`, `drug_type`, `max_phase`, `diseases`, `disease_ids`, and `source_url`. `NULL` when the target has no known drugs.

Every disease, not just the first

A drug row carries the whole list of diseases it has been tried against, and the first entry is often the least useful one. In the stored BRAF response, BELVARAFENIB lists five, and the first has no mapped `disease` at all. So the diseases arrive as list columns rather than as one picked value.

`diseases` prefers the mapped disease name and falls back to `diseaseFromSource`, which is the label the trial registry used. `disease_ids` is `NA` in the positions Open Targets could not map.

`max_phase` is the raw `maxClinicalStage`, for example `"PHASE_2"`. Turning that into "Phase 2" is presentation and belongs to whatever is presenting it.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
body <- list(data = list(target = list(
  drugAndClinicalCandidates = list(count = 1, rows = list(
    list(
      maxClinicalStage = "PHASE_2",
      drug = list(id = "CHEMBL1", name = "DRUGX", drugType = "Small molecule"),
      diseases = list(list(
        diseaseFromSource = "melanoma",
        disease = list(id = "MONDO_0005105", name = "melanoma")
```

```
      ))
    )
  ))
)))
opentargets_parse_drugs(body)
```

`opentargets_parse_matches` *Turn a disease search or lookup into a table*

Description

Pure, and it reads **both** response shapes on purpose: the `search` query returns a `hits` array, the direct `disease` lookup returns a single record. A caller resolving a term does not know in advance which query ran, so one parser reads either and the difference stops mattering downstream.

Usage

```
opentargets_parse_matches(body)
```

Arguments

body	A parsed Open Targets GraphQL response body.

Value

A tibble of `id`, `name`, `score`, `description`, and `source_url`, best match first, or `NULL` when nothing matched. `score` is `NA` for a direct lookup, which has no relevance score to report.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
hits <- list(data = list(search = list(hits = list(
  list(id = "MONDO_0018975", name = "neurofibromatosis type 1", score = 24.9)
))))
opentargets_parse_matches(hits)

single <- list(data = list(disease = list(id = "EFO_0000508", name = "neurofibroma")))
opentargets_parse_matches(single)
```

`opentargets_parse_pgx` *Turn pharmacogenomics rows into a table*

Description

Pure.

Usage

```
opentargets_parse_pgx(body)
```

Arguments

body	A parsed Open Targets GraphQL response body.

Details

`drugs` is a list column because one annotation can name several. `rsid` is `NA` where the annotation is keyed on a genotype rather than a variant, which is common.

Value

A tibble of `rsid`, `genotype_id`, `drugs`, `phenotype`, `annotation`, and `evidence_level`. `NULL` when the target has none, which is the normal case for most genes.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
body <- list(data = list(target = list(pharmacogenomics = list(
  list(
    variantRsId = "rs4244285",
    drugs = list(list(drugFromSource = "venlafaxine")),
    phenotypeText = "decreased metabolism of venlafaxine",
    evidenceLevel = "3"
  )
))))
opentargets_parse_pgx(body)
```

`opentargets_parse_targets` *Turn disease-to-target rows into a table*

Description

Pure. The discovery direction: the genes Open Targets associates with a disease, each with its overall association score.

Usage

```
opentargets_parse_targets(body, disease_id = NA_character_)
```

Arguments

`body`	A parsed Open Targets GraphQL response body.
`disease_id`	The disease id that was queried, used to build `source_url`.

Value

A tibble of `symbol`, `ensembl_id`, `score`, and `source_url`, or `NULL` when the disease is absent or has no associations.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
body <- list(data = list(disease = list(
  name = "NF1",
  associatedTargets = list(count = 1, rows = list(
    list(score = 0.9, target = list(id = "ENSG00000196712", approvedSymbol = "NF1"))
  ))
)))
opentargets_parse_targets(body, "MONDO_0018975")
```

`opentargets_pgx` *Pharmacogenomics annotations for a gene*

Description

Takes an Ensembl gene id, the same as `opentargets_gene_diseases()`.

Usage

```
opentargets_pgx(ensembl_id, ...)
```

Arguments

`ensembl_id`	An Ensembl gene id, for example `"ENSG00000141510"`.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Details

Most genes have none, so `no_data` here is the ordinary answer rather than a sign anything went wrong.

Value

A biohttp envelope whose `data` is the tibble described in `opentargets_parse_pgx()`.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: `https://platform.opentargets.org/`

Examples

```
biohttp::body_or_null(opentargets_pgx("ENSG00000165841"))
```

`opentargets_resolve_disease` *Resolve a disease term to ontology records*

Description

Free text is searched; something that already looks like an ontology id is looked up directly, which is both exact and cheaper. `opentargets_is_id()` is what decides.

Usage

```
opentargets_resolve_disease(term, limit = 5, ...)
```

Arguments

`term`	Free text, or an EFO/MONDO/HP/Orphanet id.
`limit`	How many candidates to return for a free-text search.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Details

This is the `resolve_disease` that exists twice in the family, once in `genescout/R/tools/disease_resolver.R` and once in a sibling app.

Value

A biohttp envelope whose `data` is the tibble described in `opentargets_parse_matches()`, best match first.

References

Buniello et al. (2025). Open Targets Platform: facilitating therapeutic hypotheses building in drug discovery. Nucleic Acids Research 53(D1), D1467-D1475. doi:10.1093/nar/gkae1128

Service documentation: https://platform.opentargets.org/

Examples

```
biohttp::body_or_null(opentargets_resolve_disease("neurofibromatosis type 1"))
biohttp::body_or_null(opentargets_resolve_disease("MONDO:0018975"))
```

`panelapp_all_panels` *The whole PanelApp panel index*

Description

Walks the paginated index and stacks the pages. Stops at `max_pages`, which is a guard against an unbounded walk rather than a tuned number.

Usage

```
panelapp_all_panels(max_pages = 6, page_size = 100, ...)
```

Arguments

`max_pages`	Maximum pages to fetch.
`page_size`	Panels per page.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble from `panelapp_parse_index()`, stacked across pages. A page that fails after the first is where the walk stops, keeping what was already collected; a failure on the first page is returned as-is.

