Package: genevintage (via r-universe)

October 20, 2026

Title Detect Annotation Releases and Repair Gene Names

Version 0.1.3

Description Detects the species and the 'Ensembl' <https://www.ensembl.org> or 'GENCODE' <https://www.gencodegenes.org> annotation release associated with a set of gene or transcript identifiers by scoring them against a bundled fingerprint of each release. Identifiers are then converted to gene names using the matching release. Gene symbols that spreadsheet software turned into dates or numbers ('SEPT9' to '9-Sep') are restored to the spelling of the chosen release. Also retrieves genes by class (mitochondrial, sex-linked, protein-coding, ribosomal, transfer RNA, long non-coding RNA), exon-union gene lengths, 'Entrez' cross-references, and orthologs in another species from 'Ensembl Compara'.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports stats, tools, utils

Suggests curl, Matrix, knitr, methods, rmarkdown, spelling, testthat (>= 3.0.0), withr

Config/testthat/edition 3

URL https://genevintage.saketlab.org/R/, https://github.com/saketlab/genevintage

BugReports https://github.com/saketlab/genevintage/issues

Config/roxygen2/version 8.1.0

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Saket Choudhary [aut, cre] (ORCID: <https://orcid.org/0000-0001-5202-7633>)

Maintainer Saket Choudhary <saketc@iitb.ac.in>

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

Date/Publication 2026-10-20 18:20:16 UTC

RemoteUrl https://github.com/cran/genevintage

RemoteRef HEAD

RemoteSha a8a30cafc3bcc1693366ee76a13c8673bc2ee49b

Contents

aggregate_transcripts . . . 3
annotate_genes . . . 5
build_symbol_index . . . 6
build_symbol_index_from_gtf . . . 7
coding_genes . . . 7
compare_releases . . . 9
correct_genenames . . . 10
detect_release . . . 12
detect_species . . . 13
detect_species_names . . . 14
entrez_ids . . . 15
fetch_mapping . . . 16
fetch_orthologs . . . 17
fetch_transcripts . . . 18
fetch_xrefs . . . 19
gene_biotypes . . . 20
gene_cache . . . 21
gene_ids . . . 22
gene_lengths . . . 23
gene_names . . . 25
geneid2name . . . 26
genes_by_biotype . . . 27
guess_scheme . . . 29
hemoglobin_genes . . . 30
lncrna_genes . . . 31
mhc_genes . . . 32
mito_genes . . . 34
noncoding_genes . . . 35
ortholog_species . . . 36
orthologs . . . 37
parse_ens . . . 39
qmax_violations . . . 39
resolve_species . . . 40
ribosomal_genes . . . 41
rrna_genes . . . 42
sex_genes . . . 44
species_table . . . 45

stream_exon_lengths 46
stream_gtf 46
stream_homologies 47
stream_transcripts 48
stream_xrefs 49
transcript2gene 49
trna_genes 51

Index **53**

`aggregate_transcripts` *Sum transcript-level counts to gene level*

Description

Maps each row's transcript identifier with `transcript2gene()` and sums the rows of each gene. Rows come back in the order their gene was first seen. Columns, column names and the matrix class (dense, sparse `Matrix`, data frame) are kept.

Usage

```
aggregate_transcripts(
  x,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  row_ids = NULL,
  label = c("id", "name"),
  unmapped = c("keep", "drop"),
  quiet = FALSE
)

AggregateTranscripts(
  x,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  row_ids = NULL,
  label = c("id", "name"),
  unmapped = c("keep", "drop"),
  quiet = FALSE
)
```

Arguments

x	A numeric matrix, sparse `Matrix` or data frame whose rows are transcripts.
species	Skip species detection by naming it. Any form `resolve_species()` accepts: `"homo_sapiens"`, `"Homo sapiens"`, `"Human"` or `"human"`.
release	Annotation release. Without it, chosen as above.
assembly	Assembly suffix, e.g. `"37"` for human GRCh37. Detected alongside the release when both are left `NULL`.
source	Annotation family: `"ensembl"`, or `"gencode"` / `"gencode_all"` for GENCODE's primary and scaffold-inclusive gene sets. Detected alongside the release.
mapping	Skip fetching and use this table: `fetch_transcripts()` output, or a data frame with `transcript_id`, `gene_id` and `gene_name`.
row_ids	Overrides which identifiers to map. Defaults to `rownames(x)`; set this when `x` has none.
label	Label gene rows by `"id"` (the gene identifier) or `"name"` (the gene name, with the identifier appended where two genes share one).
unmapped	`"keep"` leaves unmapped rows under their own identifier, as `geneid2name()` keeps an identifier it cannot name; `"drop"` removes them.
quiet	Suppress the messages reporting the annotation and how many rows mapped.

Details

Summing is right for counts and for TPM; it is not for effective lengths.

Value

`x` with one row per gene. A warning of class `genevintage_unmapped_transcripts` is raised when fewer than half the rows mapped; call `transcript2gene()` and pass its result as `mapping` to see which ones.

See Also

`transcript2gene()`, `geneid2name()`.

Examples

```
tx <- data.frame(
  transcript_id = c("ENSMUST00000000001", "ENSMUST00000000002"),
  id = "ENSMUSG00000000001", name = "Gnai3"
)
m <- matrix(1:6, nrow = 3, dimnames = list(
  c("ENSMUST00000000001.4", "ENSMUST00000000002.1", "ENSMUST00000999999.1"),
  c("s1", "s2")
))
aggregate_transcripts(m, mapping = tx)
```

annotate_genes *Everything the annotation knows about a set of genes*

Description

Symbol, biotype and coordinates in one table, for genes given as identifiers or as names. The caller's own token comes back in `input`, so the result joins straight onto a results table or a feature frame.

Usage

```
annotate_genes(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

AnnotateGenes(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids`	Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.
`species`, `release`, `assembly`, `source`	Skip detection by naming them. `species` accepts anything `resolve_species()` does.
`mapping`	Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.
`quiet`	Suppress the message reporting what was detected.

Details

Unmatched inputs are retained with `NA` columns. Names matching several genes yield a row for each.

Value

A data frame of `input`, `id`, `name`, `chr`, `biotype` and, where the annotation carries them, `start`, `end`, `strand` and `span`. `span` is the genomic extent from first to last base. See `stream_gtf()`.

See Also

`gene_names()` for one value per input.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
annotate_genes(c("TP53", "ENSG00000198804", "MALAT1"), mapping = hs)
```

---

build_symbol_index — *Build a compact species/release gene-symbol index from GTF mappings*

Description

Build a compact species/release gene-symbol index from GTF mappings

Usage

```
build_symbol_index(mappings, min_release = NULL)

BuildSymbolIndex(mappings, min_release = NULL)
```

Arguments

`mappings`	A named list of mapping data frames returned by `fetch_mapping()`, or one such data frame with `species` and `release` columns.
`min_release`	Optional minimum release number.

Value

A data frame with `species`, `release`, `assembly`, `source`, and `symbol`.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
hs$species <- "homo_sapiens"
hs$release <- "116"
head(build_symbol_index(hs))
```

build_symbol_index_from_gtf

Fetch GTF mappings and build a symbol index

Description

Fetch GTF mappings and build a symbol index

Usage

```
build_symbol_index_from_gtf(species, releases, source = "ensembl")

BuildSymbolIndexFromGTF(species, releases, source = "ensembl")
```

Arguments

species	Species names accepted by resolve_species().
releases	Ensembl/GENCODE release numbers to fetch.
source	Annotation family.

Value

A symbol index suitable for detect_species_names().

Examples

```
ix <- build_symbol_index_from_gtf("yeast", releases = 116)
```

coding_genes

Protein-coding genes

Description

Selects the protein_coding biotype. Immunoglobulin and T-cell receptor segments encode protein too but carry their own biotypes; ask for "immune" through genes_by_biotype() to add them.

Usage

```
coding_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
```

```
  mapping = NULL,
  quiet = FALSE
)

CodingGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids` Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

`species`, `release`, `assembly`, `source` Skip detection by naming them. `species` accepts anything `resolve_species()` does.

