

Package: drugTargetInteractions (via r-universe)

September 19, 2026

Type Package

Title drugTargetInteractions: Mining the Druggable Genome

Version 1.21.14

Date 2026-08-24

Description Utilities for identifying known drug-target interactions for a sets of query drug and/or target protein (gene) identifiers. The interaction information is obtained from a series of drug/target databases.

Depends methods, R (>= 4.1)

Imports utils, stats, RSQLite, UniProt.ws, biomaRt,ensembldb, BiocFileCache,rapports, AnnotationFilter, S4Vectors, httr2, readxl, jsonlite

Suggests RUnit, BiocStyle, knitr, rmarkdown, ggplot2, reshape2, DT, EnsDb.Hsapiens.v86, testthat (>= 3.0.0)

VignetteBuilder knitr

License Artistic-2.0

Encoding UTF-8

NeedsCompilation no

URL <https://github.com/girke-lab/drugTargetInteractions>

biocViews Cheminformatics, BiomedicalInformatics, Pharmacogenetics, Pharmacogenomics, Proteomics, Metabolomics

Config/testthat/edition 3

BugReports <https://github.com/girke-lab/drugTargetInteractions>

Config/roxygen2/version 8.0.0

Config/pak/sysreqs

make libbz2-dev libicu-dev liblzma-dev libpng-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository <https://bioc.r-universe.dev>

Date/Publication 2026-08-25 04:19:29 UTC

RemoteUrl <https://github.com/bioc/drugTargetInteractions>

RemoteRef HEAD

RemoteSha a95ecf0aec40f8aadec98ee09258d4371b359392

Contents

drugTargetInteractions-package	3
addCommonIds	4
assembleMoaTable	5
assembleMoaTargets	6
broadRepurposingHubAnnot	7
buildBroadRepurposingHubDb	9
buildGenomeWideDrugTargetTable	10
buildGtoPdbDb	12
buildHgncSymbolMap	13
buildMoaMasterTable	14
buildTtdDb	16
buildUnichemDb	17
cmpIdMapping	18
combineDrugTargets	19
combineGenomeWideDrugTargets	20
downloadBroadRepurposingHub	22
downloadChemblDb	23
downloadGtoPdb	24
downloadHgncTable	24
downloadTTD	25
downloadUniChem	26
downloadUnichemTables	27
drugTargetAnnot	28
drugTargetAnnotTable	29
drugTargetBioactivity	30
drugTargetColumnMap	31
genConfig	32
getChemblBioactivities	33
getChemblBioassay	34
getChemblDrugTarget	35
getChemblMolecule	37
getChemblTarget	38
getDgidbDrugs	39
getDgidbDrugTarget	40
getDgidbTargets	41
getDrugTarget	42
getEnsemblOrthologs	43
getEnsemblParalogs	44
getHgncGeneTable	45
getOpenTargetsDrugIds	46
getOpenTargetsDrugs	47

getOpenTargetsDrugTarget	48
getOpenTargetsIds	49
getOpenTargetsTargets	50
getParalogs	51
getPubchemDrugs	52
getPubchemDrugTarget	53
getPubchemTargets	54
getSymEnsUp	56
getUnichemMapping	57
getUniprotIDs	58
getUniprotMapping	59
gtoPdbTargetAnnot	60
listBioassayFields	61
listDrugTargetFields	62
listMoaSources	63
mergeDrugTargets	64
moaWide	65
normalizeGeneSymbols	66
processDrugage	67
queryDrugTargets	68
queryMoa	70
runDrugTarget_Annot_Bioassay	71
transformTTD	72
ttdTargetAnnot	73
Index	75

drugTargetInteractions-package

drugTargetInteractions: Mining the Druggable Genome

Description

The drugTargetInteractions package provides utilities for identifying drug-target interactions for sets of small molecule or gene/protein identifiers. The required drug-target interaction information is obtained from a local SQLite instance of the ChEMBL database.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Thomas Girke [cre, aut] (ORCID: <<https://orcid.org/0000-0003-0710-3777>>), Kevin Horan [ctb]

Maintainer: Thomas Girke <thomas.girke@ucr.edu>

addCommonIds	<i>Add shared gene and compound identifiers to per-source result tables</i>
--------------	---

Description

The source functions each return their own identifier vocabulary - ChEMBL uses UniProt accessions, Open Targets Ensembl gene IDs, DGIdb, the Broad Repurposing Hub and GtoPdb gene symbols - which leaves no column to merge or stack them on. This function adds four columns to every table so they share join keys: `hgnc_id`, `gene_symbol` (the current approved symbol), `target_uniprot` and `compound_chembl_id`.

Usage

```
addCommonIds(results, hgncTable = NULL, symbolMap = NULL, verbose = FALSE)
```

Arguments

<code>results</code>	a named list of per-source data.frames, as returned by queryDrugTargets or buildGenomeWideDrugTargetTable .
<code>hgncTable</code>	an HGNC gene table as returned by getHgncGeneTable ; downloaded (and cached) on demand when not supplied.
<code>symbolMap</code>	an old-to-current symbol map as returned by buildHgncSymbolMap ; built from <code>hgncTable</code> when not supplied.
<code>verbose</code>	logical(1); if TRUE, report how many rows each source resolved.

Details

Gene identifiers are resolved against the HGNC gene table, which maps symbols, UniProt accessions and Ensembl gene IDs to one another offline, so no network access is needed beyond the cached HGNC download. Symbols are normalized to their current approved form first, since several sources report the symbol that was current when their records were curated.

`hgnc_id` is the identifier to merge on. `target_uniprot` and `gene_symbol` are convenient but less reliable as keys: a handful of genes carry several UniProt accessions and a handful of accessions belong to more than one gene, and those cases are left NA rather than resolved to an arbitrary choice.

`compound_chembl_id` is filled for the sources that already identify drugs by ChEMBL ID: ChEMBL itself, Open Targets, and DGIdb records whose concept identifier is a ChEMBL one. TTD, the Broad Hub and GtoPdb identify compounds by PubChem CID, InChIKey and IUPHAR ligand ID, and are left NA here.

A table that already carries an `hgnc_id` keeps it; only rows where it is missing are resolved here. A genome-wide build records the gene each row was queried from, and that is a better answer than anything recoverable from the identifiers a source reports back.

Value

`results` with the four columns above added to each table, in the same order and with the same number of rows.

See Also[combineDrugTargets](#), [getHgncGeneTable](#)**Examples**

```
res <- queryDrugTargets(list(molType = "gene", idType = "symbol", ids = "FGFR1"),
                        sources = c("chembl", "opentargets", "dgidb"))
res <- addCommonIds(res)
merge(res$chembl, res$opentargets, by = c("hgnc_id", "compound_chembl_id"))
```

assembleMoaTable	<i>Assemble a drug -> mechanism-of-action table</i>
------------------	--

Description

Returns the mechanisms of action reported for each drug in [queryDrugTargets](#)'s results, one row per drug, source and mechanism. A mechanism of action describes what a drug does, so the targets it is reported to act through are returned separately by [assembleMoaTargets](#).

Usage

```
assembleMoaTable(results)
```

Arguments

results	a named list as returned by queryDrugTargets (or any similarly-shaped named list of per-source data.frames - <code>query_id</code> falls back to each source's own QueryIDs column when <code>results</code> has no "resolved" attribute). Sources present but matching zero rows contribute nothing; sources that carry no MOA are rejected with a pointer to listMoaSources .
---------	---

Details

ChEMBL, Open Targets and the Broad Repurposing Hub report mechanisms and are used here. TTD, GtoPdb and DGIdb instead report an action term such as "Inhibitor" without naming a target; [combineDrugTargets](#) collects those in its action column. [listMoaSources](#) summarises what each source provides.

Mechanism text is returned as each source wrote it, since the same mechanism is often phrased differently in different databases. Only `action_type` is standardised, with the original wording kept in `action_type_raw`. Rows from different sources are kept separate, so a mechanism reported by both ChEMBL and Open Targets appears twice.

Value

A data.frame with columns `query_id`, `drug_key`, `drug_name`, `source`, `moa_text`, `action_type_raw` and `action_type`.

See Also

[assembleMoaTargets](#), [queryMoa](#), [moaWide](#), [listMoaSources](#), [combineDrugTargets](#)

Examples

```
## A pure transformer, so it runs on any correctly-shaped per-source
## frames - no network needed. ChEMBL's two records for one mechanism
## against two targets collapse to a single MOA row.
chembl <- data.frame(
  QueryIDs = c("CHEMBL941", "CHEMBL941"), chembl_id = "CHEMBL941",
  Drug_Name = "IMATINIB", MOA = "Bcr/Abl fusion protein inhibitor",
  Action_Type = "INHIBITOR", UniProt_ID = c("P00519", "P11274"))
broad <- data.frame(
  QueryIDs = "imatinib", pert_iname = "imatinib",
  InChIKey = "KTUUFNOKKBVMGRW-UHFFFAOYSA-N",
  moa = "Bcr-Abl kinase inhibitor | KIT inhibitor", target_gene = "ABL1")
assembleMoaTable(list(chembl = chembl, broad = broad))
```

assembleMoaTargets	<i>Assemble the link from a MOA term to the target(s) it acts through</i>
--------------------	---

Description

Returns the targets each mechanism of action is reported to act through, one row per drug, source, mechanism and target. It is the companion to [assembleMoaTable](#), which returns the mechanisms themselves.

Usage

```
assembleMoaTargets(results, resolveGeneSymbol = FALSE, taxId = 9606L)
```

Arguments

results	a named list as returned by queryDrugTargets ; see assembleMoaTable .
resolveGeneSymbol	logical(1); if TRUE, fill otherwise-NA target_symbol values by resolving target_uniprot via getUniprotMapping (one extra network round trip; default FALSE). Accessions that do not resolve keep NA.
taxId	integer(1) passed to getUniprotMapping when resolveGeneSymbol = TRUE (default 9606L = human).

Details

The Broad Repurposing Hub reports a drug's mechanisms without saying which target each one acts on, so its drugs do not appear here.

Sources name targets differently: ChEMBL uses UniProt accessions and Open Targets gene symbols. `resolveGeneSymbol = TRUE` looks up the missing symbols so results from the two can be compared directly, at the cost of one extra request.

Value

A data.frame with columns query_id, drug_key, source, moa_text, target_symbol, target_uniprot and mechanism_comment (ChEMBL's free-text note on the mechanism record, available only when the ChEMBL result was fetched with fields = "all"; NA otherwise).

See Also

[assembleMoaTable](#), [listMoaSources](#)

Examples

```
chembl <- data.frame(
  QueryIDs = c("CHEMBL941", "CHEMBL941"), chembl_id = "CHEMBL941",
  Drug_Name = "IMATINIB", MOA = "Bcr/Abl fusion protein inhibitor",
  Action_Type = "INHIBITOR", UniProt_ID = c("P00519", "P11274"))
broad <- data.frame(
  QueryIDs = "imatinib", pert_iname = "imatinib",
  InChIKey = "KTUFNOKKBVMGRW-UHFFFAOYSA-N",
  moa = "Bcr-Abl kinase inhibitor", target_gene = "ABL1")
## one MOA, two targets - and Broad contributes no rows at all
assembleMoaTargets(list(chembl = chembl, broad = broad))
```

broadRepurposingHubAnnot

Query Broad Repurposing Hub drug-target annotations bidirectionally

Description

Queries the local Repurposing Hub SQLite built by [buildBroadRepurposingHubDb](#), using the same queryBy = list(molType, idType, ids) convention as [ttdTargetAnnot](#) - including its QueryIDs column: every row of the result is tagged with the original query token it matched, and query IDs that returned no rows still appear as a single row with all other fields NA.

- molType = "protein" (or "gene"), idType = "symbol" -> target -> drug. The Repurposing Hub only exposes gene symbols as targets (no accession/ID system of its own).
- molType = "cmp", idType one of "name" (pert_iname) or "broad_id" (a specific physical sample/batch ID) -> drug -> target.

"symbol"/"name" lookups are case-insensitive (the Repurposing Hub's own pert_iname values are lower-case, unlike most other sources in this package); "broad_id" is exact-match. A drug with no listed target still has one row with target_gene = NA (see [.brhExplodeTargets](#)), so it remains reachable by name/broad_id even though it can never surface via a gene-side query.

Usage

```
broadRepurposingHubAnnot(  
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),  
  brhDbPath,  
  fields = "core"  
)
```

Arguments

queryBy	list with components molType, idType, ids (character vector).
brhDbPath	character(1) path to the Repurposing Hub SQLite, e.g. from buildBroadRepurposingHubDb .
fields	"core" (default) or "all" - both return every broad_interactions column, since (unlike the REST-backed sources) there is no larger raw payload to opt into - or a character vector of column names to keep (QueryIDs is always retained). See listDrugTargetFields .

Details

idType = "broad_id" does not query broad_interactions directly - since the 2026-07-22 two-table split (see [buildBroadRepurposingHubDb](#)), broad_id lives only in broad_samples. Each queried broad_id is first resolved to its compound name(s) there, then dispatched exactly like an idType = "name" lookup, with QueryIDs kept tagged to the original broad_id rather than the resolved name. A broad_id resolves to exactly one compound in the overwhelming majority of cases; a small number (10 as of the 2025-08-18 release) resolve to more than one due to genuine naming inconsistencies in Broad's own source file (e.g. "prednisolone acetate" vs. "prednisolone-acetate" sharing one broad_id) - handled as a real one-to-many resolution (all matching compounds' full target sets are returned) rather than collapsed to one or errored on.

Value

A data.frame with columns QueryIDs, target_gene, pert_iname, clinical_phase, moa, disease_area, indication, smiles, InChIKey, pubchem_cid, structure_ambiguous (with the default fields = "core"), or a subset when fields requests specific columns. structure_ambiguous = TRUE means pert_iname covers more than one distinct structure for that row (packed ";" -joined into smiles/InChIKey/pubchem_cid, aligned segment- by-segment across the three columns) - a real name collision in the source data (different salts/stereoisomers sharing a display name), not a parsing artifact; structure-sensitive work should key on InChIKey, not pert_iname, for those rows. Sample- level fields (purity, vendor, catalog_no, qc_incompatible, expected_mass, deprecated_broad_id) moved to broad_samples in the same SQLite file - query it directly (keyed on broad_id) for that data; it is not returned here.

See Also

[buildBroadRepurposingHubDb](#), [ttdTargetAnnot](#), [listDrugTargetFields](#)

Examples

```
dbPath <- buildBroadRepurposingHubDb()  
broadRepurposingHubAnnot(list(molType = "protein", idType = "symbol",
```

```
ids = c("FGFR1", "IL1B")), dbPath)
broadRepurposingHubAnnot(list(molType = "cmp", idType = "name",
                              ids = "pemigatinib"), dbPath)
```

```
buildBroadRepurposingHubDb
```

Build (or fetch a cached) local SQLite database of Repurposing Hub drug-target annotations

Description

Downloads the Repurposing Hub flat files (see [downloadBroadRepurposingHub](#)) and writes **two** tables to a local SQLite file, each kept at its own true grain rather than flattened into one (see "Two-table design" below): `broad_interactions` (one row per (`pert_iname`, `target_gene`) drug-target edge) and `broad_samples` (physical- sample/QC metadata, one or more rows per `broad_id`). The file is cached via `BiocFileCache` under a name that includes the source files' own "`!File_date`" (e.g. `broad_repurposing_20250818.db`), so rebuilding is a no-op until the Hub actually republishes. As with [buildTtdDb](#), the package itself never ships or redistributes this file - it is built into the caller's own local cache the first time this is run.

Usage

```
buildBroadRepurposingHubDb(rerun = FALSE, config = genConfig())
```

Arguments

<code>rerun</code>	logical(1); passed to downloadBroadRepurposingHub , and also controls whether an existing cached SQLite for the current version is reused (FALSE, the default here) or rebuilt (TRUE). Deliberately defaults to FALSE for the same reason buildTtdDb does: a bare <code>buildBroadRepurposingHubDb()</code> call is routine in examples/tests/vignette chunks, and the Hub updates rarely enough that defaulting to TRUE would just accumulate redundant cached copies with no benefit. Pass <code>rerun = TRUE</code> explicitly to force a fresh check for an update.
<code>config</code>	list as returned by <code>genConfig()</code> .

Value

character(1) local file path to the SQLite database.