References

Martin et al. (2019). PanelApp crowdsources expert knowledge to establish consensus diagnostic gene panels. Nature Genetics 51(11), 1560-1565. doi:10.1038/s4158801905282

Service documentation: https://panelapp.genomicsengland.co.uk/

Examples

```
biohttp::body_or_null(panelapp_all_panels(max_pages = 2))
```

panelapp_panel *The genes on one PanelApp panel*

Description

The genes on one PanelApp panel

Usage

```
panelapp_panel(panel_id, ...)
```

Arguments

panel_id	A PanelApp panel id, for example 255.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in panelapp_parse_panel().

References

Martin et al. (2019). PanelApp crowdsources expert knowledge to establish consensus diagnostic gene panels. Nature Genetics 51(11), 1560-1565. doi:10.1038/s4158801905282

Service documentation: https://panelapp.genomicsengland.co.uk/

Examples

```
biohttp::body_or_null(panelapp_panel(255))
```

panelapp_panels *One page of the PanelApp panel index*

Description

One page of the PanelApp panel index

Usage

```
panelapp_panels(page = 1, page_size = 100, ...)
```

Arguments

page	The 1-based page number.
page_size	Panels per page.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a list of panels (the tibble from panelapp_parse_index()) and has_more, which is TRUE when PanelApp sent a next link.

References

Martin et al. (2019). PanelApp crowdsources expert knowledge to establish consensus diagnostic gene panels. Nature Genetics 51(11), 1560-1565. doi:10.1038/s4158801905282

Service documentation: https://panelapp.genomicsengland.co.uk/

Examples

```
biohttp::body_or_null(panelapp_panels())$panels
```

---

panelapp_parse_index *Turn a PanelApp panel index page into a table*

Description

Pure.

Usage

```
panelapp_parse_index(body)
```

Arguments

body	A parsed PanelApp panels/ response.

Value

A tibble of id, name, version, disorders (a list column, as a panel carries any number), and source_url, one row per panel. NULL when the page is empty.

References

Martin et al. (2019). PanelApp crowdsources expert knowledge to establish consensus diagnostic gene panels. Nature Genetics 51(11), 1560-1565. doi:10.1038/s4158801905282

Service documentation: https://panelapp.genomicsengland.co.uk/

Examples

```
body <- list(results = list(
  list(id = 255, name = "Neurofibromatosis Type 1",
       relevant_disorders = list("NF1"))
))
panelapp_parse_index(body)
```

`panelapp_parse_panel` *Turn a PanelApp panel detail into a table of genes*

Description

Pure.

Usage

```
panelapp_parse_panel(body)
```

Arguments

`body`	A parsed PanelApp panel detail response.

Details

Every gene on the panel is returned, red included. See the note in the file header on why the confidence level is not converted to a weight here.

Value

A tibble of `symbol`, `confidence` (the raw level PanelApp sent), `level` (`"green"`, `"amber"`, `"red"`, or `NA` for anything else), `hgnc_id`, and `source_url`. `NULL` when the panel has no genes.

References

Martin et al. (2019). PanelApp crowdsources expert knowledge to establish consensus diagnostic gene panels. Nature Genetics 51(11), 1560-1565. doi:10.1038/s4158801905282

Service documentation: https://panelapp.genomicsengland.co.uk/

Examples

```
body <- list(id = 255, genes = list(
  list(
    entity_name = "NF1",
    gene_data = list(gene_symbol = "NF1", hgnc_id = "HGNC:7765"),
    confidence_level = "3"
  )
))
panelapp_parse_panel(body)
```

pdbe_parse_structures *Turn a PDBe best-structures response into a table*

Description

Pure.

Usage

```
pdbe_parse_structures(body, accession = NA_character_)
```

Arguments

body	A parsed PDBe best_structures response.
accession	The UniProt accession that was queried.

Value

A tibble of pdb_id, method, resolution, coverage, and source_url, one row per distinct structure. NULL when there are none.

One entry per structure, not per chain

The response is keyed by the accession, and each value is a list of **per chain** mappings. A single PDB entry therefore recurs once for every chain it resolved, so 3 structures across 12 chains arrive as 12 records. Counting them raw overstates structural coverage several-fold.

Rows are collapsed to distinct pdb_id, keeping the first occurrence, which is PDBe's own best (highest coverage) chain for that entry.

References

Armstrong et al. (2020). PDBe: improved findability of macromolecular structure data in the PDB. Nucleic Acids Research 48(D1), D335-D343. doi:10.1093/nar/gkz990

Service documentation: https://www.ebi.ac.uk/pdbe/

Examples

```
body <- list(P21359 = list(
  list(pdb_id = "7pgp", chain_id = "F", experimental_method = "Electron Microscopy",
       resolution = 3.1, coverage = 1),
  list(pdb_id = "7pgp", chain_id = "N", experimental_method = "Electron Microscopy",
       resolution = 3.1, coverage = 1)
))
pdbe_parse_structures(body, "P21359")
```

pdbe_structures *Experimental structures for a UniProt accession*

Description

Experimental structures for a UniProt accession

Usage

```
pdbe_structures(accession, ...)
```

Arguments

accession	A UniProt accession, for example "P21359".
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in pdbe_parse_structures().

References

Armstrong et al. (2020). PDBe: improved findability of macromolecular structure data in the PDB. Nucleic Acids Research 48(D1), D335-D343. doi:10.1093/nar/gkz990

Service documentation: https://www.ebi.ac.uk/pdbe/

Examples

```
biohttp::body_or_null(pdbe_structures("P21359"))
```

pharos_parse_targets *Turn a Pharos targets response into a table*

Description

Pure. Rows come back in symbols order, one per input.

Usage

```
pharos_parse_targets(body, symbols)
```

Arguments

body	A parsed Pharos GraphQL response body.
symbols	The gene symbols that were queried, in the order asked.

Details

A TDL Pharos does not recognise is reported as `NA` rather than passed through, so a caller never has to guess whether an unexpected string is a new level or a typo.