`mapping` Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.

`quiet` Suppress the message reporting what was detected.

Value

A data frame of `id`, `name`, `chr` and `biotype`.

See Also

`noncoding_genes()`, `genes_by_biotype()`.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
coding_genes(c("TP53", "MALAT1", "XIST"), mapping = hs)

nrow(coding_genes(species = "human"))
```

compare_releases *Compare two releases of one annotation*

Description

Joins two vintages on the stable identifier and reports what differs: genes the newer release added, genes it dropped, and genes whose symbol or biotype changed. Rows that are identical in both are not returned.

Usage

```
compare_releases(
  species,
  from,
  to,
  assembly = NULL,
  source = NULL,
  quiet = FALSE
)

CompareReleases(
  species,
  from,
  to,
  assembly = NULL,
  source = NULL,
  quiet = FALSE
)
```

Arguments

species	Species in any form `resolve_species()` accepts.
from, to	The two releases, in either order.
assembly, source	Passed to `fetch_mapping()`. Both releases are read from the same assembly and annotation family to isolate release changes.
quiet	Suppress the summary message.

Details

Added and removed genes are identified by stable identifier presence. Finding a successor to a retired identifier requires a separate lookup.

Value

A data frame of `id`, `change` (`"added"`, `"removed"`, `"renamed"` or `"retyped"`), `name_from`, `name_to`, `biotype_from` and `biotype_to`.

See Also

`detect_release()` to find out which vintages you have.

Examples

```
d <- compare_releases("human", from = 110, to = 116)
table(d$change)
d[d$change == "renamed", ]
```

---

correct_genenames — *Repair gene symbols a spreadsheet has converted to dates or numbers*

Description

Excel silently reinterprets some gene symbols: `SEPT9` becomes `9-Sep`, `MARCH1` becomes `1-Mar`, `2310009E13` becomes `2.31E+13`. Several symbols can produce the same token, so candidates are checked against the caller's species and release. For example, `9-Sep` resolves to `SEPT9` against Ensembl release 95 and `SEPTIN9` against release 116.

Usage

```
correct_genenames(
  x,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  unique,
  quiet = FALSE,
  row_ids = NULL
)

CorrectGeneNames(
  x,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  unique,
  quiet = FALSE,
  row_ids = NULL
)
```

Arguments

- `x` — Gene names, some of which a spreadsheet may have converted, or a matrix, sparse `Matrix` or data frame with such names as its row names.
- `species`, `release`, `assembly`, `source` — Skip detection by naming them. `species` accepts anything `resolve_species()` does.
- `mapping` — Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.
- `unique` — Make the result unique by appending the damaged token to a repaired name that collides with another (as `gene_names()` does), then falling back to `make.unique()` for any collision that remains. Defaults to `TRUE` when `x` is a matrix, sparse `Matrix` or data frame, and to `FALSE` otherwise.
- `quiet` — Suppress the message reporting what was detected.
- `row_ids` — When `x` is a matrix, sparse `Matrix` or data frame, overrides which row identifiers to correct. Defaults to `rownames(x)`; set this when `x` has none.

Details

Tokens that are not damaged are returned unchanged, so the result drops straight back into `rownames()`. A token with no surviving candidate, or with more than one, is also returned unchanged and reported. `MARCH1` and `MARC1` both become `1-Mar`; a candidate already present undamaged elsewhere in `x` is ruled out, which usually resolves it.

Candidates are checked against an annotation: `mapping` if given, otherwise `species` at `release`. Without `release` the newest one is used, which returns current HGNC spellings (`SEPTIN9`, not `SEPT9`); stable IDs in `x` are dated instead. Without `species`, it is detected from stable IDs in `x`, and bare symbols are refused.

Renames beyond the old spelling come from the species' nomenclature authority: HGNC for human, MGI for mouse, ZFIN for zebrafish (`SEP15` to `SELENOF`, `sept9` to `septin9a`). A `mapping` without `species` is read as human nomenclature.

For a matrix, sparse `Matrix` or data frame, only row names are replaced; dimensions, row order and other attributes are preserved. Call `correct_genenames()` on `rownames(x)` to obtain the `corrected`, `ambiguous` and `unresolved` attributes.

Value

If `x` is a plain vector: a character vector the same length and order as `x`, carrying a `corrected` attribute (logical), an `ambiguous` attribute naming tokens that matched more than one symbol, and an `unresolved` attribute naming damaged tokens no symbol in this annotation produces. If `x` is a matrix, sparse `Matrix` or data frame: the same object with corrected row names and none of those attributes.

References

Ziemann M, Eren Y, El-Osta A (2016). Gene name errors are widespread in the scientific literature. *Genome Biology* 17:177.

Abeysooriya M, Soria M, Kasu MS, Ziemann M (2021). Gene name errors: Lessons not learned. *PLOS Computational Biology* 17(7):e1008984.

See Also

gene_names(), gene_ids(), geneid2name().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
correct_genenames(c("9-Sep", "1-Dec", "TP53"), mapping = hs)
m <- matrix(1:4, nrow = 2, dimnames = list(c("9-Sep", "TP53"), c("s1", "s2")))
correct_genenames(m, mapping = hs)

# release 95 resolves these as SEPT9 and DEC1
correct_genenames(c("9-Sep", "1-Dec", "TP53"), species = "human", release = 95)

# release 116 resolves these as SEPTIN9 and DELEC1
correct_genenames(c("9-Sep", "1-Dec", "TP53"), species = "human", release = 116)

# no release: the newest, so current spellings
correct_genenames(c("9-Sep", "1-Dec"), species = "human")

correct_genenames(c("1-Mar", "MTARC1"), species = "human", release = 116)
```

---

detect_release *Which Ensembl release a set of identifiers came from*

Description

Scores every release the index knows for this species, marks infeasible those that could not have produced the input, and ranks the rest by fit. See `vignette("how-it-works")` for the scoring.

Usage

```
detect_release(ids, species = NULL)

DetectRelease(ids, species = NULL)
```

Arguments

ids	Character vector of gene or transcript identifiers.
species	Species name, e.g. `"homo_sapiens"`. Detected when `NULL`. A best match at the oldest or newest indexed release raises a warning: the index covers a finite span, and a set from outside it can only be scored against the nearest edge.

Details

Transcript identifiers (ENSMUST..., FBtr...) are scored against a separate transcript index, which covers the commonly quantified species and GENCODE; for another species pass release to transcript2gene().

Value

A data frame of candidate releases, best first, with release, source (ensembl, gencode for GENCODE's primary-assembly gene set, or gencode_all for its scaffold- and patch-inclusive one), assembly, feasible, dist and n_index. Each assembly has its own row within a release. All candidates are returned because adjacent releases can have indistinguishable fingerprints.

Examples

```
ids <- readLines(system.file("extdata", "example_ids.txt.gz",
  package = "genevintage"
))
head(detect_release(ids), 3)
```

---

detect_species *Which species a set of identifiers belongs to*

Description

Matches identifier prefixes to indexed species.

Usage

```
detect_species(ids)

DetectSpecies(ids)
```

Arguments

ids	Character vector of gene or transcript identifiers.