Two-table design (2026-07-22)

Originally a single flat `broad_interactions` table, mirroring [buildTtdDb](#)'s one-table shape - but unlike TTD, the Repurposing Hub's sample-level file (sample, one row per physical vial/lot, `broad_id`) genuinely has a *different* grain than the drug-target edge the interaction table is meant to represent: a single compound (`pert_iname`) routinely has several physical samples (verified live: 67 than one `broad_id`, up to 8), and each sample can itself carry several repeat QC/purity readings. A one-table design merging sample straight into the exploded target table (via `pert_iname`,

deduplicated only by `sample[!duplicated(sample), ]`, which drops exact duplicate **rows**, not duplicate **join keys**) confirmed to multiply rows in practice - a real, shipped bug: 3.57x overall row inflation (17,996 true drug-target edges vs. 64,336 rows), with one compound-target pair duplicated 175x, driven by repeat purity readings for a single `broad_id` that survived the row-level dedup untouched. Fixed by never merging `sample` into the interaction table at all: `broad_interactions` draws its compound-level structure fields from `.brhCompoundStructure` instead (a proper one-row-per-compound reduction of `sample`, packing the rare cases where a name genuinely covers multiple structures rather than picking one arbitrarily), and everything else `sample` carries (purity, vendor, catalog_no, qc_incompatible, etc.) - which was never actually a property of a drug-target edge to begin with, only of a physical vial - moves to `broad_samples`, kept at its own natural grain. `broad_interactions`' grain is asserted, not just assumed, via `.assertUniqueKey` immediately after the reduction, so a future Repurposing Hub release that breaks this assumption fails the build loudly instead of silently reinflating results. `idType = "broad_id"` queries ([broadRepurposingHubAnnot](#)) now resolve `broad_id` -> `pert_iname` via `broad_samples` first, then dispatch into `broad_interactions` - which, given the 67 is also a correctness improvement for that query path, not just a side effect of the split.

See Also

[downloadBroadRepurposingHub](#), [broadRepurposingHubAnnot](#)

Examples

```
dbPath <- buildBroadRepurposingHubDb()
dbPath
```

```
buildGenomeWideDrugTargetTable
```

Genome-wide, checkpointed drug-target annotation build across the six annotation sources, anchored on an HGNC gene table

Description

Loops [getChemblDrugTarget](#)/[getDgidbDrugTarget](#)/[getOpenTargetsDrugTarget](#)/[ttdTargetAnnot](#)/[broadRepurposingHubAnnot](#)/[gtoPdbTargetAnnot](#) over every gene in `hgncTable` (default: all human protein-coding genes from [getHgncGeneTable](#)), in checkpointed chunks written to `outDir` as it goes - so an interrupted run resumes from the last completed chunk instead of starting over. Deliberately excludes ChEMBL's bioassay function and PubChem (see the file header and the "Genome-Wide Master Table" vignette section for why).

Usage

```
buildGenomeWideDrugTargetTable(
  hgncTable = NULL,
  sources = c("chembl", "dgidb", "opentargets", "ttd", "broad", "gtopdb"),
  ttdDbPath = NULL,
```

```

    brhDbPath = NULL,
    gtoPdbDbPath = NULL,
    outDir,
    chunkGenes = 500L,
    rerun = FALSE,
    verbose = TRUE,
    ...
)

```

Arguments

hgncTable	data.frame from getHgncGeneTable (default: fetched automatically if not supplied).
sources	character vector, any of "chembl", "dgidb", "opentargets", "ttd", "broad", "gtopdb" (default: all six).
ttdDbPath	character(1) path to a local TTD SQLite (see buildTtdDb); required if "ttd" is in sources.
brhDbPath	character(1) path to a local Broad Repurposing Hub SQLite (see buildBroadRepurposingHubDb); required if "broad" is in sources.
gtoPdbDbPath	character(1) path to a local GtoPdb SQLite (see buildGtoPdbDb); required if "gtopdb" is in sources.
outDir	character(1) directory to write checkpoint chunks and the manifest to; created if it doesn't exist.
chunkGenes	integer(1) genes per checkpoint chunk (default 500) - a coarser grouping purely for resumability; each source's own function still does its own finer-grained request batching (chunkSize, passed via ...) within one chunk.
rerun	logical(1); if FALSE (default), chunks already recorded in outDir's manifest are skipped - the resume behavior. Set TRUE to rebuild everything from scratch.
verbose	logical(1); default TRUE (unlike this package's other build*() functions) since this is a long-running job.
...	additional arguments passed through to the ChEMBL/DGIdb/ Open Targets calls (e.g. fields, chunkSize) - not forwarded to ttdTargetAnnot()/broadRepurposingHubAnnot()/gtoPdbTargetAnnot(), none of which take extra arguments.

Details

A single target-centric run already answers the reverse drug -> target question too: every row already carries whichever drug matched that gene, so filtering the assembled table by drug identifier gives that direction for free - no separate compound- centric run is needed.

Value

A named list, one data.frame per successfully-built source, each row tagged with hgnc_id/symbol/ensembl_gene_id alongside that source's own native columns (not run through [combineDrugTargets](#)'s harmonization, to keep maximum information - apply that separately if wanted). Also cached via [genConfig](#)'s BiocFileCache under a name that encodes the HGNC source snapshot and build date, so a completed build is reusable without recomputation; the cache path is recorded in attr(, "cachePath").

See Also

[getHgncGeneTable](#), [queryDrugTargets](#), [combineDrugTargets](#)

Examples

```
## Small illustrative run, not genome-wide - see the "Genome-Wide
## Master Table" vignette section for the full workflow (takes
## hours at full scale, not run here).
hgncSmall <- getHgncGeneTable()[1:3, ]
res <- buildGenomeWideDrugTargetTable(
  hgncTable = hgncSmall, sources = c("dgidb", "opentargets"),
  outDir = tempfile("dti_build_"))
names(res)
```

buildGtoPdbDb	<i>Build (or fetch a cached) local SQLite database of GtoPdb drug-target interactions</i>
---------------	---

Description

Downloads GtoPdb's bulk interaction data and HGNC mapping (see [downloadGtoPdb](#)), filters interactions to `targetSpecies == "Human"`, left-joins in the HGNC gene symbol per `targetId` (a target ID may map to more than one gene for multi-subunit complexes - kept as separate rows, not collapsed), and writes a single denormalized, indexed `gtp_interactions` table to a local SQLite file - the same one-table shape as [buildTtdDb](#)/[buildBroadRepurposingHubDb](#). The file is cached via `BiocFileCache` under a name that includes GtoPdb's own release version (e.g. `gtopdb_2026.2.db`), so rebuilding is a no-op until GtoPdb actually republishes. As with the other local-SQLite sources, the package itself never ships or redistributes this file - it is built into the caller's own local cache the first time this is run (see the file header for why that is a conservative default here, not a licensing requirement).

Usage

```
buildGtoPdbDb(rerun = FALSE, config = genConfig())
```

Arguments

<code>rerun</code>	logical(1); passed to downloadGtoPdb , and also controls whether an existing cached SQLite for the current version is reused (FALSE, the default here) or rebuilt (TRUE) - same rationale as buildTtdDb / buildBroadRepurposingHubDb : routine examples/tests/vignette chunks call this bare, and GtoPdb releases infrequently enough that defaulting to TRUE would just accumulate redundant cached copies.
<code>config</code>	list as returned by <code>genConfig()</code> .

Value

character(1) local file path to the SQLite database.

See Also

[downloadGtoPdb](#), [gtoPdbTargetAnnot](#)

Examples

```
dbPath <- buildGtoPdbDb()
dbPath
```

buildHgncSymbolMap	<i>Build a prev/alias-symbol -> current-symbol normalization map</i>
--------------------	---

Description

DGIdb and TTD (and potentially any other external gene list) echo back whatever symbol was current when *their* records were curated; a nontrivial fraction are now outdated relative to HGNC. This builds a lookup from every historical prev_symbol/ alias_symbol value to its gene's current approved symbol, so incoming symbols from those sources can be routed through it before joining back to an HGNC-anchored table.

Usage

```
buildHgncSymbolMap(hgncTable, warn = TRUE)
```

Arguments

hgncTable	data.frame as returned by getHgncGeneTable (must have symbol, prev_symbol, alias_symbol list-columns).
warn	logical(1); if TRUE (default), raise a warning() reporting how many of the map's entries are ambiguous. That count describes the HGNC snapshot as a whole, not any particular set of symbols, so functions that build a map only to translate a handful of symbols pass FALSE and let normalizeGeneSymbols report the ambiguities those symbols actually run into.

Value

A named character vector (old symbol -> current symbol). Old symbols that map to more than one current symbol (real cases exist, e.g. shared paralog aliases) keep only the first (alphabetically, for determinism) and are also reported via attr(, "ambiguous"), a named list of all candidate current symbols for each flagged old symbol.

See Also

[getHgncGeneTable](#), [normalizeGeneSymbols](#)

Examples

```
hgncTable <- getHgncGeneTable()
symbolMap <- buildHgncSymbolMap(hgncTable)
length(attr(symbolMap, "ambiguous"))
```

buildMoaMasterTable	<i>Build a drug-centric master table of mechanisms of action</i>
---------------------	--

Description

Sweeps every drug that the enumerable MOA sources annotate, and returns one row per drug and mechanism. Unlike [queryMoa](#), which looks up mechanisms for drugs you already name, this discovers the drugs as well - including the many whose mechanism is recorded without any target being named, which a gene-anchored build such as [buildGenomeWideDrugTargetTable](#) cannot reach.

Usage

```
buildMoaMasterTable(
  sources = c("chembl", "broad"),
  brhDbPath = NULL,
  resolveDrugNames = TRUE,
  includeUnknown = FALSE,
  verbose = TRUE
)
```

Arguments

<code>sources</code>	character vector, any of "chembl" and "broad" (default: both).
<code>brhDbPath</code>	character(1) path to a Broad Repurposing Hub SQLite (see buildBroadRepurposingHubDb); required if "broad" is in sources.
<code>resolveDrugNames</code>	logical(1); if TRUE (default), look up ChEMBL's preferred drug names, which costs one extra batch of requests. FALSE leaves drug_name empty on ChEMBL rows, where the identifier alone is enough.
<code>includeUnknown</code>	logical(1); ChEMBL records "Unknown" as the mechanism for some drugs. These are dropped by default as placeholders rather than mechanisms; set TRUE to keep them.
<code>verbose</code>	logical(1); report progress and per-source counts (default TRUE, since the ChEMBL sweep takes a moment).

Details

Two sources are covered. ChEMBL is read from its mechanism collection over the network; the Broad Repurposing Hub is read from the local SQLite built by [buildBroadRepurposingHubDb](#). Open Targets also records mechanisms but cannot be enumerated the same way, so use [queryMoa](#) for it.

The sources are kept side by side rather than merged: they identify drugs differently, so `drug_id` means a ChEMBL identifier on ChEMBL rows and the Repurposing Hub's own drug name on Broad rows, and the same drug can appear once per source. Compare them on `drug_name` if you need to, keeping in mind that names are not a reliable key across databases.

The Broad Hub sometimes records several mechanisms for one drug in a single "a | b" string. These are split, so each row carries one mechanism and counting or grouping the `moa` column gives real terms rather than combinations.

The two sources word mechanisms differently, and not only in capitalisation: ChEMBL names the specific protein ("Carbonic anhydrase VII inhibitor") where the Repurposing Hub names the family ("carbonic anhydrase inhibitor"). Almost no term is shared verbatim. Group or count the `moa` column within one source at a time; comparing terms across the two needs them matched deliberately, not just lower-cased.

Value

A data frame with one row per drug and mechanism: `drug_id`, `drug_name`, `moa`, `action` (ChEMBL's coarse action label, empty for Broad, which records no such field), `source`, and `has_target`, which says whether that source names a target for the mechanism. Use [assembleMoaTargets](#) to get the targets themselves.

See Also

[queryMoa](#) for mechanisms of named drugs, [assembleMoaTargets](#) for the targets a mechanism acts through, [listMoaSources](#)

Examples

```
## ChEMBL alone needs no local database
moa <- buildMoaMasterTable(sources = "chembl")
nrow(moa)
head(sort(table(moa$moa), decreasing = TRUE))      # commonest mechanisms
sum(!moa$has_target)                               # annotated, no target

## Both sources
brhDbPath <- buildBroadRepurposingHubDb(rerun = FALSE)
moa <- buildMoaMasterTable(brhDbPath = brhDbPath)
table(moa$source)
```

buildTtdDb*Build (or fetch a cached) local SQLite database of TTD interactions*

Description

Downloads the TTD flat files (see [downloadTTD](#)), parses them, and writes a single denormalized `ttd_interactions` table (TTD's target-drug relationship is a simple 1-hop mapping, unlike ChEMBL's multi-table mechanism/activity schema, so one flat, indexed table is the natural queryable shape) to a local SQLite file. The file is cached via BiocFileCache under a name that includes TTD's own release version (e.g. `ttd_10.1.01.db`), so rebuilding is a no-op until TTD actually publishes a new version. As with [downloadChEMBLdb](#), the package itself never ships or redistributes this file - it is built into the caller's own local cache the first time this is run.

Usage

```
buildTtdDb(rerun = FALSE, config = genConfig())
```

Arguments

<code>rerun</code>	logical(1); passed to downloadTTD , and also controls whether an existing cached SQLite for the current version is reused (FALSE, the default here) or rebuilt (TRUE). Deliberately defaults to FALSE, unlike most other <code>download*()/build*()</code> functions in this package (which default to <code>rerun = TRUE</code>): TTD publishes a new release only rarely, and a bare <code>buildTtdDb()</code> call is routine in examples, tests, and vignette chunks - defaulting to TRUE there meant every such run silently rebuilt and re-cached the database from scratch, accumulating redundant copies over time with no benefit. Pass <code>rerun = TRUE</code> explicitly to force a fresh check for a new TTD release.
<code>config</code>	list as returned by <code>genConfig()</code> .

Value

character(1) local file path to the SQLite database. The database additionally carries a `ttd_release` table (columns `file`, `version`, `date`) recording every downloaded TTD file's own embedded release stamp separately, since TTD does not reliably move these together across files - see the file header note. [ttdTargetAnnot](#) attaches this as `attr(result, "ttd_release")`.

See Also

[downloadTTD](#), [ttdTargetAnnot](#)

Examples

```
dbPath <- buildTtdDb()
dbPath
```

buildUnichemDb	<i>Build (or fetch a cached) local SQLite of UniChem compound cross-references</i>
----------------	--

Description

Downloads UniChem's bulk table dumps (see [downloadUnichemTables](#)), loads `reference.tsv.gz` in chunks (never all in memory at once), filters to compounds with a cross-reference to at least `minSources` distinct databases, and writes two tables to a local SQLite file: `unichem_xref(uci,src_id,compound_id)` and `unichem_sources(src_id,name,name_long,description)` (a direct copy of `source.tsv.gz`, used by [getUnichemMapping](#) to resolve source-name arguments). The file is cached via `BiocFileCache` under a name that includes the build date (UniChem's bulk dumps carry no release-version string the way TTD's flat files do); `rerun = FALSE` reuses *any* previously-built `unichem_*.db` regardless of which day it was built (see `.unichemLatestCachedDb` - an exact same-day match would otherwise silently miss yesterday's build and trigger an unwanted ~hour-long rebuild).

Usage

```
buildUnichemDb(  
  rerun = TRUE,  
  config = genConfig(),  
  minSources = 2L,  
  anchorSources = .unichemAnchorSources,  
  chunkSize = 500000L,  
  verbose = FALSE  
)
```

Arguments

<code>rerun</code>	logical(1); passed to downloadUnichemTables , and also controls whether any existing cached SQLite (see <code>.unichemLatestCachedDb</code>) is reused (FALSE) or a fresh one is built (TRUE).
<code>config</code>	list as returned by <code>genConfig()</code> .
<code>minSources</code>	integer(1) minimum distinct sources per UCI to keep (default 2) - see <code>.unichemFilterAndIndex</code> .
<code>anchorSources</code>	character vector of UniChem source names a kept UCI must reference at least one of, or NULL to disable this requirement (default: <code>.unichemAnchorSources</code> - ChEMBL, DrugBank, ChEBI, Guide to Pharmacology, DrugCentral, BindingDB, ClinicalTrials). See <code>.unichemFilterAndIndex</code> for why this matters: <code>minSources</code> alone is not enough to keep the table drug-target-relevant.
<code>chunkSize</code>	integer(1) rows read per batch while loading <code>reference.tsv.gz</code> (default 500,000).
<code>verbose</code>	logical(1); if TRUE, message progress throughout.