Value

A tibble of `symbol`, `tdl`, and `source_url`, one row per entry in `symbols`. `tdl` is one of `Tclin`, `Tchem`, `Tbio`, `Tdark`, or `NA`.

References

Kelleher et al. (2023). Pharos 2023: an integrated resource for the understudied human proteome. Nucleic Acids Research 51(D1), D1405-D1416. doi:10.1093/nar/gkac1033

Service documentation: https://pharos.nih.gov/

Examples

```
body <- list(data = list(targets = list(targets = list(
  list(sym = "NF1", tdl = "Tbio")
))))
pharos_parse_targets(body, "NF1")
```

`pharos_target` *Target Development Level for one gene*

Description

A thin wrapper over `pharos_targets()`.

Usage

```
pharos_target(symbol, ...)
```

Arguments

`symbol`	A gene symbol.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a one-row tibble.

References

Kelleher et al. (2023). Pharos 2023: an integrated resource for the understudied human proteome. Nucleic Acids Research 51(D1), D1405-D1416. doi:10.1093/nar/gkac1033

Service documentation: https://pharos.nih.gov/

Examples

```
biohttp::body_or_null(pharos_target("NF1"))
```

`pharos_targets` *Target Development Level for many genes*

Description

One request for the whole list, because Pharos's `targets` query already takes an array.

Usage

```
pharos_targets(symbols, ...)
```

Arguments

`symbols`	Gene symbols.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a tibble with one row per entry in `symbols`, in the same order. See `pharos_parse_targets()`.

References

Kelleher et al. (2023). Pharos 2023: an integrated resource for the understudied human proteome. Nucleic Acids Research 51(D1), D1405-D1416. doi:10.1093/nar/gkac1033

Service documentation: https://pharos.nih.gov/

Examples

```
biohttp::body_or_null(pharos_targets(c("NF1", "EGFR")))
```

protvar_function *Functional context for a residue*

Description

Functional context for a residue

Usage

```
protvar_function(accession, position, ...)
```

Arguments

accession	A UniProt accession, for example "P04637".
position	A protein position. Use protvar_position() to get one out of a protein-change string.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a single string. See protvar_parse_function().

References

Stephenson et al. (2024). ProtVar: mapping and contextualizing human missense variation. Nucleic Acids Research 52(W1), W140-W147. doi:10.1093/nar/gkae413

Service documentation: https://www.ebi.ac.uk/ProtVar/

Examples

```
biohttp::body_or_null(protvar_function("P04637", 175))
```

protvar_parse_function *Turn a ProtVar function response into its text*

Description

Pure. Returns the first FUNCTION comment, with citations stripped.

Usage

```
protvar_parse_function(body)
```

Arguments

body	A parsed ProtVar /function response.

Value

A single string, or NA_character_ when there is no function comment.

References

Stephenson et al. (2024). ProtVar: mapping and contextualizing human missense variation. Nucleic Acids Research 52(W1), W140-W147. doi:10.1093/nar/gkae413

Service documentation: https://www.ebi.ac.uk/ProtVar/

Examples

```
body <- list(comments = list(list(
  type = "FUNCTION",
  text = list(list(value = "Induces arrest (PubMed:11025664)."))
)))
protvar_parse_function(body)
```

---

protvar_parse_population

Turn a ProtVar population response into a table

Description

Pure. Known variants at the residue.

Usage

```
protvar_parse_population(body)
```

Arguments

body	A parsed ProtVar /population response.

Value

A tibble of change and sources, where sources is a comma-separated list of distinct source names. NULL when there are no variants.

Sources are deduplicated

A variant carries one cross-reference per supporting record, so the same source name recurs many times: the TP53 175 fixture lists NCI-TCGA four times for a single change. Reporting them raw makes a single database look like corroboration from several.

References

Stephenson et al. (2024). ProtVar: mapping and contextualizing human missense variation. Nucleic Acids Research 52(W1), W140-W147. doi:10.1093/nar/gkae413

Service documentation: https://www.ebi.ac.uk/ProtVar/

Examples

```
body <- list(variants = list(list(
  alternativeSequence = "Cys",
  xrefs = list(list(name = "NCI-TCGA"), list(name = "NCI-TCGA"))
)))
protvar_parse_population(body)
```

`protvar_population` *Known variants at a residue*

Description

Known variants at a residue

Usage

```
protvar_population(accession, position, ...)
```

Arguments

`accession`	A UniProt accession, for example `"P04637"`.
`position`	A protein position. Use `protvar_position()` to get one out of a protein-change string.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `protvar_parse_population()`.

References

Stephenson et al. (2024). ProtVar: mapping and contextualizing human missense variation. Nucleic Acids Research 52(W1), W140-W147. doi:10.1093/nar/gkae413

Service documentation: https://www.ebi.ac.uk/ProtVar/

Examples

```
biohttp::body_or_null(protvar_population("P04637", 175))
```

protvar_position *Pull a residue position out of a protein-change string*

Description

Pure. Accepts the several forms these arrive in: `"R175H"`, `"p.Arg175His"`, `"Arg175His"`, or a bare `"175"`.

Usage

```
protvar_position(variant)
```

Arguments

`variant`	A protein-change string.

Value

An integer position, or `NULL` when there is no number in it.

References

Stephenson et al. (2024). ProtVar: mapping and contextualizing human missense variation. Nucleic Acids Research 52(W1), W140-W147. doi:10.1093/nar/gkae413

Service documentation: https://www.ebi.ac.uk/ProtVar/

Examples

```
protvar_position("p.Arg175His")
protvar_position("R175H")
protvar_position("no digits here")
```

protvar_strip_citations *Strip inline citations out of a UniProt function comment*

Description

Pure.

Usage

```
protvar_strip_citations(text)
```

Arguments

text	A function comment.

Details

UniProt FUNCTION comments carry their evidence inline, as (PubMed:11025664, PubMed:12524540, ...). For a well-studied protein the citations run longer than the prose they support. This drops the citation groups and tidies the punctuation left behind, while keeping non-citation parentheticals such as (By similarity).

Value

The text with citation groups removed.