Value

A data frame of candidate species ordered by the share of ids they explain, with columns species, common, division, prefix, scheme and frac. Rows can tie at the same frac: dog breeds and mouse strains share an identifier space. Check for ties when that distinction matters.

Examples

```
detect_species(c("ENSMUSG00000051951", "ENSMUSG00000089699"))

# the prefix identifies bonobo
detect_species("ENSPPAG00000021109")[1, c("species", "common", "frac")]
```

detect_species_names *Detect species from gene symbols*

Description

Scores each indexed species/release by informative symbol overlap. Common symbols are down-weighted and Excel-prone date-like symbols are excluded. Results below the overlap or score threshold have `status = "insufficient"`; remaining results are `"ambiguous"` unless the top candidate meets the margin.

Usage

```
detect_species_names(
  symbols,
  index,
  min_overlap = 20L,
  min_score = 0.25,
  min_margin = 0.05
)

DetectSpeciesNames(
  symbols,
  index,
  min_overlap = 20L,
  min_score = 0.25,
  min_margin = 0.05
)
```

Arguments

`symbols`	Character vector of gene names.
`index`	A symbol index from `build_symbol_index()`.
`min_overlap`	Minimum informative symbols required.
`min_score`	Minimum weighted overlap fraction.
`min_margin`	Required score gap between the top two candidates.

Value

Ranked data frame with `species`, `release`, `score`, `overlap`, `margin`, and `status`.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
hs$species <- "homo_sapiens"
hs$release <- "116"
symbols <- c("TP53", "BRCA1", "GAPDH", "ACTB", "XIST", "HBB", "MALAT1", "PTPRC")
detect_species_names(symbols, build_symbol_index(hs), min_overlap = 5)
```

entrez_ids — *NCBI Entrez identifiers for a set of genes*

Description

Takes gene identifiers or gene names and returns their Entrez identifiers, detecting the species the way the rest of the package does. Cross-references are streamed once per species and release, then cached.

Usage

```
entrez_ids(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

EntrezIDs(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids`	Gene identifiers, or gene names. Names need `species`, since a name does not say which species it came from.
`species`, `release`, `assembly`, `source`	Skip detection by naming them. `species` accepts anything `resolve_species()` does.
`mapping`	Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.
`quiet`	Suppress the message reporting what was detected.

Details

A gene may carry several Entrez identifiers or none. Each row is a pair; `entrez` is `NA` where Ensembl records no cross-reference.

Cross-references are published against Ensembl releases. Identifiers dated to a GENCODE release are looked up in the newest indexed Ensembl release for that species to obtain cross-references between stable identifiers.

Value

A data frame of `input`, `id`, `name` and `entrez`, one row per pair.

See Also

`fetch_xrefs()`, `gene_names()`.

Examples

```
entrez_ids(c("ENSG00000141510", "ENSG00000012048"))
entrez_ids(c("TP53", "BRCA1"), species = "human")
```

`fetch_mapping` *Mapping table for one species and Ensembl release*

Description

Streamed from the Ensembl GTF on first use, then cached, so later calls are offline. Gene versions are read from GTF attributes or identifier suffixes.

Usage

```
fetch_mapping(
  species,
  release,
  assembly = NULL,
  source = NULL,
  refresh = FALSE
)

FetchMapping(species, release, assembly = NULL, source = NULL, refresh = FALSE)
```

Arguments

`species`	Species, in any form `resolve_species()` accepts: `"homo_sapiens"`, `"Human"` or `"human"`.
`release`	Ensembl release number.
`assembly`	Assembly suffix as the index records it, e.g. `"38"` or `"37"`. Together with `species`, `source` and `release` it selects the index row that carries the annotation's FTP root and, where an assembly is frozen, the release its annotation actually stopped at.

source	Annotation family: "ensembl", or "gencode" / "gencode_all" for GENCODE's primary and scaffold-inclusive gene sets. Needed only where one release number exists in more than one of them.
refresh	Re-download even when a cached copy exists.

Value

A data frame of id, name, chr, start, end, strand, span, id_version and biotype, suitable as the mapping argument of gene_names(). span is the genomic extent from first to last base; TPM and FPKM require exonic length. See stream_gtf().

Examples

```
m <- fetch_mapping("yeast", release = 116)
head(m)

# human release 112 ships two assemblies, so one must be named
fetch_mapping("human", release = 112, assembly = "38")
```

---

fetch_orthologs *Ortholog table for one species pair and Ensembl release*

Description

Streamed from Ensembl Compara on first use, then cached, so later calls are offline. Compara is Ensembl's, so release is an Ensembl release even when the identifiers came from GENCODE.

Usage

```
fetch_orthologs(
  species,
  to,
  release,
  kind = c("protein", "ncrna"),
  refresh = FALSE
)

FetchOrthologs(
  species,
  to,
  release,
  kind = c("protein", "ncrna"),
  refresh = FALSE
)
```

Arguments

species	Ensembl species the identifiers come from.
to	Species to find orthologs in, in any form `resolve_species()` accepts.
release	Ensembl release number.
kind	`"protein"` (the default) or `"ncrna"`: Compara publishes the two trees separately, and non-coding orthologs live only in the second.
refresh	Re-download even when a cached copy exists.

Value

A data frame of `id`, `ortholog_id`, `homology_type`, `identity`, `ortholog_identity`, `high_confidence`.

Examples

```
# first call per pair streams a Compara file and caches it; then offline
head(fetch_orthologs("human", to = "mouse", release = 116))
```

---

fetch_transcripts *Transcript-to-gene table for one species and release*

Description

Streamed from the annotation's GTF on first use, then cached, so later calls are offline.

Usage

```
fetch_transcripts(
  species,
  release,
  assembly = NULL,
  source = NULL,
  refresh = FALSE
)

FetchTranscripts(
  species,
  release,
  assembly = NULL,
  source = NULL,
  refresh = FALSE
)
```

Arguments

species	Species, in any form resolve_species() accepts: "homo_sapiens", "Human" or "human".
release	Ensembl release number.
assembly	Assembly suffix as the index records it, e.g. "38" or "37". Together with species, source and release it selects the index row that carries the annotation's FTP root and, where an assembly is frozen, the release its annotation actually stopped at.
source	Annotation family: "ensembl", or "gencode" / "gencode_all" for GENCODE's primary and scaffold-inclusive gene sets. Needed only where one release number exists in more than one of them.
refresh	Re-download even when a cached copy exists.

Value

A data frame of transcript_id, id and name, as stream_transcripts() returns it.

See Also

transcript2gene() to map a set of transcript identifiers.

Examples

```
head(fetch_transcripts("chicken", release = 116))
```

---

fetch_xrefs *Ensembl gene identifiers and their cross-references, cached*

Description

Streams the cross-reference TSV Ensembl publishes for a species and release, keeps the gene-level pairs and caches them, so later calls are offline.

Usage

```
fetch_xrefs(species, release, db = "entrez", refresh = FALSE)

FetchXrefs(species, release, db = "entrez", refresh = FALSE)
```

Arguments

species	Species in any form resolve_species() accepts.
release	Ensembl release number for the cross-reference files.
db	Cross-reference database. "entrez" is the only one indexed.
refresh	Refetch and overwrite the cached copy.

Value

A data frame of `id` and `xref`, one row per distinct pair.

See Also

`entrez_ids()` for the lookup this feeds.

Examples

```
head(fetch_xrefs("yeast", release = 116))
```

---

gene_biotypes *What biotypes an annotation contains*

Description

Use this to see what is there before filtering: the vocabulary differs by species and drifts between releases.

