Package: igblastr (via r-universe)

June 17, 2026

Title User-friendly R Wrapper to IgBLAST

Description The igblastr package provides functions to conveniently install and use a local IgBLAST installation from within R. The package also includes a set of built-in IgBLAST-compatible germline databases from OGRDB, the AIRR Community's Open Germline Receptor Database, for various organisms. It provides functions to create additional IgBLAST-compatible germline databases using reference sequences retrieved from IMGT/V-QUEST or local FASTA files supplied by the user. When possible, annotations for the V and J alleles in a new germline database are automatically generated and added to the database, so they can be used as replacements for the internal and auxiliary data provided by IgBLAST. IgBLAST is described at <https://pubmed.ncbi.nlm.nih.gov/23671333/>. IgBLAST web interface: <https://www.ncbi.nlm.nih.gov/igblast/>. OGRDB: <https://ogrdb.airr-community.org/>. IMGT/V-QUEST download site: <https://www.imgt.org/download/V-QUEST/>.

biocViews Immunology, Immunogenetics, ImmunoOncology, CellBiology

URL https://bioconductor.org/packages/igblastr

BugReports https://github.com/HyrienLab/igblastr/issues

Version 1.2.11

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.2.0), tibble, Biostrings

Imports methods, utils, stats, tools, R.utils, curl, httr, xml2, rvest, xtable, jsonlite, BiocGenerics, S4Vectors, IRanges, GenomeInfoDb

Suggests GenomicAlignments, parallel, testthat, knitr, rmarkdown, BiocStyle, ggplot2, dplyr, scales, ggseqlogo, airr

VignetteBuilder knitr

Collate utils.R internet-utils.R long_to_wide_airr.R igdata-IO.R translate_codons.R allele2gene.R parse_imgt_fasta_headers.R file-utils.R loci-utils.R db-utils.R ndm_data-IO.R compute_V_gene_delineations.R auxdata-IO.R compute_auxdata.R clean_allele_set.R V_alleles-inspect.R J_alleles-inspect.R LATIN_NAMES.R IMGT-utils.R IMGT-c_region-utils.R precompiled-igblast-utils.R cache-utils.R get_igblast_root.R edit_imgt_file.R igblast_info.R update_live_igdata.R intdata-utils.R auxdata-utils.R install_igblast.R make_blastdbs.R create_region_db.R create_germline_db.R create_c_region_db.R reset_germline_dbs.R list_germline_dbs.R use_germline_db.R reset_c_region_dbs.R list_c_region_dbs.R use_c_region_db.R install_custom_germline_db.R OGRDB-utils.R OGRDB-API.R download_OGRDB_germline_sequences.R download_OGRDB_germline_json.R install_OGRDB_germline_db.R download_IMGT_germline_sequences.R install_IMGT_germline_db.R combine_germline_dbs.R augment_germline_db.R prepare_igblastn_cmdline_args.R read_igblastn_fmt7_output.R read_igblastn_AIRR_output.R igblastn.R igbrowser.R percent_mutation.R summarizeMismatches.R OAS-utils.R zzz.R

Config/pak/sysreqs libicu-dev libxml2-dev libssl-dev zlib1g-dev

Repository https://bioc-release.r-universe.dev

Date/Publication 2026-06-17 01:45:45 UTC

RemoteUrl https://github.com/bioc/igblastr

RemoteRef RELEASE_3_23

RemoteSha 380b0389a41c94e1daee93ac6e19178812a557fb

Contents

allele2gene . . . 3
augment_germline_db . . . 4
auxdata-IO . . . 8
auxdata-utils . . . 9
combine_germline_dbs . . . 12
compute_auxdata . . . 13
compute_V_gene_delineations . . . 20
download_IMGT_germline_sequences . . . 24
download_OGRDB_germline_json . . . 26
download_OGRDB_germline_sequences . . . 30
get_igblast_root . . . 33
igblast_info . . . 34
IGBLAST_ROOT . . . 36
igblastn . . . 37
igblastr_usage_report . . . 44
igbrowser . . . 45
install_custom_germline_db . . . 46

install_igblast . . . 54
install_IMGT_germline_db . . . 55
intdata-utils . . . 59
J_alleles-inspect . . . 62
list_c_region_dbs . . . 68
list_germline_dbs . . . 69
ndm_data-IO . . . 72
OAS-utils . . . 74
parse_imgt_fasta_headers . . . 77
percent_mutation . . . 80
read_igblastn_AIRR_output . . . 82
read_igblastn_fmt7_output . . . 84
reset_c_region_dbs . . . 86
reset_germline_dbs . . . 87
summarizeMismatches . . . 88
translate_codons . . . 88
update_live_igdata . . . 91
use_c_region_db . . . 93
use_germline_dbs . . . 94
V_alleles-inspect . . . 96

Index **98**

`allele2gene` *Go from germline gene allele names to germline gene names*

Description

A simple convenience function to remove the allele suffix from a vector of germline gene allele names.

Usage

```
allele2gene(allele_names)
```

Arguments

`allele_names`	A character vector of germline gene allele names.

Details

Germline gene allele names use specific nomenclature, often including a gene name followed by an asterisk and allele number, like in IGHD3-16*03.

The `allele2gene()` function simply removes the asterisk and anything that follows it to keep the gene name only. Note that the function is *vectorized*, that is, its input can be a *vector* of germline gene allele names, in which case the corresponding germline gene names are returned in a vector of the same length as the input vector. The names on the input vector are propagated, and so are the `NA`s in it.

Value

A character vector *parallel* to the input vector.

See Also

- load_germline_sequences to load the nucleotide sequences of the gene alleles stored in a cached germline db.
- load_c_region_sequences to load the nucleotide sequences of the gene alleles stored in a cached C-region db.
- intdata_utils to access IgBLAST *internal data*.

Examples

```
allele2gene(c("IGHV1-2*04", "IGHV1-2*06", "IGHV5-51*01", "IGHD3-16*03"))

J_alleles <- load_germline_sequences("_OGRDB.human.IGH+IGK+IGL.202605",
                                     region_types="J")
names(J_alleles)
allele2gene(names(J_alleles))

C_alleles <- load_c_region_sequences("_IMGT.human.IGH+IGK+IGL.202605")
names(C_alleles)
allele2gene(names(C_alleles))
```

---

augment_germline_db *Add novel gene alleles to a germline db*

Description

WARNING: Some shortcomings were identified in the design of the augment_germline_db_[VDJ]() functions so they are now deprecated. Please do not use them. If you need to combine germline databases, please use the new combine_germline_dbs() function instead.

If you need to add your own novel gene alleles to an existing germline db, some better alternative will be provided in future versions of the package.

Three functions to add novel V, D, or J gene alleles to a germline db.

Note that these functions can also be used to combine germline databases from two different organisms. See "COMBINE GERMLINE DATABASES FROM TWO ORGANISMS" in the Examples section below for how to do this.

Usage

```
augment_germline_db_V(db_name, novel_alleles,
                      destdir=".", overwrite=FALSE, verbose=FALSE)

augment_germline_db_D(db_name, novel_alleles,
                      destdir=".", overwrite=FALSE, verbose=FALSE)
```

```
augment_germline_db_J(db_name, novel_alleles,
                      destdir=".", overwrite=FALSE, verbose=FALSE)
```

Arguments

db_name	A single string that is the name of the cached germline db that contains the set of gene alleles to augment. Use list_germline_dbs() to get the list of all cached germline dbs. The exact function used (i.e. augment_germline_db_V(), augment_germline_db_D(), or augment_germline_db_J()) determines the set of alleles to augment (i.e. alleles from the V, D, or J region).
novel_alleles	A single string that is the path to a FASTA file (possibly gz-compressed) where the novel alleles are stored. Alternatively, the novel alleles can be supplied as a *named* DNAStringSet object.
destdir	A single string that is the path to the "destination directory", that is, the directory where the augmented V-, D-, or J-region db is to be created. This directory will be created if it doesn't exist already. Note that, by default, the augmented region db will be created in the current directory.
overwrite	If the "destination directory" already contains a V-, D-, or J-region db, should it be overwritten?
verbose	Set to TRUE to have the function display some details about its internal operations.

Value

These functions don't return anything (invisible NULL).

See Also

- combine_germline_dbs to combine two existing germline dbs into a new one.
- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- list_germline_dbs to list the cached germline dbs.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## Not run:
if (!has_igblast()) install_igblast()

query <- system.file(package="igblastr", "extdata",
                     "BCR", "heavy_sequences.fasta")

use_c_region_db("_IMGT.human.IGH+IGK+IGL.202605")

## ------------------------------------------------------------------------
## USE HUMAN GERMLINE DATABASE FROM AIRR
```

```
## ------------------------------------------------------------------------

use_germline_db("_OGRDB.human.IGH+IGK+IGL.202605")

AIRR_df <- igblastn(query)

## ------------------------------------------------------------------------
## ADD NOVEL V ALLELES
## ------------------------------------------------------------------------

## 'fake_human_V_alleles.fasta' contains made-up novel V alleles:
## - 2 novel alleles for gene IGHV1-8: IGHV1-8*fake1, IGHV1-8*fake2
## - 1 novel allele for gene IGHV4-61: IGHV4-61*fake
my_novel_V_alleles <- system.file(package="igblastr", "extdata",
                                  "novel_germline_alleles",
                                  "fake_human_V_alleles.fasta")

## Take a quick look at these novel V alleles:
readDNAStringSet(my_novel_V_alleles)

## Create a new V germline database that combines the V alleles
## from _OGRDB.human.IGH+IGK+IGL.202605 with our novel V alleles:
my_Vdb_path <- file.path(tempdir(), "myVdb")
augment_germline_db_V("_OGRDB.human.IGH+IGK+IGL.202605",
                      my_novel_V_alleles,
                      destdir=my_Vdb_path)

## To use this new augmented V germline database with igblastn(),
## supply its path via the 'germline_db_V' argument:
AIRR_df2 <- igblastn(query, germline_db_V=my_Vdb_path)

## ------------------------------------------------------------------------
## A QUICK COMPARISON BETWEEN 'AIRR_df' AND 'AIRR_df2'
## ------------------------------------------------------------------------

## Index of rows where "v_call" has changed between 'AIRR_df'
## and 'AIRR_df2':
idx <- which(AIRR_df$v_call != AIRR_df2$v_call)
idx  # 2 rows

AIRR_df[idx, c("v_call", "v_cigar", "v_identity")]

AIRR_df2[idx, c("v_call", "v_cigar", "v_identity")]

## Besides these 2 rows, all the other rows are the same:
stopifnot(all.equal(AIRR_df[-idx, ], AIRR_df2[-idx, ]))

## ------------------------------------------------------------------------
## COMBINE GERMLINE DATABASES FROM TWO ORGANISMS
## ------------------------------------------------------------------------

## The augment_germline_db_[VDJ]() functions can be used to combine
## germline databases from two different organisms. This can be useful
```

```
## for example when working with BCR sequences from mice that have been
## engineered to have both mouse and some human immunoglobulin genes.
##
## To create a hybrid human/mouse V germline database, we can either:
##
## (1) Add all (or a subset of) mouse V alleles to all human V alleles.
##     This is done by extracting mouse V germline allele sequences from
##     a cached germline database and using them to augment a cached
##     germline database for human.
##
## (2) Add all (or a subset of) human V alleles to all mouse V alleles.
##     This is done by extracting human V germline allele sequences from
##     a cached germline database and using them to augment a cached
##     germline database for mouse.
##
## Note that:
## - We can choose to subset or not the V germline allele sequences
##   extracted from one V germline database before adding them to the
##   other V germline database.
## - The two approaches above are equivalent if we don't subset, that
##   is, if we combine **all** human V alleles with **all** mouse V
##   alleles.
## - However if our engineered mice only have a small known subset of
##   human immunoglobulin genes (e.g. IGHV1-2), then we might want to
##   create a hybrid human/mouse germline database that only adds the
##   human alleles for genes IGHV1-2 to the mouse V alleles. In this
##   case we need to use (2).

## Let's do (2):

db_name1 <- "_OGRDB.mouse.PWD_PhJ.IGH+IGK+IGL.202410"
db_name2 <- "_OGRDB.human.IGH+IGK+IGL.202605"

## Extract human V germline alleles:
human_V_alleles <- load_germline_sequences(db_name2, "V")

## Subset to keep only alleles for genes IGHV1-2:
idx <- grep("^IGHV[12]", names(human_V_alleles))
human_V12_alleles <- human_V_alleles[idx]

## Create a new V germline database that combines the mouse V
## alleles from 'db_name1' with the alleles in 'human_V12_alleles':
engmouseVdb_path <- file.path(tempdir(), "engmouseVdb")
augment_germline_db_V(db_name1, human_V12_alleles,
                      destdir=engmouseVdb_path)

## End(Not run)

## Then, assuming that 'query' contains BCR sequences from the
## engineered mice:
## Not run:
  use_germline_db(db_name1)
  use_c_region_db("_IMGT.mouse.IGH+IGK+IGL.202605")
```

```
  igblastn(query, germline_db_V=engmouseVdb_path, ...)

## End(Not run)

## Note that, by default, the mouse-only D and J databases that we
## selected above with 'use_germline_db(db_name1)' are being used.
## If we also want to create hybrid D and J databases, we need
## to repeat the above steps for each of them. Then we need to
## specify the paths to the 3 hybrid databases when we call igblastn():
## Not run:
  igblastn(query, germline_db_V=engmouseVdb_path,
                  germline_db_D=engmouseDdb_path,
                  germline_db_J=engmouseJdb_path,
                  ...)

## End(Not run)
```

---

auxdata-IO *Read/write IgBLAST auxiliary data files*

Description

IgBLAST auxiliary data files (a.k.a. .aux files) are tab-delimited text files used by IgBLAST to annotate germline J gene allele sequences.

The **igblastr** package provides low-level functions to read/write this type of file.

Usage

```
read_auxdata(filepath)
write_auxdata(auxdata, file="")
```

Arguments

`filepath`	The path to the IgBLAST auxiliary data file to read.
`auxdata`	A data.frame with 1 row per germline J gene allele sequence and the same columns as the data.frame returned by `read_auxdata()`. See Value section below for the details.
`file`	The path to the IgBLAST auxiliary data file to write.

Value

`read_auxdata()` returns a data.frame with 1 row per germline J gene allele sequence and the following columns:

1. `allele_name`: allele name;
2. `coding_frame_start`: first coding frame start position (0-based);
3. `chain_type`: chain type as a 2-letter code e.g. JK for "J allele from the Kappa locus" (BCR locus) or JG for "J allele from the Gamma locus" (TCR locus);

4. cdr3_end: CDR3 end position (0-based);
5. extra_bps: extra base pairs beyond J coding end.

write_auxdata() doesn't return anything (i.e. invisible NULL).

See Also

- compute_auxdata to annotate a set of germline J gene allele sequences.
- extract_auxdata_from_ogrdb_json to extract the coding frame and CDR3 end information for the J alleles annotated in an AIRR-C JSON file.
- auxdata_utils to access IgBLAST *auxiliary data*.
- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## COMING SOON...
```

---

auxdata-utils *Access IgBLAST auxiliary data*

Description

IgBLAST *auxiliary data* is expected to annotate all the known germline J gene alleles for a given organism. It is provided by NCBI and is typically included in a standard IgBLAST installation.

The *auxiliary data* informs about the coding frame and CDR3/FWR4 boundary on the J allele sequences. Although this information is not strictly needed by IgBLAST, the latter needs it in order to compute some of the AIRR fields like vj_in_frame, productive, cdr3, fwr4, etc...

get_auxdata_path() and load_auxdata() can be used to access the *auxiliary data* included in IgBLAST or in one of the *cached germline dbs* managed by **igblastr**.

We also provide compute_germline_db_auxdata() to compute the *auxiliary data* associated with the J alleles in a cached germline dbs managed by **igblastr**.

Usage

```
## Access auxiliary data:
get_auxdata_path(igblast_organism, which=c("live", "original"))
load_auxdata(igblast_organism, which=c("live", "original"))

## Compute auxiliary data associated with the J alleles in germline db:
compute_germline_db_auxdata(db_name, ...)
```

Arguments

`igblast_organism`

A single string containing the name of an "IgBLAST organism" as returned by `list_igblast_organisms()`.

Alternatively, this can be the name of a cached germline db. Note that this works only for germline dbs that include their own *auxiliary data*. You can use:

```
list_germline_dbs(with.auxdata.only=TRUE)
```

to list them. See `?list_germline_dbs` for more information.

`which`

By default, `get_auxdata_path()` and `load_auxdata()` access the "live IgBLAST data", that is, the IgBLAST data that the user has possibly updated with `update_live_igdata()`. Depending on whether updates were applied or not, the "live IgBLAST data" might differ from the original IgBLAST data.

Set `which` to `"original"` if you want to access the original IgBLAST data instead.

See `?update_live_igdata` for more information about "live" and "original" IgBLAST data.

`db_name`

A single string specifying the name of a cached germline db. Use `list_germline_dbs()` to get the list of all cached germline dbs.

`...`

Extra arguments to be passed to the internal call to `compute_auxdata()`. See `?compute_auxdata` for what arguments are supported.

Details

IgBLAST *auxiliary data* is typically included in a standard IgBLAST installation. It's located in the `optional_file/` directory which is itself a subdirectory of IgBLAST *root directory*.

The data consists of one tabulated file per organism. Each file indicates the germline J gene coding frame start position, the J gene type, and the CDR3 end position for all known germline J allele sequences. See https://ncbi.github.io/igblast/cook/How-to-set-up.html for additional details.

IMPORTANT NOTE: Using IgBLAST with *auxiliary data* that is not compatible with the germline J gene sequences in the germline db can cause it to return improper frame status or CDR3 information (other returned information will still be correct).

Value

`get_auxdata_path()` returns a single string containing the path to the *auxiliary data* included in the IgBLAST installation used by **igblastr**, for the specified organism. Not necessarily suitable to use with `igblastn()` (see WARNING below).

`load_auxdata()` returns the *auxiliary data* in a data.frame with 1 row per supplied J allele sequence and the same columns as the data.frame returned by `read_auxdata()`. See `?read_auxdata` for more information.

`compute_germline_db_auxdata()` returns the computed *auxiliary data* in a data.frame with 1 row per J allele in the germline db and the same columns as the data.frame returned by `compute_auxdata()` or `read_auxdata()`. See `?compute_auxdata` for more information.

See Also

- compute_auxdata to annotate a set of germline J gene allele sequences.
- intdata_utils to access IgBLAST *internal data*.
- update_live_igdata for more information about "live" and "original" IgBLAST data.
- list_igblast_organisms to list the organisms supported by IgBLAST (a.k.a. "IgBLAST organisms").
- list_germline_dbs to list the cached germline dbs.
- https://ncbi.github.io/igblast/cook/How-to-set-up.html for important information about IgBLAST *auxiliary data*.
- The igblastn function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## ------------------------------------------------------------------------
## Access IgBLAST auxiliary data for a given "IgBLAST organism"
## ------------------------------------------------------------------------

## IgBLAST only includes auxiliary data for the following organisms:
list_igblast_organisms()

get_auxdata_path("human")

human_auxdata <- load_auxdata("human")
head(human_auxdata)

## ------------------------------------------------------------------------
## Access the auxiliary data included in a germline db
## ------------------------------------------------------------------------

## List germline dbs with auxiliary data:
list_germline_dbs(with.auxdata.only=TRUE)

## Access the auxiliary data included in germline db
## _OGRDB.rhesus_monkey.IGH+IGK+IGL.202602 (note that this data was
## extracted from the AIRR-C JSON files provided by OGRDB for rhesus
## monkey, see '?extract_auxdata_from_ogrdb_json' for more information):
db_name <- "_OGRDB.rhesus_monkey.IGH+IGK+IGL.202602"

get_auxdata_path(db_name)

rhesus_monkey_auxdata <- load_auxdata(db_name)
head(rhesus_monkey_auxdata)

## ------------------------------------------------------------------------
## Compute the auxiliary data associated with the J alleles in a
```

```
## germline db
## ------------------------------------------------------------------------

## The auxiliary data included in germline db
## _OGRDB.rhesus_monkey.IGH+IGK+IGL.202602 can be computed
## with compute_germline_db_auxdata():
auxdata2 <- compute_germline_db_auxdata(db_name)

## Replace the negative "cdr3_end" value with NA:
replace_idx <- which(auxdata2[ , "cdr3_end"] < 0L)
auxdata2[replace_idx, "cdr3_end"] <- NA

## Sanity check:
stopifnot(identical(auxdata2, rhesus_monkey_auxdata))
```

`combine_germline_dbs` *Combine two existing germline dbs*

Description

Use the `combine_germline_dbs()` function to combine two existing germline databases, typically (but not necessarily) from two different organisms, into a new one. The new germline database gets added to **igblastr**'s persistent cache and can then be used later with `igblastn()`.

Similarly, the `combine_c_region_dbs()` function is provided to combine two existing C-region databases.

Usage

```
combine_germline_dbs(db_name, db_name1, db_name2, suffix1, suffix2,
                     overwrite=FALSE, verbose=FALSE)

combine_c_region_dbs(db_name, db_name1, db_name2, suffix1, suffix2,
                     overwrite=FALSE, verbose=FALSE)
```

Arguments

`db_name`	A single string specifying the name of the combined database. The string *must* start with "comb" and cannot contain whitespace characters.
`db_name1`, `db_name2`	COMING SOON...
`suffix1`, `suffix2`	COMING SOON...
`overwrite`	Set to `TRUE` if a database with the specified name is already installed in **igblastr**'s persistent cache, in which case the existing database will be replaced with the new one.
`verbose`	Set to `TRUE` to have the function display some details about its internal operations.

Value

combine_germline_dbs() and combine_c_region_dbs() both return the name to the newly installed database as an invisible string.

See Also

- install_IMGT_germline_db to install a germline db from IMGT.
- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- list_germline_dbs to list the cached germline dbs.
- use_germline_db to select the cached germline db to use with igblastn().
- list_c_region_dbs to list the cached C-region dbs.
- use_c_region_db to select the cached C-region db to use with igblastn().
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## TO BE EXPANDED SOON...

db_name12 <- "comb.IMGT-202614-2.human+mouse.IGH+IGK+IGL"
db_name1 <- install_IMGT_germline_db("202614-2", "Homo_sapiens",
                                     overwrite=TRUE)
db_name2 <- install_IMGT_germline_db("202614-2", "Mus_musculus",
                                     overwrite=TRUE)
combine_germline_dbs(db_name12, db_name1, db_name2, "_Hs", "_Mm")

rm_germline_db(db_name12)
```

---

compute_auxdata *Compute IgBLAST auxiliary data*

Description

Utility functions to generate *auxiliary data* for a set of germline J gene allele sequences.