References

Stephenson et al. (2024). ProtVar: mapping and contextualizing human missense variation. Nucleic Acids Research 52(W1), W140-W147. doi:10.1093/nar/gkae413

Service documentation: https://www.ebi.ac.uk/ProtVar/

Examples

```
protvar_strip_citations("Induces arrest (PubMed:11025664, PubMed:12524540).")
protvar_strip_citations("Binds DNA (By similarity).")
```

---

pubtator_entity *Build the PubTator3 entity token for a gene*

Description

Prefers the Entrez id, which names the gene exactly, and falls back to the symbol. Exported because the token is what appears in source_id, so a caller reconstructing a citation needs the same rule.

Usage

```
pubtator_entity(symbol = NULL, entrez = NULL)
```

Arguments

symbol	A gene symbol.
entrez	An NCBI Gene id. Used in preference to symbol when present.

Value

A single string such as "@GENE_7157", or NULL when neither argument is usable.

References

Wei et al. (2024). PubTator 3.0: an AI-powered literature resource for unlocking biomedical knowledge. Nucleic Acids Research 52(W1), W540-W546. doi:10.1093/nar/gkae235

Service documentation: https://www.ncbi.nlm.nih.gov/research/pubtator3/

Examples

```
pubtator_entity("TP53", 7157)
pubtator_entity("TP53")
```

---

pubtator_gene_literature

Articles PubTator3 has tagged with a gene

Description

Articles PubTator3 has tagged with a gene

Usage

```
pubtator_gene_literature(symbol = NULL, entrez = NULL, ...)
```

Arguments

symbol	A gene symbol.
entrez	An NCBI Gene id, preferred over symbol. See the note in the file header.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is a list of count, entity, results (the tibble from pubtator_parse_results(), which may be NULL), and source_url. A count of 0 is ok, not no_data.

References

Wei et al. (2024). PubTator 3.0: an AI-powered literature resource for unlocking biomedical knowledge. Nucleic Acids Research 52(W1), W540-W546. doi:10.1093/nar/gkae235

Service documentation: https://www.ncbi.nlm.nih.gov/research/pubtator3/

Examples

```
biohttp::body_or_null(pubtator_gene_literature("TP53", 7157))$count
```

pubtator_parse_count *Read the article count off a PubTator3 search response*

Description

Pure. `0` is a real count and comes back as `0L`; only an absent `count` is `NA`.

Usage

```
pubtator_parse_count(body)
```

Arguments

body	A parsed PubTator3 `search/` response.

Value

A single integer, or `NA_integer_`.

References

Wei et al. (2024). PubTator 3.0: an AI-powered literature resource for unlocking biomedical knowledge. Nucleic Acids Research 52(W1), W540-W546. doi:10.1093/nar/gkae235

Service documentation: https://www.ncbi.nlm.nih.gov/research/pubtator3/

Examples

```
pubtator_parse_count(list(count = 4180))
pubtator_parse_count(list(count = 0))
```

pubtator_parse_results *Turn PubTator3 search results into a table*

Description

Pure.

Usage

```
pubtator_parse_results(body)
```

Arguments

body	A parsed PubTator3 `search/` response.

Value

A tibble of `pmid`, `title`, `journal`, `authors` (a list column, as an article has any number), `year`, and `source_url`. `NULL` when there are no results.

References

Wei et al. (2024). PubTator 3.0: an AI-powered literature resource for unlocking biomedical knowledge. Nucleic Acids Research 52(W1), W540-W546. doi:10.1093/nar/gkae235

Service documentation: https://www.ncbi.nlm.nih.gov/research/pubtator3/

Examples

```
body <- list(results = list(list(
  pmid = 36197410,
  title = "TP53 or Not TP53",
  journal = "Clin Cancer Res"
)))
pubtator_parse_results(body)
```

---

quickgo_annotations *GO annotations for a UniProt accession*

Description

GO annotations for a UniProt accession

Usage

```
quickgo_annotations(
  accession,
  aspect = c("biological_process", "molecular_function", "cellular_component"),
  limit = 100,
  ...
)
```

Arguments

`accession`	A UniProt accession, for example `"P21359"`.
`aspect`	One of `"biological_process"`, `"molecular_function"`, or `"cellular_component"`.
`limit`	Maximum annotation rows to ask for. Remember these are evidence lines rather than terms, so this is not a term count.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `quickgo_parse_annotations()`.

References

Binns et al. (2009). QuickGO: a web-based tool for Gene Ontology searching. Bioinformatics 25(22), 3045-3046. doi:10.1093/bioinformatics/btp536

Service documentation: https://www.ebi.ac.uk/QuickGO/

Examples

```
biohttp::body_or_null(quickgo_annotations("P21359"))
```

`quickgo_parse_annotations` *Turn a QuickGO annotation response into a table*

Description

Pure.

Usage

```
quickgo_parse_annotations(body)
```

Arguments

body	A parsed QuickGO `annotation/search` response.

Value

A tibble of `go_id`, `go_name`, `evidence`, `reference`, `aspect`, and `source_url`, one row per distinct GO term. `NULL` when there are none.

References

Binns et al. (2009). QuickGO: a web-based tool for Gene Ontology searching. Bioinformatics 25(22), 3045-3046. doi:10.1093/bioinformatics/btp536

Service documentation: https://www.ebi.ac.uk/QuickGO/

Examples

```
body <- list(results = list(list(
  goId = "GO:0001937",
  goName = "negative regulation of endothelial cell proliferation",
  goEvidence = "IMP",
  reference = "PMID:16648142"
)))
quickgo_parse_annotations(body)
```

reactome_parse_pathways *Turn a Reactome pathway array into a table*

Description

Pure.

Usage

```
reactome_parse_pathways(body)
```

Arguments

body	A parsed Reactome pathways response, which is a bare array.

Value

A tibble of `pathway_id`, `name`, `in_disease`, and `source_url`, one row per pathway. `NULL` when the gene has none.