Usage

```
gene_biotypes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

GeneBiotypes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids`	Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

species, release, assembly, source
: Skip detection by naming them. species accepts anything resolve_species() does.

mapping
: Supply an annotation directly to bypass detection and downloading. A data frame with at least id, name, chr and biotype, as fetch_mapping() returns; supports local GTFs and custom builds.

quiet
: Suppress the message reporting what was detected.

Value

A data frame of biotype, class and n, commonest first.

See Also

genes_by_biotype(), coding_genes(), noncoding_genes().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
gene_biotypes(mapping = hs)

gene_biotypes(species = "human")
```

---

gene_cache *Inspect or clear the download cache*

Description

Mapping tables and ortholog pairs are streamed once and cached under tools::R_user_dir("genevintage", "cache"), so a species you have used once works offline afterwards. This reports what is cached and removes it.

Usage

```
gene_cache(clear = FALSE)

GeneCache(clear = FALSE)
```

Arguments

clear
: Remove cached files. TRUE removes all of them; a number removes those not used in that many days.

Details

The location follows R's own convention and can be moved by setting the R_USER_CACHE_DIR environment variable before the package is used.

Interrupted writes leave .rds.tmp files, listed as partial and eligible for removal.

Value

A data frame with one row per cached file (`file`, `kind`, `species`, `to`, `release`, `bytes`, `modified`), invisibly when `clear` is not `FALSE`. `to` is the second species of an ortholog pair and is `NA` for a mapping. Zero rows when nothing is cached.

Examples

```
# a scratch cache, so the example leaves yours alone
old <- Sys.getenv("R_USER_CACHE_DIR")
Sys.setenv(R_USER_CACHE_DIR = tempfile())
gene_cache() # what is cached, and how large
gene_cache(clear = 90) # drop anything untouched for 90 days
gene_cache(clear = TRUE) # drop all of it
Sys.setenv(R_USER_CACHE_DIR = old)
```

---

gene_ids — *Gene identifiers for gene names or Entrez identifiers*

Description

The reverse of `gene_names()`. Published marker lists and enrichment results carry symbols or Entrez identifiers; count matrices carry Ensembl ones.

Usage

```
gene_ids(
  x,
  from = c("auto", "name", "entrez"),
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

GeneIDs(
  x,
  from = c("auto", "name", "entrez"),
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

x	Gene names, or Entrez identifiers.
from	What x holds: "name", "entrez", or "auto" to decide from the input. All-digit input is Entrez; anything else is a name.
species, release, assembly, source	Skip detection by naming them. species accepts anything resolve_species() does.
mapping	Supply an annotation directly to bypass detection and downloading. A data frame with at least id, name, chr and biotype, as fetch_mapping() returns; supports local GTFs and custom builds.
quiet	Suppress the message reporting what was detected.

Details

Symbols and Entrez identifiers can each match multiple genes, so the table returns every match. Unmatched inputs are retained with id = NA.

Value

A data frame of input, id and name, one row per match.

See Also

gene_names() for the forward direction, entrez_ids().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
gene_ids(c("TP53", "BRCA1"), mapping = hs)

gene_ids(c("TP53", "BRCA1"), species = "human")
gene_ids(c("7157", "672"), from = "entrez", species = "human")
```

---

gene_lengths *Exon-union gene lengths for one species and release*

Description

Exon-union gene lengths for one species and release

Usage

```
gene_lengths(
  species,
  release,
  ids = NULL,
  assembly = NULL,
  source = NULL,
  refresh = FALSE
)

GeneLengths(
  species,
  release,
  ids = NULL,
  assembly = NULL,
  source = NULL,
  refresh = FALSE
)
```

Arguments

species	Species, in any form resolve_species() accepts: "homo_sapiens", "Human" or "human".
release	Ensembl release number.
ids	Optional gene identifiers or names to return lengths for, in order. Versioned identifiers match their unversioned record. Names are matched through fetch_mapping(); a name shared by several genes gets NA.
assembly	Assembly suffix as the index records it, e.g. "38" or "37". Together with species, source and release it selects the index row that carries the annotation's FTP root and, where an assembly is frozen, the release its annotation actually stopped at.
source	Annotation family: "ensembl", or "gencode" / "gencode_all" for GENCODE's primary and scaffold-inclusive gene sets. Needed only where one release number exists in more than one of them.
refresh	Re-download even when a cached copy exists.

Value

Without ids, a data frame of id and length for every gene in the annotation. With ids, a named integer vector the length of ids, NA where nothing matched.

See Also

fetch_mapping() for genomic span, stream_exon_lengths().

Examples

```
gene_lengths("human", release = 116, ids = c("ENSG00000141510", "TP53"))
```

`gene_names` *Convert gene identifiers to gene names*

Description

Returns one value per input, in the input's order. An identifier with no name in `mapping` is returned unchanged, so the result is always usable as row labels.

Usage

```
gene_names(ids, mapping, unique = FALSE, warn = TRUE)

GeneNames(ids, mapping, unique = FALSE, warn = TRUE)
```

Arguments

`ids`	Character vector of gene identifiers.
`mapping`	A data frame with `id` and `name` columns, or a named character vector of names indexed by identifier.
`unique`	Make the result unique by appending the identifier to names that would otherwise collide. Off by default, since it changes names.
`warn`	Warn when names collide or when little of the input mapped. The collision warning carries class `genevintage_duplicate_names`, so a caller that disambiguates downstream can catch and muffle just that one.

Details

Version suffixes are ignored when matching, so `ENSG00000141510.16` resolves against an unversioned mapping and the other way round.

Value

A character vector the same length as `ids`, carrying a `mapped` attribute: the logical vector of which inputs found a real name.

See Also

`fetch_mapping()` to obtain `mapping` for a species and release.

Examples

```
m <- data.frame(
  id = c("ENSG00000141510", "ENSG00000284733"),
  name = c("TP53", "ENSG00000284733")
) # second has no symbol
gene_names(c("ENSG00000141510.16", "ENSG00000284733", "ENSG00000999999"), m)
#> "TP53" "ENSG00000284733" "ENSG00000999999"
```

`geneid2name` *Convert gene identifiers to gene names, detecting species and release*

Description

Detects the species from the identifier prefixes, ranks the Ensembl releases consistent with the identifier set, fetches that release's mapping (streamed once, then cached) and converts. Identifiers with no gene name are returned unchanged.

Usage

```
geneid2name(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  unique,
  quiet = FALSE,
  row_ids = NULL
)

GeneID2Name(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  unique,
  quiet = FALSE,
  row_ids = NULL
)
```

Arguments

`ids`	Character vector of gene identifiers.

species	Skip species detection by naming it. Any form resolve_species() accepts: "homo_sapiens", "Homo sapiens", "Human" or "human".
release	Skip release detection by naming it.
assembly	Assembly suffix, e.g. "37" for human GRCh37. Detected alongside the release when both are left NULL.
source	Annotation family: "ensembl", or "gencode" / "gencode_all" for GENCODE's primary and scaffold-inclusive gene sets. Detected alongside the release.
mapping	Skip detection and use this id/name table, as accepted by gene_names().
unique	Passed to gene_names(). Defaults to TRUE when ids is a matrix, sparse Matrix or data frame, and to FALSE otherwise.
quiet	Suppress the message reporting what was detected.
row_ids	When ids is a matrix, sparse Matrix or data frame, overrides which identifiers to detect and map. Defaults to rownames(ids); set this when ids has none.