See ?load_auxdata for more information about IgBLAST *auxiliary data*.

Usage

```
## The primary utility for computing "auxiliary data":

compute_auxdata(J_alleles, codon_starts=NULL, no.warnings=FALSE)

## Additional utilities that can be used to complete the "auxiliary data"
## returned by compute_auxdata():
```

```
validate_and_complete_auxdata_with_ref(auxdata, ref_auxdata)

solve_cdr3_ends_using_fwr4_aa_comparisons(auxdata, J_alleles,
                                          max.dist=2, standout.by=5)

solve_cdr3_ends_using_fwr4_dna_comparisons(auxdata, J_alleles,
                                           max.dist=4, standout.by=15)

solve_cdr3_ends_using_fwr4_dna_PWM(auxdata, J_alleles,
                                   min.score=0.80, standout.by=0.40)
```

Arguments

J_alleles	A DNAStringSet object containing germline J gene allele sequences.
codon_starts	When supplied, must be a named integer vector *parallel* to J_alleles, that is, it must have the same length and names as J_alleles. For each allele in J_alleles, codon_starts should indicate the start position (1-based) of the first codon in the allele. This should be a number >= 1 and <= 3. NAs are allowed when the coding frame is not known. Note that the "coding frame start" is 0-based so can simply be inferred from the "codon start" by subtracting 1.
no.warnings	Set to TRUE to suppress the warnings that compute_auxdata() otherwise issues when the returned data.frame has NAs in its coding_frame_start column or negative values in its cdr3_end column.
auxdata	A data.frame as returned by compute_auxdata() that contains annotations for the alleles in J_alleles.
ref_auxdata	A data.frame containing "reference auxiliary data". Typically the *auxiliary data* provided by IgBLAST for a given organism.
max.dist	The maximum Hamming distance allowed between the content of the sliding window along the J allele to solve and the set of known FWR4 sequences. By default, solve_cdr3_ends_using_fwr4_aa_comparisons() uses max.dist=2 and solve_cdr3_ends_using_fwr4_dna_comparisons() uses max.dist=4. See Details section below for the details.
standout.by	See Details section below for the details.
min.score	The minimum score allowed for the sliding Position Weight Matrix along the the J allele to solve. See Details section below for the details.

Details

compute_auxdata() is meant to be used first, to generate *auxiliary data* from scratch for a set of germline J gene allele sequences. Note that the data.frame returned by compute_auxdata() can contain NAs in its cdr3_end column. We call these *unsolved CDR3 ends*.

validate_and_complete_auxdata_with_ref() and the solve_cdr3_ends_using_fwr4_*() functions are meant to be used on the *auxiliary data* returned by compute_auxdata() to complete it, that is, to eliminate NAs from it when possible.

The `compute_auxdata()` **function:** `compute_auxdata()` uses a motif-based approach to identify the CDR3/FWR4 junction in the supplied J allele sequences.

For human BCR germline J alleles, the FWR4 region is expected to start with the following amino acid motifs (X represents any amino acid):

- "WGXG" on the heavy chain (IGH locus);
- "FGXG" on the light chain (IGK and IGL loci).

Things might deviate slightly for other organisms, or for TCR germline J alleles. For example, for TCR germline J alleles, the FWR4 region is expected to start with the "FGXG" motif, regardless of the locus.

One complication is that the supplied J allele sequences are nucleotide sequences for which the coding frame might not be known. `compute_auxdata()` uses two distinct strategies to search for canonical motif "WGXG" or "FGXG" in a given J allele sequence, depending on whether the coding frame is known or not:

1. If the coding frame is known (i.e. a vector of codon starts is passed thru the `codon_starts` argument and it reports a non-`NA` value for the allele), then this information is used to translate the coding region of the allele to an amino acid sequence. The function then tries to locate canonical motif "WGXG" or "FGXG" on the translated sequence.
   Note that if the allele is from the IGH locus, then canonical motif "WGXG" is used. For all other loci (IGK, IGL, TRA, TRB, TRG, TRD loci), canonical motif "FGXG" is used.
2. If the coding frame is not known (i.e. the `codon_starts` argument is not set or it's set but the codon start reported for the allele is `NA`), then the search above is performed for all 3 possible coding frames, that is, for all 3 possible codon start values (1, 2, and 3).

Next, the CDR3/FWR4 junction is identified based on the result of the motif search (and the `cdr3_end` set accordingly):

- If the motif search didn't return any match, then the CDR3/FWR4 junction is considered to remain unknown. In this case, the `cdr3_end` is set to `NA`.
- If the motif search returned exactly one match, then the match is considered to mark the start of the FWR4. This means that the CDR3/FWR4 junction is now identified, and so the `cdr3_end` is set accordingly.
- If the motif search returned more than one match, then only the first match is considered to mark the start of the FWR4. All downstream matches are ignored.

Finally, the `coding_frame_start` is obtained as follow. For a given J allele:

1. If a vector of codon starts was passed thru the `codon_starts` argument and it reports a non-`NA` value for the allele, then the `coding_frame_start` (0-based) is set to the reported codon start (1-based) minus 1.
2. If the above cannot be performed, but if the CDR3/FWR4 junction of the allele was successfully identified by the motif search, then the `coding_frame_start` is inferred from the `cdr3_end`.
3. If none of the two operations above can be performed, then the `coding_frame_start` is set to `NA` with a warning.

The `validate_and_complete_auxdata_with_ref()` **function:** `validate_and_complete_auxdata_with_ref()` does two things:

1. Validation: The function validates the *auxiliary data* passed thru its first argument (`auxdata`) against the *auxiliary data* passed thru its second argument (`ref_auxdata`). This is done by

comparing the entries in the former with the entries in the latter for the J alleles that are annotated in both, and by raising an error if they disagree.

2. Completion: The function replaces each NA in auxdata with the corresponding value in ref_auxdata, if the latter is available.

The `solve_cdr3_ends_using_fwr4_*_comparisons()` **functions:** For each unsolved J allele (i.e. for each J allele with an unknown CDR3/FWR4 junction), `solve_cdr3_ends_using_fwr4_aa_comparisons()` tries to identify the CDR3/FWR4 junction as follow:

- The function moves a sliding window of 10 amino acids along the allele sequence.
- For each position p of the window, it computes the Hamming distance D(p) between the sequence in the window and the set of known FWR4 sequences.
- The "best window" is the window that minimizes D(p).
- The position Bp of the "best window" must satisfy the two following criteria in order to be considered the start of the FWR4 of the unsolved J allele:
  1. D(Bp) <= `max.dist`.
  2. D(p) >= `max.dist` + `standout.by` for any p != Bp. In other words, all other windows must have `max.dist` + `standout.by` or more mismatches with any of the known FWR4 sequences.

`solve_cdr3_ends_using_fwr4_dna_comparisons()` works the same way but at the nucleotide level i.e. it uses a sliding window of 30 nucleotides along the allele sequence. Only the positions of the window that are compatible with the coding frame are considered.

The `solve_cdr3_ends_using_fwr4_dna_PWM()` **function:** `solve_cdr3_ends_using_fwr4_dna_PWM()` uses the first 30 nucleotides of each known FWR4 to compute a 4 x 30 Position Weight Matrix (PWM) for the whole set of known FWR4 regions. Then, for each unsolved J allele (i.e. for each J allele with an unknown CDR3/FWR4 junction), it tries to identify the CDR3/FWR4 junction as follow:

- The function moves a sliding window of 30 nucleotides along the allele sequence. Only the positions of the window that are compatible with the coding frame are considered.
- For each position p of the window, it computes the score S(p) of the sequence in the window against the PWM.
- The "best window" is the window that maximizes S(p).
- The position Bp of the "best window" must satisfy the two following criteria in order to be considered the start of the FWR4 of the unsolved J allele:
  1. S(Bp) >= `min.score`.
  2. S(p) <= `min.score` - `standout.by` for any p != Bp.

Note that the `PWM()` function from the **Biostrings** package is used to compute the Position Weight Matrix. See `?Biostrings::PWM` for more information.

Value

All the functions documented in this man page return a data.frame with 1 row per supplied J allele and the same columns as the data.frame returned by `read_auxdata()`. See `?read_auxdata` for more information.

See Also

- print_J_alleles to display a set of J alleles sequences where the sequences are justified w.r.t. their CDR3/FWR4 junction.
- load_auxdata to access IgBLAST *auxiliary data*.
- load_germline_sequences to load the nucleotide sequences of the gene alleles stored in a cached germline db.
- compute_V_gene_delineations to annotate a set of germline V gene allele sequences.
- https://ncbi.github.io/igblast/cook/How-to-set-up.html for important information about IgBLAST *auxiliary data*.
- DNAStringSet and AAStringSet objects in the **Biostrings** package.
- The igblastn function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## ---------------------------------------------------------------------
## 1. BASIC compute_auxdata() EXAMPLE
## ---------------------------------------------------------------------

## Let's load the J allele sequences from _OGRDB.human.IGH+IGK+IGL.202605:
db_name <- "_OGRDB.human.IGH+IGK+IGL.202605"
J_alleles <- load_germline_sequences(db_name, region_types="J")
J_alleles  # DNAStringSet object

computed_auxdata <- compute_auxdata(J_alleles)
head(computed_auxdata)

## The CDR3/FWR4 junction was found for all the alleles:
anyNA(computed_auxdata$cdr3_end)  # FALSE

## This is also reflected by print_J_alleles() which is able to display
## the CDR3/FWR4 junction for all alleles:
print_J_alleles(J_alleles, computed_auxdata, translate=TRUE)

## Note that 'computed_auxdata' is identical to the "auxiliary data"
## included in the germline db (which is provided by AIRR/OGRDB):
stopifnot(identical(computed_auxdata, load_auxdata(db_name)))

## Also note that 'computed_auxdata' is in agreement with the "auxiliary
## data" provided by IgBLAST for human, except for alleles IGHJ6*02 and
## IGHJ6*03 (the 'extra_bps' column contains incorrect values for these
## two alleles, 0's instead of 1's):
human_auxdata0 <- load_auxdata("human", which="original")
bad_alleles <- c("IGHJ6*02", "IGHJ6*03")
subset(human_auxdata0, allele_name %in% bad_alleles)

## We manually correct this:
fixme <- human_auxdata0[ , "allele_name"] %in% bad_alleles
```

```
human_auxdata0[fixme, "extra_bps"] <- 1L  # replace 0L with 1L

## Now the data in 'computed_auxdata' matches exactly the corresponding
## data in 'human_auxdata0':
m <- match(computed_auxdata[ , "allele_name"],
           human_auxdata0[ , "allele_name"])
human_auxdata <- human_auxdata0[m, ]
rownames(human_auxdata) <- NULL
stopifnot(identical(computed_auxdata, human_auxdata))

## ---------------------------------------------------------------------
## 2. ADVANCED compute_auxdata() EXAMPLE
## ---------------------------------------------------------------------

## In this example we're going to annotate some mouse J alleles that we
## retrieve directly from IMGT/V-QUEST.

## We start by downloading the IMGT FASTA files for mouse to a
## temporary directory:
dir.create(mouse_destdir <- tempfile("fasta_dir_"))
download_IMGT_germline_sequences("202614-2",
                                 organism="Mus_musculus",
                                 destdir=mouse_destdir)
## List the downloaded files:
list.files(mouse_destdir)  # 7 FASTA files

## Let's load the J allele sequences for the lambda locus (IGLJ.fasta
## file):
J_alleles <- readDNAStringSet(file.path(mouse_destdir, "IGLJ.fasta"))
J_alleles  # only 8 sequences

## First attempt at annotating the J alleles with compute_auxdata():
auxdata1 <- compute_auxdata(J_alleles)  # emits warning!

## Two alleles could not be annotated:
auxdata1

## We can improve the situation by supplying the codon starts to
## compute_auxdata(). These are provided by IMGT in the IMGT FASTA
## headers:
parsed_headers <- parse_imgt_fasta_headers(names(J_alleles))
parsed_headers[1:3, ]
codon_starts <- as.integer(parsed_headers[ , "codon_start"])

## Set the allele names on the 'codon_starts' vector:
names(codon_starts) <- parsed_headers[ , "allele_name"]
codon_starts

## Second attempt at annotating the J alleles with compute_auxdata():
auxdata2 <- compute_auxdata(J_alleles, codon_starts=codon_starts)
auxdata2

## Supplying the codon starts had the effect of eliminating the NAs
```

```
## that we saw in the "coding_frame_start" and "extra_bps" columns
## of 'auxdata1'.

## However, alleles IGLJ2P*01 and IGLJ3P*01 are still unsolved.
## This is reflected by print_J_alleles() which cannot display
## the CDR3/FWR4 junction for these two alleles:
print_J_alleles(J_alleles, auxdata2, translate=TRUE)

## In the next examples below, we're going to use
## validate_and_complete_auxdata_with_ref() and the
## solve_cdr3_ends_using_fwr4_*_comparisons() functions to try
## and solve these alleles.

## ---------------------------------------------------------------------
## 3. validate_and_complete_auxdata_with_ref()
## ---------------------------------------------------------------------

igblast_auxdata <- load_auxdata("mouse")

## Note that IGLJ2P*01 is not annotated in 'igblast_auxdata'
## but 'IGLJ3P*01' is:
"IGLJ2P*01"
"IGLJ3P*01"

## validate_and_complete_auxdata_with_ref() replaces each NA in 'auxdata2'
## with the corresponding value in 'igblast_auxdata', if the latter is
## available:
auxdata3 <-
    validate_and_complete_auxdata_with_ref(auxdata2, igblast_auxdata)
auxdata3  # IGLJ3P*01 is solved!

## Note that the function also compares the data in 'auxdata2' with the
## data in 'igblast_auxdata' (for the J alleles that are annotated in
## both), and raises an error if they disagree.

## ---------------------------------------------------------------------
## 4. solve_cdr3_ends_using_fwr4_aa_comparisons() and
##    solve_cdr3_ends_using_fwr4_dna_comparisons()
## ---------------------------------------------------------------------

## Finally IGLJ2P*01 can be solved with:
auxdata4a <-
    solve_cdr3_ends_using_fwr4_aa_comparisons(auxdata3, J_alleles)
auxdata4a
stopifnot(!anyNA(auxdata4a$cdr3_end))

## or with:
auxdata4b <-
    solve_cdr3_ends_using_fwr4_dna_comparisons(auxdata3, J_alleles)
stopifnot(identical(auxdata4a, auxdata4b))

print_J_alleles(J_alleles, auxdata4a, translate=TRUE,
                igblast_organism="mouse")
```

```
## ------------------------------------------------------------------------
## 5. solve_cdr3_ends_using_fwr4_dna_PWM()
## ------------------------------------------------------------------------

## In this example we're going to annotate some ferret J alleles that we
## retrieve directly from IMGT/V-QUEST.

dir.create(ferret_destdir <- tempfile("fasta_dir_"))
download_IMGT_germline_sequences("202614-2",
                                 organism="Mustela_putorius_furo",
                                 destdir=ferret_destdir)
## List the downloaded files:
list.files(ferret_destdir)  # 7 FASTA files

## Let's load the J allele sequences for the kappa locus (IGKJ.fasta
## file):
J_alleles <- readDNAStringSet(file.path(ferret_destdir, "IGKJ.fasta"))
J_alleles  # only 5 sequences

## Extract their codon starts:
parsed_headers <- parse_imgt_fasta_headers(names(J_alleles))
codon_starts <- as.integer(parsed_headers[ , "codon_start"])
names(codon_starts) <- parsed_headers[ , "allele_name"]

## First attempt at annotating the J alleles with compute_auxdata():
auxdata1 <- compute_auxdata(J_alleles, codon_starts=codon_starts)
auxdata1

## Allele IGKJ2*01 didn't get solved:
print_J_alleles(J_alleles, auxdata1, translate=TRUE)

## Sill doesn't solve IGKJ2*01:
solve_cdr3_ends_using_fwr4_aa_comparisons(auxdata1, J_alleles)

## However, the following calls are able to solve IGKJ2*01:
auxdata2a <-
    solve_cdr3_ends_using_fwr4_dna_comparisons(auxdata1, J_alleles)
auxdata2a
stopifnot(!anyNA(auxdata2a$cdr3_end))

auxdata2b <- solve_cdr3_ends_using_fwr4_dna_PWM(auxdata1, J_alleles)
auxdata2b
stopifnot(identical(auxdata2a, auxdata2b))

print_J_alleles(J_alleles, auxdata2a, translate=TRUE)
```

---

compute_V_gene_delineations

Compute V gene delineations

Description

A utility function to compute the FWR/CDR boundaries (a.k.a. V gene delineations) for a set of germline V gene allele sequences. Note that the input sequences must be *gapped*, that is, they must contain the so-called "IMGT gaps" whose purpose is to convey the delineation information.

Usage