References

Milacic et al. (2024). The Reactome Pathway Knowledgebase 2024. Nucleic Acids Research 52(D1), D672-D678. doi:10.1093/nar/gkad1025

Service documentation: https://reactome.org/

Examples

```
body <- list(list(
  stId = "R-HSA-5658442",
  displayName = "Regulation of RAS by GAPs",
  isInDisease = FALSE
))
reactome_parse_pathways(body)
```

reactome_pathways *Reactome pathways for a gene symbol*

Description

Reactome pathways for a gene symbol

Usage

```
reactome_pathways(symbol, species = 9606, ...)
```

Arguments

symbol	A gene symbol, for example "NF1".
species	The NCBI taxon id. Defaults to human.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in reactome_parse_pathways().

References

Milacic et al. (2024). The Reactome Pathway Knowledgebase 2024. Nucleic Acids Research 52(D1), D672-D678. doi:10.1093/nar/gkad1025

Service documentation: https://reactome.org/

Examples

```
biohttp::body_or_null(reactome_pathways("NF1"))
```

---

string_map_ids *The STRING identifier map for a set of symbols*

Description

Maps each queried symbol to STRING's own preferredName. Needed to interpret network edges; see string_reconcile_edges().

Usage

```
string_map_ids(symbols, species = STRING_HUMAN, ...)
```

Arguments

symbols	Gene symbols.
species	An NCBI taxon id. Defaults to human.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in string_parse_ids().

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
biohttp::body_or_null(string_map_ids(c("SEPTIN9", "TP53")))
```

`string_network` *The interaction network within a set of genes*

Description

High-confidence edges among the genes you asked about, with the endpoints reconciled back to your symbols.

Usage

```
string_network(
  symbols,
  species = STRING_HUMAN,
  required_score = STRING_MIN_SCORE,
  reconcile = TRUE,
  ...
)
```

Arguments

`symbols`	Gene symbols. At least two, since one gene has no network.
`species`	An NCBI taxon id. Defaults to human.
`required_score`	STRING's 0-1000 combined-score threshold.
`reconcile`	Whether to fetch the identifier map and translate edge endpoints back into your symbols. Costs one extra request and is skipped automatically when there are no edges to translate.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a list of `edges` (see `string_parse_network()`), `queried`, `n_query`, `truncated`, and `n_dropped`.

Knowing what was asked

The result carries `queried`, the exact symbol set STRING was asked about after the 500-identifier cap, plus `truncated` and `n_dropped`. That is what lets a caller tell a gene measured to have no partners from a gene that was never sent. Reporting the second as the first would invent a negative result.

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
res <- string_network(c("TP53", "NF1", "EGFR"))
biohttp::body_or_null(res)$edges
```

`string_parse_ids` *Turn a STRING identifier-map response into a table*

Description

Pure. One row per queried identifier.

Usage

```
string_parse_ids(body)
```

Arguments

`body`	A parsed STRING `get_string_ids` response, a JSON array.

Value

A tibble of `query`, `preferred`, and `string_id`, with `query` and `preferred` upper-cased.

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
body <- list(list(
  queryItem = "SEPTIN9", preferredName = "SEPT9",
  stringId = "9606.ENSP00000329125"
))
string_parse_ids(body)
```

`string_parse_network` *Turn a STRING network response into an edge table*

Description

Pure. Edge endpoints come back in STRING's `preferredName` space; see `string_reconcile_edges()` for why that matters.

Usage

```
string_parse_network(body)
```

Arguments

`body`	A parsed STRING `network` response, a JSON array of edges.

Value

A tibble of `gene_a`, `gene_b`, and `score`, upper-cased. A zero-row tibble when there are no edges, because "no high-confidence edges" is a real answer about a set rather than an absence of one.

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
body <- list(list(
  preferredName_A = "TP53", preferredName_B = "NF1", score = 0.88
))
string_parse_network(body)
```

string_parse_partners *Turn STRING interaction-partner rows into a table*

Description

Pure. Sorted by combined score, strongest first.

Usage

```
string_parse_partners(body)
```

Arguments

body	A parsed STRING `interaction_partners` response, a JSON array.

Value

A tibble of `partner`, `score`, and the four evidence channels `experimental`, `database`, `coexpression`, and `textmining`. `NULL` when there are no partners.

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
body <- list(list(
  preferredName_B = "SFN", score = 0.999,
  escore = 0.981, dscore = 0.75, ascore = 0, tscore = 0.859
))
string_parse_partners(body)
```

string_partners *Interaction partners for one gene*

Description

Interaction partners for one gene

Usage

```
string_partners(symbol, species = STRING_HUMAN, limit = 25, ...)
```

Arguments

symbol	A gene symbol.
species	An NCBI taxon id. Defaults to human.
limit	How many partners to return.
...	Passed to biohttp::get_json(), for example throttle.

Value

A biohttp envelope whose data is the tibble described in string_parse_partners().

On STRING's content type

STRING serves JSON as text/json rather than application/json. A client that trusts the content-type header rejects a perfectly good body. biohttp parses with check_type = FALSE, so this works, but it is the reason not to "tidy" that up.

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
biohttp::body_or_null(string_partners("TP53"))
```

---

string_reconcile_edges

Rewrite edge endpoints back into the queried symbol space

Description

Pure, and it exists because of a real and quiet failure.

Usage

```
string_reconcile_edges(edges, id_map)
```

Arguments

edges	An edge tibble from string_parse_network().
id_map	An identifier map from string_parse_ids(), or NULL.

Details

STRING's canonical `preferredName` lags HGNC. Query `SEPTIN9` and the edges come back naming `SEPT9`, and the `network` endpoint does **not** echo the query term. A caller matching edges against the symbols it asked about therefore finds nothing for that gene and records it as having no partners, which is indistinguishable from a real isolate.

Passing the map from `string_map_ids()` through here translates the endpoints back. An endpoint with no mapping is left alone, and an empty map leaves the edges untouched, so reconciliation can never turn a good answer into a worse one.

Value

The edge tibble, with endpoints translated where a mapping existed.