Value

If ids is a plain vector: a character vector the same length as ids, carrying mapped, species, source, release and assembly attributes. If ids is a matrix, sparse Matrix or data frame: the same object with only its row names replaced by gene names; dimensions, row order and other attributes are preserved.

See Also

detect_species(), detect_release(), gene_names().

Examples

```
# an excerpt of human Ensembl 116; drop `mapping` to detect and fetch the real one
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
ids <- c("ENSG00000141510", "ENSG00000284616", "ENSG00000012048")
geneid2name(ids, mapping = hs) # the unnamed lncRNA keeps its id

m <- matrix(1:6, nrow = 3, dimnames = list(ids, c("s1", "s2")))
geneid2name(m, mapping = hs)

geneid2name(ids) # species and release detected, then fetched
```

---

genes_by_biotype *Genes of a given biotype, or class of biotypes*

Description

Genes of a given biotype, or class of biotypes

Usage

```
genes_by_biotype(
  biotype,
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

GenesByBiotype(
  biotype,
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`biotype` One or more Ensembl biotypes (`"lncRNA"`, `"miRNA"`, `"snoRNA"`, ...) or class names: `"coding"`, `"noncoding"`, `"pseudogene"`, `"immune"` (immunoglobulin and T-cell receptor segments) or `"other"`. `gene_biotypes()` lists what an annotation actually has.

`ids` Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

`species`, `release`, `assembly`, `source` Skip detection by naming them. `species` accepts anything `resolve_species()` does.

`mapping` Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.

`quiet` Suppress the message reporting what was detected.

Value

A data frame of `id`, `name`, `chr` and `biotype`, shaped as `mito_genes()` returns.

See Also

`gene_biotypes()` to see the vocabulary first.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
genes_by_biotype(c("miRNA", "snoRNA"), mapping = hs)
genes_by_biotype("pseudogene", mapping = hs) # a whole class

genes_by_biotype("lncRNA", species = "human")
```

---

guess_scheme *Which identifier scheme a vector of IDs uses*

Description

Which identifier scheme a vector of IDs uses

Usage

```
guess_scheme(ids, min_purity = 0.5)

GuessScheme(ids, min_purity = 0.5)
```

Arguments

ids	Character vector of identifiers.
min_purity	Smallest matching fraction to accept a scheme.

Value

A list with `scheme`, `purity` and the per-scheme fractions.

Examples

```
guess_scheme(c("ENSG00000141510", "ENSG00000012048"))$scheme
guess_scheme(c("TP53", "BRCA1"))$scheme

# purity is how a mixed vector announces itself
guess_scheme(c("ENSG00000141510", "TP53"))$purity
```

hemoglobin_genes *Hemoglobin subunit genes*

Description

The set behind a haemoglobin-contamination metric in single-cell work, beside `mito_genes()` and `ribosomal_genes()`. Named per clade: `HBA1/HBB` in human, `Hba-a1/Hbb-bs` in mouse, `hbaa1/hbba1` in zebrafish.

Usage

```
hemoglobin_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

HemoglobinGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids`	Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.
`species`, `release`, `assembly`, `source`	Skip detection by naming them. `species` accepts anything `resolve_species()` does.
`mapping`	Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.
`quiet`	Suppress the message reporting what was detected.

Details

Subunit-letter matching excludes HBEGF, HBS1L and HBP1. Requiring protein_coding also excludes haemoglobin pseudogenes such as HBAP1, HBBP1 and HBZP1.

Value

A data frame of id, name, chr and biotype, shaped as mito_genes() returns.

See Also

mito_genes(), ribosomal_genes().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
hemoglobin_genes(mapping = hs)
```

---

lncrna_genes *Long non-coding RNA genes*

Description

Matches lncRNA biotype spellings across species and releases, including lncRNA, lincRNA, antisense, processed_transcript and 3prime_overlapping_ncrna.

Usage

```
lncrna_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

lncRNAGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

- `ids` — Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.
- `species, release, assembly, source` — Skip detection by naming them. `species` accepts anything `resolve_species()` does.
- `mapping` — Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.
- `quiet` — Suppress the message reporting what was detected.

Details

Generic `ncRNA` entries are excluded. If the annotation contains them and the subset has no lncRNA matches, a message points to `genes_by_biotype("ncRNA")`.

Value

A data frame of `id`, `name`, `chr` and `biotype`, shaped as `mito_genes()` returns.

See Also

`noncoding_genes()` for every non-coding class, `gene_biotypes()`.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
lncrna_genes(mapping = hs)

nrow(lncrna_genes(species = "human"))
table(lncrna_genes(species = "zebrafish")$biotype) # antisense and lincRNA
```

---

mhc_genes — *Major histocompatibility complex genes*

Description

The MHC, which human calls HLA. Every clade names it differently: `HLA-A` in human, `H2-K1` in mouse, `mhc1uba` in zebrafish, so matching `"^HLA-"` finds nothing outside primates.

Usage

```
mhc_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

MHCGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids` Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

`species`, `release`, `assembly`, `source` Skip detection by naming them. `species` accepts anything `resolve_species()` does.

`mapping` Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.

`quiet` Suppress the message reporting what was detected.

Details

Requires `protein_coding` and an MHC family name pattern. The patterns exclude regional genes such as `B2M`, `TAP1`, `TAP2` and `MICA`. Annotations without matching genes return an empty result.

Value

A data frame of `id`, `name`, `chr` and `biotype`, shaped as `mito_genes()` returns.

See Also

`genes_by_biotype()` for the immunoglobulin and T-cell receptor segments, which are a different question.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
mhc_genes(mapping = hs)

mhc_genes(species = "human") # coding HLA genes
mhc_genes(species = "mouse") # H2-K1, H2-D1, H2-Aa, ...
```

`mito_genes` *Mitochondrial genes*

Description

Selects genes by mitochondrial sequence names in the annotation for use in mitochondrial count fractions. Sequence-name matching supports the naming conventions of different species.

Usage

```
mito_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

MitoGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids` Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

`species`, `release`, `assembly`, `source` Skip detection by naming them. `species` accepts anything `resolve_species()` does.

mapping	Supply an annotation directly to bypass detection and downloading. A data frame with at least id, name, chr and biotype, as fetch_mapping() returns; supports local GTFs and custom builds.
quiet	Suppress the message reporting what was detected.

Value

A data frame of id, name, chr and biotype, one row per mitochondrial gene. When ids were given, an input column comes first holding the caller's own token, identifier or gene name as passed, so rownames(m) %in% mito_genes(rownames(m))$input subsets a matrix directly whichever kind its row names are.

See Also

sex_genes() for the sex chromosomes.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
counts <- matrix(1:72, ncol = 2, dimnames = list(hs$id, c("s1", "s2")))
mito_genes(mapping = hs)
mt <- mito_genes(rownames(counts), mapping = hs)
colSums(counts[mt$input, ]) / colSums(counts) # the mitochondrial fraction

mito_genes(species = "human") # the mitochondrial genes
```

---

noncoding_genes *Non-coding RNA genes*

Description

Genes transcribing functional RNA: lncRNA, miRNA, snRNA, snoRNA, rRNA and the rest. Pseudogenes are a separate class; ask for "pseudogene" through genes_by_biotype() when you want them.

Usage

```
noncoding_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

```
NoncodingGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

- `ids` — Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.
- `species`, `release`, `assembly`, `source` — Skip detection by naming them. `species` accepts anything `resolve_species()` does.
- `mapping` — Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.
- `quiet` — Suppress the message reporting what was detected.

Value

A data frame of `id`, `name`, `chr` and `biotype`.