```
compute_V_gene_delineations(gapped_V_alleles, as.IRangesList=FALSE)
```

Arguments

`gapped_V_alleles`	A DNAStringSet object containing germline V gene allele *gapped* sequences.
`as.IRangesList`	Experts only! Set to `TRUE` to return the delineations in an IRangesList object instead of a data.frame. In this case, the returned IRangesList object will have 1 list element per supplied V allele sequence and will carry the following metadata columns (accessible with `mcols()`): `seq_len`, `coding_frame_start`, `starting_gap`, `all_gaps_in_frame`, `all_gaps_contained`. See Value section below for what these columns contain.

Details

See https://www.imgt.org/IMGTScientificChart/Numbering/IMGTIGVLsuperfamily.html and https://www.imgt.org/IMGTScientificChart/Numbering/IMGT-Kabat_part1.html for how gaps are used in the context of "IMGT unique numbering for V-DOMAIN and V-LIKE-DOMAIN".

Note that the annotation returned by `compute_V_gene_delineations()` can be used by IgBLAST as a substitute to the IgBLAST-provided *internal data*.

See `?load_intdata` for more information about IgBLAST *internal data*.

Value

Returns the computed *internal data* in a data.frame with 1 row per supplied V allele sequence and a set of columns similar to the data.frame returned by `read_ndm_data()`. With the following differences:

- The data.frame returned by `compute_V_gene_delineations()` has no `chain_type` column.
- The data.frame returned by `compute_V_gene_delineations()` has the following additional columns:
  1. `seq_len`: length of the sequence after removal of the gaps;
  2. `starting_gap`: size (in number of nucleotides) of gap block located at the very beginning of the sequence if any (0 if no such block);
  3. `all_gaps_in_frame`: indicates whether the gap blocks align with the underlying coding frame or not;
  4. `all_gaps_contained`: `TRUE` if the gap blocks don't cross the FWR/CDR boundaries, and `FALSE` otherwise.

See Also

- https://www.imgt.org/IMGTScientificChart/Numbering/IMGTIGVLsuperfamily.html for how the "IMGT unique numbering for V-DOMAIN and V-LIKE-DOMAIN" works.
- load_intdata to access IgBLAST *internal data*.
- compute_auxdata to annotate a set of germline J gene allele sequences.
- DNAStringSet objects in the **Biostrings** package.
- The igblastn function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## ---------------------------------------------------------------------
## 1. USING HUMAN GERMLINE V GENE ALLELE SEQUENCES FROM OGRDB
## ---------------------------------------------------------------------
## Visit https://ogrdb.airr-community.org/germline_sets/Homo%20sapiens
## for the list of human germline sets available at OGRDB.

## Download OGRDB germline gene allele sequences for human locus IGK
## to a temporary directory:
germline_set <- c(`IGKappa_VJ`=4)

dir.create(tmpdir <- tempfile())
download_OGRDB_germline_sequences("Homo sapiens", germline_set,
                                  destdir=tmpdir)

## List the downloaded files:
list.files(tmpdir)  # 2 FASTA files (one per IMGT group)

## The V allele sequences are **gapped**:
IGKV_alleles <- readDNAStringSet(file.path(tmpdir, "IGKV.fasta"))
as.character(IGKV_alleles[1:3])

## Compute the V gene delineations for 'IGKV_alleles':
IGKV_gene_delinea <- compute_V_gene_delineations(IGKV_alleles)
head(IGKV_gene_delinea)

## As a sanity check, we're going to compare 'IGKV_gene_delinea' with
## the V gene delineations provided by the corresponding AIRR-C JSON
## file from OGRDB.

download_OGRDB_germline_json("Homo sapiens", germline_set,
                             destdir=tmpdir)

list.files(tmpdir)  # 1 JSON file

json_path <- file.path(tmpdir, "IGKappa_VJ.json")
IGKV_gene_delinea2 <- extract_intdata_from_ogrdb_json(json_path)
head(IGKV_gene_delinea2)
```

```
## Keep only the shared columns:
shared_colnames <- intersect(colnames(IGKV_gene_delinea),
                             colnames(IGKV_gene_delinea2))
IGKV_gene_delinea  <- IGKV_gene_delinea[ , shared_colnames]
IGKV_gene_delinea2 <- IGKV_gene_delinea2[ , shared_colnames]

## Let's use same_alleles_annot(), a small undocumented helper function,
## to verify that 'IGKV_gene_delinea' and 'IGKV_gene_delinea2' contain
## the same delineation data (possibly with their rows in different
## order):
stopifnot(same_alleles_annot(IGKV_gene_delinea, IGKV_gene_delinea2))

## Remove temporary directory:
unlink(tmpdir, recursive=TRUE)

## ---------------------------------------------------------------------
## 2. USING MOUSE GERMLINE V GENE ALLELE SEQUENCES FROM OGRDB
## ---------------------------------------------------------------------
## Visit https://ogrdb.airr-community.org/germline_sets/Mus%20musculus
## for the list of mouse germline sets available at OGRDB.

## Download OGRDB germline V gene allele sequences for locus IGL of
## mouse strain MSM/MsJ to a temporary directory:
germline_set <- c(`MSM/MsJ IGLV`=1)

dir.create(tmpdir <- tempfile())
download_OGRDB_germline_sequences("Mus musculus", germline_set,
                                  destdir=tmpdir)

## List the downloaded files:
list.files(tmpdir)  # 1 FASTA file

## Compute the V gene delineations for 'IGKV_alleles':
IGLV_alleles <- file.path(tmpdir, "IGLV.fasta")
IGLV_gene_delinea <- compute_V_gene_delineations(IGLV_alleles)
IGLV_gene_delinea

## Note that for the first two alleles, the length of the **ungapped**
## sequence (seq_len) is smaller than the FWR3 end position (fwr3_end):
IGLV_gene_delinea[ , c("fwr3_end", "seq_len")]

## In other words, the sequences of alleles IGLV0-37NY*00 and
## IGLV0-COOR*00 end before the end of their FWR3.

## This is reflected in the V gene delineations provided by the
## corresponding AIRR-C JSON file from OGRDB:
download_OGRDB_germline_json("Mus musculus", germline_set,
                             destdir=tmpdir)
list.files(tmpdir)
json_path <- file.path(tmpdir, "IGLV.json")
IGLV_gene_delinea2 <- extract_intdata_from_ogrdb_json(json_path)
IGLV_gene_delinea2
```

```
## As you can see, when a V allele sequence ends before the end of
## its FWR3, the FWR3 end position reported in the AIRR-C JSON file
## is the position of the last nucleotide in the (ungapped) sequence.

## Remove temporary directory:
unlink(tmpdir, recursive=TRUE)
```

`download_IMGT_germline_sequences` *Download germline sequences from IMGT*

Description

The `download_IMGT_germline_sequences()` function downloads germline V/D/J gene allele sequences (nucleotides) from the IMGT/V-QUEST download site for a given IMGT/V-QUEST release and organism. The sequences are downloaded in several FASTA files, one per *IMGT group*.

CONDITIONS OF USE AND LICENSE: The IMGT data is provided to the academic users and NPO's (Not for Profit Organization(s)) under the CC BY-NC-ND 4.0 license. See https://creativecommons.org/licenses/by-nc-nd/4.0/. Any other use of IMGT material, from the private sector, needs a financial arrangement with CNRS.

Usage

```
download_IMGT_germline_sequences(release, organism="Homo sapiens",
                                 tcr.db=FALSE,
                                 destdir=".", overwrite=FALSE,
                                 recache=FALSE, ...)

## Related utilities:
list_IMGT_releases(recache=FALSE)
list_IMGT_organisms(release)
IMGT_is_up()
```

Arguments

`release`	A single string specifying the IMGT/V-QUEST release to download the germline sequences from (or to list the organisms from for `list_IMGT_organisms()`). Use `list_IMGT_releases()` to get the list of all releases.
`organism`	A single string specifying the latin name of the organism for which to download the germline sequences.
`tcr.db`	Should the germline sequences to download be B-cell Receptor (a.k.a. BCR) sequences or T-cell Receptor (a.k.a. TCR) sequences? By default (i.e. when `tcr.db` is omitted or set to `FALSE`), the V/D/J allele sequences for the three BCR loci (IGH, IGK, IGL) will be downloaded. This will result in the 7 following FASTA files being downloaded:

```
IGHV.fasta, IGHD.fasta, IGHJ.fasta,
IGKV.fasta, IGKJ.fasta,
IGLV.fasta, IGLJ.fasta
```

When `tcr.db` is set to `TRUE`, the V/D/J allele sequences for the four TCR loci (TRA, TRB, TRG, TRD) will be downloaded instead. This will result in the 10 following FASTA files being downloaded:

```
TRAV.fasta, TRAJ.fasta,
TRBV.fasta, TRBD.fasta, TRBJ.fasta,
TRGV.fasta, TRGJ.fasta,
TRDV.fasta, TRDD.fasta, TRDJ.fasta
```

Note that:

- The names of the downloaded files follow the <IMGT group>.fasta convention, where <IMGT group> is one of the 7 *IMGT groups* with the "IG" prefix or one of the 10 *IMGT groups* with the "TR" prefix.
- IMGT doesn't always provide germline sequences for all *IMGT groups*. This means that, depending on the IMGT/V-QUEST release and organism, some FASTA files can be missing.

`destdir` A single string that is the path to the directory where the FASTA files containing the germline sequences should be downloaded.

`overwrite` `download_IMGT_germline_sequences()` won't replace existing files, unless `overwrite` is set to `TRUE`.

`recache` `download_IMGT_germline_sequences()` and `list_IMGT_releases()` both use a caching mechanism for the data that they download from IMGT/V-QUEST. In the case of `download_IMGT_germline_sequences()`, the downloaded files are cached on disk so the caching is persistent across R sessions. In the case of `list_IMGT_releases()`, the caching is in memory so does not persist across sessions.

Set `recache` to `TRUE` to force a new download (and recaching) of the data.

`...` Extra arguments to be passed to the internal call to `download.file()`. See ?`download.file` in the **utils** package for more information.

Value

`download_IMGT_germline_sequences()` returns the names of the downloaded FASTA files in an invisible character vector.

`list_IMGT_releases()` returns the list of IMGT/V-QUEST releases in a character vector. The releases are sorted from newest to oldest (latest release is first).

`list_IMGT_organisms()` returns the list of organisms included in the specified IMGT/V-QUEST release in a character vector.

`IMGT_is_up()` returns `TRUE` or `FALSE`, indicating whether the IMGT website at https://www.imgt.org is up and running or down.

See Also

- install_IMGT_germline_db to install a germline db from IMGT.
- install_custom_germline_db to install a germline db from user-supplied gene allele sequences.
- download_OGRDB_germline_sequences to download germline sequences from OGRDB.
- The IMGT website: https://www.imgt.org/.
- The IMGT/V-QUEST download site: https://www.imgt.org/download/V-QUEST/.

Examples

```
if (IMGT_is_up()) {
  ## As of April 10, 2026, the latest IMGT/V-QUEST release is 202614-2:
  list_IMGT_releases()

  list_IMGT_organisms("202614-2")

  ## Download Mouse BCR germline gene allele sequences from IMGT/V-QUEST
  ## 202614-2 to a temporary directory:
  dir.create(destdir <- tempfile("fasta_dir_"))
  download_IMGT_germline_sequences("202614-2", organism="Mus musculus",
                                   destdir=destdir)

  ## List the downloaded files:
  list.files(destdir)  # 7 FASTA files

  ## Remove temporary directory:
  unlink(destdir, recursive=TRUE)

  ## Download Mouse TCR germline gene allele sequences from IMGT/V-QUEST
  ## 202614-2 to a temporary directory:
  dir.create(destdir <- tempfile("fasta_dir_"))
  download_IMGT_germline_sequences("202614-2", organism="Mus musculus",
                                   tcr.db=TRUE, destdir=destdir)

  ## List the downloaded files:
  list.files(destdir)  # 10 FASTA files

  ## Remove temporary directory:
  unlink(destdir, recursive=TRUE)
}
```

---

download_OGRDB_germline_json

Download AIRR-C JSON files from OGRDB

Description

The download_OGRDB_germline_json() function downloads the AIRR-C JSON files assciated with the specified germline sets from OGRDB.

The extract_intdata_from_ogrdb_json() function extracts the V gene delineations for the V alleles annotated in an AIRR-C JSON file. Note that it's a simple reimplementation in R of the Python script makeogrannote.py included in IgBLAST.

The extract_auxdata_from_ogrdb_json() function extracts the coding frame and CDR3 end information for the J alleles annotated in an AIRR-C JSON file.

If you use OGRDB in your research, please cite:

```
The current landscape of adaptive immune receptor genomic and
repertoire data: OGRDB and VDJbase
Lees, Peres, Klein, Amos et al., Nucleic Acids Research, November 2025.
https://doi.org/10.1093/nar/gkaf1094
```

Usage

```
download_OGRDB_germline_json(organism, germline_sets,
                             source_set=FALSE,
                             destdir=".", overwrite=FALSE,
                             recache=FALSE, ...)

extract_intdata_from_ogrdb_json(json_path, extra_fields=NULL)
extract_auxdata_from_ogrdb_json(json_path, extra_fields=NULL)
```

Arguments

organism	A single string specifying the latin name of the organism for which to download the AIRR-C JSON file, e.g. "Homo sapiens". See https://ogrdb.airr-community.org/germline_sets for the organisms that are currently available at OGRDB.
germline_sets	A named integer vector. The names on the vector must be the names of the germline sets to download and the values in the vector must be their versions. Go to https://ogrdb.airr-community.org/germline_sets, select a Species, and click on "Show Germline Sets" to see the list of germline sets available for that species and their versions.
source_set	Whether to download the AIRR-C JSON files assciated with the *Source Sets* instead of the *Reference Sets*. Only applies when organism is Homo sapiens.
destdir	A single string that is the path to the directory where the AIRR-C JSON files should be downloaded.
overwrite	download_OGRDB_germline_json() won't replace existing files, unless overwrite is set to TRUE.
recache	download_OGRDB_germline_json() uses a caching mechanism for the data that gets downloaded from OGRDB. The data is cached on disk so the caching is persistent across R sessions. Set recache to TRUE to force a new download (and recaching) of the data.

...	Extra arguments to be passed to the internal call to download.file(). See ?download.file in the **utils** package for more information.
json_path	The path to an AIRR-C JSON file as one obtained with download_OGRDB_germline_json().
extra_fields	NULL (the default) or a character vector of valid AIRR-C "AlleleDescription fields" to extract from the AIRR-C JSON file and to include in the returned data.frame as additional columns. See https://docs.airr-community.org/en/latest/datarep/germline.html#alleledescription-fields for the list of valid "AlleleDescription fields".

Value

download_OGRDB_germline_json() returns the list of AIRR-C JSON files that it produced in an invisible character vector that carries the names of the corresponding germline sets.

extract_intdata_from_ogrdb_json() returns the V gene delineations in a data.frame with 1 row per V allele in the AIRR-C JSON file and with the same columns as the data.frame returned by load_intdata(). See ?load_intdata for the details. If extra_fields was specified, the returned data.frame will have 1 additional column per field in extra_fields.

extract_auxdata_from_ogrdb_json() returns the coding frame and CDR3 end information in a data.frame with 1 row per J allele in the AIRR-C JSON file and with the same columns as the data.frame returned by load_auxdata(). See ?load_auxdata for the details. If extra_fields was specified, the returned data.frame will have 1 additional column per field in extra_fields.

See Also

- download_OGRDB_germline_sequences to download germline sequences from OGRDB.
- write_ndm_data to write the data.frame returned by extract_intdata_from_ogrdb_json() in *ndm* format.
- write_auxdata to write the data.frame returned by extract_auxdata_from_ogrdb_json() in *.aux* format.
- compute_V_gene_delineations to annotate a set of germline V gene allele sequences.
- compute_auxdata to annotate a set of germline J gene allele sequences.
- intdata_utils to access IgBLAST *internal data*.
- auxdata_utils to access IgBLAST *auxiliary data*.
- The OGRDB website: https://ogrdb.airr-community.org/.

Examples

```
## ------------------------------------------------------------------------
## download_OGRDB_germline_json()
## ------------------------------------------------------------------------
## We're going to use download_OGRDB_germline_json() to download the
## AIRR-C JSON files for mouse strain PWD/PhJ from OGRDB.
## Visit https://ogrdb.airr-community.org/germline_sets/Mus%20musculus
## for the list of germline sets available for mouse.

## We need the five following germline sets in order to cover all the
```

```
## BCR loci/regions for mouse strain PWD/PhJ:
germline_sets <- c(`PWD/PhJ IGH`=2, `PWD/PhJ IGKV`=1, `PWD/PhJ IGLV`=1,
                   `IGKJ (all strains)`=1, `IGLJ (all strains)`=1)

## Let's download them to a temporary directory:
dir.create(temp_json_dir <- tempfile("json_dir_"))
download_OGRDB_germline_json("Mus musculus", germline_sets,
                             destdir=temp_json_dir)

## List the downloaded files:
list.files(temp_json_dir)  # 5 AIRR-C JSON files (one per germline set)

## ---------------------------------------------------------------------
## extract_intdata_from_ogrdb_json()
## ---------------------------------------------------------------------
## Let's use extract_intdata_from_ogrdb_json() to extract the V gene
## delineations for the V alleles annotated in the AIRR-C JSON files.
## Note that germline sets "IGKJ (all strains)" and "IGLJ (all strains)"
## don't annotate any V alleles so we exclude files IGKJ.json and
## IGLJ.json:

json_path <- file.path(temp_json_dir, "IGH.json")
IGHV_intdata <- extract_intdata_from_ogrdb_json(json_path)
head(IGHV_intdata)

json_path <- file.path(temp_json_dir, "IGKV.json")
IGKV_intdata <- extract_intdata_from_ogrdb_json(json_path)
head(IGKV_intdata)

json_path <- file.path(temp_json_dir, "IGLV.json")
IGLV_intdata <- extract_intdata_from_ogrdb_json(json_path)
IGLV_intdata

## Combine the 3 data.frames:
intdata <- rbind(IGHV_intdata, IGKV_intdata, IGLV_intdata)
dim(intdata)

## 'intdata' should be the same as the "internal data" included in
## germline db _OGRDB.mouse.PWD_PhJ.IGH+IGK+IGL.202410:
intdata0 <- load_intdata("_OGRDB.mouse.PWD_PhJ.IGH+IGK+IGL.202410")

## Let's use same_alleles_annot(), a small undocumented helper function,
## to verify that 'intdata' and 'intdata0' contain the same data (possibly
## with their rows in different order):
stopifnot(same_alleles_annot(intdata, intdata0))

## Note that calling extract_intdata_from_ogrdb_json() on an AIRR-C JSON
## file with no V alleles returns a 0-row data.frame with a warning:
json_path <- file.path(temp_json_dir, "IGKJ.json")
IGKJ_intdata <- extract_intdata_from_ogrdb_json(json_path)  # warning!
stopifnot(nrow(IGKJ_intdata) == 0)
json_path <- file.path(temp_json_dir, "IGLJ.json")
IGLJ_intdata <- extract_intdata_from_ogrdb_json(json_path)  # warning!
```

```
stopifnot(nrow(IGLJ_intdata) == 0)

## ---------------------------------------------------------------------
## extract_auxdata_from_ogrdb_json()
## ---------------------------------------------------------------------
## Let's use extract_auxdata_from_ogrdb_json() to extract the coding
## frame and CDR3 end information for the J alleles annotated in the
## AIRR-C JSON files. Note that germline sets "PWD/PhJ IGKV" and
## "PWD/PhJ IGLV" don't annotate any J alleles so we exclude files
## IGKV.json and IGLV.json:

json_path <- file.path(temp_json_dir, "IGH.json")
IGHJ_auxdata <- extract_auxdata_from_ogrdb_json(json_path)
IGHJ_auxdata

json_path <- file.path(temp_json_dir, "IGKJ.json")
IGKJ_auxdata <- extract_auxdata_from_ogrdb_json(json_path)
IGKJ_auxdata

json_path <- file.path(temp_json_dir, "IGLJ.json")
IGLJ_auxdata <- extract_auxdata_from_ogrdb_json(json_path)
IGLJ_auxdata

## Combine the 3 data.frames:
auxdata <- rbind(IGHJ_auxdata, IGKJ_auxdata, IGLJ_auxdata)
dim(auxdata)

## 'auxdata' should be the same as the "auxiliary data" included
## in germline db _OGRDB.mouse.PWD_PhJ.IGH+IGK+IGL.202410:
auxdata0 <- load_auxdata("_OGRDB.mouse.PWD_PhJ.IGH+IGK+IGL.202410")
stopifnot(same_alleles_annot(auxdata, auxdata0))

## Note that calling extract_auxdata_from_ogrdb_json() on an AIRR-C JSON
## file with no J alleles returns a 0-row data.frame with a warning:
json_path <- file.path(temp_json_dir, "IGKV.json")
IGKV_auxdata <- extract_auxdata_from_ogrdb_json(json_path)  # warning!
stopifnot(nrow(IGKV_auxdata) == 0)
json_path <- file.path(temp_json_dir, "IGLV.json")
IGLV_auxdata <- extract_auxdata_from_ogrdb_json(json_path)  # warning!
stopifnot(nrow(IGLV_auxdata) == 0)

## Remove temporary directory:
unlink(temp_json_dir, recursive=TRUE)
```

---

download_OGRDB_germline_sequences

Download germline sequences from OGRDB

Description

The `download_OGRDB_germline_sequences()` function downloads germline V/D/J gene allele sequences (nucleotides) from OGRDB for the specified germline sets. The sequences are downloaded in several FASTA files, one per *IMGT group*.

If you use OGRDB in your research, please cite:

```
The current landscape of adaptive immune receptor genomic and
repertoire data: OGRDB and VDJbase
Lees, Peres, Klein, Amos et al., Nucleic Acids Research, November 2025.
https://doi.org/10.1093/nar/gkaf1094
```

Usage

```
download_OGRDB_germline_sequences(organism, germline_sets,
                                  gapped=TRUE, source_set=FALSE,
                                  destdir=".", overwrite=FALSE,
                                  recache=FALSE, ...)
```

Arguments

`organism` A single string specifying the latin name of the organism for which to download the germline sequences, e.g. `"Homo sapiens"`. See https://ogrdb.airr-community.org/germline_sets for the organisms that are currently available at OGRDB.

`germline_sets` A named integer vector. The names on the vector must be the names of the germline sets to download and the values in the vector must be their versions. Go to https://ogrdb.airr-community.org/germline_sets, select a Species, and click on "Show Germline Sets" to see the list of germline sets available for that species and their versions.

Note that each OGRDB germline set corresponds to one or more *IMGT groups*, where an *IMGT group* is a combination of locus and region type. For example:

- Germline set `IGH_VDJ` corresponds to *IMGT groups* IGHV, IGHD, and IGHJ.
- Germline set `IGLambda_VJ` corresponds to *IMGT groups* IGLV and IGLJ.
- Germline set `PWD/PhJ IGH` corresponds to *IMGT groups* IGHV, IGHD, and IGHJ.
- Germline set `PWD/PhJ IGKV` corresponds to *IMGT group* IGKV.

For reference, the 10 *IMGT groups* for B-cell Receptors are:

```
IGHV, IGHD, IGHJ, IGHC,
IGKV, IGKJ, IGKC,
IGLV, IGLJ, IGLC
```

IMPORTANT: The germline sets in your `germline_sets` vector must correspond to distinct *IMGT groups*.