Details

This is a substantially heavier operation than the package's other `build*Db()/download*Db()` functions - live-tested 2026-07-17: ~1.5GB download, ~166M source rows to load/filter/index. Expect this to take real time (minutes, not seconds) and several GB of temporary disk space for the unfiltered staging table before filtering shrinks it down.

Value

character(1) local file path to the SQLite database.

See Also

[downloadUnichemTables](#), [getUnichemMapping](#)

Examples

```
dbPath <- buildUnichemDb(verbose = TRUE)
dbPath
```

cmpIdMapping

cmpIdMapping

Description

Function to generate compound ID mappings UniChem.

This function requires the ID mapping files "src1src2.txt.gz", "src1src22.txt.gz", and "src1src7.txt.gz" to exist in a directory called "downloads" before being run. These can be generated with the [downloadUniChem](#) function.

It will do some processing on these files and output an RDS file at outfile. This file can then be used in other functions, such as [drugTargetAnnot](#).

Usage

```
cmpIdMapping(outfile=file.path(config$resultsPath,"cmp_ids.rds"), rerun=TRUE, config=genConfig())
```

Arguments

outfile	Path to output file.
rerun	If true, runs processing, otherwise does nothing.
config	General configuration. See genConfig .

Value

Generates an RDS file at outfile.

Author(s)

Thomas Girke

See Also[downloadUniChem](#)**Examples**

```
cmpIdMapping("cmp_ids.rds", rerun=FALSE)
```

combineDrugTargets	Combine queryDrugTargets results into one table
--------------------	---

Description

Row-binds a small set of canonical columns across whichever sources are present in results, mapping each source's own column names to a shared vocabulary (see `.dtiCombineColMap`). This is deliberately just column-name alignment and row append - not deduplication, not cross-source identity resolution (the same compound or target appearing under different native IDs in different sources is not merged). A more principled harmonized container (an S4 class, analysed but not yet built) is a separate, deferred phase; this function is the "just append them" version to use in the meantime, and every source's full original data remains available unchanged in results itself.

Usage

```
combineDrugTargets(  
  results,  
  columns = c("query_id", "gene_symbol", "drug_name", "action", "source"),  
  resolveGeneSymbol = FALSE,  
  taxId = 9606L  
)
```

Arguments

results	a named list as returned by queryDrugTargets (or any similarly-shaped named list of per-source data.frames - <code>query_id</code> falls back to each source's own QueryIDs column when results has no "resolved" attribute).
columns	character vector of canonical columns to include, any of "query_id", "gene_symbol", "drug_name", "action", "source" (default: all five), plus the shared join keys "hgnc_id", "target_uniprot" and "compound_chembl_id". The three keys are not part of any source's own output, so asking for one derives it with addCommonIds , which reads the cached HGNC gene table.

`resolveGeneSymbol` logical(1); if TRUE, fill ChEMBL's otherwise-NA `gene_symbol` by resolving its UniProt_ID column via [getUniprotMapping](#) (one extra network round trip - default FALSE so `combineDrugTargets()` is network-free by default when results already has everything it needs). Only has an effect when "`gene_symbol`" is also in columns.

`taxId` integer(1) passed to [getUniprotMapping](#) when `resolveGeneSymbol` = TRUE (default 9606L = human).

Details

`action` is a best-effort common label, not a perfectly aligned concept: ChEMBL's `Action_Type` and Open Targets' `action_type` are categorical mechanism labels (e.g. "INHIBITOR"); TTD's MOA, DGIdb's `interaction_types`, the Broad Repurposing Hub's `moa` (free text, e.g. "fgfr inhibitor", not a controlled vocabulary) and GtoPdb's `action` (e.g. "Inhibition") are similar. PubChem is not one of the combinable sources at all (see `.dtiMetaSources`): its bioactivity data has no mechanism-of-action concept to align to in the first place.

Value

A single `data.frame` with columns `columns`, one row per row of every source in results.

See Also

[queryDrugTargets](#)

Examples

```
res <- queryDrugTargets(list(molType = "gene", idType = "symbol", ids = "FGFR1"),
                        sources = c("chembl", "dgidb", "opentargets"))
combineDrugTargets(res)
combineDrugTargets(res, resolveGeneSymbol = TRUE) ## fills ChEMBL's gene_symbol too
```

`combineGenomeWideDrugTargets`

Append a genome-wide build's per-source tables into one table

Description

[buildGenomeWideDrugTargetTable](#) returns one table per source, each keeping that source's own columns. This stacks them into a single table: columns holding the same content under different names are aligned onto one shared column, and columns belonging to only one source are carried through as their own columns, empty for every other source's rows.

Usage

```
combineGenomeWideDrugTargets(  
  results,  
  colMap = drugTargetColumnMap(),  
  native = TRUE,  
  keys = TRUE,  
  hgncTable = NULL,  
  verbose = FALSE  
)
```

Arguments

results	a named list of per-source data.frames from buildGenomeWideDrugTargetTable , or any similarly-shaped list such as queryDrugTargets ' output.
colMap	the column mapping to apply; default drugTargetColumnMap() , the curated mapping the package ships. Rows with active = FALSE are ignored.
native	logical(1); if TRUE (default), columns belonging to a single source are carried through as source.column (ttid.Smiles, opentargets.disease_id) rather than dropped, so nothing in the build is lost. Set FALSE for just the shared columns.
keys	logical(1); if TRUE (default), make sure hgnc_id and compound_chembl_id are present, deriving with addCommonIds only what is missing. Values already in results are never overwritten.
hgncTable	optional HGNC table from getHgncGeneTable , passed to addCommonIds when keys have to be derived.
verbose	logical(1); report per-source row counts and any columns the mapping does not cover (default FALSE).

Details

Which columns hold the same content is decided by colMap, an ordinary data.frame from [drugTargetColumnMap](#) that you can edit to change the alignment or to add groups of your own.

Each row keeps the HGNC gene it was queried from, in hgnc_id, symbol and ensembl_gene_id. Those are taken from the build unchanged. symbol is HGNC's current symbol for that gene; gene_symbol is what the source itself called it, and the two can differ where a source reports an older symbol.

Some columns share a concept without sharing a vocabulary. max_phase is the clearest case: ChEMBL reports a number from -1 to 4, Open Targets a token such as "PHASE_3", TTD and the Broad Repurposing Hub a phrase such as "Approved". Read that column together with source.

Value

A single data.frame: the gene identity columns present in results, then source, then one column per shared group in colMap, then compound_chembl_id, then each source's own remaining columns when native = TRUE.

See Also

[buildGenomeWideDrugTargetTable](#), [drugTargetColumnMap](#), [addCommonIds](#), [mergeDrugTargets](#)

Examples

```
res <- buildGenomeWideDrugTargetTable(
  hgncTable = getHgncGeneTable()[1:3, ],
  sources = c("dgidb", "opentargets"), outDir = tempfile("dti_"))

## Shared columns only
combineGenomeWideDrugTargets(res, native = FALSE)

## Everything, with each source's own columns kept
tbl <- combineGenomeWideDrugTargets(res)
table(tbl$source)
```

downloadBroadRepurposingHub

Download the Broad Repurposing Hub drug/sample annotation flat files

Description

Downloads (or reuses previously cached copies of) the two Repurposing Hub flat files via the package's existing `.downloadFile()/BiocFileCache` infrastructure - the same mechanism `downloadTTD()` uses. Files land in the local `BiocFileCache` (see `.getCache()`, `rappdirs::user_cache_dir(appname = "drugTargetInteractions")`) - no Repurposing Hub data is bundled with or downloaded by the package itself until this is called explicitly.

Usage

```
downloadBroadRepurposingHub(rerun = TRUE, config = genConfig())
```

Arguments

<code>rerun</code>	logical(1); if TRUE (default), check for updates and (re)download as needed; if FALSE, use whatever is already in the local cache without checking upstream.
<code>config</code>	list as returned by <code>genConfig()</code> ; unused here but accepted for consistency with the rest of the package's API.

Details

`repo-hub.broadinstitute.org` has been observed to serve an incomplete TLS certificate chain (see this file's header comment) - if the first download attempt fails for any reason, this function retries once with a CA bundle extended to include the one legitimate, publicly-issued intermediate certificate the server itself omits (`.brhWithSupplementalCa()`); a genuine failure (network down, file moved, etc.) still surfaces as an error after that retry.

Value

A named list of local file paths: drug, sample.

See Also

[buildBroadRepurposingHubDb](#)

Examples

```
paths <- downloadBroadRepurposingHub()
paths
```

downloadChEMBLdb	<i>downloadChEMBLdb</i>
------------------	-------------------------

Description

Download ChEMBL sqlite db for use by several other functions in the package.

Usage

```
downloadChEMBLdb(version=.chemblLatestVersion(),rerun=TRUE,config=genConfig())
```

Arguments

version	The ChEMBL version to download. Defaults to the latest release, resolved live from ChEMBL's REST status endpoint (see getChemblMolecule and related REST accessors); pass an explicit integer to pin to an older release instead.
rerun	If TRUE, the file will be downloaded, otherwise do nothing.
config	The configuration object. This gives the location to put the downloaded chembl db.

Value

No return value.

Author(s)

Kevin Horan

Examples

```
downloadChEMBLdb(27)
```

downloadGtoPdb	<i>Download GtoPdb's bulk interactions (REST) and HGNC mapping (flat file) data</i>
----------------	---

Description

Downloads (or reuses previously cached copies of) GtoPdb's bulk /services/interactions REST response and its GtP_to_HGNC_mapping.tsv flat file via the package's existing .downloadFile()/BiocFileCache infrastructure - the same mechanism downloadTTD()/downloadBroadRepurposingHub() use. Files land in the local BiocFileCache (see .getCache(), rappdirs::user_cache_dir(appname = "drugTargetInteractions")) - no GtoPdb data is bundled with or downloaded by the package itself until this is called explicitly.

Usage

```
downloadGtoPdb(rerun = TRUE, config = genConfig())
```

Arguments

rerun	logical(1); if TRUE (default), check for updates and (re)download as needed; if FALSE, use whatever is already in the local cache without checking upstream.
config	list as returned by genConfig(); unused here but accepted for consistency with the rest of the package's API.

Value

A named list of local file paths: interactions, hgncMapping.

See Also

[buildGtoPdbDb](#)

Examples

```
paths <- downloadGtoPdb()
paths
```

downloadHgncTable	<i>Download (and cache) an HGNC complete-gene-set TSV snapshot</i>
-------------------	--

Description

Download (and cache) an HGNC complete-gene-set TSV snapshot

Usage

```
downloadHgncTable(
  archiveFile = NULL,
  current = FALSE,
  rerun = TRUE,
  config = genConfig(),
  verbose = FALSE
)
```

Arguments

archiveFile	character(1) or NULL; a specific HGNC archive filename (quarterly or monthly, see https://storage.googleapis.com/public-download-files/hgnc/archive/archive/quarterly/tsv/) to reproduce a particular snapshot. NULL (default) uses the package's pinned quarterly snapshot (see .hgncPinnedQuarterlyFile) - fixed across package versions so results stay reproducible; the pin is updated as a package maintenance task, not a user setting.
current	logical(1); if TRUE, ignore archiveFile and download HGNC's always-latest rolling file instead (updated Tuesdays/Fridays) - convenient, but re-running the same code later can yield different results.
rerun	logical(1); if TRUE (default), check for a newer remote version even if already cached (only matters for current = TRUE; pinned/archive snapshots never change once published). If FALSE, reuse whatever is cached without checking.
config	list from genConfig .
verbose	logical(1).

Value

character(1) local file path to the cached TSV.

See Also

[getHgncGeneTable](#)

Examples

```
hgncPath <- downloadHgncTable()
```

Description

Downloads (or reuses previously cached copies of) the TTD target, drug, drug-target-mapping, drug cross-matching and drug-disease indication flat files via the package's existing `.downloadFile()/BiocFileCache` infrastructure - the same mechanism `downloadChEMBLdb()` and `downloadUniChem()` use. Files land in the local `BiocFileCache` (see `.getCache()`, `rappdirs::user_cache_dir(appname = "drugTargetInteractions")`) - no TTD data is bundled with or downloaded by the package itself until this is called explicitly.

Usage

```
downloadTTD(rerun = TRUE, config = genConfig())
```

Arguments

<code>rerun</code>	logical(1); if TRUE (default), check for updates and (re)download as needed; if FALSE, use whatever is already in the local cache without checking upstream.
<code>config</code>	list as returned by <code>genConfig()</code> ; unused here but accepted for consistency with the rest of the package's API.

Value

A named list of local file paths: targets, drugs, mapping, crossmatch, drugDisease.

See Also

[buildTtdDb](#)

Examples

```
paths <- downloadTTD()
paths
```

downloadUniChem

downloadUniChem

Description

Downloads UniChem compound ID mappings from <https://www.ebi.ac.uk/unichem/ucquery/listSources>. Mappings are downloaded for DrugBank, PubChem, and ChEBI.

Usage

```
downloadUniChem(rerun=TRUE, config=genConfig())
```

Arguments

<code>rerun</code>	If true, downloads the files, else does nothing.
<code>config</code>	General configuration. See genConfig .

Value

Generates the following output files: "src1src2.txt.gz", "src1src22.txt.gz", and "src1src7.txt.gz". These correspond to mappings from ChEMBL to DrugBank, PubChem, and ChEBI, respectively.

Author(s)

Thomas Girke

References

<https://www.ebi.ac.uk/unicem/ucquery/listSources>

See Also

[drugTargetAnnotTable](#)

Examples

```
downloadUniChem(rerun=TRUE)
```

downloadUniChemTables *Download the UniChem table dumps needed for compound cross-referencing*

Description

Downloads (or reuses previously cached copies of) UniChem's reference (UCI/source/compound-ID triples, ~1.5GB) and source (source-ID/name lookup, ~3KB) table dumps via the package's existing `downloadFile()/BiocFileCache` infrastructure - the same mechanism `downloadChEMBLdb()` and `downloadTTD()` use. `structure.tsv.gz` (actual chemical structures, ~13.4GB) is deliberately not downloaded - see the file header.

Usage

```
downloadUniChemTables(rerun = TRUE, config = genConfig())
```

Arguments

<code>rerun</code>	logical(1); if TRUE (default), check for updates and (re)download as needed; if FALSE, use whatever is already in the local cache without checking upstream.
<code>config</code>	list as returned by <code>genConfig()</code> ; unused here but accepted for consistency with the rest of the package's API.

Value

A named list of local file paths: reference, source.

See Also[buildUnichemDb](#)**Examples**

```
paths <- downloadUnichemTables()
paths
```

drugTargetAnnot	<i>drugTargetAnnot</i>
-----------------	------------------------

Description

Function to query known drug-target annotations.

Usage

```
drugTargetAnnot(queryBy=list(molType=NULL, idType=NULL, ids=NULL), cmpid_file=file.path(config$resul
```

Arguments

queryBy	A list defining the query, as described in queryBy .
cmpid_file	Path to a compound ID mapping file, generated by cmpIdMapping .
config	General configuration. See genConfig .

Value

Returns the query results as a data frame.

Author(s)

Thomas Girke

See Also[queryBy](#) [cmpIdMapping](#)**Examples**

```
# These are just sample files included in the package.
# You should use your own data files.
config = genConfig(chemblDbPath=
  system.file("extdata", "chembl_sample.db", package="drugTargetInteractions"),
  resultsPath =
  system.file("extdata", "results", package="drugTargetInteractions"))

queryBy <- list(molType="cmp", idType="chembl_id", ids=c("CHEMBL1233058", "CHEMBL1200916", "CHEMBL437765"))
qresult <- drugTargetAnnot(queryBy, config=config)
```

drugTargetAnnotTable *drugTargetAnnotTable*

Description

Generates a drug target annotation TSV file. This file includes target information from ChEMBL, drugbank, pubchem, and chembi.

This function requires the ID mapping files "src1src2.txt.gz", "src1src22.txt.gz", and "src1src7.txt.gz" to exist in a directory called "downloads" before being run. These can be generated with the [downloadUniChem](#) function.

Usage

```
drugTargetAnnotTable(outfile, rerun=TRUE, config=genConfig())
```

Arguments

outfile	The name of the output file to write the results to.
rerun	If true, download and generate output file. Otherwise do nothing.
config	General configuration. See genConfig .

Value

Writes output file to outfile.