See Also

`coding_genes()`, `genes_by_biotype()`.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
table(noncoding_genes(mapping = hs)$biotype)
```

`ortholog_species` *Species Compara publishes homologies for*

Description

Lists the species with Compara files for a release and division so callers can check availability before streaming a pair.

Usage

```
ortholog_species(release, division = "vertebrates")

OrthologSpecies(release, division = "vertebrates")
```

Arguments

release	Ensembl release number. Under Ensembl Genomes, that division's own release number.
division	Which division's Compara to list. Defaults to vertebrates.

Value

A character vector of species names.

Examples

```
"mus_musculus" %in% ortholog_species(116)
```

---

orthologs *Find orthologs of gene identifiers in another species*

Description

Detects the species from identifiers and defaults to its newest indexed Ensembl release. Fetches that release's Compara orthologs against `to` (streamed once, then cached) and returns the orthologous identifiers with their gene names.

Usage

```
orthologs(
  ids,
  to,
  species = NULL,
  release = NULL,
  one2one = TRUE,
  names = TRUE,
  kind = c("protein", "ncrna"),
  quiet = FALSE
)

Orthologs(
  ids,
  to,
  species = NULL,
  release = NULL,
```

```
  one2one = TRUE,
  names = TRUE,
  kind = c("protein", "ncrna"),
  quiet = FALSE
)
```

Arguments

ids	Character vector of gene identifiers or gene names. A symbol can label multiple genes, so all matches are included. Supply species for input containing only names.
to	Species to find orthologs in, in any form resolve_species() accepts: "mus_musculus", "Mouse" or "mouse".
species, release	Skip detection by naming them.
one2one	Keep only one-to-one orthologs. See details.
names	Also look up gene names for both sides of the pair, which costs a streamed mapping for each species. Both are named from the same Ensembl release, so the two halves of a row describe one vintage.
kind	Passed to fetch_orthologs().
quiet	Suppress the message reporting what was detected.

Details

A gene can have several orthologs or none. With one2one = TRUE (the default) only one-to-one orthologs are returned, so there is at most one row per input identifier and the result can be used directly as a lookup. Set it to FALSE to see one-to-many and many-to-many relationships as well.

Value

A data frame with one row per ortholog relationship: id and name for the query gene, ortholog_id and ortholog_name for its counterpart in to (the two name columns when names), then homology_type, identity, ortholog_identity and high_confidence. Identifiers with no ortholog in to are absent, so the result is usually shorter than ids.

See Also

geneid2name() for names within one species.

Examples

```
orthologs(c("ENSG00000141510", "ENSG00000012048"), to = "mus_musculus")
```

parse_ens — *Parse Ensembl identifiers into their parts*

Description

Handles unversioned and versioned IDs in one vector, _PAR_Y-style tags, and non-Ensembl input (returned as NA rows).

Usage

```
parse_ens(x)

ParseEns(x)
```

Arguments

x	Character vector of identifiers.

Value

A data frame with one row per input: id, prefix, feature, num, id_version, tag.

Examples

```
parse_ens(c(
  "ENSG00000141510.16", "ENSMUSG00000059552",
  "ENSG00000182378.14_PAR_Y", "TP53"
))
```

qmax_violations — *Where the index violates its own ordering invariant*

Description

Identifier numbers are assigned upward, so qmax should not fall as releases advance within one species and assembly. Ensembl retires genes at the top of the range occasionally, so only a fall of more than 1% is reported. Only numerically ordered schemes are checked.

Usage

```
qmax_violations(fp = .fp())

QmaxViolations(fp = .fp())
```

Arguments

fp	A fingerprint index. Defaults to the shipped one.

Value

A character vector of violation messages, empty when the index is sound.

Examples

```
# report ordering violations in the shipped index
qmax_violations()
```

---

resolve_species *Resolve a species name*

Description

Accepts the Ensembl name (`"homo_sapiens"`), the scientific name (`"Homo sapiens"`), Ensembl's display name (`"Zebrafish"`, `"Giant panda"`) or a common shorthand (`"human"`, `"mouse"`, `"rat"`, `"fly"`, `"yeast"`). Matching ignores case, punctuation and underscores.

Usage

```
resolve_species(x, quiet = FALSE)

ResolveSpecies(x, quiet = FALSE)
```

Arguments

`x`	A species name in any of the accepted forms.
`quiet`	Suppress the message reporting a partial match.

Details

An unambiguous partial match is accepted and reported; an ambiguous one is refused with the candidates. The `"mouse"` alias selects `mus_musculus` even when strain genomes also match the word.

Value

The Ensembl species name, e.g. `"homo_sapiens"`.

Examples

```
resolve_species("human")
resolve_species("Zebrafish")
```

ribosomal_genes *Ribosomal genes*

Description

`which` selects the ribosomal set:

Usage

```
ribosomal_genes(
  ids = NULL,
  which = c("protein", "rrna", "mito", "all"),
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

RibosomalGenes(
  ids = NULL,
  which = c("protein", "rrna", "mito", "all"),
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids` Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

`which` Which ribosomal set to return. See details.

`species`, `release`, `assembly`, `source` Skip detection by naming them. `species` accepts anything `resolve_species()` does.

`mapping` Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.

`quiet` Suppress the message reporting what was detected.

Details

- "protein" (the default): the genes encoding the ribosome's proteins, the set behind a single-cell ribosomal-content metric.
- "rrna": the RNA components, by biotype (rRNA and Mt_rRNA).
- "mito": the mitoribosome's proteins, named with an M prefix. These are nuclear genes; mito_genes() does not return them.
- "all": every one of the above.

The "^RP[SL]" symbol pattern is restricted to protein_coding genes to exclude pseudogenes and lncRNAs. A name rule also excludes S6 kinases (RPS6KA1 to RPS6KL1) and RPS19BP1.

Value

A data frame of id, name, chr and biotype, shaped as mito_genes() returns.

See Also

mito_genes(), genes_by_biotype().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
counts <- matrix(1:72, ncol = 2, dimnames = list(hs$id, c("s1", "s2")))
rb <- ribosomal_genes(rownames(counts), mapping = hs)
colSums(counts[rb$input, ]) / colSums(counts) # ribosomal content
ribosomal_genes(mapping = hs, which = "rrna")

ribosomal_genes(species = "human", which = "rrna")
```

---

rrna_genes *Ribosomal RNA genes*

Description

The RNA components of the ribosome, by biotype: rRNA in the nucleus and Mt_rRNA in the mitochondrion.

Usage

```
rrna_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
```

```
  quiet = FALSE
)

rRNAGenes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

`ids` Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, `species` must be named and the whole mitochondrial gene set for its newest indexed release is returned.

`species`, `release`, `assembly`, `source` Skip detection by naming them. `species` accepts anything `resolve_species()` does.

`mapping` Supply an annotation directly to bypass detection and downloading. A data frame with at least `id`, `name`, `chr` and `biotype`, as `fetch_mapping()` returns; supports local GTFs and custom builds.

`quiet` Suppress the message reporting what was detected.

Details

`rRNA_pseudogene` belongs with the pseudogenes. Ask for it with `genes_by_biotype("rRNA_pseudogene")`.

Value

A data frame of `id`, `name`, `chr` and `biotype`, shaped as `mito_genes()` returns.

See Also

`ribosomal_genes()` for the ribosome's *proteins*, `trna_genes()`.