Here is an example of valid `germline_sets` vector for Mus musculus:

```
germline_sets <- c(`PWD/PhJ IGH`=2,
`PWD/PhJ IGKV`=1, `IGKJ (all strains)`=1,
`PWD/PhJ IGLV`=1, `IGLJ (all strains)`=1)
```

gapped — By default `download_OGRDB_germline_sequences()` will download the *gapped* V allele sequences. Set `gapped` to `FALSE` to download the *ungapped* V allele sequences instead.

source_set — Whether to download the germline sequences from the *Source Sets* instead of the *Reference Sets*. Only applies when `organism` is Homo sapiens.

Note that the AIRR-community/OGRDB maintainers recommend to use the *Reference Sets* for AIRR-seq analysis. See for example https://ogrdb.airr-community.org/germline_set/75. See also https://wordpress.vdjbase.org/index.php/ogrdb/explanation-of-germline-set-formats/ for a brief explanation of the different germline set formats available from OGRDB.

destdir — A single string that is the path to the directory where the FASTA files containing the germline sequences should be downloaded.

overwrite — `download_OGRDB_germline_sequences()` won't replace existing files, unless `overwrite` is set to `TRUE`.

recache — `download_OGRDB_germline_sequences()` uses a caching mechanism for the data that gets downloaded from OGRDB. The data is cached on disk so the caching is persistent across R sessions.

Set `recache` to `TRUE` to force a new download (and recaching) of the data.

... — Extra arguments to be passed to the internal call to `download.file()`. See ?`download.file` in the **utils** package for more information.

Value

The function returns the list of FASTA files that it produced in an invisible character vector that carries the names of the corresponding germline sets.

See Also

- `install_custom_germline_db` to install a germline db from user-supplied gene allele sequences.
- `download_IMGT_germline_sequences` to download germline sequences from IMGT.
- `download_OGRDB_germline_json` to download AIRR-C JSON files from OGRDB.
- The OGRDB website: https://ogrdb.airr-community.org/.
- https://wordpress.vdjbase.org/index.php/ogrdb/explanation-of-germline-set-formats/ for a brief explanation of the different germline set formats available from OGRDB.

Examples

```
## ------------------------------------------------------------------------
## DOWNLOAD RHESUS MONKEY GERMLINE SEQUENCES FROM OGRDB
## ------------------------------------------------------------------------
## Visit https://ogrdb.airr-community.org/germline_sets/Macaca%20mulatta
## for the list of germline sets available for rhesus monkey.

## Download germline gene allele sequences for all BCR loci/regions
## for rhesus monkey to a temporary directory:
germline_sets <- c(IGH_VDJ=2, IGK_VJ=2, IGL_VJ=2)
```

```
dir.create(destdir <- tempfile("fasta_dir_"))
download_OGRDB_germline_sequences("Macaca mulatta", germline_sets,
                                  destdir=destdir)

## List the downloaded files:
list.files(destdir)  # 7 FASTA files (one per IMGT group)

## Remove temporary directory:
unlink(destdir, recursive=TRUE)

## ------------------------------------------------------------------
## DOWNLOAD MOUSE GERMLINE SEQUENCES FROM OGRDB
## ------------------------------------------------------------------
## Visit https://ogrdb.airr-community.org/germline_sets/Mus%20musculus
## for the list of germline sets available for mouse.

## Download germline gene allele sequences for all BCR loci/regions
## for mouse strain PWD/PhJ to a temporary directory:
germline_sets <- c(`PWD/PhJ IGH`=2,
                   `PWD/PhJ IGKV`=1, `IGKJ (all strains)`=1,
                   `PWD/PhJ IGLV`=1, `IGLJ (all strains)`=1)

dir.create(destdir <- tempfile("fasta_dir_"))
download_OGRDB_germline_sequences("Mus musculus", germline_sets,
                                  destdir=destdir)

## List the downloaded files:
list.files(destdir)  # 7 FASTA files (one per IMGT group)

## Remove temporary directory:
unlink(destdir, recursive=TRUE)
```

`get_igblast_root` *Control IgBLAST installation to use*

Description

Get (or set) the IgBLAST installation used (or to be used) by the **igblastr** package.

Usage

```
get_igblast_root()
set_igblast_root(version_or_path)
```

Arguments

`version_or_path`
: A single string that is either a version number (e.g. `"1.22.0"`) or the path to an IgBLAST installation.

Details

`set_igblast_root` can be used to set or change the path to the IgBLAST installation to use. This can be an *internal* or *external* installation.

In the former case, `version_or_path` should be the version of an existing *internal* installation. The setting will be persistent.

In the latter case, it should be the full path (absolute or relative) to the *root directory* of a valid *external* installation. Note that the setting won't be persistent i.e. it won't be remembered across R sessions. See `?IGBLAST_ROOT` for how to set the *external* IgBLAST installation to use in **igblastr** in a persistent manner.

Value

`get_igblast_root()` returns a single string containing the path to the *root directory* of the IgBLAST installation used by **igblastr**.

`set_igblast_root()` returns a single string containing the path to the *root directory* of the newly selected IgBLAST installation. The string is returned invisibly.

See Also

- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- `install_igblast` to perform an *internal* IgBLAST installation.
- `igblast_info` to collect basic information about the IgBLAST installation used by the **igblastr** package.
- IGBLAST_ROOT to set the *external* IgBLAST installation to be used by the **igblastr** package in a persistent manner.
- IgBLAST is described at `https://pubmed.ncbi.nlm.nih.gov/23671333/`.

Examples

```
if (!has_igblast()) install_igblast()

get_igblast_root()
```

---

igblast_info *Check IgBLAST used by igblastr*

Description

Collect basic information about the IgBLAST installation used by the **igblastr** package, or about any IgBLAST installation on the user machine.

Usage

```
igblast_info(igblast_root=get_igblast_root())

igblast_build(igblast_root=get_igblast_root())
igblastn_version(igblast_root=get_igblast_root(), raw.version=FALSE)
makeblastdb_version(igblast_root=get_igblast_root(), raw.version=FALSE)
list_igblast_organisms(igblast_root=get_igblast_root())

has_igblast()
```

Arguments

igblast_root	A single string that is the path to an IgBLAST installation. By default igblast_root is set to get_igblast_root(), which is the path to the IgBLAST installation used by the **igblastr** package. See ?get_igblast_root for more information. Note that the supplied string must contain the path to the *root directory* of an IgBLAST installation, that is, to a directory with a bin subdirectory in it that has the igblastn, igblastp, and makeblastdb *standalone executables* (on Windows these executables are files named igblastn.exe, igblastp.exe, and makeblastdb.exe, respectively).

raw.version	By default (i.e. when raw.version is omitted or set to FALSE), igblastn_version() and makeblastdb_version() return the version string of the igblastn and makeblastdb *standalone executables* included in IgBLAST. This string is extracted from the output produced by system commands:
	`igblastn -version`
	and
	`makeblastdb -version`
	When raw.version is set to TRUE, igblastn_version() and makeblastdb_version() return the *full ouput* produced by the above commands.

Value

igblast_info() returns a named list containing basic information about the IgBLAST installation.

igblast_build() returns a single string containing IgBLAST build information.

By default, igblastn_version() returns a single string containing the version of the igblastn *standalone executable* included in IgBLAST.

By default, makeblastdb_version() returns a single string containing the version of the makeblastdb *standalone executable* included in IgBLAST.

list_igblast_organisms() returns a character vector that lists the organisms for which IgBLAST provides *internal data*. Note that this is obtained by simply listing the content of the internal_data directory located in the IgBLAST installation.

has_igblast() returns TRUE or FALSE.

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- install_igblast to perform an *internal* IgBLAST installation.
- get_igblast_root to get (or set) the IgBLAST installation used (or to be used) by the **igblastr** package.
- IGBLAST_ROOT to set the *external* IgBLAST installation to be used by the **igblastr** package in a persistent manner.
- intdata_utils to access IgBLAST *internal data*.
- auxdata_utils to access IgBLAST *auxiliary data*.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

igblast_info()

list_igblast_organisms()
```

IGBLAST_ROOT *Use an external IgBLAST installation*

Description

Select the *external* IgBLAST installation to use in **igblastr** in a persistent manner.

Details

The **igblastr** package can use 2 types of IgBLAST installation:

1. Internal (a.k.a. igblastr-managed): refers to an installation obtained with install_igblast().
2. External: refers to an installation that is not managed by the **igblastr** package. This is usually an installation that was manually performed by you or a system administrator on your machine. It can be a system-wide installation or a per-user installation.

To use an *external* installation of IgBLAST in **igblastr**, set environment variable IGBLAST_ROOT to the path of the installation. Note that this must be the path to the *root directory* of the IgBLAST installation, that is, to a directory with a bin subdirectory in it that has the igblastn, igblastp, and makeblastdb *standalone executables* (on Windows these executables are files named igblastn.exe, igblastp.exe, and makeblastdb.exe, respectively).

This can be done within your current R session with Sys.setenv(IGBLAST_ROOT="path/to/igblast_root") for testing. However, this won't be remembered across R sessions.

To set IGBLAST_ROOT in a persistent manner, define it outside R. The exact way to do this is OS-dependent e.g. on Linux and Mac you can define it in your user's .profile by adding the following line to it:

```
export IGBLAST_ROOT="path/to/igblast_root"
```

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- install_igblast to perform an *internal* IgBLAST installation.
- igblast_info to collect basic information about the IgBLAST installation used by the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

---

igblastn *BLAST for BCR/Ig and TCR sequences*

Description

The igblastn() function is a wrapper to the igblastn *standalone executable* included in IgBLAST. This is the main function in the **igblastr** package.

Usage

```
igblastn(query, outfmt="AIRR",
         germline_db_V="auto", germline_db_V_seqidlist=NULL,
         germline_db_D="auto", germline_db_D_seqidlist=NULL,
         germline_db_J="auto", germline_db_J_seqidlist=NULL,
         organism="auto", c_region_db="auto",
         custom_internal_data="auto", auxiliary_data="auto",
         domain_system=c("imgt", "kabat"), ig_seqtype="auto",
         ...,
         out=NULL, parse.out=TRUE,
         show.in.browser=FALSE, show.command.only=FALSE)

igblastn_help(long.help=FALSE, show.in.browser=FALSE)
```

Arguments

query — A character vector containing the paths to the input files (FASTA), or a *named* DNAStringSet object.

If a character vector, then query must be of length >= 1 and each vector element must be the path to a FASTA file (possibly gz-compressed). In the context of IgBLAST, the DNA sequences in the FASTA files are referred to as *the query sequences*, and the sequence names found in the description lines of the FASTA records are referred to as *the query sequence ids*.

Note that the query sequences are typically (but not always) stored in a single file, in which case query will be a single string. If more than one FASTA file is specified via query, then igblastn() will concatenate all the files together and pass the resulting file to the igblastn *standalone executable*.

If query is a DNAStringSet object, then it must have names on it. These will be considered the query sequence ids.

outfmt
: One of "AIRR", 3, 4, 7, or 19. "AIRR" is the default and is an alias for 19. outfmt can also be a string describing a customized format 7 e.g. "7 qseqid sseqid pident nident length score".

  See ?list_outfmt7_specifiers for more information about customizing format 7.

germline_db_V
: "auto" (the default), or the path to a V-region db.

  Note that, by default (i.e. when germline_db_V is omitted or set to "auto"), igblastn() uses the V-region db that belongs to the cached germline db that is currently selected.

  See ?use_germline_db for how to select the cached germline db to use with igblastn().

germline_db_D
: Same as germline_db_V but for the D-region db.

germline_db_J
: Same as germline_db_V but for the J-region db.

germline_db_V_seqidlist, germline_db_D_seqidlist, germline_db_J_seqidlist
: Restrict search of germline database to list of gene alleles. A list of gene alleles can be specified either as a character vector of gene allele identifiers (e.g. IGHV3-23*01, IGHV3-23*04, etc...) or as the path to a file containing the identifiers (one identifier per line). In the latter case, a file object must be passed to the germline_db_V_seqidlist, germline_db_D_seqidlist, or germline_db_J_seqidlist argument. The file object will typically be constructed with something like file("path/to/some/file").

organism
: "auto" (the default), or the organism associated with the query sequences. Supported organisms include human, mouse, rat, rabbit and rhesus_monkey. Use list_igblast_organisms() to obtain this list programmatically.

  Note that, by default (i.e. when organism is omitted or set to "auto"), igblastn() infers the organism from the name of the cached germline db that is currently selected.

  See ?use_germline_db for how to select the cached germline db to use with igblastn().

c_region_db
: "auto" (the default), NULL, or the path to a C-region db.

  Note that, by default (i.e. when c_region_db is omitted or set to "auto"), igblastn() uses the cached C-region db that is currently selected.

  See ?use_c_region_db for how to select the cached C-region db to use with igblastn().

custom_internal_data
: "auto" (the default), or a data.frame containing custom FWR/CDR annotation, or the path to a file containing custom FWR/CDR annotation, or NULL.

  By default (i.e. when custom_internal_data and germline_db_V are both omitted or set to "auto"), igblastn() will use:

  - the *internal data* included in the cached germline db that is currently selected, if it has any;
  - the organism-specific IgBLAST *internal data* from NCBI if the cached germline db that is currently selected does not include its own *internal data*.

Use `list_germline_dbs(with.intdata.only=TRUE)` to list the cached germline dbs that include their own *internal data*.

See `?use_germline_db` for how to select the cached germline db to use with `igblastn()`.

Note that when `custom_internal_data` is set to `NULL`, `igblastn()` always uses the organism-specific IgBLAST *internal data* from NCBI.

`auxiliary_data` `"auto"` (the default), or a data.frame containing auxiliary data, or the path to a file containing auxiliary data (.aux file), or `NULL`.

By default (i.e. when `auxiliary_data` and `germline_db_J` are both omitted or set to `"auto"`), `igblastn()` will use:

- the *auxiliary data* included in the cached germline db that is currently selected, if it has any;
- the organism-specific IgBLAST *auxiliary data* from NCBI if the cached germline db that is currently selected does not include its own *auxiliary data*.

Use `list_germline_dbs(with.auxdata.only=TRUE)` to list the cached germline dbs that include their own *auxiliary data*.

See `?use_germline_db` for how to select the cached germline db to use with `igblastn()`.

IMPORTANT NOTE: When `auxiliary_data` is set to `NULL`, then no *auxiliary data* is used. In this case, `igblastn()` can emit a significant number of the following warning:

```
Warning: Auxilary data file could not be found
```

and various columns of the returned AIRR-formatted tibble (e.g. columns `vj_in_frame`, `productive`, `cdr3`, `fwr4`, and others) will be filled with `NA`s.

`domain_system` Set to `"imgt"` or `"kabat"`.

`ig_seqtype` Set to `"Ig"` or `"TCR"` depending on whether the query sequences are BCR/Ig or TCR sequences.

Note that, by default (i.e. when `ig_seqtype` is omitted or set to `"auto"`), the value of `ig_seqtype` is inferred from the germline loci that appear in the name of the cached germline db that is currently selected (this name can be obtained with `use_germline_db()`). If these are BCR/Ig germline loci (i.e. IGH, IGK, IGL), then the inferred value will be `"Ig"`. If they are TCR germline loci (TRA, TRB, TRG, TRD), then it will be `"TCR"`.

See `?use_germline_db` for how to select the cached germline db to use with `igblastn()`.

`...` Extra arguments to be passed to the `igblastn` *standalone executable*. The list of valid arguments can be displayed with `igblastn_help()`.

Note that the argument/value pairs must be passed to the `igblastn()` function in the usual R fashion. For example, what would be passed as `-num_alignments 1 -num_threads 8` when invoking the `igblastn` *standalone executable* in a terminal should be passed as `num_alignments_V=1, num_threads=8` when calling the `igblastn()` function:

```
igblastn(query, num_alignments_V=1, num_threads=8)
```

For options that don't require a value (e.g. `-extend_align5end`, `-extend_align3end`, `-ungapped`, etc...), pass the empty string (or a white string) to the argument. For example:

```
igblastn(query, extend_align5end="", extend_align3end="")
```

`out` `NULL` (the default), or the path to the file where the `igblastn` *standalone executable* should write its output.

Note that, by default (i.e. when `out` is omitted or set to `NULL`), `igblastn()` instructs the `igblastn` *standalone executable* to write its output to a temporary file.

`parse.out` Whether `igblastn()` should parse the plain-text output produced by the `igblastn` *standalone executable* or not, before returning it to the user. `TRUE` by default.

If set to `FALSE`, then `igblastn()` returns the output as-is in a character vector, with one line per element in the vector. Note that `igblastn()` sets the `"igblastn_raw_output"` class attribute on this character vector, which allows compact display of the vector (this is achieved via a dedicated `print()` method defined in the **igblastr** package). The class attribute can be dropped with `unclass()`.

`show.in.browser`

For `igblastn()`: Whether the output of the `igblastn` *standalone executable* should also be displayed in a browser or not (in addition to being returned by the `igblastn()` function call). `FALSE` by default.

For `igblastn_help()`: Whether the help printed by the `igblastn` *standalone executable* (when invoked with the -h or -help argument) should be displayed in a browser or not. `FALSE` by default.

`show.command.only`

`TRUE` or `FALSE`. If set to `TRUE`, `igblastn()` won't invoke the `igblastn` *standalone executable* and instead will display the full command that shows how it would have invoked it. Note that the command is also returned in an invisible character vector. `FALSE` by default.

`long.help` `TRUE` or `FALSE`. If set to `FALSE` (the default), the `igblastn` *standalone executable* is invoked with the -h argument. Otherwise, it's invoked with the -help argument.

Value

`igblastn()` captures the output produced by the `igblastn` *standalone executable* and returns it as:

- A tibble with 1 row per query sequence if `outfmt` is `"AIRR"` or `19` and `parse.out` is `TRUE`.
- A nested list with two top-level components (`records` and `footer`) if `outfmt` is `7` (or a customized format 7) and `parse.out` is `TRUE`. See `?read_igblastn_fmt7_output` for more information.
- A character vector with class attribute `"igblastn_raw_output"` on it in all other cases, that is, if `parse.out` is `FALSE` or `outfmt` is `3` or `4`. See the `parse.out` argument above for more information.

Note

By default, the NCBI BLAST+ and IgBLAST programs will "call home" to report usage when they run on a computer with internet access. See https://www.ncbi.nlm.nih.gov/books/NBK569851/ for the details. This can induce a significant slowdown in some situations e.g. when the `igblastn` *standalone executable* is called in a loop on a small set of query sequences at each iteration.

For this reason, the "call home" feature is disabled in **igblastr** by default, unless environment variable BLAST_USAGE_REPORT is set to true. See `?igblastr_usage_report` for more information.

See Also

- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.
- `install_igblast` to perform an *internal* IgBLAST installation.
- `igblast_info` to collect basic information about the IgBLAST installation used by the **igblastr** package.
- `install_IMGT_germline_db` to install a germline db from IMGT.
- `install_custom_germline_db` to install a germline db from user-supplied gene allele sequences.
- `use_germline_db` to select the cached germline db to use with `igblastn()`.
- `use_c_region_db` to select the cached C-region db to use with `igblastn()`.
- `igbrowser` to display the annotated sequences returned by `igblastn()` in a browser.
- `percent_mutation` to compute the percent mutation in the V, D, J segments of a set of BCR or TCR sequences.
- `list_outfmt7_specifiers` for how to customize output format 7.
- `list_igblast_organisms` to list the organisms supported by IgBLAST.
- igblastr_usage_report to turn "Usage Reporting" on or off.
- DNAStringSet objects implemented in the **Biostrings** package.
- tibble objects implemented in the **tibble** package.

Examples

```
if (!has_igblast()) install_igblast()

igblast_info()

## ------------------------------------------------------------------------
## Access query sequences and select germline and C-region dbs to use
## ------------------------------------------------------------------------

## Files 'heavy_sequences.fasta' and 'light_sequences.fasta' included
## in igblastr contain 250 paired heavy- and light- chain sequences (125
## sequences in each file) downloaded from OAS (the Observed Antibody
## Space database):
filenames <- paste0(c("heavy", "light"), "_sequences.fasta")
query <- system.file(package="igblastr", "extdata", "BCR", filenames)
```

```
## Install Human germline db from IMGT:
db_name <- install_IMGT_germline_db("202614-2", "Homo_sapiens",
                                    overwrite=TRUE)

## Select germline db to use with igblastn():
use_germline_db(db_name)

## Select C-region db to use with igblastn():
use_c_region_db("_IMGT.human.IGH+IGK+IGL.202605")

## ---------------------------------------------------------------------
## Call igblastn()
## ---------------------------------------------------------------------

## We don't specify the 'outfmt' argument so output will be in AIRR
## format:
AIRR_df <- igblastn(query)
AIRR_df

## The result is a tibble with one row per query sequence:
class(AIRR_df)
dim(AIRR_df)

## You can call igbrowser() on 'AIRR_df' to visualize the annotated
## sequences in a browser. See '?igbrowser'.

## Note that this tibble can easily be converted to an ordinary data.frame
## with 'as.data.frame()', or to a DataFrame with 'as(., "DataFrame")':
as(AIRR_df, "DataFrame")

## To call igblastn() on a subset of the FASTA file, load the file as a
## DNAStringSet object with Biostrings::readDNAStringSet(), then subset
## the object, and finally pass the result of the subsetting operation
## to igblastn():
query_21_30 <- readDNAStringSet(query)[21:30]
query_21_30  # a DNAStringSet object with 10 sequences
igblastn(query_21_30)

## ---------------------------------------------------------------------
## Parallel computing
## ---------------------------------------------------------------------

## The igblastn standalone executable included in IgBLAST supports
## multithreading via command line argument -num_threads. To use it
## in igblastn(), set argument 'num_threads' to the desired value:
AIRR_df2 <- igblastn(query, num_threads=4)
stopifnot(identical(AIRR_df, AIRR_df2))

## Unfortunately, in our experience, using 'num_threads' on Linux
## doesn't achieve any significant speedup.

## Alternatively, one can parallelize execution at the R level using
```

```
## standard tools from the parallel or BiocParallel package. In this
## case it's the responsibility of the user to split the input in
## smaller batches and combine the results obtained for each batch:
if (.Platform$OS.type != "windows") {
  library(parallel)
  ## Split 'query' in 10 batches of 25 sequences each:
  batches <- split(readDNAStringSet(query), rep(1:10, each=25))
  AIRR_df3 <- mclapply(seq_along(batches),
                       function(i) igblastn(batches[[i]]),
                       mc.cores=4)
  ## Combine the results:
  AIRR_df3 <- do.call(rbind, AIRR_df3)
  stopifnot(identical(AIRR_df, AIRR_df3))
}

## Note that the actual speedup will depend on many factors like
## hardware, number and size of the batches, number of cores used,
## etc...