Author(s)

Thomas Girke

See Also

[downloadUniChem](#)

Examples

```
config = genConfig(chemblDbPath=
  system.file("extdata", "chembl_sample.db", package="drugTargetInteractions"))
drugTargetAnnotTable(outfile="drugTargetAnnot.xls", config=config)
```

drugTargetBioactivity *drugTargetBioactivity*

Description

Function to query bioactivity data by target or compound ids

Usage

```
drugTargetBioactivity( queryBy=list(molType=NULL, idType=NULL, ids=NULL), cmpid_file=file.path(config
```

Arguments

queryBy	A list defining the query, as described in queryBy .
cmpid_file	Path to a compound ID mapping file, generated by cmpIdMapping .
config	General configuration. See genConfig .

Value

Returns results as a data frame.

Author(s)

Thomas Girke

See Also

[queryBy](#) [cmpIdMapping](#)

Examples

```
config = genConfig(chemblDbPath=
  system.file("extdata", "chembl_sample.db", package="drugTargetInteractions"),
  resultsPath =
  system.file("extdata", "results", package="drugTargetInteractions"))

queryBy <- list(molType="protein", idType="uniprot", ids=c("P05979", "P35354", "P33033", "Q8VCT3", "P29475", "P
qresult <- drugTargetBioactivity( queryBy, config=config)

queryBy <- list(molType="cmp", idType="molregno", ids=c("101036", "101137", "1384464"))
qresult <- drugTargetBioactivity( queryBy, config=config)

queryBy <- list(molType="cmp", idType="DrugBank_ID", ids=c("DB00945", "DB00316", "DB01050"))
qresult <- drugTargetBioactivity(queryBy, config=config)

queryBy <- list(molType="cmp", idType="PubChem_ID", ids=c("2244", "3672", "1983"))
qresult <- drugTargetBioactivity(queryBy, config=config)
```

drugTargetColumnMap	<i>Column mapping used to append per-source tables into one table</i>
---------------------	---

Description

Every drug-target source names its columns differently, so stacking their tables row-wise requires knowing which of them hold the same content. This function returns that mapping as an ordinary data.frame, one row per (canonical column, source), which you can inspect, edit, write to a file and pass back to [combineGenomeWideDrugTargets](#).

Usage

```
drugTargetColumnMap(results = NULL, propose = TRUE)
```

Arguments

results	optional named list of per-source data.frames, as returned by buildGenomeWideDrugTargetTable or queryDrugTargets . When supplied, proposed and unmatched rows are added for the columns those tables carry.
propose	logical(1); set FALSE to report only the unmatched columns of results and skip name-based proposals. Ignored when results is NULL.

Details

Called with no arguments it returns the curated mapping the package ships. Given a result list it additionally inspects the columns those tables actually carry and reports two further kinds of row: "proposed" groups it found by matching column names after normalising case and punctuation, and "unmatched" columns that belong to no group at all. Proposed rows come back with active = FALSE and have no effect until you set them TRUE - column names that look alike are not always the same content, so nothing detected this way is applied on your behalf.

hgnc_id and compound_chembl_id are not part of the mapping. They are cross-source keys that [addCommonIds](#) resolves against HGNC and UniChem, using ambiguity rules a column mapping cannot express.

Value

A data.frame with columns canonical (the shared column name; NA for unmatched rows), source, column (that source's own column name), status ("curated", "proposed" or "unmatched") and active (whether [combineGenomeWideDrugTargets](#) will use the row).

See Also

[combineGenomeWideDrugTargets](#), [addCommonIds](#)

Examples

```
drugTargetColumnMap()

## Review what a real build carries beyond the curated map
res <- buildGenomeWideDrugTargetTable(
  hgncTable = getHgncGeneTable()[1:3, ],
  sources = c("dgidb", "opentargets"), outDir = tempfile("dti_"))
m <- drugTargetColumnMap(res)
subset(m, status != "curated")

## Accept one proposal and align a group of your own
m$active[m$canonical %in% "druglikeness"] <- TRUE
combineGenomeWideDrugTargets(res, colMap = m)
```

genConfig

genConfig

Description

Create a default configuration object.

Usage

```
genConfig( chemblDbPath = "chembl.db", downloadPath = "downloads", resultsPath = "results" )
```

Arguments

chemblDbPath	Path or filename of ChEMBL SQLite db file.
downloadPath	The name of a directory to put downloaded files in.
resultsPath	The name of a directory to put output files in.

Value

A config object that can be passed to ther functions.

Author(s)

Kevin Horan

Examples

```
config = genConfig()
```

getChEMBLBioactivities

Retrieve bioactivity measurements from the ChEMBL REST API

Description

Pulls activity rows for a given ChEMBL target ID, optionally filtered to a standard activity type (e.g. IC50, Ki). Paginates through the REST activity resource up to maxRows.

Usage

```
getChEMBLBioactivities(
  targetChEMBLId,
  standardType = NA,
  maxRows = 1000L,
  pageSize = 1000L,
  verbose = FALSE
)
```

Arguments

targetChEMBLId	character(1), e.g. "ChEMBL1862".
standardType	character(1) or NA; e.g. "IC50". NA = all.
maxRows	integer(1) cap on returned rows.
pageSize	integer(1) rows per REST page (max 1000).
verbose	logical(1) progress messages.

Value

A data.frame with columns molecule_chembl_id, target_chembl_id, standard_type, standard_relation, standard_value, standard_units, pchembl_value, assay_chembl_id, assay_description.
Empty on failure.

See Also

getChEMBLTarget

Examples

```
act <- getChEMBLBioactivities("ChEMBL1862", standardType = "IC50",
                             maxRows = 50)
head(act)
```

getChEMBLBioassay

Query raw ChEMBL bioassay measurements via the queryBy interface

Description

Bidirectional, batched REST equivalent of the package's legacy local-ChEMBL-SQLite `drugTargetBioactivity()` query (R/`drugTargetAnnotations_Fct.R`): given one or more UniProt accessions, returns every bioassay measurement recorded against ChEMBL targets matching those accessions (target -> drug bioassay direction); given one or more ChEMBL molecule IDs, returns every measurement recorded for those compounds (drug -> target bioassay direction). This is the raw-measurement counterpart to [getChEMBLDrugTarget](#)'s curated `drug_mechanism` annotations - see the "Bioassay Queries" vignette section for the annotation-vs-bioassay distinction. Unlike [getChEMBLBioactivities](#) (a simple single-target convenience lookup, kept as-is), this batches arbitrarily many IDs and supports both query directions, matching [getChEMBLDrugTarget](#)'s interface.

Usage

```
getChEMBLBioassay(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  standardType = NA,
  fields = "core",
  verbose = FALSE,
  chunkSize = 200L
)
```

Arguments

<code>queryBy</code>	named list with components <code>molType</code> , <code>idType</code> and <code>ids</code> , same vocabulary as getChEMBLDrugTarget : <code>molType = "protein"</code> with <code>idType</code> containing "Uniprot" for target -> drug bioassay by UniProt accession(s) in <code>ids</code> ; <code>molType = "cmp"</code> with <code>idType = "chembl_id"</code> for drug -> target bioassay by ChEMBL molecule ID(s).
<code>standardType</code>	character(1) or NA; restrict to one measurement type (e.g. "IC50"). NA (default) keeps all types.
<code>fields</code>	"core" (default), "all", or a character vector of column names - see getChEMBLDrugTarget 's <code>fields</code> argument for the general mechanism; use listBioassayFields to browse what's available for this function specifically.
<code>verbose</code>	logical(1); if TRUE, message progress per batch.
<code>chunkSize</code>	integer(1), same batching convention as getChEMBLDrugTarget .

Value

A data.frame with columns `QueryIDs`, `chembl_id`, `Drug_Name`, `ChEMBL_TID`, `UniProt_ID`, `Organism`, `Desc`, `assay_chembl_id`, `assay_description`, `standard_type`, `standard_relation`, `standard_value`, `standard_units`, `pchembl_value` (with `fields = "core"`, the default), plus further `activity.-` prefixed columns when `fields` requests more. Query IDs that return no rows still appear as a single row with all other fields NA.

See Also

[getChEMBLDrugTarget](#), [getChEMBLBioactivities](#), [listBioassayFields](#)

Examples

```
## target -> drug bioassay: FGFR1
getChEMBLBioassay(list(molType = "protein", idType = "Uniprot",
                      ids = "P11362"), standardType = "IC50")
## drug -> target bioassay: dasatinib
getChEMBLBioassay(list(molType = "cmp", idType = "chembl_id",
                      ids = "CHEMBL1421"))
```

getChEMBLDrugTarget	<i>Query known drug-target annotations via the ChEMBL REST API</i>
---------------------	--

Description

REST-API equivalent of [drugTargetAnnot](#) for the package's local ChEMBL SQLite path: given one or more UniProt accessions, returns the drugs annotated (via ChEMBL's `drug_mechanism` table) as acting on that target (target -> drug); given one or more ChEMBL molecule IDs, returns the targets annotated for that compound's mechanism of action (drug -> target). Reproduces `drugTargetAnnot()`'s join across `drug_mechanism`, `molecule_dictionary`, `target_dictionary`, `target_components`, `component_sequences` and `drug_indication` using the ChEMBL REST target, mechanism, molecule and drug_indication resources, so a local ChEMBL SQLite download is not required.

Usage

```
getChEMBLDrugTarget(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  verbose = FALSE,
  chunkSize = 200L,
  fields = "core",
  unichemDbPath = NULL
)
```

Arguments

queryBy	named list with components <code>molType</code> , <code>idType</code> and <code>ids</code> , matching drugTargetAnnot 's interface. <code>molType = "protein"</code> with <code>idType</code> containing "Uniprot" (e.g. "Uniprot") queries target -> drug by UniProt accession(s) in <code>ids</code> . <code>molType = "cmp"</code> with <code>idType = "chembl_id"</code> queries drug -> target by ChEMBL molecule ID(s) in <code>ids</code> . Other <code>idType</code> values (<code>molregno</code> , <code>PubChem_ID</code> , <code>DrugBank_ID</code>) are not yet supported over REST. <code>ids</code> can be arbitrarily large: internally the query is chunked (see <code>chunkSize</code>) and each chunk paginated, so a query of e.g. 2000 UniProt accessions or ChEMBL IDs works the same as one.
verbose	logical(1); if TRUE, message progress per batch.

chunkSize	integer(1) max IDs sent per HTTP request via a <code><field>__in=id1,id2,...</code> filter. ChEMBL's REST server caps the request-line length at ~4KB, which limits a single filter to roughly 250-400 IDs depending on ID length (http2 percent-encodes the separating commas, eating into that budget) — 200 leaves comfortable margin for both short legacy IDs (e.g. "CHEMBL25") and longer modern ones (e.g. "CHEMBL5291677"). Requests are also client-side throttled (see <code>.dtiApiGET</code>) to be a polite API citizen; ChEMBL does not publish a hard rate limit.
fields	"core" (default), "all", or a character vector of column names. "core" returns exactly the curated columns listed under Value below (unchanged from previous versions). "all" additionally returns every other field ChEMBL's target/mechanism/molecule/drug_indication REST resources provide for the matched records, source-prefixed (e.g. <code>target.target_type</code> , <code>mechanism.mechanism_comment</code> , <code>molecule.molecule_properties.alogp</code>) - no extra API calls, these are already fetched and simply no longer discarded. A character vector requests just those extra fields (QueryIDs is always included). See listDrugTargetFields to browse what's available.
unichemDbPath	optional path to a local UniChem SQLite (see buildUnichemDb), used to resolve the PubChem_CID column via getUnichemMapping - ChEMBL's own REST API has no PubChem CID field on the molecule record (its <code>cross_references</code> field is sparse/curator-populated and does not carry PubChem, confirmed live). PubChem_CID is always present in the output; when <code>unichemDbPath = NULL</code> (the default - a pre-built UniChem SQLite is a multi-hundred-MB local resource, not something a routine REST query should require), it is left NA and a message points to <code>buildUnichemDb()</code> .

Details

Only ChEMBL's own native identifiers are supported directly: UniProt accession for the target direction, ChEMBL molecule ID for the drug direction. Translating a different starting identifier (Ensembl gene ID, PubChem CID, DrugBank ID, ...) into one of these is left to a separate, source-agnostic ID-translation step (see [cmpIdMapping](#) for the compound side and [getUniprotIDs/getParalogs](#) for the protein side) so that this function's interface stays uniform across sources.

Value

A `data.frame` with columns `QueryIDs`, `chembl_id`, `Drug_Name`, `PubChem_CID` (see `unichemDbPath`), `MOA`, `Action_Type`, `Max_Phase`, `First_Approval`, `ChEMBL_TID`, `UniProt_ID`, `Desc`, `Organism`, `Mesh_Indication` (with `fields = "core"`, the default), plus further source-prefixed columns when `fields` requests more. Query IDs that return no rows still appear as a single row with all other fields NA, so callers can always confirm which of their input IDs were resolved.

Salt and parent compounds

In the drug → target direction, ChEMBL files a drug's mechanism records against whichever molecule form the curator used, which is frequently a salt rather than the parent normally used to refer to the drug - imatinib (CHEMBL941) has no records of its own, only records on the mesylate salt CHEMBL1642. Both `molecule_chembl_id` and `parent_molecule_chembl_id` are therefore queried and unioned, and results are keyed back to the ID that was asked for, so `chembl_id/Drug_Name`

describe that compound ("IMATINIB") rather than whichever salt carried the record. Querying a salt ID directly still returns that salt. Under `fields = "all"` the underlying record is still visible unchanged in `mechanism.molecule_chembl_id`.

See Also

[drugTargetAnnot](#), [getChEMBLMolecule](#), [listDrugTargetFields](#)

Examples

```
## target -> drug: FGFR1
getChEMBLDrugTarget(list(molType = "protein", idType = "Uniprot",
                        ids = "P11362"))
## drug -> target for a compound whose mechanisms ChEMBL files on a
## salt form (imatinib) - resolved via the parent, see above
getChEMBLDrugTarget(list(molType = "cmp", idType = "chembl_id",
                        ids = "CHEMBL941"))
## drug -> target: dasatinib
getChEMBLDrugTarget(list(molType = "cmp", idType = "chembl_id",
                        ids = "CHEMBL1421"))
## every available field
getChEMBLDrugTarget(list(molType = "cmp", idType = "chembl_id",
                        ids = "CHEMBL1421"), fields = "all")
```

getChEMBLMolecule	<i>Retrieve molecule records from the ChEMBL REST API</i>
-------------------	---

Description

Queries the public ChEMBL REST endpoint for one or more molecules by ChEMBL ID and returns a tidy data.frame of the most commonly used identifiers, structures and computed properties. Complements the package's local-SQLite path ([downloadChEMBLDb](#)) for cases where only a handful of compounds are needed or a local database is not available.

Usage

```
getChEMBLMolecule(chemblIds, verbose = FALSE, chunkSize = 200L)
```

Arguments

chemblIds	character vector of ChEMBL molecule IDs (e.g. "CHEMBL25").
verbose	logical(1); if TRUE, message progress per batch.
chunkSize	integer(1) max IDs looked up per HTTP request (see getChEMBLDrugTarget for why this is capped around 200-400 rather than sent in one request). Large chemblIds vectors are chunked and paginated automatically.

Value

A data.frame with one row per input ID (duplicates and input order preserved) and columns: chembl_id, pref_name, molecule_type, max_phase, first_approval, canonical_smiles, standard_inchi_key, mw_freebase, alogp, hba, hbd, psa, rtb, qed_weighted. IDs that fail to resolve yield a row of NAs.

See Also

[getChEMBLDrugTarget](#), [downloadChEMBLdb](#)

Examples

```
## Requires internet access to www.ebi.ac.uk
df <- getChEMBLMolecule(c("CHEMBL25", "CHEMBL1201585"))
df[, c("chembl_id", "pref_name", "max_phase")]
```

getChEMBLTarget

Search ChEMBL for a target by gene symbol or name

Description

Uses the ChEMBL REST target resource to resolve a query string (gene symbol, protein or preferred name) to ChEMBL target IDs.

Usage

```
getChEMBLTarget(query, organism = "Homo sapiens", limit = 25L)
```

Arguments

query	character(1) search term, e.g. "FGFR1".
organism	character(1) optional NCBI organism name filter, e.g. "Homo sapiens"; NA to disable.
limit	integer(1) maximum number of hits to return.