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
rrna_genes(mapping = hs)

rrna_genes(species = "human")
```

sex_genes *Sex-chromosome genes*

Description

Sex chromosome names vary by clade: mammals use X and Y, birds Z and W, and C. elegans X. Annotations without recognised sex chromosomes return no rows.

Usage

```
sex_genes(
  ids = NULL,
  which = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

SexGenes(
  ids = NULL,
  which = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

ids	Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, species must be named and the whole mitochondrial gene set for its newest indexed release is returned.
which	Restrict to one chromosome: "X", "Y", "Z" or "W". Defaults to every sex chromosome the annotation has.
species, release, assembly, source	Skip detection by naming them. species accepts anything resolve_species() does.
mapping	Supply an annotation directly to bypass detection and downloading. A data frame with at least id, name, chr and biotype, as fetch_mapping() returns; supports local GTFs and custom builds.
quiet	Suppress the message reporting what was detected.

Value

A data frame of id, name, chr and biotype, one row per gene, shaped as mito_genes() returns.

See Also

mito_genes().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
sex_genes(mapping = hs, which = "Y")

sex_genes(species = "human", which = "Y") # Y-linked genes
```

---

species_table *Species the index covers, with their common names*

Description

Species the index covers, with their common names

Usage

```
species_table(pattern = NULL)

SpeciesTable(pattern = NULL)
```

Arguments

pattern	Optional: keep only species whose Ensembl or common name matches, ignoring case and punctuation.

Value

A data frame of species, common, taxonomy_id and division.

Examples

```
head(species_table("mouse"))
```

stream_exon_lengths *Exon-union lengths from a GTF, streamed without writing it to disk*

Description

Keeps only exon lines. Gene ids are read from the gene_id attribute, so a GENCODE GTF yields versioned ids, as its gene lines do in stream_gtf().

Usage

```
stream_exon_lengths(url, chunk = 1e+05)

StreamExonLengths(url, chunk = 1e+05)
```

Arguments

url	A .gtf.gz URL.
chunk	Lines to decompress per iteration.

Value

A data frame of id and length (bases).

Examples

```
stream_exon_lengths(paste0(
  "https://ftp.ensembl.org/pub/release-116/gtf/",
  "saccharomyces_cerevisiae/",
  "Saccharomyces_cerevisiae.R64-1-1.63.gtf.gz"
))
```

stream_gtf *Pull gene records out of a GTF without ever writing it to disk*

Description

Decompresses the stream in chunks and retains gene lines. If there are none, reads the first record per gene from the remaining feature lines.

Usage

```
stream_gtf(url, chunk = 1e+05)

StreamGTF(url, chunk = 1e+05)
```

Arguments

`url`	A `.gtf.gz` URL.
`chunk`	Lines to decompress per iteration.

Value

A data frame of `id`, `name`, `chr`, `start`, `end`, `strand`, `span`, `id_version` and `biotype`. `span` is the genomic extent from first to last base; TPM and FPKM require exonic length.

Examples

```
# any Ensembl-shaped GTF; nothing is written to disk
stream_gtf(paste0(
  "https://ftp.ensembl.org/pub/release-116/gtf/",
  "saccharomyces_cerevisiae/",
  "Saccharomyces_cerevisiae.R64-1-1.63.gtf.gz"
))
```

`stream_homologies` *Stream one species' orthologs against one other species*

Description

Filters the Compara stream in chunks to retain orthologs for the requested target species, limiting memory to the selected pair.

Usage

```
stream_homologies(url, to, swap = FALSE, chunk = 2e+05)

StreamHomologies(url, to, swap = FALSE, chunk = 2e+05)
```

Arguments

`url`	A Compara `homologies.tsv.gz` URL.
`to`	Ensembl species name to keep, e.g. `"mus_musculus"`.
`swap`	Reverse the pair when reading it from the target species' file, including the direction of one-to-many relationships.
`chunk`	Lines to decompress per iteration.

Value

A data frame of `id`, `ortholog_id`, `homology_type`, `identity`, `ortholog_identity`, `high_confidence`.

Examples

```
url <- paste0(
  "https://ftp.ensembl.org/pub/release-116/tsv/ensembl-compara/",
  "homologies/mus_musculus/",
  "Compara.116.protein_default.homologies.tsv.gz"
)
head(stream_homologies(url, to = "homo_sapiens"))
```

`stream_transcripts` *Transcript-to-gene records from a GTF, streamed without writing it to disk*

Description

Keeps `transcript` lines. GTFs before Ensembl release 75 have none, so a chunk without them is read from its exon and CDS lines, which carry the same `transcript_id`, `gene_id` and `gene_name`; duplicates are dropped.

Usage

```
stream_transcripts(url, chunk = 1e+05)

StreamTranscripts(url, chunk = 1e+05)
```

Arguments

`url`	A `.gtf.gz` URL.
`chunk`	Lines to decompress per iteration.

Value

A data frame of `transcript_id`, `id` (the gene) and `name` (the gene name, `NA` where the annotation gives none). Identifiers keep whatever version suffix the GTF writes: GENCODE versions them, Ensembl does not.

See Also

`fetch_transcripts()` for the cached table of one release.

Examples

```
stream_transcripts(paste0(
  "https://ftp.ensembl.org/pub/release-116/gtf/",
  "saccharomyces_cerevisiae/",
  "Saccharomyces_cerevisiae.R64-1-1.63.gtf.gz"
))
```

stream_xrefs — *Stream one cross-reference TSV, keeping the gene-level pairs*

Description

Deduplicate gene-level pairs from transcript rows while streaming to limit memory to distinct pairs.

Usage

```
stream_xrefs(url, db = "EntrezGene", chunk = 2e+05)

StreamXrefs(url, db = "EntrezGene", chunk = 2e+05)
```

Arguments

`url`	A `.tsv.gz` cross-reference URL.
`db`	The `db_name` to keep, e.g. `"EntrezGene"`.
`chunk`	Lines to decompress per iteration.

Value

A data frame of `id` and `xref`.

Examples

```
stream_xrefs(paste0(
  "https://ftp.ensembl.org/pub/release-116/tsv/saccharomyces_cerevisiae/",
  "Saccharomyces_cerevisiae.R64-1-1.63.entrez.tsv.gz"
))
```

transcript2gene — *Map transcript identifiers to their genes*

Description

Detects the species from the identifier prefix (`ENSMUST`, `ENSGALT`, ...), fetches the annotation's transcript table (streamed once, then cached) and returns each transcript's gene identifier and gene name. Version suffixes are ignored when matching, so `ENSMUST00000178537.2` resolves against an unversioned annotation and the other way round.

Usage

```
transcript2gene(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

Transcript2Gene(
  ids,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

ids	Character vector of transcript identifiers.
species	Skip species detection by naming it. Any form `resolve_species()` accepts: `"homo_sapiens"`, `"Homo sapiens"`, `"Human"` or `"human"`.
release	Annotation release. Without it, chosen as above.
assembly	Assembly suffix, e.g. `"37"` for human GRCh37. Detected alongside the release when both are left `NULL`.
source	Annotation family: `"ensembl"`, or `"gencode"` / `"gencode_all"` for GENCODE's primary and scaffold-inclusive gene sets. Detected alongside the release.
mapping	Skip fetching and use this table: `fetch_transcripts()` output, or a data frame with `transcript_id`, `gene_id` and `gene_name`.
quiet	Suppress the message reporting what was detected.

Details

Without `release`, `detect_release()` dates the transcript identifiers against a transcript fingerprint index of Ensembl and GENCODE for the commonly quantified species. `source` and `assembly` narrow the candidates.

When the best-scoring release leaves any id unmapped, the best release of each other annotation family (Ensembl, GENCODE, GENCODE ALL) is fetched too and the best-covering one kept: a filtered transcriptome (cDNA only) dates by its newest id but leaves its family open.