## ---------------------------------------------------------------------
## TCR analysis
## ---------------------------------------------------------------------

## NCBI IgBLAST can also be used for TCR sequence analysis, and so does
## igblastn().

## File 'SRR11341217.fasta.gz' included in igblastr contains 10,875 human
## beta chain TCR transcripts running from 5' of reverse transcription
## reaction to beginning of constant region:
filename <- "SRR11341217.fasta.gz"
query <- system.file(package="igblastr", "extdata", "TCR", filename)

## For this example, we're only keeping the first 100 sequences:
query <- head(readDNAStringSet(query), n=100)

## Install Human TCR germline db from IMGT:
db_name <- install_IMGT_germline_db("202614-2", "Homo_sapiens",
                                    tcr.db=TRUE, overwrite=TRUE)

## Select germline db to use with igblastn():
use_germline_db(db_name)

## Select C-region db to use with igblastn():
use_c_region_db("_IMGT.human.TRA+TRB+TRG+TRD.202605")

## Call igblastn(). Note that the 'ig_seqtype' argument will be
## automatically set to "TCR" (see documentation of the 'ig_seqtype'
## argument above in this man page for more information):
AIRR_df <- igblastn(query)

## ---------------------------------------------------------------------
## More examples
## ---------------------------------------------------------------------
```

```
## See '?read_igblastn_fmt7_output' for more examples.
```

`igblastr_usage_report` *Turn "Usage Reporting" on or off*

Description

By default, the NCBI BLAST+ and IgBLAST programs will "call home" to report usage when they run on a computer connected to the internet. See https://www.ncbi.nlm.nih.gov/books/NBK569851/ for the details. This can induce a significant slowdown in some situations e.g. when the `igblastn` *standalone executable* is called in a loop on a small set of query sequences at each iteration.

For this reason, the "call home" feature is disabled in **igblastr** by default, unless environment variable BLAST_USAGE_REPORT is set to true.

More precisely, the "call home" feature is controlled by global option `igblastr_usage_report` in **igblastr**. On package startup, this option is set to `TRUE` if environment variable BLAST_USAGE_REPORT is set to true. Otherwise (i.e. if BLAST_USAGE_REPORT is not set, or is set to false or gibberish) it is set to `FALSE`.

Details

The user can change the value of global option `igblastr_usage_report` any time with:

```
options(igblastr_usage_report=TRUE)
```

or with:

```
options(igblastr_usage_report=FALSE)
```

To get the value of this option, use:

```
getOption("igblastr_usage_report")
```

Note that changing the value of a global option interactively with `options(...)` won't be remembered across R sessions. For a persistent change, you can either:

- Put the `options(...)` command in your .Rprofile file. See `?Rprofile` for more information. Note that this is the standard way of setting global options persistently.
- In the particular case of global option `igblastr_usage_report` an alternative is to define environment variable BLAST_USAGE_REPORT outside R. The exact way to do this is OS-dependent e.g. on Linux and Mac you can define it in your user's `.profile` by adding the following line to it:

```
export BLAST_USAGE_REPORT=true
```

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## Check current status of usage reporting:
getOption("igblastr_usage_report")

## Turn on usage reporting:
options(igblastr_usage_report=TRUE)

## Turn off usage reporting:
options(igblastr_usage_report=FALSE)
```

---

igbrowser *Display annotated BCR sequences in a browser*

Description

Use igbrowser() to display the annotated BCR sequences returned by igblastn() in a browser. For each sequence, the V, D, and J segments are shown as well as the FWR1-4 and CDR1-3 regions. Additionally, the C segments are shown if the C-region information is available.

Usage

```
igbrowser(AIRR_df, show.full.sequence=FALSE, dna.coloring=TRUE,
          Vcolor="#FFDDD2", Dcolor="#CFC", Jcolor="#CEF", Ccolor="#EEC",
          FWRcolor="#C9D", CDRcolor="#EE4")
```

Arguments

AIRR_df	The AIRR-formatted data.frame or tibble returned by igblastn(). Note that calling igbrowser() on a data.frame with thousands of rows is quite resource-intensive (it can even crash your browser!), so in this case we recommend subsetting the data.frame before passing it to igbrowser() to keep the number of rows under 2000.
show.full.sequence	By default, the part of the BCR sequences upstream of the V region is not shown. Set show.full.sequence to TRUE to show it.
dna.coloring	Whether the nucleotides in the BCR sequences (sequence column in AIRR_df) should be colored or not.
Vcolor, Dcolor, Jcolor, Ccolor	The background colors of the V, D, J, and C segments of the BCR sequences. Note that the C segments are shown only if AIRR_df contains C-region information.

`FWRcolor`, `CDRcolor`
: The background colors of the Framework Regions (FWR1-4) and Complementarity-Determining Regions (CDR1-3), respectively.

Value

0 or the error code returned by the internal call to `browseURL()`, invisibly.

See Also

- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.
- tibble objects implemented in the **tibble** package.

Examples

```
if (!has_igblast()) install_igblast()

query <- system.file(package="igblastr", "extdata",
                     "BCR", "heavy_sequences.fasta")
use_germline_db("_OGRDB.human.IGH+IGK+IGL.202605")

## ------------------------------------------------------------------------
## With C regions
## ------------------------------------------------------------------------

use_c_region_db("_IMGT.human.IGH+IGK+IGL.202605")
AIRR_df <- igblastn(query)
igbrowser(AIRR_df)

## By default, the part of the sequences upstream of the V region is
## not shown. Use 'show.full.sequence=TRUE' to show the full sequences:
igbrowser(AIRR_df, show.full.sequence=TRUE)

## ------------------------------------------------------------------------
## No C regions
## ------------------------------------------------------------------------

use_c_region_db("")
AIRR_df2 <- igblastn(query)
igbrowser(AIRR_df2)
```

`install_custom_germline_db` *Install a germline db from user-supplied gene allele sequences*

Description

The `install_custom_germline_db()` function creates and installs an IgBLAST-compatible germline database from a set of user-supplied FASTA files containing germline V/D/J gene allele sequences (nucleotides).

The new database gets installed in **igblastr**'s persistent cache and can be used later with `igblastn()`.

Usage

```
install_custom_germline_db(db_name, fasta_dir, tcr.db=FALSE, loci="auto",
                           imgt.fasta.headers=FALSE,
                           gapped=FALSE, intdata=NULL,
                           auxdata=NULL, ref_auxdata=NULL,
                           disambiguate.allele.names=FALSE,
                           overwrite=FALSE, verbose=FALSE)
```

Arguments

`db_name` A single string specifying the name of the germline db to install. The string *must* start with "cus" and cannot contain whitespace characters.

`fasta_dir` The path to the directory containing the FASTA files to import in the database. Note that the files to import *must* be named:

- IGH[VDJ].fasta, IGK[VJ].fasta, and IGL[VJ].fasta for a BCR germline loci database;
- TRA[VJ].fasta, TRB[VDJ].fasta, TRG[VJ].fasta, and TRD[VDJ].fasta for a TCR germline loci database.

See the `tcr.db` argument below for more details.

Note that the FASTA header lines are expected to contain the allele names:

- either on their own if the header lines have no `|`,
- or in the 2nd field if the header lines have fields separated with `|`.

`tcr.db` Should the database to install be populated with allele sequences from the BCR (B-cell Receptor) or TCR (T-cell Receptor) germline loci?

The BCR germline loci are: IGH, IGK, IGL.

The TCR germline loci are: TRA, TRB, TRG, TRD.

By default, the V/D/J allele sequences from the BCR germline loci will be used. More precisely, the 7 following FASTA files will be expected to be present in `fasta_dir`:

```
IGHV.fasta, IGHD.fasta, IGHJ.fasta,
IGKV.fasta, IGKJ.fasta,
IGLV.fasta, IGLJ.fasta
```

The files that are effectively present will be imported in the database, with a warning if some of them are missing.

If `tcr.db` is set to `TRUE`, then the V/D/J allele sequences from the TCR germline loci will be used instead. More precisely, the 10 following FASTA files will be expected to be present in `fasta_dir`:

```
TRAV.fasta, TRAJ.fasta,
TRBV.fasta, TRBD.fasta, TRBJ.fasta,
TRGV.fasta, TRGJ.fasta,
TRDV.fasta, TRDD.fasta, TRDJ.fasta
```

The files that are effectively present will be imported in the database, with a warning if some of them are missing.

`loci` By default, the database to install will be populated with the allele sequences from all the BCR or TCR loci. However, if you want to restrict the database to specific loci, you can use the `loci` argument to specify these loci. The subset of loci can be specified as a character vector with one element per locus (e.g. `"IGH"` or `c("TRA", "TRB")`), or as a +-separated list in a single string (e.g. `"TRA+TRB"`).

`imgt.fasta.headers`

COMING SOON...

`gapped` Set to `TRUE` if the supplied V allele sequences are *gapped*, in which case the gaps will be removed when the sequences get imported in the database.

`intdata` `NULL` (the default), or `"auto"`, or a data.frame containing custom *internal data*, or the path to a file containing custom *internal data*.

By default, `install_custom_germline_db()` doesn't include any *internal data* in the new database.

To include *internal data* in the new db, you can set `intdata` to `"auto"`, in which case `install_custom_germline_db()` will compute and add the *internal data* to the database. Note that this is only supported if the V allele sequences in the supplied FASTA files have gaps and `gapped` is set to `TRUE`.

Alternatively, you can supply your own custom *internal data* as a data.frame or by passing the path to a file containing *internal data*.

Note that the presence of *internal data* in the new db will be indicated in the `intdata` column of the data.frame returned by `list_germline_dbs()`.

If a cached germline db includes its own *internal data*, then `igblastn()` will use it (as *custom internal data*) instead of the *internal data* provided by IgBLAST. See documentation of the `custom_internal_data` argument in `?igblastn` for more information.

Also see `?intdata_utils` for more information about IgBLAST *internal data*.

`auxdata` `NULL` (the default), or `"auto"`, or a data.frame containing custom *auxiliary data*, or the path to a file containing custom *auxiliary data*.

By default, `install_custom_germline_db()` doesn't include any *auxiliary data* in the new database.

To include *auxiliary data* in the new db, you can set `auxdata` to `"auto"`, in which case `install_custom_germline_db()` will try to compute and add the *auxiliary data* to the database.

Alternatively, you can supply your own custom *auxiliary data* as a data.frame or by passing the path to a file containing *auxiliary data*.

Note that the presence of *auxiliary data* in the new db will be indicated in the `auxdata` column of the data.frame returned by `list_germline_dbs()`.

If a cached germline db includes its own *auxiliary data*, then `igblastn()` will use it instead of the *auxiliary data* provided by IgBLAST. See documentation of the `auxiliary_data` argument in `?igblastn` for more information.

`ref_auxdata` Use only if `auxdata` is set to `"auto"` (`ref_auxdata` is ignored otherwise).

`ref_auxdata` can be `NULL` (the default) or a data.frame containing *auxiliary data*. If the latter, then:

- `ref_auxdata` would have typically be obtained with `load_auxdata(organism)` where `organism` is one of the organisms returned by `list_igblast_organisms()` (e.g. `"mouse"`).
- `install_custom_germline_db()` will use the `ref_auxdata` data.frame internally to verify that the computed *auxiliary data* is *concordant* (i.e. in agreement) with `ref_auxdata`. An error will be raised that shows the disagreements if there are any.
- Furthermore, `install_custom_germline_db()` will also use the `ref_auxdata` data.frame internally to complete the computed *auxiliary data*. The completion process consists in eliminating missing values in the computed *auxiliary data* where possible. This is done by replacing each missing value with the corresponding value from the *auxiliary data* provided by IgBLAST, if the latter is available.

`disambiguate.allele.names`

Note that *repeated* alleles (i.e. alleles with identical ungapped DNA sequences *and* names) are dropped. More precisely, only the first allele in each group of *repeated* alleles is kept.

If, after dropping the *repeated* alleles, the names of the remaining alleles are not unique, then an error will be raised, unless `disambiguate.allele.names` is set to `TRUE`, in which case the *ambiguous* allele names will be disambiguated by adding a suffix to them.

Finally, note that we only look at *repeated* alleles or *ambiguous* allele names within the same region type e.g. within the union of IGHV.fasta, IGKV.fasta, and IGLV.fasta (V regions), or within the union of TRBD.fasta and TRDD.fasta (D regions). But we never look at *repeated* alleles or *ambiguous* allele names across IGHV.fasta and IGHJ.fasta, or across TRBV.fasta and TRBD.fasta (these are very very unlikely to occur anyways).

`overwrite` Set to `TRUE` if a germline db with the specified name is already installed in **igblastr**'s persistent cache, in which case the existing db will be replaced with the new one.

`verbose` Set to `TRUE` to have the function display some details about its internal operations.

Value

`install_custom_germline_db()` returns the name to the newly installed germline db as an invisible string.

Note

`install_custom_germline_db()` creates the local database by performing the instructions provided at https://ncbi.github.io/igblast/cook/How-to-set-up.html.

See Also

- install_IMGT_germline_db to install a germline db from IMGT.
- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- list_germline_dbs to list the cached germline dbs.
- use_germline_db to select the cached germline db to use with igblastn().
- load_germline_sequences to load the nucleotide sequences of the gene alleles stored in a cached germline db.
- load_intdata to access IgBLAST *internal data*.
- download_OGRDB_germline_sequences to download germline sequences from OGRDB.
- download_IMGT_germline_sequences to download germline sequences from IMGT.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## ------------------------------------------------------------------------
## A. WITH UNGAPPED GERMLINE V GENE ALLELE SEQUENCES
## ------------------------------------------------------------------------

## In this admittedly artificial example, we're going to create and
## install a germline db from the gene allele sequences in built-in
## germline db _OGRDB.human.IGH+IGK+IGL.202605. More precisely, we're
## going to:
##   1. Dump built-in germline db _OGRDB.human.IGH+IGK+IGL.202605
##      in 7 FASTA files.
##   2. Use these FASTA files to create and install a new germline db.
##   3. Check that the new db contains the same sequences as the
##      original db.

## --- 1. Dump _OGRDB.human.IGH+IGK+IGL.202605 in 7 FASTA files ---

dump_germline_db <- function(db_name, destdir) {
  for (region_type in c("V", "D", "J")) {
    alleles <- load_germline_sequences(db_name, region_types=region_type)
    for (locus in c("IGH", "IGK", "IGL")) {
      loc_alleles <- alleles[grep(locus, names(alleles))]
      if (length(loc_alleles) == 0L)
        next
      fasta_file <- paste0(locus, region_type, ".fasta")
      writeXStringSet(loc_alleles, file.path(destdir, fasta_file))
    }
  }
}

db_name0 <- "_OGRDB.human.IGH+IGK+IGL.202605"
dir.create(exA_fasta_dir <- tempfile())
```

```
dump_germline_db(db_name0, exA_fasta_dir)

list.files(exA_fasta_dir)  # 7 FASTA files

## --- 2. Use FASTA files to install new germline db ---

exA_db_name <- "cus.exA.human.IGH+IGK+IGL"
install_custom_germline_db(exA_db_name, exA_fasta_dir)
list_germline_dbs()

## --- 3. Check that new and original dbs have the same sequences ---

stopifnot(identical(
    as.character(load_germline_sequences(db_name0)),
    as.character(load_germline_sequences(exA_db_name))
))

## ------------------------------------------------------------------------
## B. WITH GAPPED GERMLINE V GENE ALLELE SEQUENCES
## ------------------------------------------------------------------------

## In this example, we're going to download human germline gene allele
## **gapped** sequences from AIRR-community/OGRDB, and use them to create
## and install a new germline db. More precisely, we're going to:
##   1. Use download_OGRDB_germline_sequences() to download the
##      gapped sequences for the following "Reference Sets" from
##      AIRR-community/OGRDB:
##        - IGH_VDJ version 10
##        - IGKappa_VJ version 5
##        - IGLambda_VJ version 4
##      Each "Reference Set" corresponds to a BCR germline locus. For
##      each set, download_OGRDB_germline_sequences() will download the
##      FASTA file provided by AIRR-community/OGRDB that contains the
##      gapped allele sequences for that locus, and split the file into
##      one FASTA file per region type.
##   2. Use these FASTA files to create and install a new germline db.
##   3. Check that the new db contains the same sequences
##      as built-in germline db _OGRDB.human.IGH+IGK+IGL.202605.

## --- 1. Download human gapped sequences from AIRR-community/OGRDB ---

dir.create(exB_fasta_dir <- tempfile())
germline_sets <- c(IGH_VDJ=10, IGKappa_VJ=5, IGLambda_VJ=4)

download_OGRDB_germline_sequences("Homo sapiens", germline_sets,
                                  destdir=exB_fasta_dir)

list.files(exB_fasta_dir)  # 7 FASTA files

## Note that the V allele sequences have gaps:
IGKV_alleles <- readDNAStringSet(file.path(exB_fasta_dir, "IGKV.fasta"))
as.character(IGKV_alleles[1:3])
```

```
## --- 2. Use FASTA files to install new germline db ---

exB_db_name <- "cus.exB.human.IGH+IGK+IGL"
install_custom_germline_db(exB_db_name, exB_fasta_dir, gapped=TRUE)
list_germline_dbs()

## --- 3. Check that the new db contains the same sequences ---
## ---          as _OGRDB.human.IGH+IGK+IGL.202605          ---

## The Reference Sets we used to populate the new db (IGH_VDJ version 10,
## IGKappa_VJ version 5, IGLambda_VJ version 4) are the same that were
## used to populate built-in germline db _OGRDB.human.IGH+IGK+IGL.202605.
## Let's check that the two dbs contain the same sequences:
stopifnot(identical(
    as.character(load_germline_sequences(db_name0)),
    as.character(load_germline_sequences(exB_db_name))
))

## ------------------------------------------------------------------------
## C. CREATE AND INSTALL GERMLINE DB WITH COMPUTED "INTERNAL DATA"
## ------------------------------------------------------------------------

## The gaps that germline gene allele sequence providers like IMGT and
## AIRR-community/OGRDB inject in the V allele sequences reflect their
## FWR/CDR boundaries. When the 'intdata' argument is set to "auto",
## install_custom_germline_db() takes advantage of this to compute and
## add the "internal data" associated with the V alleles to the new db:
exC_db_name <- "cus.exC.human.IGH+IGK+IGL"
install_custom_germline_db(exC_db_name, exB_fasta_dir,
                           gapped=TRUE, intdata="auto")
list_germline_dbs()  # the 'intdata' col indicates the presence
                     # of "internal data" in the new db

## Note that the "internal data" in a germline db can be loaded with
## load_intdata() (see '?load_intdata' for more information):
intdataC <- load_intdata(exC_db_name)
head(intdataC)

## Check that the "internal data" in the new db is the same as in
## _OGRDB.human.IGH+IGK+IGL.202605:
stopifnot(identical(intdataC, load_intdata(db_name0)))

## ------------------------------------------------------------------------
## D. CREATE AND INSTALL GERMLINE DB WITH COMPUTED "INTERNAL DATA"
##    AND "AUXILIARY DATA"
## ------------------------------------------------------------------------

## In this example, we're going to download mouse germline gene allele
## **gapped** sequences from IMGT/V-QUEST, and use them to create
## and install a new germline db with "internal data" and auxiliary data".
## More precisely, we're going to:
##   1. Use download_IMGT_germline_sequences() to download IMGT FASTA
##      files containing the gapped sequences for mouse.
```

```
##   2. Use these FASTA files to create and install a new germline db.
##   3. Install IMGT-202614-2.Mus_musculus.IGH+IGK+IGL and compare it
##      with the germline db obtained in 2.
##   4. Leverage the IgBLAST-provided "auxiliary data" to complete
##      the "auxiliary data" computed by install_IMGT_germline_db().

if (IMGT_is_up()) {

  ## --- 1. Download mouse gapped sequences from IMGT ---

  list_IMGT_releases()
  dir.create(exD_fasta_dir <- tempfile())
  download_IMGT_germline_sequences("202614-2", organism="Mus musculus",
                                   destdir=exD_fasta_dir)

  list.files(exD_fasta_dir)  # 7 FASTA files

  ## Note that the V allele sequences have gaps:
  IGKV_alleles <- readDNAStringSet(file.path(exD_fasta_dir, "IGKV.fasta"))
  as.character(IGKV_alleles[1:3])

  ## --- 2. Use FASTA files to install new germline db ---

  exD_db_name <- "cus.exD.mouse.IGH+IGK+IGL"
  install_custom_germline_db(exD_db_name, exD_fasta_dir,
                             imgt.fasta.headers=TRUE,
                             gapped=TRUE, intdata="auto", auxdata="auto")
  list_germline_dbs()  # the 'intdata' and 'auxdata' cols indicate the
                       # presence of "internal data" and "auxiliary data"
                       # in the new db

  ## --- 3. Install IMGT-202614-2.Mus_musculus.IGH+IGK+IGL and ---
  ## ---       compare it with cus.exD.mouse.IGH+IGK+IGL       ---

  db_name <- install_IMGT_germline_db("202614-2", "Mus musculus",
                                      overwrite=TRUE)

  ## Let's check that the two dbs contain the same sequences:
  stopifnot(identical(
    as.character(load_germline_sequences(db_name)),
    as.character(load_germline_sequences(exD_db_name))
  ))

  ## Let's check that the two dbs contain the same "internal data":
  intdataD <- load_intdata(exD_db_name)
  stopifnot(identical(intdataD, load_intdata(db_name)))

  ## However the two dbs do NOT contain the same "auxiliary data":
  auxdataD <- load_auxdata(exD_db_name)
  identical(auxdataD, load_auxdata(db_name))  # FALSE!