References

Szklarczyk et al. (2023). The STRING database in 2023: protein-protein association networks and functional enrichment analyses for any sequenced genome of interest. Nucleic Acids Research 51(D1), D638-D646. doi:10.1093/nar/gkac1000

Service documentation: https://string-db.org/

Examples

```
edges <- tibble::tibble(gene_a = "SEPT9", gene_b = "TP53", score = 0.9)
map <- tibble::tibble(
  query = "SEPTIN9", preferred = "SEPT9", string_id = "9606.ENSP00000329125"
)
string_reconcile_edges(edges, map)
```

---

uniprot_diseases *Diseases UniProt curates for an accession*

Description

Diseases UniProt curates for an accession

Usage

```
uniprot_diseases(accession, ...)
```

Arguments

`accession`	A UniProt accession, for example `"P04637"`.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `uniprot_parse_diseases()`.

References

The UniProt Consortium (2025). UniProt: the Universal Protein Knowledgebase in 2025. Nucleic Acids Research 53(D1), D609-D617. doi:10.1093/nar/gkae1010

Service documentation: https://www.uniprot.org/

Examples

```
biohttp::body_or_null(uniprot_diseases("P04637"))
```

`uniprot_features` *Sequence features for an accession*

Description

From the EBI Proteins API, which is a different host to `uniprot_diseases()`.

Usage

```
uniprot_features(accession, types = UNIPROT_FEATURE_TYPES, ...)
```

Arguments

`accession`	A UniProt accession.
`types`	Feature types to request. Defaults to the set worth showing when placing a variant in a protein's architecture.
`...`	Passed to `biohttp::get_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is the tibble described in `uniprot_parse_features()`.

References

Nightingale et al. (2017). The Proteins API: accessing key integrated protein and genome information. Nucleic Acids Research 45(W1), W539-W544. doi:10.1093/nar/gkx237

The data it serves is UniProt's. The UniProt Consortium (2025). UniProt: the Universal Protein Knowledgebase in 2025. Nucleic Acids Research 53(D1), D609-D617. doi:10.1093/nar/gkae1010

Service documentation: https://www.ebi.ac.uk/proteins/api/doc/

Examples

```
biohttp::body_or_null(uniprot_features("P15056"))
```

`uniprot_features_at` *The features spanning a residue position*

Description

Pure. Which annotated domains, sites, or regions contain a given residue, which is the question a variant reviewer actually asks.

Usage

```
uniprot_features_at(features, position)
```

Arguments

`features`	A feature tibble from `uniprot_parse_features()`.
`position`	A residue position.

Value

The rows of `features` whose `begin`/`end` span `position`.

References

Nightingale et al. (2017). The Proteins API: accessing key integrated protein and genome information. Nucleic Acids Research 45(W1), W539-W544. doi:10.1093/nar/gkx237

The data it serves is UniProt's. The UniProt Consortium (2025). UniProt: the Universal Protein Knowledgebase in 2025. Nucleic Acids Research 53(D1), D609-D617. doi:10.1093/nar/gkae1010

Service documentation: https://www.ebi.ac.uk/proteins/api/doc/

Examples

```
features <- uniprot_parse_features(list(features = list(
  list(type = "DOMAIN", description = "Kinase", begin = "457", end = "717")
)))
uniprot_features_at(features, 600)
uniprot_features_at(features, 100)
```

`uniprot_parse_diseases` *Turn a UniProtKB entry into a curated-disease table*

Description

Pure. Reads the `DISEASE` comments of an entry.

Usage

```
uniprot_parse_diseases(body)
```

Arguments

`body`	A parsed UniProtKB entry.

Value

A tibble of `id`, `name`, `acronym`, `mim`, `causal`, and `source_url`. `NULL` when the entry curates no disease involvement.

The causal distinction

UniProt's disease notes separate two very different claims. "The disease is caused by variants affecting the gene" is a Mendelian statement; "may be involved in the pathogenesis" is far weaker. Both arrive as `DISEASE` comments, so treating them alike would promote a speculative association to a causal one. `causal` carries the distinction, read from the note text.

References

The UniProt Consortium (2025). UniProt: the Universal Protein Knowledgebase in 2025. Nucleic Acids Research 53(D1), D609-D617. doi:10.1093/nar/gkae1010

Service documentation: https://www.uniprot.org/

Examples

```
body <- list(comments = list(list(
  commentType = "DISEASE",
  disease = list(
    diseaseAccession = "DI-00218", diseaseId = "Li-Fraumeni syndrome",
    acronym = "LFS", diseaseCrossReference = list(database = "MIM", id = "151623")
  ),
  note = list(texts = list(list(value = "The disease is caused by variants")))
)))
uniprot_parse_diseases(body)
```

`uniprot_parse_features` *Turn an EBI Proteins features response into a table*

Description

Pure. Sorted by start position.

Usage

```
uniprot_parse_features(body)
```

Arguments

`body`	A parsed EBI Proteins `features` response.

Value

A tibble of `type`, `label`, `description`, `begin`, and `end`. A zero-row tibble when the entry has no features of the requested types, because "this protein has no annotated domains" is a real answer.

References

Nightingale et al. (2017). The Proteins API: accessing key integrated protein and genome information. Nucleic Acids Research 45(W1), W539-W544. doi:10.1093/nar/gkx237

The data it serves is UniProt's. The UniProt Consortium (2025). UniProt: the Universal Protein Knowledgebase in 2025. Nucleic Acids Research 53(D1), D609-D617. doi:10.1093/nar/gkae1010

Service documentation: https://www.ebi.ac.uk/proteins/api/doc/

Examples

```
body <- list(features = list(
  list(type = "DOMAIN", description = "Protein kinase", begin = "457", end = "717")
))
uniprot_parse_features(body)
```

`variantvalidator_normalize` *Validate and normalize one HGVS variant*

Description

Validate and normalize one HGVS variant

Usage

```
variantvalidator_normalize(
  hgvs,
  build = "GRCh38",
  throttle = VARIANTVALIDATOR_THROTTLE,
  ...
)
```

Arguments

`hgvs`	An HGVS transcript variant.
`build`	A genome build.
`throttle`	A throttle spec. Defaults to the documented 4 per second.
`...`	Passed to `biohttp::get_json()`.

Value

A biohttp envelope whose `data` is the tibble described in `variantvalidator_parse()`.