Value

A data.frame with columns target_chembl_id, pref_name, target_type, organism. Empty data.frame if nothing matches or the API is unreachable.

See Also

[getChEMBLDrugTarget](#)

Examples

```
getChEMBLTarget("FGFR1", organism = "Homo sapiens")
```

getDgidbDrugs*Retrieve DGIdb drug interactions for one or more genes*

Description

Queries the DGIdb interactions(`geneNames: ...`) GraphQL field and returns a tidy data.frame of gene-drug interaction rows. This is the target -> drug direction; see [getDgidbTargets](#) for the reverse.

Usage

```
getDgidbDrugs(  
  genes,  
  pageSize = 500L,  
  maxRows = 5000L,  
  verbose = FALSE,  
  fields = "core"  
)
```

Arguments

<code>genes</code>	character vector of HGNC gene symbols (case-insensitive; unmatched symbols simply contribute no rows, no error).
<code>pageSize</code>	integer(1) rows per GraphQL page (default 500).
<code>maxRows</code>	integer(1) cap on total returned rows (default 5000).
<code>verbose</code>	logical(1); if TRUE, message progress per chunk/page.
<code>fields</code>	"core" (default), "all", or a character vector of column names. "core" returns exactly the curated columns above (unchanged from previous versions). "all" also returns further fields DGIdb's GraphQL API exposes on the matched Interaction/Drug/Gene/Source/ Publication records, source-prefixed (e.g. <code>drug.id</code> , <code>source.citation</code> , <code>publication.pmid</code>) - these are requested from the API in the same query (GraphQL only returns what's asked for), so <code>fields = "all"</code> does cost a wider response, not an extra request. A character vector requests just those extra columns. See listDrugTargetFields to browse what's available.

Value

A data.frame with columns `gene_name`, `drug_name`, `drug_concept_id` (e.g. "chembl:CHEMBL1201585" - DGIdb's own pick of one external ID, not a stable DGIdb-native ID), `drug_aliases` ("; "- collapsed cross-reference IDs DGIdb knows for that drug, e.g. "CHEMBL:CHEMBL941; DRUGBANK:DB00619; ... " - more reliable for linking to other sources than matching on drug name alone), `drug_approved`, `interaction_types`, `directionality`, `interaction_score`, `evidence_score`, `sources` (upstream provenance), `db`. Empty if none / offline.

See Also

[getDgidbTargets](#), [getDgidbDrugTarget](#)

Examples

```
df <- getDgidbDrugs(c("FGFR1", "KLB"))
table(df$gene_name)
```

getDgidbDrugTarget	<i>Query DGIdb interaction data via the uniform queryBy interface</i>
--------------------	---

Description

Thin queryBy-dispatching wrapper over [getDgidbDrugs](#) (target -> drug) and [getDgidbTargets](#) (drug -> target), matching the queryBy = list(molType, idType, ids) interface used by [drugTargetAnnot](#), [getChEMBLDrugTarget](#) and [getPubchemDrugTarget](#). Since DGIdb normalizes matched names to its own canonical casing (see the file header note), the returned QueryIDs column is matched back to the original queryBy\$ids token *case-insensitively* rather than by verbatim string equality.

Usage

```
getDgidbDrugTarget(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  ...
)
```

Arguments

queryBy	named list with components molType, idType and ids. molType = "gene" with idType = "symbol" queries target -> drug (ids = gene symbols). molType = "cmp" with idType = "name" queries drug -> target (ids = drug names). Matches getPubchemDrugTarget 's vocabulary for the same concepts.
...	additional arguments passed through to getDgidbDrugs / getDgidbTargets (e.g. pageSize, maxRows, verbose, and fields - see their fields argument for "core"/"all"/ custom column selection).

Value

A data.frame in the same column shape as [getDgidbDrugs](#) / [getDgidbTargets](#), plus a leading QueryIDs column echoing the original queryBy\$ids token each row case-insensitively matched to. Query IDs that returned no rows still appear as a single row with all other fields NA.

See Also

[getDgidbDrugs](#), [getDgidbTargets](#), [getChEMBLDrugTarget](#), [getPubchemDrugTarget](#), [listDrugTargetFields](#)

Examples

```
## target -> drug: FGFR1, KLB
getDgldbDrugTarget(list(molType = "gene", idType = "symbol",
                        ids = c("FGFR1", "KLB")))

## drug -> target: imatinib
getDgldbDrugTarget(list(molType = "cmp", idType = "name",
                        ids = "imatinib"))

## every available field
getDgldbDrugTarget(list(molType = "cmp", idType = "name",
                        ids = "imatinib"), fields = "all")
```

getDgldbTargets	<i>Retrieve DGldb target interactions for one or more drugs</i>
-----------------	---

Description

Queries the DGldb interactions(`drugNames: ...`) GraphQL field and returns a tidy data.frame of drug-gene interaction rows. This is the drug -> target direction; see [getDgldbDrugs](#) for the reverse. Same columns/shape as [getDgldbDrugs](#) since Interaction nodes always carry both endpoints.

Usage

```
getDgldbTargets(
  drugs,
  pageSize = 500L,
  maxRows = 5000L,
  verbose = FALSE,
  fields = "core"
)
```

Arguments

<code>drugs</code>	character vector of drug names (case-insensitive; unmatched names simply contribute no rows, no error).
<code>pageSize</code>	integer(1) rows per GraphQL page (default 500).
<code>maxRows</code>	integer(1) cap on total returned rows (default 5000).
<code>verbose</code>	logical(1); if TRUE, message progress per chunk/page.
<code>fields</code>	"core", "all", or a character vector of column names, matching getDgldbDrugs 's <code>fields</code> argument.

Value

A data.frame; see [getDgldbDrugs](#) for columns.

See Also

[getDgidbDrugs](#), [getDgidbDrugTarget](#)

Examples

```
df <- getDgidbTargets(c("imatinib", "aspirin"))
table(df$drug_name, useNA = "no")
```

getDrugTarget

getDrugTarget

Description

This function allows you to query a subset of the data fetched by [drugTargetAnnotTable](#).

Usage

```
getDrugTarget(dt_file=file.path(config$resultsPath,"drugTargetAnnot.xls"), queryBy=list(molType=NULL,
id_mapping=c(chembl="chembl_id", pubchem="PubChem_ID", uniprot="UniProt_ID"), columns=config=
```

Arguments

dt_file	The drug target annotation file. This can be generated with drugTargetAnnotTable .
queryBy	A list defining the query, as described in queryBy .
id_mapping	A list providing the id columns for ChEMBL, PubChem, and UniProt. It should contain the fields "chembl", "pubchem", and "uniprot", each with the column name of the respective id number in the drug target annotation file. See default value above for an example.
columns	A list of column indexes to select as a subset of the final result set.
config	General configuration. See genConfig .

Value

Returns the query result as a data frame.

Author(s)

Thomas Girke

See Also

[drugTargetAnnotTable](#)

Examples

```

config = genConfig(chemblDbPath=
  system.file("extdata", "chembl_sample.db", package="drugTargetInteractions"),
  resultsPath =
  system.file("extdata", "results", package="drugTargetInteractions"))
id_mapping <- c(chembl="chembl_id", pubchem="PubChem_ID", uniprot="UniProt_ID", drugbank="DrugBank_ID")

queryBy <- list(molType="cmp", idType="chembl", ids=c("CHEMBL25", "CHEMBL1742471"))
getDrugTarget(queryBy=queryBy, id_mapping=id_mapping,
  columns=c(1,5,8,16,17),config=config)

queryBy <- list(molType="cmp", idType="pubchem", ids=c("2244", "65869", "2244"))
getDrugTarget(queryBy=queryBy, id_mapping=id_mapping,
  columns=c(1,5,8,16,17),config=config)

queryBy <- list(molType="protein", idType="uniprot", ids=c("P43166", "P00915", "P43166"))
getDrugTarget(queryBy=queryBy, id_mapping=id_mapping,
  columns=c(1,5,8,16,17),config=config)

```

getEnsemblOrthologs	<i>Retrieve cross-species orthologs for one or more genes from Ensembl</i>
---------------------	--

Description

Unlike [getParalogs](#) (biomaRt, human paralogs only), this also covers true cross-species orthologs - see [getEnsemblParalogs](#) for the shared rationale and the type/taxonomy_level interpretation.

Usage

```

getEnsemblOrthologs(
  genes,
  species = "human",
  targetSpecies = NULL,
  condensed = FALSE,
  verbose = FALSE
)

```

Arguments

genes	character vector of gene symbols.
species	character(1) Ensembl species name or alias for genes (default "human").
targetSpecies	character vector of Ensembl species names/aliases to restrict orthologs to (e.g. "mouse"), or NULL (default) for all species with a called ortholog.
condensed	logical(1); see getEnsemblParalogs .
verbose	logical(1); if TRUE, message progress per gene.

Value

A data.frame; see [getEnsemblParalogs](#) for columns (homolog_species varies here). Genes with no orthologs simply contribute no rows.

See Also

[getEnsemblParalogs](#), [getParalogs](#)

Examples

```
getEnsemblOrthologs("NLRP3", targetSpecies = "mouse")
```

getEnsemblParalogs	<i>Retrieve within-species paralogs for one or more genes from Ensembl</i>
--------------------	--

Description

Direct replacement for [getParalogs](#) (biomaRt), which is left in place unmodified as a historical/backup option but is not the recommended path going forward - see the `idTranslation.R` file header for the full rationale.

Usage

```
getEnsemblParalogs(
  genes,
  species = "human",
  condensed = FALSE,
  verbose = FALSE
)
```

Arguments

genes	character vector of gene symbols.
species	character(1) Ensembl species name or alias for genes (default "human").
condensed	logical(1); if TRUE, use the faster format=condensed response shape (gene/protein IDs + type only, no perc_id/perc_pos) - much faster for large gene families (see file header), useful when only an existence check or the relationship type is needed.
verbose	logical(1); if TRUE, message progress per gene.

Details

type (e.g. "other_paralog", "within_species_paralog", "gene_split") and taxonomy_level come from Ensembl Compara's gene-tree reconciliation, not raw sequence-identity thresholding - already a more principled "relatedness" signal than perc_id alone. perc_id/perc_pos are useful for ranking within a type tier.

Value

A data.frame with columns query_gene, homolog_id, homolog_protein_id, homolog_species (always the same as species here), type, taxonomy_level, perc_id, perc_pos (the latter two NA when condensed = TRUE). Genes with no paralogs simply contribute no rows.

See Also

[getEnsemblOrthologs](#), [getParalogs](#)

Examples

```
getEnsemblParalogs("FGFR1")
```

getHgncGeneTable	<i>Load the HGNC complete gene set as a gene-centric data.frame</i>
------------------	---

Description

Reads the cached HGNC TSV (see [downloadHgncTable](#)), optionally restricts to protein-coding genes (~19,200 of them), and reshapes the pipe-separated multi-value columns (prev_symbol/alias_symbol/uniprot_ids) into list-columns - one row per approved HGNC gene report. Isoform-suffixed UniProt accessions (e.g. "P12345-2") are stripped defensively, though HGNC's own cross-references are already canonical-only.

Usage

```
getHgncGeneTable(
  proteinCodingOnly = TRUE,
  archiveFile = NULL,
  current = FALSE,
  rerun = FALSE,
  config = genConfig(),
  verbose = FALSE
)
```

Arguments

proteinCodingOnly
logical(1); if TRUE (default), keep only rows where locus_group == "protein-coding gene".

archiveFile, current, rerun, config, verbose
passed to [downloadHgncTable](#); rerun defaults to FALSE here (unlike downloadHgncTable()'s own default) since this higher-level function is the one called routinely.

Value

A data.frame with columns hgnc_id, symbol, prev_symbol (list-column), alias_symbol (list-column), entrez_id, ensembl_gene_id, uniprot_ids (list-column), locus_group. attr(., "hgncSource") records the source filename for provenance.

See Also

[downloadHgncTable](#), [buildHgncSymbolMap](#), [buildGenomeWideDrugTargetTable](#)

Examples

```
hgncTable <- getHgncGeneTable()
nrow(hgncTable)
hgncTable[hgncTable$symbol == "FGFR1", c("symbol", "uniprot_ids")]
```

getOpenTargetsDrugIds *Resolve many drug names to ChEMBL IDs in batched GraphQL requests*

Description

Vectorised, batched the same way [getOpenTargetsIds](#) batches target-symbol resolution. Open Targets drug IDs *are* ChEMBL IDs.

Usage

```
getOpenTargetsDrugIds(names, chunkSize = 100L, pause = 0.1, verbose = FALSE)
```

Arguments

names	character vector of drug names (ChEMBL IDs pass through unresolved - checked via a plain <code>"^CHEMBL[0-9]+\$"</code> regex).
chunkSize	integer(1) names per HTTP request (default 100).
pause	numeric(1) seconds to sleep between requests (politeness).
verbose	logical(1) progress messages.

Value

A named character vector (names = input drug names) of ChEMBL IDs; unresolved names are NA. Order matches the input.

See Also

[getOpenTargetsTargets](#)

Examples

```
getOpenTargetsDrugIds(c("aspirin", "imatinib", "NOT_A_DRUG"))
```

getOpenTargetsDrugs	<i>Retrieve known drugs / clinical candidates for one or more targets from Open Targets (target -> drug)</i>
---------------------	---

Description

Accepts many genes (symbols and/or Ensembl IDs), resolves symbols in bulk via [getOpenTargetsIds](#), then pulls drugAndClinicalCandidates for up to chunkSize targets per HTTP request using GraphQL aliasing. One input target can expand into several output rows (a drug may act on the target via more than one mechanism and may be developed against more than one disease); the cartesian expansion is controlled by expand. Note that agonist / undrugged-direct targets (e.g. FGF21, NLRP3, TFEB, ADIPOR) legitimately return zero rows.

Usage

```
getOpenTargetsDrugs(
  genes,
  expand = c("mechanism", "disease", "drug"),
  chunkSize = 25L,
  pause = 0.1,
  verbose = FALSE,
  fields = "core"
)
```

Arguments

genes	character vector of gene symbols and/or Ensembl gene IDs.
expand	character(1); "mechanism" (default) emits one row per drug x mechanism-of-action and collapses diseases into a single semicolon-delimited string; "disease" emits one row per drug x disease and collapses mechanisms; "drug" emits one row per drug with both mechanisms and diseases collapsed.
chunkSize	integer(1) targets per HTTP request (default 25; kept modest so per-request payloads and server load stay reasonable).
pause	numeric(1) seconds to sleep between requests.
verbose	logical(1) progress messages.
fields	"core" (default), "all", or a character vector of column names. "core" returns exactly the curated columns above (unchanged from previous versions). "all" also returns further fields Open Targets' GraphQL API exposes on the matched Target/Drug records, prefixed target./drug. (e.g. target.biotype, drug.description) - requested in the same query (GraphQL only returns what's asked for), so fields = "all" costs a wider response, not an extra request. A character vector requests just those extra columns. See listDrugTargetFields to browse what's available.

Details

The Open Targets GraphQL schema evolves between platform releases; this function targets the v4 schema in which rows are of type `ClinicalTargetFromTarget` (fields `maxClinicalStage`, `drug`, `diseases`) and mechanism data lives under `drug.mechanismsOfAction.rows`. If Open Targets changes these names, update `.dtiDrugSelection`.

Value

A `data.frame` with columns `ensembl_id`, `approved_symbol`, `drug_id` (ChEMBL), `drug_name`, `drug_type`, `max_clinical_stage`, `mechanism_of_action`, `action_type`, `disease_id`, `disease_name`. Genes with no known drugs simply contribute no rows. Empty `data.frame` if nothing / offline.

See Also

[getOpenTargetsIds](#), [getOpenTargetsTargets](#), [getOpenTargetsDrugTarget](#)

Examples

```
drugs <- getOpenTargetsDrugs(c("FGFR1", "KLB"))
head(drugs[, c("drug_name", "mechanism_of_action", "max_clinical_stage")])
```

`getOpenTargetsDrugTarget`

Query Open Targets drug/target data via the uniform queryBy interface

Description

Thin `queryBy`-dispatching wrapper over [getOpenTargetsDrugs](#) (`target -> drug`) and [getOpenTargetsTargets](#) (`drug -> target`), matching the `queryBy = list(molType, idType, ids)` interface used by [drugTargetAnnot](#), [getChEMBLDrugTarget](#), [getPubchemDrugTarget](#) and [getDgidbDrugTarget](#). Unlike [getDgidbDrugTarget](#) (which joins on a case-normalized display name), this joins the `QueryIDs` column on the stable resolved ID (Ensembl gene ID / ChEMBL ID) that Open Targets already returns verbatim in every output row, so no casing ambiguity arises; the tradeoff is one extra batched ID-resolution call inside the wrapper (see the file header note).