  ## Note the 4 NAs in the "cdr3_end" column:
  auxdataD
```

```
  ## where 'load_auxdata(db_name)' only has one:
  load_auxdata(db_name)

  ## --- 4. Leverage the IgBLAST-provided "auxiliary data" ---
  ## ---     to complete the "auxiliary data" computed     ---
  ## ---           by install_IMGT_germline_db()           ---

  ## When calling install_IMGT_germline_db(), we can pass some "reference
  ## auxiliary data" for mouse to the 'ref_auxdata' argument. The
  ## reference data we will use is the "auxiliary data" provided by IgBLAST
  ## for mouse. install_custom_germline_db() will then use this data
  ## internally to verify that the computed "auxiliary data" is concordant
  ## with the reference data and to complete it. See documentation of the
  ## 'ref_auxdata' argument above in this man page for more information.

  igblast_mouse_auxdata <- load_auxdata("mouse")
  install_custom_germline_db(exD_db_name, exD_fasta_dir,
                             imgt.fasta.headers=TRUE,
                             gapped=TRUE, intdata="auto", auxdata="auto",
                             ref_auxdata=igblast_mouse_auxdata,
                             overwrite=TRUE)

  ## Now cus.exD.mouse.IGH+IGK+IGL and
  ## IMGT-202614-2.Mus_musculus.IGH+IGK+IGL contain the
  ## same "auxiliary data":
  auxdataD <- load_auxdata(exD_db_name)
  stopifnot(identical(auxdataD, load_auxdata(db_name)))

  ## So the two dbs are now exactly the same: same allele names and
  ## sequences, same "internal data", and same "auxiliary data"!
}

## ------------------------------------------------------------------------
## Remove the 4 custom dbs
## ------------------------------------------------------------------------

rm_germline_db(exA_db_name)
rm_germline_db(exB_db_name)
rm_germline_db(exC_db_name)
rm_germline_db(exD_db_name)
```

`install_igblast` *Install IgBLAST*

Description

Download and install a pre-compiled IgBLAST from NCBI FTP site for use with **igblastr**.

Usage

```
install_igblast(release="LATEST", overwrite=FALSE, ...)
```

Arguments

release	A single string specifying the IgBLAST release version to install. For example "LATEST" (recommended), or one of the IgBLAST release versions listed at https://ftp.ncbi.nih.gov/blast/executables/igblast/release/ (e.g. "1.21.0"). Note that old versions have not been tested and are not guaranteed to be compatible with the **igblastr** package.
overwrite	Set to TRUE to reinstall if the specified IgBLAST release version is already installed.
...	Extra arguments to be passed to the internal call to download.file(). See ?download.file in the **utils** package for more information.

Value

The path to the *root directory* of the IgBLAST installation, as an invisible string.

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IGBLAST_ROOT to use an *external* IgBLAST installation.
- igblast_info to collect basic information about the IgBLAST installation used by the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

igblast_info()
```

---

install_IMGT_germline_db

Install a germline db from IMGT

Description

The install_IMGT_germline_db() function creates and installs an IgBLAST-compatible germline database from germline V, D, and J gene sequences available on IMGT/V-QUEST download site: https://www.imgt.org/download/V-QUEST/

The new database gets installed in **igblastr**'s persistent cache and can be used later with igblastn().

CONDITIONS OF USE AND LICENSE: The IMGT data is provided to the academic users and NPO's (Not for Profit Organization(s)) under the CC BY-NC-ND 4.0 license. See https://creativecommons.org/licenses/by-nc-nd/4.0/. Any other use of IMGT material, from the private sector, needs a financial arrangement with CNRS.

Usage

```
install_IMGT_germline_db(release, organism="Homo sapiens", tcr.db=FALSE,
                         loci="auto",
                         auto.intdata=TRUE, auto.auxdata=TRUE,
                         overwrite=FALSE, verbose=FALSE, ...)

## Can only be used to "subset" a built-in C-region database (see
## advanced example in Examples section below):
install_IMGT_c_region_db(organism, loci,
                         disambiguate.allele.names=FALSE,
                         overwrite=FALSE, verbose=FALSE)
```

Arguments

`release`, `organism`, `tcr.db`
: See ?download_IMGT_germline_sequences for the documentation of these arguments. Note that, in addition to the organism latin name (e.g. "Homo sapiens"), `install_IMGT_c_region_db()` also accepts the common name of the organism (e.g. "human").

`loci`
: By default, the database to install will be populated with the allele sequences from all the BCR or TCR loci. However, if you want to restrict the database to specific loci, you can use the `loci` argument to specify these loci. The subset of loci can be specified as a character vector with one element per locus (e.g. `"IGH"` or `c("TRA", "TRB")`), or as a `+`-separated list in a single string (e.g. `"TRA+TRB"`).

`auto.intdata`
: By default, `install_IMGT_germline_db()` will also compute and add the *internal data* (i.e. the FWR/CDR boundaries on the V alleles) to the new db. Note that `install_IMGT_germline_db()` uses the *gapped* V allele sequences provided by IMGT for that. You can skip this step by setting `auto.intdata` to `FALSE`.

  The presence of *internal data* in the new db will be indicated in the `intdata` column of the data.frame returned by `list_germline_dbs()`.

  If a cached germline db includes its own *internal data*, then `igblastn()` will use it (as *custom internal data*) instead of the *internal data* provided by IgBLAST. See documentation of the `custom_internal_data` argument in ?igblastn for more information.

  Also see ?intdata_utils for more information about IgBLAST *internal data*.

`auto.auxdata`
: COMING SOON...

`overwrite`
: Set to `TRUE` to reinstall if the requested database is already installed.

`verbose`
: Set to `TRUE` to have the function display some details about its internal operations.

`...`
: Extra arguments to be passed to the internal call to `download.file()`. See ?download.file in the **utils** package for more information.

`disambiguate.allele.names`
: See documentation of the `disambiguate.allele.names` in ?install_custom_germline_db.

Details

`install_IMGT_germline_db()` installs the newly created blast database in **igblastr**'s persistent cache. The database can then be used later with `igblastn()`.

The following naming scheme is used to form the name of the installed database:

```
IMGT-<release>.<organism>.<loci>
```

where:

1. <release> is the IMGT/V-QUEST release e.g. 202614-2 or 202449-1. Use `list_IMGT_releases()` to get the list of releases currently available at IMGT/V-QUEST.
2. <organism> is the latin name (a.k.a. binomial name) of the organism with all spaces replaced with underscores (_). For example Homo_sapiens or Macaca_mulatta. Use `list_IMGT_organisms("<release>")` to get the list of organisms included in a given IMGT/V-QUEST release. Note that, starting with release 202405-2, IMGT/V-QUEST provides BCR and TCR germline gene allele sequences for mouse strain C57BL6J (Mus_musculus_C57BL6J).
3. <loci> is a string obtained by concatenating the germline loci together separated with the + sign. For example IGH+IGK+IGL or TRA+TRB+TRG+TRD. The list of loci depends on whether the germline gene allele sequences for BCR or TCR were requested. See `tcr.db` argument above for more information. Note that for some IMGT/V-QUEST releases/organisms, only a subset of the loci are available. For example, in release 202343-3, the only TCR germline loci available for Mus_musculus_C57BL6J are TRA and TRB. This will be automatically reflected in the name of the installed germline db.

Value

`install_IMGT_germline_db()` returns the name to the newly installed germline db as an invisible string.

`install_IMGT_c_region_db()` returns the name to the newly installed C-region db as an invisible string.

Note

`install_IMGT_germline_db()` creates the local database by performing the instructions provided at https://ncbi.github.io/igblast/cook/How-to-set-up.html.

See Also

- `install_custom_germline_db` to install a germline db from user-supplied gene allele sequences.
- `list_IMGT_releases` to list IMGT/V-QUEST releases.
- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- `list_germline_dbs` to list the cached germline dbs.
- `use_germline_db` to select the cached germline db to use with `igblastn()`.
- The IMGT website: https://www.imgt.org/.
- The IMGT/V-QUEST download site: https://www.imgt.org/download/V-QUEST/.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

if (IMGT_is_up()) {
  ## ------------------------------------------------------------------
  ## BASIC EXAMPLES
  ## ------------------------------------------------------------------

  ## As of April 10, 2026, the latest IMGT/V-QUEST release is 202614-2:
  list_IMGT_releases()

  list_IMGT_organisms("202614-2")

  ## Download human BCR germline gene allele sequences from IMGT/V-QUEST
  ## release 202614-2, and store them in a cached germline database:
  install_IMGT_germline_db("202614-2", organism="Homo sapiens",
                           overwrite=TRUE)

  ## List the cached germline databases:
  list_germline_dbs()

  ## Select newly installed germline db to use with igblastn():
  use_germline_db("IMGT-202614-2.Homo_sapiens.IGH+IGK+IGL")

  ## Download human TCR germline gene allele sequences from IMGT/V-QUEST
  ## release 202614-2, and store them in a cached germline database:
  install_IMGT_germline_db("202614-2", organism="Homo sapiens",
                           tcr.db=TRUE, overwrite=TRUE)
  list_germline_dbs()

  ## ------------------------------------------------------------------
  ## ADVANCED EXAMPLES
  ## ------------------------------------------------------------------

  ## Install an IMGT database for a subset of TCR loci:
  install_IMGT_germline_db("202614-2", organism="Homo sapiens",
                           loci="TRA+TRB", overwrite=TRUE)
  list_germline_dbs()

  ## Install the corresponding C-region database:
  install_IMGT_c_region_db("human", "TRA+TRB", overwrite=TRUE)
  list_c_region_dbs()

  ## Note that install_IMGT_c_region_db() can only be used to "subset"
  ## a built-in C-region database.

  ## ------------------------------------------------------------------
  ## WARNING about disagreements between the igblastr-generated
  ## "internal data" that install_IMGT_germline_db() includes in a
  ## germline db and the "internal data" provided by IgBLAST for the
  ## corresponding organism
  ## ------------------------------------------------------------------
```

```
  ## 1. The igblastr-generated "internal data" for the rat and rhesus
  ##    monkey IG V alleles from IMGT has disagreements with the
  ##    "internal data" provided by IgBLAST for these organisms!
  suppressMessages(install_IMGT_germline_db("202614-2", "Rattus norvegicus",
                                            overwrite=TRUE))
  suppressMessages(install_IMGT_germline_db("202614-2", "Macaca mulatta",
                                            overwrite=TRUE))

  ## 2. The igblastr-generated "internal data" for the human and mouse
  ##    TR V alleles from IMGT has disagreements with the "internal data"
  ##    provided by IgBLAST for these organisms!
  suppressMessages(install_IMGT_germline_db("202614-2", "Homo sapiens",
                                            tcr.db=TRUE, overwrite=TRUE))
  suppressMessages(install_IMGT_germline_db("202614-2", "Mus musculus",
                                            tcr.db=TRUE, overwrite=TRUE))
}
```

`intdata-utils` *Access IgBLAST internal data*

Description

IgBLAST *internal data* is expected to annotate all the known germline V gene alleles for a given organism. It is provided by NCBI and is typically included in a standard IgBLAST installation.

The *internal data* informs about the FWR/CDR boundaries on the V allele sequences, also known as the V gene delineations. This information is needed and used internally by IgBLAST.

`get_intdata_path()` and `load_intdata()` can be used to access the *internal data* included in IgBLAST or in one of the *cached germline dbs* managed by **igblastr**.

`V_genes_with_varying_fwrcdr_boundaries()` can be used to identify germline V genes for which the FWR/CDR boundaries are not the same across all alleles.

Usage

```
## Access internal data:
get_intdata_path(igblast_organism, for.aa=FALSE,
                 domain_system=c("imgt", "kabat"),
                 which=c("live", "original"))
load_intdata(igblast_organism, for.aa=FALSE,
             domain_system=c("imgt", "kabat"),
             which=c("live", "original"))

## Show disagreements (if any) between igblastr-generated "internal data"
## included in a germline db and IgBLAST-provided "internal data":
show_intdata_disagreements(db_name)

## Identify V genes with varying FWR/CDR boundaries across alleles:
V_genes_with_varying_fwrcdr_boundaries(intdata, V_segment=NULL)
```

Arguments

igblast_organism	A single string containing the name of an "IgBLAST organism" as returned by list_igblast_organisms(). Alternatively, this can be the name of a cached germline db. Note that this works only for germline dbs that include their own *internal data*. You can use: `list_germline_dbs(with.intdata.only=TRUE)` to list them. See ?list_germline_dbs for more information.
for.aa	By default, the data.frame returned by load_intdata() contains FWR/CDR boundaries reported with respect to the nucleotide sequences of the germline V alleles. Setting for.aa to TRUE will return a data.frame where they are reported with respect to the amino acid sequences of the germline V alleles.
domain_system	Domain system to be used for segment annotation. Must be "imgt" (the default) or "kabat".
which	By default, get_intdata_path() and load_intdata() access the "live IgBLAST data", that is, the IgBLAST data that the user has possibly updated with update_live_igdata(). Depending on whether updates were applied or not, the "live IgBLAST data" might differ from the original IgBLAST data. Set which to "original" if you want to access the original IgBLAST data instead. See ?update_live_igdata for more information about "live" and "original" IgBLAST data.
db_name	A single string specifying the name of a cached germline db.
intdata	A data.frame as returned by load_intdata().
V_segment	The name of a V gene segment. This can be set to "fwr1", "cdr1", "fwr2", "cdr2", or "fwr3". By default (i.e. when V_segment is omitted or set to NULL), V_genes_with_varying_fwrcdr_boundarie will identify V genes for which any segment has varying boundaries across alleles. Otherwise, it will identify V genes for which the specified segment has varying boundaries.

Details

IgBLAST *internal data* is typically included in a standard IgBLAST installation. It's located in the internal_data/ directory which is itself a subdirectory of IgBLAST *root directory*.

Value

get_intdata_path() returns a single string containing the path to the *internal data* included in the IgBLAST installation used by **igblastr**, for the specified organism.

load_intdata() returns the *internal data* in a data.frame with 1 row per germline V gene allele sequence and the same columns as the data.frame returned by read_ndm_data(). See ?read_ndm_data for more information.

show_intdata_disagreements() doesn't return anything (i.e. invisible NULL).

V_genes_with_varying_fwrcdr_boundaries() returns a character vector containing the names of the germline V genes for which the FWR/CDR boundaries are not the same across all alleles.

See Also

- compute_V_gene_delineations to annotate a set of germline V gene allele sequences.
- auxdata_utils to access IgBLAST *auxiliary data*.
- update_live_igdata for more information about "live" and "original" IgBLAST data.
- list_igblast_organisms to list the organisms supported by IgBLAST (a.k.a. "IgBLAST organisms").
- list_germline_dbs to list the cached germline dbs.
- The igblastn function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

igblast_info()

## ---------------------------------------------------------------------
## Access IgBLAST internal data for a given "IgBLAST organism"
## ---------------------------------------------------------------------

## IgBLAST only includes internal data for the following organisms:
list_igblast_organisms()

get_intdata_path("rabbit")

rabbit_intdata <- load_intdata("rabbit")
head(rabbit_intdata)

rabbit_intdata2 <- load_intdata("rabbit", for.aa=TRUE)
head(rabbit_intdata2)

## The values in the "end" cols in 'rabbit_intdata' are exactly 3 times
## those in the "end" cols in 'rabbit_intdata2':
end_colnames <- grep("_end$", colnames(rabbit_intdata), value=TRUE)
stopifnot(identical(rabbit_intdata [ , end_colnames],
                    rabbit_intdata2[ , end_colnames] * 3L))

## ---------------------------------------------------------------------
## Access the internal data included in a germline db
## ---------------------------------------------------------------------

## List germline dbs with internal data:
list_germline_dbs(with.intdata.only=TRUE)

## Access the internal data included in germline db
## _OGRDB.human.IGH+IGK+IGL.202605 (this data was inferred from
## the gaps in the germline V gene allele sequences provided by
## AIRR-community/OGRDB):
```

```
db_name <- "_OGRDB.human.IGH+IGK+IGL.202605"

get_intdata_path(db_name)

human_intdata <- load_intdata(db_name)
head(human_intdata)

## ------------------------------------------------------------------------
## show_intdata_disagreements()
## ------------------------------------------------------------------------

show_intdata_disagreements(db_name)

## See last examples in '?install_IMGT_germline_db' for some examples
## of germline dbs with disagreements.

## ------------------------------------------------------------------------
## V_genes_with_varying_fwrcdr_boundaries()
## ------------------------------------------------------------------------

## Note that the alleles of a given germline V gene don't necessarily
## share the same FWR/CDR boundaries. You can use utility function
## V_genes_with_varying_fwrcdr_boundaries() to identify them:
human_intdata0 <- load_intdata("human")
var_genes <- V_genes_with_varying_fwrcdr_boundaries(human_intdata0)
var_genes  # 4 genes

## Display rows associated with gene IGHV4-31:
subset(human_intdata0, allele2gene(allele_name) == "IGHV4-31")

## Human germline V genes for which the CDR1 boundaries are not the
## same across all alleles:
var_genes <- V_genes_with_varying_fwrcdr_boundaries(human_intdata0,
                                                    V_segment="cdr1")
var_genes  # 2 genes

## Display rows associated with the genes in 'var_genes':
subset(human_intdata0, allele2gene(allele_name) %in% var_genes)
```

`J_alleles-inspect` *Basic inspection of J allele sequences*

Description

A small set of utilities for basic inspection of J allele sequences.

The main function is `print_J_alleles()` which can be used to display a set of J alleles sequences where the sequences are colored and justified w.r.t. their CDR3/FWR4 junction.

Note that all of the functions documented in this man page require access to the *auxiliary data* associated with the J alleles to inspect. See `auxdata` argument below for how to obtain this data.

Usage

```
print_J_alleles(J_alleles, auxdata, translate=FALSE,
                igblast_organism=NA, filler=".", cdr3fwr4_sep=" ")

translate_J_alleles(J_alleles, auxdata)
J_allele_has_stop_codon(J_alleles, auxdata)
translate_fwr4(J_alleles, auxdata, max.codons=NA)
```

Arguments

J_alleles	A DNAStringSet object containing germline J gene allele sequences. Alternatively, this can be the name of a cached germline db, in which case the auxdata argument doesn't need to (and shouldn't) be supplied. Note that this works only for germline dbs that include their own *auxiliary data*. You can use: `list_germline_dbs(with.auxdata.only=TRUE)` to list them. See ?list_germline_dbs for more information.
auxdata	A data.frame as returned by load_auxdata(), compute_auxdata(), or compute_germline_db_auxdata that contains annotations for the alleles in J_alleles.
translate	By default, print_J_alleles() displays the nucleotide sequences. Set translate to TRUE to display the amino acid sequences instead.
igblast_organism	If the alleles in J_alleles are from an "IgBLAST organism" (see ?list_igblast_organisms), then the name of the organism can be passed to igblast_organism. In this case print_J_alleles() will display whether each allele is also annotated in the *auxiliary data* provided by IgBLAST for this organism.
filler	The character to use to pad the sequences on both ends. This padding achieves the horizontal alignment of all sequences w.r.t. the CDR3/FWR4 junction.
cdr3fwr4_sep	The string to use to separate the CDR3 portions of the sequences from their FWR4 portions.
max.codons	The maximum number of FWR4 codons to translate. By default (i.e. when max.codons is NA) all the FWR4 codons are translated.