References

Freeman et al. (2018). VariantValidator: accurate validation, mapping, and formatting of sequence variation descriptions. Human Mutation 39(1), 61-68. doi:10.1002/humu.23348

Service documentation: `https://variantvalidator.org/`

Examples

```
biohttp::body_or_null(variantvalidator_normalize("NM_000546.6:c.215C>G"))
```

variantvalidator_parse *Turn a VariantValidator response into a table*

Description

Pure.

Usage

```
variantvalidator_parse(body, submitted = NA_character_)
```

Arguments

body	A parsed VariantValidator response.
submitted	The HGVS string that was sent.

Value

A one-row tibble of `resolved`, `submitted`, `normalized`, `gene`, `protein`, `chrom`, `pos`, `ref`, `alt`, and `warnings`, where `warnings` is a list column. `NULL` when there is no record at all.

The response is keyed by the answer, not by a fixed name

VariantValidator returns an object whose key is the **normalized** variant, which is not known before the call. `flag` and `metadata` are the only fixed keys, so the record is found by removing those rather than by looking up a name. A parser expecting a fixed key finds nothing.

References

Freeman et al. (2018). VariantValidator: accurate validation, mapping, and formatting of sequence variation descriptions. Human Mutation 39(1), 61-68. doi:10.1002/humu.23348

Service documentation: https://variantvalidator.org/

Examples

```
body <- list(
  flag = "gene_variant",
  metadata = list(),
  `NM_000546.6:c.215C>G` = list(gene_symbol = "TP53")
)
variantvalidator_parse(body, "NM_000546.6:c.215C>G")
```

vep_default_options *The request flags VEP is asked for by default*

Description

Pure. The default for the options argument of vep_variants() and vep_variants_all(). Start from this and add to it rather than replacing it, because vcf_string is one of the identities results are matched on and mane is what vep_pick_transcript() chooses by.

Usage

```
vep_default_options()
```

Value

A named list of flags.

Supported flags

Every entry is a VEP request parameter, sent as name=value on the query string. TRUE and 1 send name=1; FALSE, 0 and NULL omit the flag. Names are case sensitive, exactly as VEP spells them. The flags this package parses are:

- AlphaMissense, mane, numbers, vcf_string: the defaults.
- af, af_gnomade, af_gnomadg: colocated variant frequencies, read by vep_parse_colocated().
- CADD, SpliceAI, REVEL: per-transcript predictor scores, read by vep_parse_element().
- hgvs: hgvsc and hgvsp notation per transcript.
- canonical: marks the canonical transcript.
- pick, pick_allele_gene: ask VEP to return one transcript per variant or per allele and gene, rather than all of them.
- protein, domains, variant_class: extra transcript annotation, carried through untouched in the response for a caller parsing it directly.
- LoF: LOFTEE, read into the lof column.

dbNSFP is refused. It returns comma-joined multi-transcript strings in dbNSFP's own order, not aligned to the transcript being reported, so the values silently belong to a different transcript than the rest of the row.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
vep_default_options()
c(vep_default_options(), list(CADD = 1, REVEL = 1))
```

`vep_default_throttle` *The default rate limit for VEP requests*

Description

Ensembl publishes no formal limit for the REST service, but an unpaced burst of the 100 or so requests one case's coding lane needs reliably trips a 429 partway through. One request per second is conservative enough to avoid that while still finishing a typical chunk count in a few minutes; `vep_variants_all()` applies it by default.

Usage

```
vep_default_throttle()
```

Value

A throttle spec, see `biohttp::req_defaults()`'s `throttle` argument.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
vep_default_throttle()
```

`vep_key` *Build the variant key VEP results are matched on*

Description

Pure. `vep_variants()` returns a `key` column built this way, so this is what a caller uses to join the result back onto its own variant table.

Usage

```
vep_key(chrom, pos, ref, alt)
```

Arguments

chrom	A chromosome, with or without a chr prefix.
pos	A 1-based position.
ref, alt	Reference and alternate alleles.

Value

A single string, chrom-pos-ref-alt.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
vep_key("7", 140753336, "A", "T")
vep_key("chr7", 140753336, "a", "t")
```

---

vep_parse_batch *Turn a VEP batch response into a table*

Description

Pure. One row per requested variant, in the order asked.

Usage

```
vep_parse_batch(body, keys)
```

Arguments

body	A parsed VEP response, an array of elements.
keys	Variant keys from vep_key(), in the order asked.

Value

A tibble with one row per entry in keys: a key column, the columns of vep_parse_element(), then those of vep_parse_colocated().

Matched by identity, not by position

VEP does not promise that results come back in the order they were sent. Zipping the response onto the input by index therefore assigns consequences to the wrong variants, silently. Elements are matched on the key built from the echoed input line, which is the exact string that was sent.

Indels are renumbered

VEP left-trims and renumbers an indel, so the `start` and `allele_string` it reports do not rebuild the key the caller asked with. An insertion sent as `1 55516888 . T TA . . .` comes back as start 55516889 and `-/A`, and a delins can come back re-anchored in `vcf_string` too. The echoed `input` is the identity that survives, so it is matched first, then `vcf_string`, and the rebuilt key only for a response carrying neither.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

---

vep_parse_colocated *Turn the colocated variants of a VEP element into a table row*

Description

Pure. A VEP element lists the known variants at the same site under `colocated_variants`: the dbSNP record, and COSMIC and HGMD entries beside it. The dbSNP record is the one carrying population frequencies and clinical significance, so that is the record read. When several dbSNP records are listed, the one with frequencies wins.

Usage

```
vep_parse_colocated(element)
```

Arguments

`element`	One parsed VEP result element.

Details

Frequencies are only present when `af`, `af_gnomade` or `af_gnomadg` was requested, see `vep_default_options()`. `gnomadg_af_max` and `gnomade_af_max` are the largest per-population frequency of that source, which is what a rarity filter wants rather than the overall frequency. `clin_sig` is the significance of the element's own allele where VEP reports it per allele, so the alleles of a multi-allelic site do not share one answer.

Value

A one-row tibble of `rsid`, `gnomadg_af`, `gnomade_af`, `gnomadg_af_max`, `gnomade_af_max`, and `clin_sig`, all `NA` when the element has no dbSNP record.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
element <- list(
  allele_string = "G/C",
  colocated_variants = list(list(
    id = "rs1042522",
    frequencies = list(C = list(gnomadg = 0.62, gnomadg_afr = 0.38)),
    clin_sig = list("benign")
  ))
)
vep_parse_colocated(element)
```

---

vep_parse_element *Turn one VEP element into a table row*

Description

Pure.