Usage

```
getOpenTargetsDrugTarget(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  ...
)
```

Arguments

queryBy named list with components molType, idType and ids. molType = "gene" with idType = "symbol" queries target -> drug (ids = gene symbols and/or Ensembl gene IDs). molType = "cmp" with idType = "name" queries drug -> target (ids = drug names and/or ChEMBL IDs). Matches [getPubchemDrugTarget](#)/[getDgidbDrugTarget](#)'s vocabulary for the same concepts.

... additional arguments passed through to [getOpenTargetsDrugs](#) / [getOpenTargetsTargets](#) (e.g. expand, chunkSize, verbose, and fields - see their fields argument for "core"/"all"/custom column selection).

Value

A data.frame in the same column shape as [getOpenTargetsDrugs](#) / [getOpenTargetsTargets](#), plus a leading QueryIDs column echoing the original queryBy\$ids token each row resolved from. Query IDs that returned no rows still appear as a single row with all other fields NA.

See Also

[getOpenTargetsDrugs](#), [getOpenTargetsTargets](#), [getChemblDrugTarget](#), [getPubchemDrugTarget](#), [getDgidbDrugTarget](#), [listDrugTargetFields](#)

Examples

```
## target -> drug: FGFR1, KLB
getOpenTargetsDrugTarget(list(molType = "gene", idType = "symbol",
                             ids = c("FGFR1", "KLB")))

## drug -> target: aspirin
getOpenTargetsDrugTarget(list(molType = "cmp", idType = "name",
                             ids = "aspirin"))

## every available field
getOpenTargetsDrugTarget(list(molType = "cmp", idType = "name",
                             ids = "aspirin"), fields = "all")
```

getOpenTargetsIds	<i>Resolve many gene symbols to Ensembl IDs in batched GraphQL requests</i>
-------------------	---

Description

Uses GraphQL field-aliasing to resolve up to chunkSize symbols per HTTP request, so resolving many genes takes a handful of requests rather than one per gene.

Usage

```
getOpenTargetsIds(symbols, chunkSize = 100L, pause = 0.1, verbose = FALSE)
```

Arguments

symbols	character vector of gene symbols (Ensembl gene IDs pass through unresolved - checked via a plain "^ENSG[0-9]+\$" regex).
chunkSize	integer(1) symbols per HTTP request (default 100).
pause	numeric(1) seconds to sleep between requests (politeness).
verbose	logical(1) progress messages.

Value

A named character vector (names = input symbols) of Ensembl gene IDs; unresolved symbols are NA. Order matches the input.

See Also

[getOpenTargetsDrugs](#)

Examples

```
getOpenTargetsIds(c("FGFR1", "KLB", "NOT_A_GENE"))
```

`getOpenTargetsTargets` *Retrieve targets for one or more drugs from Open Targets (drug -> target)*

Description

Accepts many drugs (names and/or ChEMBL IDs), resolves names in bulk via [getOpenTargetsDrugIds](#), then pulls mechanismsOfAction for up to chunkSize drugs per HTTP request using GraphQL aliasing. This is the drug -> target counterpart of [getOpenTargetsDrugs](#). A single mechanism-of-action row can list more than one target (e.g. aspirin's "Cyclooxygenase inhibitor" mechanism lists both PTGS1 and PTGS2); the cartesian expansion is controlled by expand.

Usage

```
getOpenTargetsTargets(  
  drugs,  
  expand = c("target", "mechanism"),  
  chunkSize = 25L,  
  pause = 0.1,  
  verbose = FALSE,  
  fields = "core"  
)
```

Arguments

drugs	character vector of drug names and/or ChEMBL IDs.
expand	character(1); "target" (default) emits one row per mechanism-of-action x target; "mechanism" emits one row per mechanism-of-action and collapses targets into a single semicolon-delimited string.
chunkSize	integer(1) drugs per HTTP request (default 25).
pause	numeric(1) seconds to sleep between requests.
verbose	logical(1) progress messages.
fields	"core", "all", or a character vector of column names, matching getOpenTargetsDrugs 's fields argument (same target./drug.-prefixed extra columns, from the opposite query direction).

Value

A data.frame with columns chembl_id, drug_name, drug_type, max_clinical_stage, mechanism_of_action, action_type, moa_target_name, target_id (Ensembl), approved_symbol. Drugs with no known targets simply contribute no rows. Empty data.frame if nothing / offline.

See Also

[getOpenTargetsDrugIds](#), [getOpenTargetsDrugs](#), [getOpenTargetsDrugTarget](#)

Examples

```
tgts <- getOpenTargetsTargets("aspirin")
tgts[, c("approved_symbol", "mechanism_of_action")]
```

getParalogs	<i>getParalogs</i>
-------------	--------------------

Description

Using biomaRt, obtain for query genes the corresponding UniProt IDs as well as paralogs. Query genes can be Gene Names or ENSEMBL Gene IDs from H sapiens. The result is similar to IDMs and SSNNs from [getUniprotIDs](#) function, but instead of UNIREF clusters, biomaRt's paralogs are used to obtain SSNNs.

Usage

```
getParalogs(queryBy)
```

Arguments

queryBy	A list defining the query, as described in queryBy .
---------	--

Value

Returns a list with the paralogs for the given genes.

Author(s)

Thomas Girke

See Also

[getUniprotIDs queryBy](#)

Examples

```
queryBy <- list(molType="gene", idType="external_gene_name", ids=c("ZPBP", "MAPK1", "EGFR"))

#requires network connection and is slow
result <- getParalogs(queryBy)
```

getPubchemDrugs

Retrieve PubChem bioactivities for one or more genes (target -> drug)

Description

Ports the validated R_Py_code/pubchem_fetch.py reference: resolves each gene symbol to a human NCBI GeneID, pulls its concise bioactivity table, keeps Active + numeric + potency-endpoint rows, collapses to the most potent row per CID (capped at maxCids), and looks up each surviving CID's name and SMILES.

Usage

```
getPubchemDrugs(
  genes,
  taxid = 9606L,
  keepActivities = .dtiPotencyEndpoints,
  maxCids = 400L,
  pause = 0.34,
  verbose = FALSE,
  fields = "core"
)
```

Arguments

genes	character vector of HGNC gene symbols (or NCBI GeneIDs, which pass through unresolved).
taxid	integer(1) NCBI taxonomy ID for symbol resolution (default 9606 = human).

keepActivities	character vector of Activity Name values to keep (default IC50/Ki/Kd/EC50/AC50/Potency); NULL keeps all.
maxCids	integer(1) cap on CIDs kept per gene (default 400).
pause	numeric(1) seconds between requests (NCBI: <=3 req/s without an API key).
verbose	logical(1) progress messages.
fields	"core" (default), "all", or a character vector of column names. "core" returns exactly the curated columns above (unchanged from previous versions). "all" also returns every other column PubChem's concise bioactivity table provides, prefixed activity. (e.g. activity.PubMed_ID, activity.RNAi - no extra API calls, already fetched). A character vector requests just those extra columns. See listDrugTargetFields to browse what's available.

Value

A data.frame with columns gene_symbol, geneid, target_accession, cid, drug_name (~27 breadth), canonical_smiles, activity_name, activity_value_uM, assay_name, db. Empty if none / offline.

See Also

[getPubchemTargets](#), [getPubchemDrugTarget](#)

Examples

```
df <- getPubchemDrugs(c("FGFR1", "KLB"))
table(df$gene_symbol)
```

getPubchemDrugTarget *Query PubChem bioactivity data via the uniform queryBy interface*

Description

Thin queryBy-dispatching wrapper over [getPubchemDrugs](#) (target -> drug) and [getPubchemTargets](#) (drug -> target), matching the queryBy = list(molType, idType, ids) interface used by [drugTargetAnnot](#) and [getChEMBLDrugTarget](#), so a future cross-source ID-translation/meta layer can dispatch to any source function the same way. PubChem's two directions are genuinely asymmetric (no shared bulk filter covers both, unlike ChEMBL's drug_mechanism join), so this only adds a QueryIDs column and unmatched-ID NA-row padding on top of the two underlying functions - it does not change their query logic.

Usage

```
getPubchemDrugTarget(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  ...
)
```

Arguments

queryBy named list with components `molType`, `idType` and `ids`. `molType = "gene"` with `idType = "symbol"` queries target -> drug; `ids` may be HGNC gene symbols or NCBI GeneIDs (numeric strings pass through unresolved), matching [getPubchemDrugs](#). `molType = "cmp"` with `idType = "name"` queries drug -> target; `ids` may be compound names or PubChem CIDs (numeric strings pass through unresolved), matching [getPubchemTargets](#). Other identifier types (UniProt accession, ChEMBL ID, PubChem CID as the **only** form, DrugBank ID, ...) are not yet supported over this wrapper.

... additional arguments passed through to [getPubchemDrugs](#) / [getPubchemTargets](#) (e.g. `taxid`, `keepActivities`, `verbose`, and `fields` - see their `fields` argument for `"core"/"all"/` custom column selection).

Value

A data.frame in the same column shape as [getPubchemDrugs](#) / [getPubchemTargets](#), plus a leading `QueryIDs` column echoing the original `queryBy$ids` token each row resolved from. Query IDs that returned no rows still appear as a single row with all other fields NA, so callers can always confirm which of their input IDs were resolved.

See Also

[getPubchemDrugs](#), [getPubchemTargets](#), [getChEMBLDrugTarget](#), [listDrugTargetFields](#)

Examples

```
## target -> drug: FGFR1, KLB
getPubchemDrugTarget(list(molType = "gene", idType = "symbol",
                          ids = c("FGFR1", "KLB")))

## drug -> target: aspirin
getPubchemDrugTarget(list(molType = "cmp", idType = "name",
                          ids = "aspirin"))

## every available field
getPubchemDrugTarget(list(molType = "cmp", idType = "name",
                          ids = "aspirin"), fields = "all")
```

getPubchemTargets	<i>Retrieve PubChem bioactivities for one or more drugs (drug -> target)</i>
-------------------	---

Description

Drug -> target counterpart of [getPubchemDrugs](#): resolves each drug name to a CID, pulls its compound-centric assay-summary table (batched), keeps Active + numeric + potency-endpoint rows, collapses to the most potent row per (CID, GeneID) pair, and resolves each target GeneID to a gene symbol - filtered to `taxid` (default human), since compound assay data spans whatever species each assay used (see file header note).

Usage

```
getPubchemTargets(  
  drugs,  
  taxid = 9606L,  
  keepActivities = .dtiPotencyEndpoints,  
  maxCidsPerBatch = 50L,  
  pause = 0.2,  
  verbose = FALSE,  
  fields = "core"  
)
```

Arguments

drugs	character vector of drug/compound names (or PubChem CIDs, which pass through unresolved).
taxid	integer(1) or NULL; keep only targets from this NCBI taxonomy ID (default 9606 = human), or all species if NULL.
keepActivities	character vector of Activity Name values to keep (default IC50/Ki/Kd/EC50/AC50/Potency); NULL keeps all.
maxCidsPerBatch	integer(1) CIDs per assaysummary HTTP request (default 50).
pause	numeric(1) seconds between name-resolution requests.
verbose	logical(1) progress messages.
fields	"core" (default), "all", or a character vector of column names, matching getPubchemDrugs 's fields argument (same activity.*-prefixed extra columns from the compound-centric assaysummary table).

Value

A data.frame with columns cid, drug_name, gene_symbol, geneid, taxid, target_accession, activity_name, activity_value_uM, assay_name, db. Empty if none / offline.

See Also

[getPubchemDrugs](#), [getPubchemDrugTarget](#)

Examples

```
df <- getPubchemTargets("aspirin")  
df[, c("drug_name", "gene_symbol", "activity_name", "activity_value_uM")]
```

`getSymEnsUp`*Gene to Protein ID Mappings*

Description

The `getSymEnsUp` function returns for a query of gene or protein IDs a mapping table containing: ENSEMBL Gene IDs, Gene Names/Symbols, UniProt IDs and ENSEMBL Protein IDs. Subsequent slots contain the corresponding named character vectors. Internally, the function uses the `ensemldb` package.

Usage

```
getSymEnsUp(EnsDb = "EnsDb.Hsapiens.v86", ids, idtype)
```

Arguments

<code>EnsDb</code>	EnsDb instance of <code>ensemldb</code> package
<code>ids</code>	Character vector with IDs matching the type specified under <code>idtype</code>
<code>idtype</code>	Character vector of length one containing one of: <code>GENE_NAME</code> , <code>ENSEMBL_GENE_ID</code> or <code>UNIPROT_ID</code>

Value

List object with following components:

<code>idDF</code>	ID mapping data.frame
<code>ens_gene_id</code>	named character vector
<code>up_ens_id</code>	named character vector
<code>up_gene_id</code>	named character vector

Author(s)

Thomas Girke

Examples

```
gene_name <- c("CA7", "CFTR")
getSymEnsUp(EnsDb="EnsDb.Hsapiens.v86", ids=gene_name, idtype="GENE_NAME")
```

getUnichemMapping	<i>Translate compound identifiers between databases via a local UniChem SQLite</i>
-------------------	--

Description

Genuinely source-agnostic: from/to accept any source name (or raw numeric source ID) present in the unichem_sources table of the database built by [buildUnichemDb](#) - not a fixed enum hand-coded per database. A source this package has no dedicated accessor for (e.g. "gtopdb", "drugcentral", or any source added to a future UniChem refresh) works identically to "chembl" or "pubchem" the moment it is present in the underlying data - no code changes needed. Implemented as a single SQL self-join on the shared UniChem structure cluster ID (UCI), which is what makes this generic: the query never references a specific source by name in its logic, only in the two parameter values.

Usage

```
getUnichemMapping(ids, from, to, dbPath)
```

Arguments

ids	character vector of source-database identifiers.
from	character(1) source database name (e.g. "chembl", "pubchem", "drugbank", "chebi") or numeric source ID, matching unichem_sources\$name/src_id.
to	character(1) target database name or numeric source ID.
dbPath	character(1) path to the UniChem SQLite, e.g. from buildUnichemDb .

Value

A data.frame with columns From and To. IDs with no cross-reference to to contribute no row (the underlying table only carries compounds with ≥ 2 source cross-references in the first place - see [buildUnichemDb](#) - so an unmapped ID may mean either no to-side entry exists, or the compound had no cross-reference to *any* other source and was excluded from the table entirely).

See Also

[buildUnichemDb](#), [getUniprotMapping](#)

Examples

```
dbPath <- buildUnichemDb()
getUnichemMapping("CHEMBL25", from = "chembl", to = "pubchem", dbPath)
getUnichemMapping("CHEMBL25", from = "chembl", to = "drugbank", dbPath)
```

getUniprotIDs	<i>Retrieve UniProt IDs via ID and Cluster Mappings</i>
---------------	---

Description

The following returns for a set of query IDs (e.g. Ensembl gene IDs) the corresponding UniProt IDs based on two independent approaches: ID mappings (IDMs) and sequence similarity nearest neighbors (SSNNs) using UNIREF clusters. Note, the 'keys' or query IDs (e.g. ENSEMBL genes) can only be reliably maintained in the SSNN results when 'chunksize=1' since batch queries for protein clusters with 'UnitProt.ws' will often drop the query IDs. To address this, the query result contains an extra 'QueryID' column when 'chunksize=1', but not when it is set to a different value than 1.

The [getParalogs](#) function is similar but it uses biomaRt's paralogs instead of UNIREF clusters.

Usage

```
getUniprotIDs(taxId = 9606, kt = "ENSEMBL", keys, seq_cluster = "UNIREF90", chunksize=20)
```

Arguments

taxId	An NCBI taxonomy ID
kt	Should be either "ENSEMBL" or "UNIPROTKB".
keys	Query IDs.
seq_cluster	Which cluster to use. Should be one of 'UNIREF100', 'UNIREF90', 'UNIREF50'.
chunksize	Queries are done in batches, this parameter sets the size of each batch.

Value

Returns a list of data.