Value

print_J_alleles() returns the CDR3 and FWR4 portions of the J allele sequences in an invisible DataFrame with 1 row per J allele.

translate_J_alleles() returns a named character vector with 1 amino acid sequence per supplied allele. The vector contains an NA for any allele that is not annotated in auxdata or for which auxdata$coding_frame_start has an NA. The names on it are the names of the supplied alleles.

J_allele_has_stop_codon() returns a named logical vector with 1 value per supplied allele. The vector contains an NA for any allele that is not annotated in auxdata or for which auxdata$coding_frame_start has an NA. The names on it are the names of the supplied alleles.

translate_fwr4() returns a named character vector with 1 amino acid sequence per supplied allele. The vector contains an NA for any allele that is not annotated in auxdata or for which auxdata$cdr3_end has an NA.

See Also

- auxdata_utils to access IgBLAST *auxiliary data*.
- intdata_utils to access IgBLAST *internal data*.
- V_alleles_inspect for basic inspection of V allele sequences.
- `update_live_igdata` for more information about "live" and "original" IgBLAST data.
- DNAStringSet and AAStringSet objects in the **Biostrings** package.
- The `translate_codons` function which is used internally by `print_J_alleles()`, `translate_J_alleles()`, and `translate_fwr4()`.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

igblast_info()

## ------------------------------------------------------------------------
## 1. A QUICK LOOK AT THE GERMLINE J GENE ALLELES PROVIDED BY OGRDB
## ------------------------------------------------------------------------

list_germline_dbs()

## --- Human germline J gene alleles ---

db_name <- "_OGRDB.human.IGH+IGK+IGL.202605"

## The 'also_in_IgBLAST_auxdata' column indicates whether an allele is
## also annotated in the "auxiliary data" provided by IgBLAST for human
## (i.e. in the optional_file/human_gl.aux file shipped with IgBLAST):
print_J_alleles(db_name)
print_J_alleles(db_name, translate=TRUE)

## Note that the above shows that:
## - All FWR4 regions from the IGH locus (heavy chain) start with
##   the "WGXG" motif.
## - All FWR4 regions from the IGK and IGL loci (light chain) start
##   with the "FGXG" motif.

## --- Mouse germline J gene alleles ---

db_name <- "_OGRDB.mouse.MSM_MsJ.IGH+IGK+IGL.202603"

## None of the mouse J alleles from OGRDB are annotated in the "auxiliary
## data" provided by IgBLAST for mouse (i.e. in optional_file/mouse_gl.aux).
print_J_alleles(db_name, translate=TRUE)

## The "auxiliary data" included in this germline db is provided by OGRDB.
## In fact, the "auxiliary data" included in **all** the OGRDB built-in
## dbs is provided by OGRDB.
```

```
## ---------------------------------------------------------------------
## 2. A QUICK LOOK AT THE GERMLINE J GENE ALLELES PROVIDED BY IMGT
## ---------------------------------------------------------------------

## --- For "IgBLAST organisms" ---

## Mouse:
db_name <- install_IMGT_germline_db("202614-2", "Mus_musculus",
                                    overwrite=TRUE)
print_J_alleles(db_name, translate=TRUE)

## Note that:
## - The annotations for the 3 alleles not annotated in
##   IgBLAST "auxiliary data" for mouse were computed by
##   install_IMGT_germline_db().
## - The CDR3 of allele IGHJ3*02 contains a stop codon (indicated by
##   character "*").

## Rat:
db_name <- suppressWarnings(
  install_IMGT_germline_db("202614-2", "Rattus_norvegicus", overwrite=TRUE)
)
print_J_alleles(db_name, translate=TRUE)

## Notes:
## - The ?> and <? tags around a sequence indicate an allele for which
##   the CDR3/FWR4 junction is unknown.
## - Even though allele IGKJ3*01 is annotated in the "auxiliary data"
##   provided by IgBLAST for rat (in optional_file/rat_gl.aux), its
##   cdr3_end value is missing:
igblast_auxdata <- load_auxdata("rat")
igblast_auxdata
subset(igblast_auxdata, is.na(cdr3_end))

## --- For non-IgBLAST organisms ---

## Gorilla:
db_name <- install_IMGT_germline_db("202614-2", "Gorilla_gorilla_gorilla",
                                    overwrite=TRUE)
print_J_alleles(db_name, translate=TRUE)

## Note that, in this case:
## - There's no 'also_in_IgBLAST_auxdata' column.
## - install_IMGT_germline_db() took care of computing the annotations
##   for all the alleles.

## Pig:
db_name <- install_IMGT_germline_db("202614-2", "Sus_scrofa",
                                    overwrite=TRUE)
print_J_alleles(db_name, translate=TRUE)

## ---------------------------------------------------------------------
## 3. MORE ABOUT print_J_alleles()
```

```
## ---------------------------------------------------------------------

## print_J_alleles() also accepts an arbitrary set of J allele sequences
## and their annotations (via arguments 'J_alleles' and 'auxdata',
## respectively):

db_name <- "IMGT-202614-2.Mus_musculus.IGH+IGK+IGL"
J_alleles <- load_germline_sequences(db_name, region_types="J")
auxdata <- load_auxdata(db_name)
print_J_alleles(J_alleles, auxdata, igblast_organism="mouse")

## Note that the above is equivalent to 'print_J_alleles(db_name)'.

## print_J_alleles() returns a DataFrame with the CDR3 and FWR4
## portions of the J allele sequences:

DF <- suppressMessages(print_J_alleles(db_name, translate=TRUE))
DF

## ---------------------------------------------------------------------
## 4. FWR4 CONSENSUS SEQUENCES
## ---------------------------------------------------------------------

## The above can be used to determine the consensus amino acid sequence
## for mouse FWR4 regions:

fwr4_aa <- setNames(DF$fwr4, DF$allele_name)
fwr4_aa <- fwr4_aa[width(fwr4_aa) != 0L]
IGHJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "IGH"]
IGKJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "IGK"]
IGLJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "IGL"]
consensusString(IGHJ_fwr4_aa)  # WGQGT?VTVSS  (starts with "WGXG" motif)
consensusString(IGKJ_fwr4_aa)  # FG?GTKLEIK   (starts with "FGXG" motif)
consensusString(IGLJ_fwr4_aa)  # FGGGT?LTVL   (starts with "FGXG" motif)

## See '?consensusString' in the Biostrings package for more information
## about the consensusString() function.

## Let's take a look at the FWR4 sequences for human TCR J alleles:
db_name <- suppressWarnings(
  install_IMGT_germline_db("202614-2", "Homo_sapiens", tcr.db=TRUE,
                           overwrite=TRUE)
)
print_J_alleles(db_name, translate=TRUE)

DF <- suppressMessages(print_J_alleles(db_name, translate=TRUE))
fwr4_aa <- setNames(DF$fwr4, DF$allele_name)
fwr4_aa <- fwr4_aa[width(fwr4_aa) != 0L]
TRAJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "TRA"]
TRBJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "TRB"]
TRGJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "TRG"]
TRDJ_fwr4_aa <- fwr4_aa[substr(names(fwr4_aa), 1, 3) == "TRD"]
consensusString(TRAJ_fwr4_aa)  # FG?GT?L?V?P
```

```
consensusString(TRBJ_fwr4_aa)  # FG?GTRLTVL
consensusString(TRGJ_fwr4_aa)  # FG?GT?LIVTSP
consensusString(TRDJ_fwr4_aa)  # FGKGT?L?VEP

## Note that, unlike with BCR J alleles where the FWR4 is expected to
## start with the "WGXG" and "FGXG" motifs on the heavy- and light-chain,
## respectively, for human TCR J alleles it's expected to start with
## the "FGXG" motif regardless of the locus.

## ---------------------------------------------------------------------
## 5. translate_J_alleles()
## ---------------------------------------------------------------------

db_name <- install_IMGT_germline_db("202614-2", "Oryctolagus cuniculus",
                                    overwrite=TRUE)
J_alleles <- load_germline_sequences(db_name, region_types="J")
auxdata <- load_auxdata(db_name)
head(auxdata)

## The 'coding_frame_start' column in 'auxdata' contains integer
## values that are >= 0 and <= 2. They indicate how many nucleotides
## precede the first codon on each allele sequence. In other words,
## this is the number of nucleotides that we need to trim on the 5'
## end of the germline J allele sequence before we start translating
## it. translate_J_alleles() uses this information to translate the
## DNA sequences in 'J_alleles':
J_aa <- translate_J_alleles(J_alleles, auxdata)
J_aa

## No sequence in 'J_aa' should contain the letter "*" which is used
## by translate_J_alleles() to represent a stop codon. However, one
## J allele in IMGT-202614-2.Oryctolagus_cuniculus.IGH+IGK+IGL seems
## to disobey:
has_stop_codon <- grepl("*", J_aa, fixed=TRUE)
auxdata[has_stop_codon, ]  # coding_frame_start = 0 for IGKJ1-2*04
J_alleles[has_stop_codon]  # first codon (TGA) is a stop codon
J_aa[has_stop_codon]       # indeed!

## ---------------------------------------------------------------------
## 6. ABOUT THE "WGXG" AND "FGXG" MOTIFS
## ---------------------------------------------------------------------

## The FWR4 region is expected to start with the following amino acid
## motifs (X represents any amino acid):
##   - "WGXG" on the heavy chain
##   - "FGXG" on the light chain

## Let's use translate_fwr4() to extract and translate the first 4
## codons of the FWR4 region:
fwr4_head <- translate_fwr4(J_alleles, auxdata, max.codons=4)

## We expect to see the "WGXG" and "FGXG" motifs here, and most of the
## time we do:
```

```
has_motif <- grepl("[FW]G.G", fwr4_head)
table(has_motif)

## However, there are a few exceptions:
fwr4_head[!has_motif]
```

`list_c_region_dbs` *List cached C-region dbs*

Description

List the *cached C-region dbs*, that is, the IgBLAST-compatible C-region databases currently installed in **igblastr**'s persistent cache.

Usage

```
## List the cached C-region dbs:
list_c_region_dbs(builtin.only=FALSE, names.only=FALSE, long.listing=FALSE)

## Remove a C-region db from igblastr's persistent cache:
rm_c_region_db(db_name)
```

Arguments

`builtin.only`	By default `list_c_region_dbs()` returns the list of all cached C-region dbs, including built-in C-region dbs. Set `builtin.only` to `TRUE` to return only the list of built-in C-region dbs. Note that built-in dbs are prefixed with an underscore (_).
`names.only`	By default `list_c_region_dbs()` returns the list of cached C-region dbs in a data.frame with one db per row. Set `names.only` to `TRUE` to return the db names only, in which case they are returned in a character vector.
`long.listing`	`TRUE` or `FALSE`. If set to `TRUE`, then `list_c_region_dbs()` returns a named list with one list element per C-region db. Each list element is a named integer vector that indicates the number of C-region sequences per locus. Ignored if `names.only` is set to `TRUE`.
`db_name`	A single string specifying the name of the C-region db to remove from **igblastr**'s persistent cache. This cannot be a built-in db.

Details

Cached germline dbs and C-region dbs: The **igblastr** package provides utility functions to manage the *cached germline dbs* and *cached C-region dbs* to use with `igblastn()`.

Terminology used across **igblastr** documentation:

- A *cached germline db* contains the nucleotide sequences of the germline V, D, and J gene alleles for a given organism.

- A *cached C-region db* contains the nucleotide sequences of the constant gene regions for a given organism.

This man page documents utilities that manage cached C-region dbs.

See ?list_germline_dbs for utilities that manage cached germline dbs.

Built-in dbs: The **igblastr** package comes with preinstalled cached germline and C-region dbs that are called *built-in dbs*.

Built-in dbs have their name prefixed with an underscore (_).

Note that the built-in C-region dbs from IMGT were downloaded from https://www.imgt.org/vquest/refseqh.html#constant-sets and included in the **igblastr** package on the date indicated by the suffix of the db name.

Value

`list_c_region_dbs()` returns the list of all cached C-region dbs in a data.frame with one db per row (if `names.only` is `FALSE`, which is the default), or in a character vector (if `names.only` is `TRUE`). Column C in the data.frame indicates the number of C-region sequences in each db.

`rm_c_region_db()` does not return anything (invisible `NULL`).

See Also

- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- `use_c_region_db` to select the cached C-region db to use with `igblastn()`.
- `reset_c_region_dbs` to reset the cached C-region dbs.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## 7 built-in C-region dbs (prefixed with an underscore):
list_c_region_dbs()
list_c_region_dbs(names.only=TRUE)    # db names only
list_c_region_dbs(long.listing=TRUE)  # long listing
```

---

list_germline_dbs *List cached germline dbs*

Description

List the *cached germline dbs*, that is, the IgBLAST-compatible germline databases currently installed in **igblastr**'s persistent cache.

Usage

```
## List the cached germline dbs:
list_germline_dbs(builtin.only=FALSE,
                  with.intdata.only=FALSE,
                  with.auxdata.only=FALSE,
                  names.only=FALSE, long.listing=FALSE)

## Remove a germline db from igblastr's persistent cache:
rm_germline_db(db_name)
```

Arguments

builtin.only	By default list_germline_dbs() returns the list of all cached germline dbs, including built-in germline dbs. Set builtin.only to TRUE to return only the list of built-in germline dbs. Note that built-in dbs are prefixed with an underscore (_).
with.intdata.only	Set with.intdata.only to TRUE to return only the list of germline dbs that include their own *internal data* (i.e. the FWR/CDR boundaries on the V alleles). Note that whether a germline db includes its own *internal data* or not is reported in the intdata column. If a cached germline db includes its own *internal data*, then igblastn() will use it (as *custom internal data*) instead of the *internal data* provided by IgBLAST for the corresponding organism. See documentation of the custom_internal_data argument in ?igblastn for more information.
with.auxdata.only	Set with.auxdata.only to TRUE to return only the list of germline dbs that include their own *auxiliary data* (i.e. annotations of the J alleles). Note that whether a germline db includes its own *auxiliary data* or not is reported in the auxdata column. If a cached germline db includes its own *auxiliary data*, then igblastn() will use it instead of the *auxiliary data* provided by IgBLAST for the corresponding organism. See documentation of the auxiliary_data argument in ?igblastn for more information.
names.only	By default list_germline_dbs() returns the list of cached germline dbs in a data.frame with one db per row. Set names.only to TRUE to return the db names only, in which case they are returned in a character vector.
long.listing	TRUE or FALSE. If set to TRUE, then list_germline_dbs() returns a named list with one list element per germline db. Each list element is an integer matrix that indicates the number of germline gene allele sequences per locus and region type. Ignored if names.only is set to TRUE.
db_name	A single string specifying the name of the germline db to remove from **igblastr**'s persistent cache. This cannot be a built-in db.

Details

Cached germline dbs and C-region dbs: The **igblastr** package provides utility functions to manage the *cached germline dbs* and *cached C-region dbs* to use with `igblastn()`.

Terminology used across **igblastr** documentation:

- A *cached germline db* contains the nucleotide sequences of the germline V, D, and J gene alleles for a given organism.
- A *cached C-region db* contains the nucleotide sequences of the constant gene regions for a given organism.

This man page documents utilities that manage cached germline dbs.

See `?list_c_region_dbs` for utilities that manage cached C-region dbs.

Built-in dbs: The **igblastr** package comes with preinstalled cached germline and C-region dbs that are called *built-in dbs*.

Built-in dbs have their name prefixed with an underscore (_).

Note that additional dbs can easily be installed with functions like `install_IMGT_germline_db` or `install_custom_germline_db`.

AIRR-community/OGRDB built-in cached dbs: Note that the built-in germline dbs starting with `_AIRR` are made of the AIRR-community/OGRDB germline sets available at https://ogrdb.airr-community.org/germline_sets/Homo%20sapiens for human, at https://ogrdb.airr-community.org/germline_sets/Mus%20musculus for mouse, and at https://ogrdb.airr-community.org/germline_sets/Macaca%20mulatta for rhesus monkey.

Each AIRR db is populated with the latest germline datasets that were available at AIRR-community/OGRDB at the time indicated by the date (in YYYYMM format) embedded in the db name.

The AIRR dbs with the `.src` suffix contain the *Source Sets*. The AIRR dbs without the `.src` suffix contain the *Reference Sets*. See https://github.com/HyrienLab/igblastr/tree/devel/inst/extdata/germline_sets/AIRR/human/202410/README.md for more information.

The AIRR-community/OGRDB maintainers recommend to use the *Reference Sets* for AIRR-seq analysis. See for example https://ogrdb.airr-community.org/germline_set/75

Value

`list_germline_dbs()` returns the list of all cached germline dbs in a data.frame with one db per row (if `names.only` is `FALSE`, which is the default), or in a character vector (if `names.only` is `TRUE`). Columns V, D, J in the data.frame indicate the number of germline gene allele sequences for each region in each db.

`rm_germline_db()` does not return anything (invisible `NULL`).

See Also

- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- `use_germline_db` to select the cached germline db to use with `igblastn()`.
- `install_IMGT_germline_db` to install a germline db from IMGT.
- `install_custom_germline_db` to install a germline db from user-supplied gene allele sequences.

- intdata_utils to access IgBLAST *internal data*.
- `reset_germline_dbs` to reset the cached germline dbs.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## Get list of built-in germline dbs only.
list_germline_dbs(builtin.only=TRUE)
list_germline_dbs(builtin.only=TRUE, names.only=TRUE)  # db names only

## Get list of germline dbs that include their own internal data:
list_germline_dbs(with.intdata.only=TRUE)
list_germline_dbs(with.intdata.only=TRUE, names.only=TRUE)

## Long listing:
list_germline_dbs(long.listing=TRUE)
list_germline_dbs(with.intdata.only=TRUE, long.listing=TRUE)

if (IMGT_is_up()) {
  ## Install Mouse germline db from IMGT:
  install_IMGT_germline_db("202614-2", "Homo sapiens", overwrite=TRUE)

  list_germline_dbs()  # all germline dbs

  ## Select germline db to use with igblastn():
  db_name <- "IMGT-202614-2.Homo_sapiens.IGH+IGK+IGL"
  use_germline_db(db_name)  # select germline db to use

  use_germline_db()  # get current selection
}
```

`ndm_data-IO` *Read/write "ndm" files*

Description

The *ndm* (Nucleotide Domain Mapping) file format is a specialized, tab-delimited text file used by IgBLAST to define the FWR/CDR boundaries on the germline V gene allele sequences.

The **igblastr** package provides low-level functions to read/write data in *ndm* format.

Usage

```
read_ndm_data(filepath)
write_ndm_data(ndm_data, file="", check.data=FALSE)

validate_ndm_rows(ndm_data, allow.repeated.rows=FALSE)
```

Arguments

filepath	The path to the "ndm" file to read.
ndm_data	A data.frame with 1 row per germline V gene allele sequence and the same columns as the data.frame returned by read_ndm_data(). See Value section below for the details.
file	The path to the "ndm" file to write.
check.data	COMING SOON...
allow.repeated.rows	COMING SOON...

Value

read_ndm_data() returns a data.frame with 1 row per germline V gene allele sequence and the following columns:

- allele_name: allele name;
- fwr1_start, fwr1_end: FWR1 start/end positions (1-based);
- cdr1_start, cdr1_end: CDR1 start/end positions (1-based);
- fwr2_start, fwr2_end: FWR2 start/end positions (1-based);
- cdr2_start, cdr2_end: CDR2 start/end positions (1-based);
- fwr3_start, fwr3_end: FWR3 start/end positions (1-based);
- chain_type: chain type as a 2-letter code e.g. VK for "V allele from the Kappa locus" (BCR locus) or VG for "V allele from the Gamma locus" (TCR locus);
- coding_frame_start: first coding frame start position (0-based).

write_ndm_data() doesn't return anything (i.e. invisible NULL).

validate_ndm_rows() returns a logical vector with 1 value per row in the ndm_data data.frame indicating whether the data in that row is valid or not.

See Also

- compute_V_gene_delineations to annotate a set of germline V gene allele sequences.
- extract_intdata_from_ogrdb_json to extract the V gene delineations for the V alleles annotated in an AIRR-C JSON file.
- intdata_utils to access IgBLAST *internal data*.
- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## COMING SOON...
```

OAS-utils *Download and manipulate OAS data*

Description

Some utility functions to query the Observed Antibody Space database, a.k.a. OAS, and to download and manipulate data from OAS.

OAS's homepage: https://opig.stats.ox.ac.uk/webapps/oas/

Note that OAS has two databases: the "Unpaired Sequences" database and the "Paired Sequences" database. Some of the utilities documented in this man page only work for data retrieved from the latter.