Usage

```
vep_parse_element(element)
```

Arguments

element	One parsed VEP result element.

Value

A one-row tibble.

AlphaMissense is nested

It is at `transcript_consequences[].alphamissense$am_pathogenicity`, per transcript. There is nothing at the top level. Hoisting the read out of the transcript is the single easiest way to get `NA` everywhere and conclude the API does not serve it.

Optional columns

`hgvsc`, `hgvsp`, `canonical`, `cadd_phred`, `cadd_raw`, `revel`, the `spliceai_*` scores and `lof` are only populated when the matching flag was requested, see `vep_default_options()`. Otherwise they are `NA`. VEP marks only the canonical transcript, so `canonical` is `TRUE` on that transcript and `NA` everywhere else, including when the flag was never asked. `spliceai_max` is the largest of the four SpliceAI delta scores.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
element <- list(
  most_severe_consequence = "missense_variant",
  transcript_consequences = list(list(
    gene_symbol = "BRAF", consequence_terms = list("missense_variant"),
    alphamissense = list(am_pathogenicity = 0.99, am_class = "pathogenic")
  ))
)
vep_parse_element(element)
```

---

vep_pick_transcript *Choose which transcript to report for a variant*

Description

Pure. A variant hits many transcripts and only one row can be reported.

Usage

```
vep_pick_transcript(element)
```

Arguments

element	One parsed VEP result element.

Details

The order is MANE Select first, because that is the community's designated representative transcript; then whichever transcript carries the variant's most severe consequence; then the first VEP listed. Taking the first outright reports an arbitrary transcript, often a non-coding one, which makes the consequence look milder than it is.

Value

One transcript consequence, or NULL when there are none.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
element <- list(
  most_severe_consequence = "missense_variant",
  transcript_consequences = list(
    list(consequence_terms = list("intron_variant")),
    list(consequence_terms = list("missense_variant"), mane_select = "NM_1")
  )
)
vep_pick_transcript(element)$mane_select
```

---

vep_region *Build a VEP region string from variant components*

Description

Pure. VEP's region endpoint takes a VCF-like string.

Usage

```
vep_region(chrom, pos, ref, alt)
```

Arguments

`chrom`	A chromosome, with or without a `chr` prefix.
`pos`	A 1-based position.
`ref`, `alt`	Reference and alternate alleles.

Value

A single string.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
vep_region("7", 140753336, "A", "T")
```

`vep_variants` *Consequence predictions for many variants*

Description

Consequence predictions for many variants

Usage

```
vep_variants(chrom, pos, ref, alt, options = vep_default_options(), ...)
```

Arguments

`chrom`, `pos`, `ref`, `alt`	Variant components, all the same length.
`options`	A named list of VEP request flags. See `vep_default_options()` for the supported flags and what each one adds to the result.
`...`	Passed to `biohttp::post_json()`, for example `throttle`.

Value

A biohttp envelope whose `data` is a tibble with one row per variant, in the order asked. See `vep_parse_batch()`.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
biohttp::body_or_null(vep_variants("7", 140753336, "A", "T"))
biohttp::body_or_null(vep_variants(
  "7", 140753336, "A", "T",
  options = c(vep_default_options(), list(af_gnomadg = 1, CADD = 1))
))
```

vep_variants_all *Consequence predictions for any number of variants*

Description

vep_variants() refuses more than the 200 VEP accepts per POST. This chunks the input at chunk_size and dispatches the chunks through biohttp::post_json_many(), so a variant table of any length is a handful of requests rather than a loop written at every call site.

Usage

```
vep_variants_all(
  chrom,
  pos,
  ref,
  alt,
  chunk_size = VEP_BATCH,
  options = vep_default_options(),
  throttle = vep_default_throttle(),
  ...
)
```

Arguments

chrom, pos, ref, alt	Variant components, all the same length.
chunk_size	Variants per request, at most VEP_BATCH (200).
options	A named list of VEP request flags. See vep_default_options() for the supported flags and what each one adds to the result.
throttle	A throttle spec passed to biohttp::post_json_many(). Defaults to vep_default_throttle(); pass NULL to disable pacing.
...	Passed to biohttp::post_json_many(), for example max_active.

Value

A biohttp envelope whose data is a tibble with one row per variant, in the order asked: the columns of vep_parse_batch() plus status, which is "ok" for a row whose chunk was answered and the failing envelope's status otherwise.

A failed chunk is rows of NA, not a failed call

Following gnomad_constraints(). Each chunk's envelope is inspected on its own; a chunk that failed yields a row of NA per variant carrying the envelope status in status, while the chunks that succeeded are parsed as usual. Partial failure is the normal case across many requests, and taking the whole call down would throw away the answers that did arrive. Only when every chunk failed is the first failing envelope returned, so an outage still reads as one.

Sorted and deduplicated before chunking

The input is put in genome order and a repeated variant is sent once, not once per occurrence, before it is split into chunks. Neither changes the contract: the return is still one row per input, in input order, and a repeated variant's positions all carry the same answer (which they did not necessarily do before, when duplicates could land in different chunks and see different transient failures). What it does change is how many requests go out: a coding lane's ~1500 multiallelic-split duplicates stop costing a request each, and a stable chunk order gives `biohttp`'s per-chunk cache a real chance to hit between two calls that share a background.

References

McLaren et al. (2016). The Ensembl Variant Effect Predictor. Genome Biology 17, 122. doi:10.1186/s1305901609744

Service documentation: https://rest.ensembl.org/

Examples

```
biohttp::body_or_null(vep_variants_all(
  c("7", "17"),
  c(140753336, 7676154),
  c("A", "G"),
  c("T", "C")
))
```

Index