Author(s)

Thomas Girke

See Also

[getParalogs UniProt.ws](#)

Examples

```
keys <- c("ENSG00000145700", "ENSG00000135441", "ENSG00000120071", "ENSG00000120088", "ENSG00000185829", "ENSG00000120088")
res_list100 <- getUniprotIDs(taxId=9606, kt="ENSEMBL", keys=keys, seq_cluster="UNIREF100")
```

getUniprotMapping	<i>Map identifiers between databases via the UniProt REST ID Mapping API</i>
-------------------	--

Description

Direct replacement for `getUniprotIDs` (UniProt.ws), which is left in place unmodified as a historical/backup option but is not the recommended path going forward - see the `idTranslation.R` file header for the full rationale (reliability and a real organism-filtering bug in UniProt.ws that this direct implementation avoids by construction, since it calls the same REST service UniProt.ws itself wraps).

Usage

```
getUniprotMapping(
  ids,
  from,
  to,
  taxId = 9606L,
  pollInterval = 1,
  maxWait = 120,
  verbose = FALSE
)
```

Arguments

<code>ids</code>	character vector of source-database identifiers (e.g. gene symbols, UniProt accessions, Ensembl gene IDs). Deduplicated internally before submission.
<code>from</code>	character(1) source database name, in UniProt's own vocabulary (e.g. "Gene_Name", "Ensembl", "UniProtKB_AC-ID"). See https://www.uniprot.org/help/id_mapping for the full list.
<code>to</code>	character(1) target database name (e.g. "UniProtKB-Swiss-Prot" for canonical reviewed entries only, "UniProtKB" for reviewed+unreviewed, "Ensembl", "Gene_Name").
<code>taxId</code>	integer(1) or NULL; NCBI taxonomy ID to restrict matches to one organism (default 9606L = human), or NULL to span all organisms matching <code>ids</code> . Applied server-side at job-submission time (confirmed live to filter correctly, unlike the historical UniProt.ws bug noted above).
<code>pollInterval</code>	numeric(1) seconds between job-status polls.
<code>maxWait</code>	numeric(1) seconds to wait for job completion before erroring.
<code>verbose</code>	logical(1); if TRUE, message the job ID and size on submission.

Details

Submits an asynchronous ID mapping job, polls until it finishes, and returns every (From, To) pair found, paginating through all result pages automatically.

Value

A `data.frame` with columns `From` and `To`. IDs that don't map contribute no row (matching the underlying API's behavior - there is no per-ID failure signal to NA-pad against, only an aggregate list of unmapped IDs the API does not currently surface in the paginated JSON results). Note: `to = "Ensembl"` results carry a version suffix (e.g. "ENSG00000077782.24"), unlike the bare IDs returned elsewhere in this package - strip `sub("\\..*$", "", x)` if a bare ID is needed.

See Also

[getUniprotIDs](#)

Examples

```
getUniprotMapping(c("FGFR1", "KLB"), from = "Gene_Name",
                  to = "UniProtKB-Swiss-Prot")
getUniprotMapping("P11362", from = "UniProtKB_AC-ID", to = "Ensembl")
```

`gtoPdbTargetAnnot`

Query GtoPdb drug-target interactions bidirectionally

Description

Queries the local GtoPdb SQLite built by [buildGtoPdbDb](#), using the same `queryBy = list(molType, idType, ids)` convention as [ttdTargetAnnot/broadRepurposingHubAnnot](#) - including its `QueryIDs` column: every row of the result is tagged with the original query token it matched, and query IDs that returned no rows still appear as a single row with all other fields NA.

- `molType = "protein"` (or `"gene"`), `idType` one of `"symbol"` (HGNC gene symbol) or `"gtp_target_id"` (GtoPdb's own numeric target ID) -> target -> drug.
- `molType = "cmp"`, `idType` one of `"name"` (ligand name) or `"gtp_ligand_id"` (GtoPdb's own numeric ligand ID) -> drug -> target.

`"symbol"/"name"` lookups are case-insensitive; the `"gtp_target_id"/"gtp_ligand_id"` ID lookups are exact-match.

Usage

```
gtoPdbTargetAnnot(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  gtoPdbDbPath,
  fields = "core"
)
```

Arguments

queryBy	list with components molType, idType, ids (character vector).
gtoPdbDbPath	character(1) path to the GtoPdb SQLite, e.g. from buildGtoPdbDb .
fields	"core" (default) or "all" - both return every gtp_interactions column, since (unlike the REST-backed sources) there is no larger raw payload to opt into - or a character vector of column names to keep (QueryIDs is always retained). See listDrugTargetFields .

Value

A data.frame with columns QueryIDs, target_gene, targetId, targetName, species, primaryTarget, ligandId, ligandName, type, action, affinity, affinityParameter, selectivity, refIds (with the default fields = "core"), or a subset when fields requests specific columns.

See Also

[buildGtoPdbDb](#), [ttdTargetAnnot](#), [broadRepurposingHubAnnot](#), [listDrugTargetFields](#)

Examples

```
dbPath <- buildGtoPdbDb()
gtoPdbTargetAnnot(list(molType = "protein", idType = "symbol",
                      ids = c("FGFR1", "IL1B")), dbPath)
gtoPdbTargetAnnot(list(molType = "cmp", idType = "name",
                      ids = "pemigatinib"), dbPath)
```

listBioassayFields	<i>List the fields available from a bioassay source under fields = "all"</i>
--------------------	--

Description

The bioassay-track counterpart of [listDrugTargetFields](#), kept as a separate function rather than adding new source keys there - this package deliberately keeps drug-target *annotation* data (curated mechanism-of-action calls: ChEMBL/DGIdb/Open Targets/TTD, see [listDrugTargetFields](#)) and raw *bioassay* measurements (individual assay results: ChEMBL/PubChem, this function) as two distinct tracks everywhere, including field discovery. See the "Bioassay Queries" vignette section for the full distinction.

Usage

```
listBioassayFields(source = c("chembl", "pubchem"), queryBy = NULL, ...)
```

Arguments

source	character(1); one of "chembl" (see getChemblBioassay) or "pubchem" (see getPubchemDrugTarget).
queryBy	optional queryBy list to probe live instead of returning the static documented list (see listDrugTargetFields 's queryBy argument for the same tradeoff).
...	additional arguments passed to the source function when queryBy is supplied (e.g. standardType for getChemblBioassay , verbose).

Value

character vector of column names.

See Also

[getChemblBioassay](#), [getPubchemDrugTarget](#), [listDrugTargetFields](#)

Examples

```
listBioassayFields("chembl")
```

listDrugTargetFields	<i>List the fields available from a drug-target source under fields = "all"</i>
----------------------	---

Description

A quick reference for what [getChemblDrugTarget\(\)](#), [getPubchemDrugTarget\(\)](#), [getDgidbDrugTarget\(\)](#), [getOpenTargetsDrugTarget\(\)](#), [ttdTargetAnnot\(\)](#), [broadRepurposingHubAnnot\(\)](#) and [gtoPdbTargetAnnot\(\)](#) can return when called with fields = "all" (or requested individually via fields = c(...)). By default this returns a documented list with no network/database call; pass queryBy to instead run a live single-query probe through the matching source function and report the columns it actually returned (slower, but exact for that query's shape - useful since some fields only appear for certain record types, e.g. protein-family ChEMBL targets vs single proteins; for "ttd"/"broad"/"gtopdb" this is exact rather than best-effort either way, see [ttdTargetAnnot](#) et al.). The local-SQLite sources ("ttd", "broad", "gtopdb") additionally require their database path (ttdDbPath, brhDbPath, gtoPdbDbPath respectively) passed via ... when queryBy is supplied.

Usage

```
listDrugTargetFields(
  source = c("chembl", "pubchem", "dgidb", "opentargets", "ttd", "broad", "gtopdb"),
  queryBy = NULL,
  ...
)
```

Arguments

source	character(1); one of "chembl", "pubchem", "dgidb", "opentargets", "ttd", "broad", "gtopdb".
queryBy	optional queryBy list (see getChemblDrugTarget) to probe live instead of returning the static documented list.
...	additional arguments passed to the source function when queryBy is supplied (e.g. verbose, or the local-SQLite sources' database path).

Value

character vector of column names.

See Also

[getChemblDrugTarget](#), [getPubchemDrugTarget](#), [getDgidbDrugTarget](#), [getOpenTargetsDrugTarget](#), [ttdTargetAnnot](#), [broadRepurposingHubAnnot](#), [gtoPdbTargetAnnot](#)

Examples

```
listDrugTargetFields("chembl")
listDrugTargetFields("ttd")
```

listMoaSources	<i>Which sources carry mechanism-of-action terms, and what the others store</i>
----------------	---

Description

Reference table behind [assembleMoaTable](#)'s source selection, including the distinct-value counts that separate a MOA field from an action-type field. All figures measured from the sources themselves rather than taken from their documentation.

Usage

```
listMoaSources()
```

Value

A data.frame with one row per data source and columns source, column, kind, n_distinct, used_for_moa and note.

See Also

[assembleMoaTable](#), [combineDrugTargets](#)

Examples

```
listMoaSources()
```

mergeDrugTargets	<i>Join the per-source tables side by side, one row per gene-drug pair</i>
------------------	--

Description

The horizontal counterpart to [combineDrugTargets](#), which stacks the sources on top of one another. This one places them side by side: one row per key, with each source's own columns appended and prefixed by the source name.

Usage

```
mergeDrugTargets(
  results,
  by = c("hgnc_id", "compound_chembl_id"),
  columns = NULL,
  colMap = drugTargetColumnMap(),
  collapse = c("list", "string"),
  sep = " | ",
  verbose = FALSE
)
```

Arguments

results	a named list of per-source data.frames, as returned by queryDrugTargets or buildGenomeWideDrugTargetTable . Shared keys are added with addCommonIds if not already present.
by	character vector of key columns, any of "hgnc_id", "compound_chembl_id", "gene_symbol", "target_uniprot".
columns	character vector of columns to carry, given without the source prefix; the default carries every column each source has apart from the keys and QueryIDs. A shared column name from colMap ("drug_name") selects whatever each source calls it, so you do not have to name all six variants; a source's own column name ("Drug_Name") also works.
colMap	the column mapping used to resolve shared names in columns; default drugTargetColumnMap() .
collapse	"list" (default) for list-columns, or "string" to paste each cell's values together with sep.
sep	separator used when collapse = "string".
verbose	logical(1); if TRUE, report per-source row counts and anything dropped for want of a key.

Details

Every source reports several rows for the same gene-drug pair - one per mechanism in ChEMBL, one per disease in Open Targets, one per assay in GtoPdb - so the values are collapsed to the

distinct ones per key before the sources are put together. Collapsed values are returned as list-columns, which keep them addressable with `lengths()` and `unlist()`; use `collapse = "string"` to get them as delimited text instead, which is what you want for writing the table to a file.

The key is set by `by`. The default, `c("hgnc_id", "compound_chembl_id")`, gives one row per gene-drug pair; `by = "hgnc_id"` gives one row per gene, with each source's drugs collapsed into that gene's cell. Only three sources currently identify compounds by ChEMBL ID (see [addCommonIds](#)), so a query including TTD, the Broad Hub or GtoPdb keyed on a compound will drop their rows, reporting how many; key on `"hgnc_id"` to keep them.

`n_sources` counts the sources that returned a row for that key, which is not the same as the sources that found a drug. A genome-wide build queries every gene everywhere, so most of its genes carry a row from every source with the drug columns empty. To count the sources that actually found something, test the cells: `lengths(x$chembl_Drug_Name) > 0`.

Value

A [DataFrame](#) with one row per distinct `by` combination: the key columns first, then any gene identity the tables carry (`symbol`, `ensembl_gene_id` - once, not once per source), then `n_sources` and `sources`, then each source's carried columns prefixed with its name.

See Also

[combineDrugTargets](#) for the row-append form, [addCommonIds](#) for the keys themselves.

Examples

```
res <- queryDrugTargets(list(molType = "gene", idType = "symbol", ids = "FGFR1"),
  sources = c("chembl", "opentargets"))
## Each source's own column names...
mergeDrugTargets(res, columns = c("Drug_Name", "Action_Type", "action_type"))
## ...or the shared names, which resolve to the same columns.
mergeDrugTargets(res, columns = c("drug_name", "action"))
```

moaWide

Collapse a MOA table to one row per drug

Description

Reshapes [assembleMoaTable](#)'s output column-wise, with a list-column of the distinct MOA terms each drug carries.

Usage

```
moaWide(x)
```

Arguments

`x` a `data.frame` as returned by [assembleMoaTable](#).

Value

A data.frame with one row per query_id x drug_key, columns query_id, drug_key, drug_name, n_sources, n_moa, and the list-columns sources, moa_text and action_type.

See Also

[assembleMoaTable](#)

Examples

```
chembl <- data.frame(
  QueryIDs = "CHEMBL941", chembl_id = "CHEMBL941",
  Drug_Name = "IMATINIB",
  MOA = c("Bcr/Abl fusion protein inhibitor",
    "Stem cell growth factor receptor inhibitor"),
  Action_Type = "INHIBITOR", UniProt_ID = c("P00519", "P10721"))
wide <- moaWide(assembleMoaTable(list(chembl = chembl)))
wide[, c("drug_key", "drug_name", "n_sources", "n_moa")]
wide$moa_text[[1]]
```

normalizeGeneSymbols *Normalize gene symbols to their current HGNC-approved form*

Description

Passes already-current symbols straight through; resolves anything else via a prev/alias-symbol map (see [buildHgncSymbolMap](#)); anything still unresolved becomes NA, with the unmapped input symbols reported via attr(., "unmapped") rather than silently dropped.

Usage

```
normalizeGeneSymbols(symbols, symbolMap = NULL, hgncTable = NULL)
```

Arguments

symbols	character vector of gene symbols to normalize.
symbolMap	named character vector as returned by buildHgncSymbolMap ; built from hgncTable if not supplied.
hgncTable	data.frame as returned by getHgncGeneTable ; used to build symbolMap (and to recognise already-current symbols) when symbolMap is not supplied directly. Fetched automatically if neither argument is given.

Value

character vector, same length/order as symbols. Symbols that could not be resolved at all are NA and are listed in attr(., "unmapped"). Symbols that resolve to more than one current symbol keep the first (alphabetically) and are listed in attr(., "ambiguous"), with a warning(). Only the symbols given are considered: a symbol that is already current is passed straight through and never counts as ambiguous, so translating unaffected symbols is silent.

See Also

[buildHgncSymbolMap](#), [getHgncGeneTable](#)

Examples

```
hgncTable <- getHgncGeneTable()
normalizeGeneSymbols(c("FGFR1", "ABL"), hgncTable = hgncTable)
```

processDrugage	<i>processDrugage</i>
----------------	-----------------------

Description

Download Drug Age data from genomics.senescence.info/drugs. Process the data and write it out as a TSV spreadsheet.

Usage

```
processDrugage(drugagefile=file.path(config$resultsPath,"drugage_id_mapping.xls"), redownloaddrugage
```

Arguments

drugagefile	The name of the output file.
redownloaddrugage	If true, download the data file. Otherwise assume the file is already downloaded.
config	General configuration. See genConfig .

Value

Output is written to drugagefile.

Author(s)

Thomas Girke

Examples

```
tryCatch({
  config = genConfig(chemblDbPath=
    system.file("extdata", "chembl_sample.db", package="drugTargetInteractions"))
  processDrugage("drugage_id_mapping.xls", TRUE, config)
},
error=function(e){
  message("Failed to run processDrugage(), please try again later")
}
)
```

queryDrugTargets	<i>Query drug-target interactions across multiple sources from any recognised starting identifier</i>
------------------	---

Description

The tying-together piece of the ID-translation layer (see `idTranslation.R`, `unichemAccess.R`): resolves `queryBy$ids` to whatever native identifier each requested source needs (see `.resolveGeneIds/` `.resolveCompoundIds` for how), then dispatches to that source's own bidirectional *annotation* query function (`getChEMBLDrugTarget`, `getDgIdbDrugTarget`, `getOpenTargetsDrugTarget`, `ttdTargetAnnot`, `broadRepurposingHubAnnot`, `gtoPdbTargetAnnot`). Named `queryDrugTargets()`, not `getDrugTarget()` - that name is already taken by an older, unrelated function in `drugTargetAnnotations_Fct.R`.