A species outside that index, a set too small to date (a handful of ids), or dated releases explaining less than 90% of the input fall back to coverage: the newest Ensembl release of each assembly is tried, newest first, and the best-covering one kept. Chicken needs this: GRCg7b (release 107 on) renumbered every identifier, so GRCg6a transcripts only map at release 106 or earlier.

Value

A data frame with one row per input, in order: `input`, `transcript_id` (as the annotation writes it), `gene_id` and `gene_name`. Unmapped inputs have `NA` in the last three. A gene without a name has its identifier as `gene_name`. Attributes `species`, `source`, `release` and `assembly` record the annotation used.

See Also

`aggregate_transcripts()` to sum a transcript-level matrix to genes.

Examples

```
tx <- data.frame(
  transcript_id = c("ENST00000269305", "ENST00000357654"),
  id = c("ENSG00000141510", "ENSG00000012048"), name = c("TP53", "BRCA1")
)
transcript2gene(c("ENST00000269305.9", "ENST00000357654.9"), mapping = tx)

transcript2gene(c("ENSMUST00000178537.2", "ENSMUST00000178862.2"))
transcript2gene(c("ENSGALT00000000003", "ENSGALT00000000004"))
```

---

trna_genes *Transfer RNA genes*

Description

Ensembl names them `Mt_tRNA` on the mitochondrial chromosome and `tRNA` in the nucleus. Which of those an annotation actually contains varies: the vertebrate GTFs carry only the mitochondrial tRNAs, since nuclear tRNA genes are predicted separately and lie outside the gene set. Invertebrate, fungal and plant annotations include the nuclear ones. The `chr` and `biotype` columns say which kind each row is.

Usage

```
trna_genes(
  ids = NULL,
  species = NULL,
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)

tRNAGenes(
  ids = NULL,
  species = NULL,
```

```
  release = NULL,
  assembly = NULL,
  source = NULL,
  mapping = NULL,
  quiet = FALSE
)
```

Arguments

ids	Optional character vector of gene identifiers or gene names. When given, the species and release are detected from them and only these genes are considered. When omitted, species must be named and the whole mitochondrial gene set for its newest indexed release is returned.
species, release, assembly, source	Skip detection by naming them. species accepts anything resolve_species() does.
mapping	Supply an annotation directly to bypass detection and downloading. A data frame with at least id, name, chr and biotype, as fetch_mapping() returns; supports local GTFs and custom builds.
quiet	Suppress the message reporting what was detected.

Value

A data frame of id, name, chr and biotype, shaped as mito_genes() returns.

See Also

ribosomal_genes(), genes_by_biotype().

Examples

```
hs <- read.delim(system.file("extdata", "example_mapping.tsv", package = "genevintage"))
trna_genes(mapping = hs)

trna_genes(species = "human") # the mitochondrial tRNAs
table(trna_genes(species = "yeast")$chr) # nuclear, by chromosome
```

Index

aggregate_transcripts, 3
aggregate_transcripts(), *51*
AggregateTranscripts (aggregate_transcripts), 3
annotate_genes, 5
AnnotateGenes (annotate_genes), 5

build_symbol_index, 6
build_symbol_index(), *14*
build_symbol_index_from_gtf, 7
BuildSymbolIndex (build_symbol_index), 6
BuildSymbolIndexFromGTF (build_symbol_index_from_gtf), 7

coding_genes, 7
coding_genes(), *21*, *36*
CodingGenes (coding_genes), 7
compare_releases, 9
CompareReleases (compare_releases), 9
correct_genenames, 10
CorrectGeneNames (correct_genenames), 10

detect_release, 12
detect_release(), *10*, *27*, *50*
detect_species, 13
detect_species(), *27*
detect_species_names, 14
detect_species_names(), *7*
DetectRelease (detect_release), 12
DetectSpecies (detect_species), 13
DetectSpeciesNames (detect_species_names), 14

entrez_ids, 15
entrez_ids(), *20*, *23*
EntrezIDs (entrez_ids), 15

fetch_mapping, 16
fetch_mapping(), *5*, *6*, *8*, *9*, *11*, *15*, *21*, *23–25*, *28*, *30*, *32*, *33*, *35*, *36*, *41*, *43*, *44*, *52*
fetch_orthologs, 17
fetch_orthologs(), *38*
fetch_transcripts, 18
fetch_transcripts(), *4*, *48*, *50*
fetch_xrefs, 19
fetch_xrefs(), *16*
FetchMapping (fetch_mapping), 16
FetchOrthologs (fetch_orthologs), 17
FetchTranscripts (fetch_transcripts), 18
FetchXrefs (fetch_xrefs), 19

gene_biotypes, 20
gene_biotypes(), *28*, *32*
gene_cache, 21
gene_ids, 22
gene_ids(), *12*
gene_lengths, 23
gene_names, 25
gene_names(), *6*, *11*, *12*, *16*, *17*, *22*, *23*, *27*
GeneBiotypes (gene_biotypes), 20
GeneCache (gene_cache), 21
GeneID2Name (geneid2name), 26
geneid2name, 26
geneid2name(), *4*, *12*, *38*
GeneIDs (gene_ids), 22
GeneLengths (gene_lengths), 23
GeneNames (gene_names), 25
genes_by_biotype, 27
genes_by_biotype(), *7*, *8*, *21*, *33*, *35*, *36*, *42*, *52*
GenesByBiotype (genes_by_biotype), 27
guess_scheme, 29
GuessScheme (guess_scheme), 29

hemoglobin_genes, 30
HemoglobinGenes (hemoglobin_genes), 30

lncrna_genes, 31
lncRNAGenes (lncrna_genes), 31

make.unique(), *11*
mhc_genes, 32
MHCGenes (mhc_genes), 32
mito_genes, 34
mito_genes(), *28*, *30–33*, *42*, *43*, *45*, *52*
MitoGenes (mito_genes), 34

noncoding_genes, 35
noncoding_genes(), *8*, *21*, *32*
NoncodingGenes (noncoding_genes), 35

ortholog_species, 36
Orthologs (orthologs), 37
orthologs, 37
OrthologSpecies (ortholog_species), 36

parse_ens, 39
ParseEns (parse_ens), 39

qmax_violations, 39
QmaxViolations (qmax_violations), 39

resolve_species, 40
resolve_species(), *4*, *5*, *7–9*, *11*, *15*, *16*, *18*, *19*, *21*, *23*, *24*, *27*, *28*, *30*, *32–34*, *36*, *38*, *41*, *43*, *44*, *50*, *52*
ResolveSpecies (resolve_species), 40
ribosomal_genes, 41
ribosomal_genes(), *30*, *31*, *43*, *52*
RibosomalGenes (ribosomal_genes), 41
rrna_genes, 42
rRNAGenes (rrna_genes), 42

sex_genes, 44
sex_genes(), *35*
SexGenes (sex_genes), 44
species_table, 45
SpeciesTable (species_table), 45
stream_exon_lengths, 46
stream_exon_lengths(), *24*
stream_gtf, 46
stream_gtf(), *6*, *17*, *46*
stream_homologies, 47
stream_transcripts, 48
stream_transcripts(), *19*
stream_xrefs, 49
StreamExonLengths (stream_exon_lengths), 46
StreamGTF (stream_gtf), 46
StreamHomologies (stream_homologies), 47
StreamTranscripts (stream_transcripts), 48
StreamXrefs (stream_xrefs), 49

Transcript2Gene (transcript2gene), 49
transcript2gene, 49
transcript2gene(), *3*, *4*, *13*, *19*
trna_genes, 51
trna_genes(), *43*
tRNAGenes (trna_genes), 51