Usage

```
queryDrugTargets(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  sources = names(.dtiMetaSources),
  ttdDbPath = NULL,
  brhDbPath = NULL,
  gtoPdbDbPath = NULL,
  unichemDbPath = NULL,
  taxId = 9606L,
  verbose = FALSE,
  ...
)
```

Arguments

queryBy	named list with components <code>molType</code> , <code>idType</code> and <code>ids</code> . <code>molType</code> is "gene" or "protein" (accepted interchangeably) for the target -> drug direction, or "cmp" for drug -> target. <code>idType</code> is one of "symbol", "uniprot", "ensembl" for the gene/ protein side (see <code>.resolveGeneIds</code>), or one of "name", "chembl_id", "pubchem_id", "drugbank_id", "chebi_id" for the compound side (see <code>.resolveCompoundIds</code>) - not each source's own native vocabulary, which this function translates to internally.
sources	character vector, any of "chembl", "dgidb", "opentargets", "ttd", "broad", "gtopdb" (default: all six annotation sources; PubChem is not included here, see Details).
ttdDbPath	character(1) path to a local TTD SQLite (see <code>buildTtdDb</code>); required if "ttd" is in <code>sources</code> , and the function stops before querying anything if it is missing. Not built automatically - a TTD build is a real, deliberate operation, not something to trigger silently from inside a dispatcher.
brhDbPath	character(1) path to a local Broad Repurposing Hub SQLite (see <code>buildBroadRepurposingHubDb</code>); required if "broad" is in <code>sources</code> . Not built automatically, same rationale as <code>ttdDbPath</code> .

gtoPdbDbPath	character(1) path to a local GtoPdb SQLite (see buildGtoPdbDb); required if "gtopdb" is in sources. Not built automatically, same rationale as ttdDbPath.
unichemDbPath	character(1) path to a local UniChem SQLite (see buildUnichemDb); required whenever compound-side resolution needs it (structured-ID-to-structured-ID or structured-ID-to-name translation - see .resolveCompoundIds). Not built automatically - buildUnichemDb takes on the order of an hour.
taxId	integer(1) NCBI taxonomy ID for gene/protein-side resolution (default 9606L = human); passed to getUniprotMapping .
verbose	logical(1); if TRUE, message resolution and per-source progress, and why a source was skipped.
...	additional arguments passed through to each dispatched source function (e.g. expand for getOpenTargetsDrugTarget , or fields - supported by every dispatchable source - to request "core"/"all" columns or a specific subset; passed identically to every source named in sources, so mixing sources whose column sets differ is only useful with fields = "core"/"all" or a subset common to all of them).

Details

PubChem is intentionally not one of the dispatchable sources: its REST accessors return raw bioassay measurements, not curated drug-target annotations, a different data type from what this function and [combineDrugTargets](#) harmonize. Use [getPubchemDrugTarget](#) directly, or [getChEMBLBioassay](#) for ChEMBL's own bioassay data - see the "Bioassay Queries" vignette section.

Returns a named list, one element per successfully-queried source - **not** one unified table. Cross-source schema harmonization (e.g. a proper S4 container) is a separate, deferred phase; this function's job is ID resolution and dispatch, not schema design. A source contributes no list element at all (rather than an empty data.frame) when none of queryBy\$ids resolved to that source's native ID type, or when the source's own function raised an error (e.g. a required local database path was not supplied) - set verbose = TRUE to see why.

Value

A named list, one element (a data.frame) per source that returned a result. attr(result, "resolved") holds the per-source ID-resolution vectors (named by the original queryBy\$ids, as returned by [.resolveGeneIds](#)/ [.resolveCompoundIds](#)) for tracing which input ID led to which rows.

See Also

[getChEMBLDrugTarget](#), [getDgidbDrugTarget](#), [getOpenTargetsDrugTarget](#), [ttdTargetAnnot](#), [broadRepurposingHubAnnot](#), [gtoPdbTargetAnnot](#), [getPubchemDrugTarget](#), [getChEMBLBioassay](#), [getUniprotMapping](#), [getUnichemMapping](#)

Examples

```
## target -> drug, starting from an Ensembl gene ID, all live sources
res <- queryDrugTargets(
  list(molType = "gene", idType = "ensembl", ids = "ENSG00000077782"),
  sources = c("chembl", "dgidb", "opentargets"))
```

```

names(res)
res$chembl

## drug -> target, starting from a DrugBank ID (needs a UniChem SQLite)
res2 <- queryDrugTargets(
  list(molType = "cmp", idType = "drugbank_id", ids = "DB00945"),
  sources = "chembl", unichemDbPath = buildUnichemDb(rerun = FALSE))

```

queryMoa	<i>Query the MOA-carrying sources and assemble their MOA table in one call</i>
----------	--

Description

Convenience wrapper: [queryDrugTargets](#) followed by [assembleMoaTable](#). Call the two separately when you also want the target link ([assembleMoaTargets](#)) or each source's full native columns, both of which this wrapper discards.

Usage

```
queryMoa(queryBy, sources = names(.dtiMoaSourceSpec), ...)
```

Arguments

queryBy	a list(molType, idType, ids) as accepted by queryDrugTargets .
sources	character vector of sources to query; defaults to the three that carry MOA terms (see listMoaSources). One of them, the Broad Repurposing Hub, is read from a local SQLite, so a default call needs <code>brhDbPath</code> (see buildBroadRepurposingHubDb); without it the call stops rather than returning a table quietly missing one of the three MOA sources. Pass <code>sources = c("chembl", "opentargets")</code> to query only the two that need no local database.
...	further arguments passed to queryDrugTargets , e.g. <code>brhDbPath</code> , <code>unichemDbPath</code> , <code>verbose</code> .

Value

The data.frame described in [assembleMoaTable](#).

Note

Querying compounds by `idType = "name"` routes through [queryDrugTargets](#)'s compound resolution, which reaches ChEMBL and Open Targets only via a ChEMBL ID - so those two need `unichemDbPath` (see [buildUnichemDb](#)) when the query starts from a name. Without it they are skipped and the table is assembled from whatever remains.

See Also

[assembleMoaTable](#), [assembleMoaTargets](#)

Examples

```
queryMoa(list(molType = "cmp", idType = "chembl_id", ids = "ChEMBL941"),
          sources = c("chembl", "opentargets"))
```

```
runDrugTarget_Annot_Bioassay
      runDrugTarget_Annot_Bioassay
```

Description

Meta-function to obtain in one step both drug-target annotation and bioassay data.

Usage

```
runDrugTarget_Annot_Bioassay(res_list, up_col_id="ID", ens_gene_id, cmpid_file=file.path(config$resu
```

Arguments

<code>res_list</code>	Object obtained from <code>getUniprotIDs</code> function.
<code>up_col_id</code>	Column name in <code>data.frames</code> of <code>res_list</code> containing uniprot IDs, usually one of: <code>'ID'</code> , <code>'ID_up_sp'</code> or <code>'ID_up_sp_tr'</code> .
<code>ens_gene_id</code>	Named character vector with ENSEMBL gene IDs in name slot and gene symbols or other ID type in value slot
<code>cmpid_file</code>	Path to CMP ID mapping file, often named <code>cmp_ids.rds</code> .
<code>config</code>	General configuration. See genConfig .
<code>...</code>	Slot to pass on additional arguments.

Value

List with two components each containing a `data.frame`. The first one (Annotation) contains drug-target annotation data, and the second one (Bioassay) contains drug-target bioassay data.

Author(s)

Thomas Girke

References

References to be added...

See Also

See also: `drugTargetAnnot` and `drugTargetBioactivity`

Examples

```

config = genConfig(chemblDbPath=
  system.file("extdata", "chembl_sample.db", package="drugTargetInteractions"),
  resultsPath =
  system.file("extdata", "results", package="drugTargetInteractions"))

## (1) Translate gene symbols to ENSEMBL gene IDs
ensembl_gene_id <- c("ENSG00000001626", "ENSG000000168748")
idMap <- getSymEnsUp(EnsDb="EnsDb.Hsapiens.v86", ids=ensembl_gene_id, idtype="ENSEMBL_GENE_ID")
ens_gene_id <- idMap$ens_gene_id

## (2a) Retrieve UniProt IDs with both IDMs and SSNN paralogs
queryBy <- list(molType="gene", idType="ensembl_gene_id", ids=names(ens_gene_id))

#this function is slow and requires a network connection
res_list <- getParalogs(queryBy)

## (3) Obtain Drug-Target Annotation and Bioassay Data
drug_target_list <- runDrugTarget_Annot_Bioassay(res_list=res_list,
  up_col_id="ID_up_sp", ens_gene_id,config=config )

```

transformTTD

*transformTTD***Description**

Integration with Therapeutic Target Database (TTD). This function downloads a data file from idr-blab.org and returns it as a data frame.

Usage

```
transformTTD(ttdfile=file.path(config$downloadPath,"TTD_IDs.txt"), redownloadTTD=TRUE,config=genConf
```

Arguments

ttdfile	The name of the output file to write the downloaded file to.
redownloadTTD	If true, data file will be downloaded again. If false, we assume the file already exists at ttdfile.
config	General configuration. See genConfig .

Value

Returns a data frame with TTD data in it.

Author(s)

Thomas Girke

Examples

```
ttd=tryCatch(
  transformTTD(),
  error=function(e){
    message("Failed to download TTD file, please try again later")
  }
)
```

ttdTargetAnnot*Query TTD target-drug interactions bidirectionally*

Description

Queries the local TTD SQLite built by [buildTtdDb](#), using the same queryBy = list(molType, idType, ids) convention as the package's ChEMBL-SQL [drugTargetAnnot](#) - including its QueryIDs column: every row of the result is tagged with the original query token it matched, and query IDs that returned no rows still appear as a single row with all other fields NA, so callers can always confirm which of their input IDs were resolved (unlike the standalone R_Py_code/ttdAccess.R reference this was ported from, which returned only the matched rows with no such bookkeeping).

- molType = "protein", idType one of "symbol", "uniprot", "ttd_target_id" -> target -> drug
- molType = "cmp", idType one of "name", "ttd_drug_id" -> drug -> target

Symbol/name lookups are case-insensitive; ID lookups ("uniprot", "ttd_target_id", "ttd_drug_id") are exact-match. Note: TTD's "uniprot" field (and the col used for idType = "uniprot" lookups here) is the UniProt *mnemonic entry name* (e.g. "FGFR1_HUMAN"), NOT the accession number (e.g. "P11362") used by the package's ChEMBL-SQL side ([drugTargetAnnot](#)) - a real cross-source naming inconsistency confirmed to persist even in TTD's own dedicated UniProt file, not a bug; the separate Uniprot_acc column carries a resolved accession where available (see [buildTtdDb](#) for how, and its uniprot_source sibling column for why a given row may still be NA), so query by idType = "uniprot" still needs the mnemonic, but downstream joins to other sources can use Uniprot_acc directly.

Usage

```
ttdTargetAnnot(
  queryBy = list(molType = NULL, idType = NULL, ids = NULL),
  ttdDbPath,
  fields = "core"
)
```

Arguments

queryBy	list with components molType, idType, ids (character vector).
ttdDbPath	character(1) path to the TTD SQLite, e.g. from buildTtdDb .
fields	"core" (default) or "all" - both return every ttd_interactions column, since (unlike the REST-backed sources) there is no larger raw payload to opt into - or a character vector of column names to keep (QueryIDs is always retained). See listDrugTargetFields .

Value

A data.frame with columns QueryIDs, TargetID, GeneName, Uniprot, TargetType, DrugID, DrugName, Smiles, Highest_status, MOA, Uniprot_acc, uniprot_source, molecule_type, PubChem_CID, PubChem_SID, CAS, ChEBI_ID, Indication (with the default fields = "core"), or a subset when fields requests specific columns. Also carries attr(result, "ttd_release"), a data.frame of every source TTD file's own (file, version, date) release stamp (see [buildTtdDb](#)); NULL for a ttdDbPath built before this attribute was added.

See Also

[buildTtdDb](#), [drugTargetAnnot](#), [listDrugTargetFields](#)

Examples

```
dbPath <- buildTtdDb()
ttdTargetAnnot(list(molType = "protein", idType = "symbol",
  ids = c("FGFR1", "IL1B")), dbPath)
ttdTargetAnnot(list(molType = "cmp", idType = "name",
  ids = "Pemigatinib"), dbPath)
```

Index

- * **package**
 - drugTargetInteractions-package, 3
- * **utilities**
 - getSymEnsUp, 56
 - runDrugTarget_Annot_Bioassay, 71
- addCommonIds, 4, 19, 21, 22, 31, 64, 65
- assembleMoaTable, 5, 6, 7, 63, 65, 66, 70
- assembleMoaTargets, 5, 6, 6, 15, 70
- broadRepurposingHubAnnot, 7, 10, 60, 61, 63, 68, 69
- buildBroadRepurposingHubDb, 7, 8, 9, 11, 12, 14, 15, 23, 68, 70
- buildGenomeWideDrugTargetTable, 4, 10, 14, 20–22, 31, 46, 64
- buildGtoPdbDb, 11, 12, 24, 60, 61, 69
- buildHgncSymbolMap, 4, 13, 46, 66, 67
- buildMoaMasterTable, 14
- buildTtdDb, 9, 11, 12, 16, 26, 68, 73, 74
- buildUnichemDb, 17, 28, 36, 57, 69, 70
- cmpIdMapping, 18, 28, 30, 36
- combineDrugTargets, 5, 6, 11, 12, 19, 63–65, 69
- combineGenomeWideDrugTargets, 20, 31
- DataFrame, 65
- downloadBroadRepurposingHub, 9, 10, 22
- downloadChEMBLDb, 16, 23, 37, 38
- downloadGtoPdb, 12, 13, 24
- downloadHgncTable, 24, 45, 46
- downloadTTD, 16, 25
- downloadUniChem, 18, 19, 26, 29
- downloadUnichemTables, 17, 18, 27
- drugTargetAnnot, 18, 28, 35, 37, 40, 48, 53, 73, 74
- drugTargetAnnotTable, 27, 29, 42
- drugTargetBioactivity, 30
- drugTargetColumnMap, 21, 22, 31, 64
- drugTargetInteractions
 - (drugTargetInteractions-package), 3
- drugTargetInteractions-package, 3
- genConfig, 11, 18, 25, 26, 28–30, 32, 42, 67, 71, 72
- getChEMBLBioactivities, 33, 34, 35
- getChEMBLBioassay, 34, 62, 69
- getChEMBLDrugTarget, 10, 34, 35, 35, 37, 38, 40, 48, 49, 53, 54, 63, 68, 69
- getChEMBLMolecule, 23, 37, 37
- getChEMBLTarget, 33, 38
- getDgidbDrugs, 39, 40–42
- getDgidbDrugTarget, 10, 40, 40, 42, 48, 49, 63, 68, 69
- getDgidbTargets, 39, 40, 41
- getDrugTarget, 42
- getEnsemblOrthologs, 43, 45
- getEnsemblParalogs, 43, 44, 44
- getHgncGeneTable, 4, 5, 10–13, 21, 25, 45, 66, 67
- getOpenTargetsDrugIds, 46, 50, 51
- getOpenTargetsDrugs, 47, 48–51
- getOpenTargetsDrugTarget, 10, 48, 48, 51, 63, 68, 69
- getOpenTargetsIds, 46–48, 49
- getOpenTargetsTargets, 46, 48, 49, 50
- getParalogs, 36, 43–45, 51, 58
- getPubchemDrugs, 52, 53–55
- getPubchemDrugTarget, 40, 48, 49, 53, 53, 55, 62, 63, 69
- getPubchemTargets, 53, 54, 54
- getSymEnsUp, 56
- getUnichemMapping, 17, 18, 36, 57, 69
- getUniprotIDs, 36, 51, 52, 58, 59, 60
- getUniprotMapping, 6, 20, 57, 59, 69
- gtoPdbTargetAnnot, 10, 13, 60, 63, 68, 69
- listBioassayFields, 34, 35, 61

`listDrugTargetFields`, 8, 36, 37, 39, 40, 47,
49, 53, 54, 61, 62, 62, 74
`listMoaSources`, 5–7, 15, 63, 70
`mergeDrugTargets`, 22, 64
`moaWide`, 6, 65
`normalizeGeneSymbols`, 13, 66
`processDrugage`, 67
`queryBy`, 28, 30, 42, 51, 52
`queryBy (drugTargetAnnot)`, 28
`queryDrugTargets`, 4–6, 12, 19–21, 31, 64,
68, 70
`queryMoa`, 6, 14, 15, 70
`runDrugTarget_Annot_Bioassay`, 71
`transformTTD`, 72
`ttdTargetAnnot`, 7, 8, 10, 16, 60–63, 68, 69,
73
`UniProt.ws`, 58