Usage

```
## Read metadata/data from a single OAS unit file:
read_OAS_csv_metadata(file)
read_OAS_csv(file, skip=1, ...)
extract_sequences_from_paired_OAS_df(df, add.prefix=FALSE)

## Basic query of OAS website:
list_paired_OAS_studies(as.df=FALSE, recache=FALSE)
list_paired_OAS_units(study, as.df=FALSE, recache=FALSE)
download_paired_OAS_units(study, units=NULL, destdir=".", ...)

## Read metadata/data from a batch of downloaded OAS unit files:
extract_metadata_from_OAS_units(dir=".", pattern="\\.csv\\.gz$")
extract_sequences_from_paired_OAS_units(dir=".", pattern="\\.csv\\.gz$")

## Check OAS website:
OAS_is_up()
```

Arguments

file	A single string that is the path to an *OAS unit file*.
skip	The number of lines of the data file to skip before beginning to read data. The first line in an OAS unit file contains metadata in JSON format, so must always be skipped.
...	For read_OAS_csv(): Extra arguments to be passed to the internal call to read.table(). See ?read.table in the **utils** package for more information. For download_paired_OAS_units(): Extra arguments to be passed to the internal call to download.file(). See ?download.file in the **utils** package for more information.
df	The data.frame or tibble returned by read_OAS_csv().

add.prefix — TRUE or FALSE. Should the names on the returned DNAStringSet object be the original sequence ids as-is (this is the default), or should the heavy_chain_ and light_chain_ prefixes be added to them?

extract_sequences_from_paired_OAS_df() returns a DNAStringSet object with the sequence ids as names. The sequence ids are obtained from the sequence_id_heavy and sequence_id_light columns of the supplied data.frame or tibble. By default, they are propagated as-is to the DNAStringSet object, which makes it difficult to recognize which chain (heavy or light) the antibody sequences are coming from. Setting add.prefix to TRUE will add the heavy_chain_ or light_chain_ prefix to the names on the DNAStringSet object, hence making it easy to identify which chain a given antibody sequence is coming from.

as.df — TRUE or FALSE. By default, i.e. when as.df is FALSE, list_paired_OAS_studies() and list_paired_OAS_units() return the list of studies or units in a character vector. Alternatively you can set as.df to TRUE to get the list in a 3-column data.frame that contains a directory index as displayed at https://opig.stats.ox.ac.uk/webapps/ngsdb/paired/ or at https://opig.stats.ox.ac.uk/webapps/ngsdb/paired/Jaffe_2022/csv/.

recache — TRUE or FALSE. list_paired_OAS_studies() and list_paired_OAS_units() both cache the information retrieved from OAS website for the duration of the R session (note that this caching is done in memory so it does not persist across sessions). Set recache to TRUE to force a new retrieval (and recaching) of the results.

study — A single string containing the name of a study as returned by list_paired_OAS_studies().

units — NULL, or a character vector that must be a subset of list_paired_OAS_units(study) in which case the download will be restricted to these units only.

destdir — A single string that is the path to the directory where the OAS unit files are to be downloaded.

dir — A single string that is the path to a directory containing OAS unit files. This will typically be the same as destdir above if the unit files were downloaded with download_paired_OAS_units().

pattern — Regular expression passed to the internal call to list.files() to obtain the list of OAS unit files located in dir. No reason to change this unless you know what you are doing.

Details

OAS delivers data in the form of *OAS unit files*. These files are typically obtained by running the bulk_download.sh script that OAS generates based on one's search criteria. They are compressed CSV (comma-separated values) files with the .csv.gz extension.

OAS unit files can vary a lot in size: from only a few KB to 25 MB or more.

The first line in an OAS unit file contains metadata in JSON format (which means that these files cannot strictly be considered CSV files).

The CSV data is MiAIRR-compliant (see The "MiAIRR format" paper in the References section below).

Value

`read_OAS_csv_metadata()` extracts the metadata from the specified OAS unit file and returns it in a named list.

`read_OAS_csv()` extracts the data from the specified OAS unit file and returns it in a tibble. The tibble has 1 row per antibody sequence if the data is unpaired (i.e. comes from the "Unpaired Sequences" database), or 1 row per sequence pair if the data is paired (i.e. comes from the "Paired Sequences" database).

`extract_sequences_from_paired_OAS_df()` returns the sequence pairs in a named DNAStringSet object where the names are the sequence ids. See `add.prefix` above for how the sequence ids are obtained.

`list_paired_OAS_studies()` returns the list of studies that populate the "Paired Sequences" database in a character vector. This list can be seen here: https://opig.stats.ox.ac.uk/webapps/ngsdb/paired/.

`list_paired_OAS_units()` returns the list of all the OAS unit files that belong to a given study from the "Paired Sequences" database.

`download_paired_OAS_units()` does not return anything (invisible `NULL`).

`extract_metadata_from_OAS_units()` returns the metadata of all the OAS unit files found in the specified directory in a data.frame with 1 row per file.

`extract_sequences_from_paired_OAS_units()` extracts the sequence pairs from all the OAS unit files found in the specified directory and returns them in a named DNAStringSet object where the names are the sequence ids. The sequence ids are obtained by prefixing the original sequence ids found in the files with the name of the unit followed by `_heavy_chain_` or `_light_chain_`.

`OAS_is_up()` returns `TRUE` or `FALSE`, indicating whether the OAS website at https://opig.stats.ox.ac.uk/webapps/oas/ is up and running or down.

References

- The OAS paper:

  Tobias H. Olsen, Fergus Boyles, Charlotte M. Deane. Observed Antibody Space: A diverse database of cleaned, annotated, and translated unpaired and paired antibody sequences. Protein Science (2021). https://doi.org/10.1002/pro.4205

- The "MiAIRR format" paper:

  Rubelt, F., Busse, C., Bukhari, S. et al. Adaptive Immune Receptor Repertoire Community recommendations for sharing immune-repertoire sequencing data. Nat Immunol 18, 1274–1278 (2017). https://doi.org/10.1038/ni.3873

See Also

- OAS's homepage at: https://opig.stats.ox.ac.uk/webapps/oas/
- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- tibble objects implemented in the **tibble** package.
- DNAStringSet objects implemented in the **Biostrings** package.

Examples

```
if (OAS_is_up()) {
  list_paired_OAS_studies()

  list_paired_OAS_units("Eccles_2020")

  ## Import all the pairs of antibody sequences from the Eccles_2020 study:

  download_dir <- tempdir()
  download_paired_OAS_units("Eccles_2020", destdir=download_dir)

  metadata <- extract_metadata_from_OAS_units(download_dir)
  metadata  # data.frame with 1 row per unit file

  sequences <- extract_sequences_from_paired_OAS_units(download_dir)
  sequences  # DNAStringSet object

  ## Odd indices correspond to heavy chain sequences and even indices
  ## to light chain sequences:

  head(names(sequences))

  sequences[1:2]  # 1st pair
  sequences[3:4]  # 2nd pair
  sequences[5:6]  # 3rd pair
  # etc...
}
```

---

parse_imgt_fasta_headers

Parse IMGT FASTA headers

Description

The IMGT FASTA headers contain 15 fields separated by |. See https://www.imgt.org/IMGTindex/Fasta.php.

`parse_imgt_fasta_headers()` takes a vector of headers and parse them into a 15-column matrix with 1 row per header.

Usage

```
parse_imgt_fasta_headers(headers)
```

Arguments

headers	A character vector of IMGT FASTA headers.

Value

A 15-column character matrix with 1 row per header. The column names on the matrix are:

1. `IMGT_acc`: IMGT/LIGM-DB accession number(s);
2. `allele_name`: IMGT gene and allele name;
3. `organism`: species/organism;
4. `func`: IMGT allele functionality;
5. `region`: exon(s), region name(s), or extracted label(s);
6. `startend_in_IMGT_acc`: start and end positions in the IMGT/LIGM-DB accession number(s);
7. `nb_nuc`: number of nucleotides in the IMGT/LIGM-DB accession number(s);
8. `codon_start`: codon start, or 'NR' (not relevant) for non coding labels;
9. `extra_nuc_5prime`: +n: number of nucleotides (nt) added in 5' compared to the corresponding label extracted from IMGT/LIGM-DB;
10. `extra_nuc_3prime`: +n or -n: number of nucleotides (nt) added or removed in 3' compared to the corresponding label extracted from IMGT/LIGM-DB;
11. `nuc_corrected`: +n, -n, and/or nS: number of added, deleted, and/or substituted nucleotides to correct sequencing errors, or `not corrected` if non corrected sequencing errors;
12. `nb_aa`: number of amino acids (AA): this field indicates that the sequence is in amino acids;
13. `nb_chars`: number of characters in the sequence: nt (or AA)+IMGT gaps=total;
14. `partial`: partial (if it is);
15. `revcomp`: reverse complementary (if it is).

See Also

- https://www.imgt.org/IMGTindex/Fasta.php for the format of IMGT FASTA headers.
- download_IMGT_germline_sequences to download germline sequences from IMGT.
- compute_V_gene_delineations to annotate a set of germline V gene allele sequences.

Examples

```
if (IMGT_is_up()) {
  ## ------------------------------------------------------------------
  ## Retrieve IMGT FASTA files for mouse and parse their headers
  ## ------------------------------------------------------------------

  ## As of April 10, 2026, the latest IMGT/V-QUEST release is 202614-2:
  list_IMGT_releases()

  list_IMGT_organisms("202614-2")

  ## Download Mouse BCR germline gene allele sequences from IMGT/V-QUEST
  ## 202614-2 to a temporary directory:
  dir.create(destdir <- tempfile("fasta_dir_"))
  download_IMGT_germline_sequences("202614-2", organism="Mus musculus",
```

```
                                destdir=destdir)

## List the downloaded files:
list.files(destdir)  # 7 FASTA files

## Load the V allele sequences (gapped):
IMGT_groups <- c("IGHV", "IGKV", "IGLV")
V_fasta_files <- file.path(destdir, paste0(IMGT_groups, ".fasta"))
gapped_V_alleles <- readDNAStringSet(V_fasta_files)

## 'gapped_V_alleles' is a DNAStringSet object that carries the IMGT
## FASTA headers as names:
gapped_V_alleles              # 865 V alleles
names(gapped_V_alleles)[1:4]  # first 4 headers

## Let's parse the headers:
parsed_V_headers <- parse_imgt_fasta_headers(names(gapped_V_alleles))
dim(parsed_V_headers)    # 865 x 15 matrix
parsed_V_headers[1:4, ]  # first 4 rows

## ---------------------------------------------------------------------
## Scrutinize IMGT V alleles reported as "partial in 5'"
## ---------------------------------------------------------------------

## The presence of "partial in 5'" in the 14th field ('partial') of
## a header indicates a sequence that is truncated at the 5' end:
partial <- parsed_V_headers[ , "partial"]
table(partial)
is_partial_in_5prime <- grepl("in 5", partial)
table(is_partial_in_5prime)  # 35 sequences marked as "partial in 5'"

## The corresponding sequences are expected to start with a gap:
start_with_gap <- grepl("^\\.", as.character(gapped_V_alleles))

## However, it seems that 1 V allele (IGHV8-9*02) is marked
## as "partial in 5'" but does NOT start with a gap:
table(is_partial_in_5prime, start_with_gap)
not_ok <- is_partial_in_5prime & !start_with_gap
parsed_V_headers[not_ok, ]  # partial in 5'
gapped_V_alleles[not_ok]  # does NOT start with a gap!

not_ok_V_allele <- unname(parsed_V_headers[not_ok, "allele_name"])

stopifnot(identical(not_ok_V_allele, "IGHV8-9*02"))  # sanity check

## ---------------------------------------------------------------------
## Scrutinize IMGT V alleles reported as "partial in 3'"
## ---------------------------------------------------------------------

## The presence of "partial in 3'" in the 14th field ('partial') of
## a header indicates a sequence that is truncated at the 3' end:
is_partial_in_3prime <- grepl("in 3", partial)
table(is_partial_in_3prime)  # 72 sequences marked as "partial in 3'"
```

```
  ## Note that the FWR3 always ends at position 312 on a gapped V allele
  ## sequence. This means that any gapped sequence shorter than that should
  ## be considered truncated at the 3' end:
  gapped_seq_lens <- lengths(gapped_V_alleles)
  has_incomplete_fwr3 <- gapped_seq_lens < 312

  ## However, it seems that 12 V alleles have an incomplete FWR3 but
  ## are NOT marked as "partial in 3'":
  table(has_incomplete_fwr3, is_partial_in_3prime)
  not_ok <- has_incomplete_fwr3 & !is_partial_in_3prime
  gapped_V_alleles[not_ok]  # incomplete FWR3
  parsed_V_headers[not_ok, ]  # NOT marked as partial in 3'

  parsed_V_headers[not_ok, "allele_name"]  # suspect alleles

  stopifnot(sum(not_ok) == 12L)  # sanity check

  ## ---------------------------------------------------------------------
  ## A quick look at the IMGT header of J alleles
  ## ---------------------------------------------------------------------

  ## Load the J allele sequences:
  IMGT_groups <- c("IGHJ", "IGKJ", "IGLJ")
  J_fasta_files <- file.path(destdir, paste0(IMGT_groups, ".fasta"))
  J_alleles <- readDNAStringSet(J_fasta_files)

  ## 'J_alleles' is a DNAStringSet object that carries the IMGT
  ## FASTA headers as names:
  J_alleles              # 27 J alleles
  names(J_alleles)[1:4]  # first 4 headers

  ## Let's parse the headers:
  parsed_J_headers <- parse_imgt_fasta_headers(names(J_alleles))
  dim(parsed_J_headers)    # 27 x 15 matrix
  parsed_J_headers[1:4, ]  # first 4 rows

  ## Codon start is the start position of the first codon:
  as.integer(parsed_J_headers[ , "codon_start"])

  ## Remove temporary directory:
  unlink(destdir, recursive=TRUE)
}
```

`percent_mutation` *Computes percent mutation in V, D, J segments*

Description

Computes the percent mutation in the V, D, J segments of a set of BCR or TCR sequences, at the nucleotide or amino acid levels.

Usage

```
percent_mutation(AIRR_df, for.aa=FALSE, as.matrix=FALSE)
```

Arguments

AIRR_df	The AIRR-formatted data.frame or tibble returned by igblastn().
for.aa	By default, the percent mutation is computed at the nucleotide level. Set for.aa to TRUE to compute it at the amino acid level.
as.matrix	By default, the function returns the results in a data.frame. Set as.matrix to TRUE to get the results in a matrix.

Details

The percent mutation in the V segment is computed as follow.

For each sequence in AIRR_df:

1. First the number of mutations is obtained by computing the Hamming distance d between the string reported in the v_sequence_alignment column and the string reported in the v_germline_alignment column. Note that the two strings are guaranteed to *always* have the same length L.
2. The percent mutation is 100 * d / L.

If for.aa is set to TRUE, then replace v_sequence_alignment and v_germline_alignment with v_sequence_alignment_aa and v_germline_alignment_aa above.

The percent mutation in the D and J segments is computed in the same manner using the corresponding d_* and j_* columns.

Value

By default, the function returns a data.frame or tibble with 1 row per row in AIRR_df, and with the following columns:

1. sequence_id: the sequence_id column taken as-is from AIRR_df;
2. locus: the locus column taken as-is from AIRR_df;
3. v_perc_mut: the percent mutation in the V segment of the BCR or TCR sequences;
4. d_perc_mut: the percent mutation in the D segment of the BCR or TCR sequences;
5. j_perc_mut: the percent mutation in the J segment of the BCR or TCR sequences.

If for.aa is set to TRUE, then columns 3 to 5 are named v_perc_mut_aa, d_perc_mut_aa, and j_perc_mut_aa.

If as.matrix is set to TRUE, then the function returns a 3-column numeric matrix with 1 row per row in AIRR_df. The rownames on the matrix are the sequence ids from AIRR_df$sequence_id. The colnames on the matrix are [vdj]_perc_mut if for.aa is FALSE, or [vdj]_perc_mut_aa if it's TRUE.

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.
- tibble objects implemented in the **tibble** package.

Examples

```
if (!has_igblast()) install_igblast()

query <- system.file(package="igblastr", "extdata",
                     "BCR", "heavy_sequences.fasta")
use_germline_db("_OGRDB.human.IGH+IGK+IGL.202605")

AIRR_df <- igblastn(query)

## Percent mutation in V, D, J segments at the amino acid level:
percent_mutation(AIRR_df, for.aa=TRUE)

## Get the results in a 3-column matrix:
m1 <- percent_mutation(AIRR_df, for.aa=TRUE, as.matrix=TRUE)
m1[1:4, ]
colMeans(m1)

## Percent mutation in V, D, J segments at the nucleotide level:
percent_mutation(AIRR_df)
m2 <- percent_mutation(AIRR_df, as.matrix=TRUE)
m2[1:4, ]
colMeans(m2)

## Note that columns 'v_identity', 'd_identity', and 'j_identity'
## in 'AIRR_df' are equal to '1 - v_perc_mut', '1 - d_perc_mut',
## and '1- j_perc_mut', respectively:
ident <- AIRR_df[ , c("v_identity", "d_identity", "j_identity")]
ident

## Sanity check:
m <- as.matrix(ident)
stopifnot(max((m + m2 - 100)^2, na.rm=TRUE) < 1e-6)
```

---

read_igblastn_AIRR_output

igblastn output format 19 (AIRR format)

Description

Read igblastn output format 19 (AIRR format). This is the output produced by igblastn when outfmt is set to "AIRR" or 19.

This format is sometimes called "Rearrangement summary report" or simply "AIRR rearrangement tabular" format. See for example IgBLAST web interface at https://www.ncbi.nlm.nih.gov/igblast/.

Usage

```
read_igblastn_AIRR_output(out)
```

Arguments

out	The path to a file containing the output produced by igblastn when outfmt is set to "AIRR" or 19.

Value

A data.frame with 1 row per query sequence and many columns.

The columns are standard "Rearrangement Schema" fields, which are defined and documented by the AIRR Community at https://docs.airr-community.org/en/latest/datarep/rearrangements.html#fields

See Also

- The "Rearrangement Schema" at https://docs.airr-community.org/en/latest/datarep/rearrangements.html
- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- read_igblastn_fmt7_output to read and parse igblastn output format 7.
- IgBLAST web interface at https://www.ncbi.nlm.nih.gov/igblast/.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## ------------------------------------------------------------------------
## Access query sequences and select germline and C-region dbs to use
## ------------------------------------------------------------------------

## Files 'heavy_sequences.fasta' and 'light_sequences.fasta' included
## in igblastr contain 250 paired heavy- and light- chain sequences (125
## sequences in each file) downloaded from OAS (the Observed Antibody
## Space database):
filenames <- paste0(c("heavy", "light"), "_sequences.fasta")
query <- system.file(package="igblastr", "extdata", "BCR", filenames)

## Keep only the first 10 sequences from each file:
query <- c(head(readDNAStringSet(query[[1L]]), n=10),
           head(readDNAStringSet(query[[2L]]), n=10))

## Select the germline and C-region dbs to use with igblastn():
```

```
use_germline_db("_OGRDB.human.IGH+IGK+IGL.202605")
use_c_region_db("_IMGT.human.IGH+IGK+IGL.202605")

## ------------------------------------------------------------------------
## Call igblastn()
## ------------------------------------------------------------------------

out <- tempfile()
igblastn(query, out=out)
AIRR_df <- read_igblastn_AIRR_output(out)

class(AIRR_df)
dim(AIRR_df)  # 1 row per query sequence

tibble(AIRR_df)
```

`read_igblastn_fmt7_output` *igblastn output format 7*

Description

Read and parse `igblastn` output format 7. This is the output produced by `igblastn` when `outfmt` is set to 7.

This format is sometimes called "Tabular with comment lines" or simply "Tabular" format. See for example IgBLAST web interface at https://www.ncbi.nlm.nih.gov/igblast/.

Usage

```
read_igblastn_fmt7_output(out)

## Related utilities:
parse_outfmt7(out_lines)
list_outfmt7_specifiers()
```

Arguments

`out`	The path to a file containing the output produced by `igblastn` when `outfmt` is set to 7.
`out_lines`	The character vector returned by `igblatsn(query, outfmt=7, parse.out=FALSE, ...)`.

Value

`read_igblastn_fmt7_output(out)` returns `parse_outfmt7(readLines(out))`.

`parse_outfmt7(out_lines)` returns the parsed form of `out_lines` in a list.

`list_outfmt7_specifiers()` returns the list of format specifiers supported by `igblastn` formatting option 7.

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- read_igblastn_AIRR_output to read igblastn output format 19 (AIRR format).
- IgBLAST web interface at https://www.ncbi.nlm.nih.gov/igblast/.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

## ------------------------------------------------------------------------
## Access query sequences and select germline and C-region dbs to use
## ------------------------------------------------------------------------

## Files 'heavy_sequences.fasta' and 'light_sequences.fasta' included
## in igblastr contain 250 paired heavy- and light- chain sequences (125
## sequences in each file) downloaded from OAS (the Observed Antibody
## Space database):
filenames <- paste0(c("heavy", "light"), "_sequences.fasta")
query <- system.file(package="igblastr", "extdata", "BCR", filenames)

## Keep only the first 10 sequences from each file:
query <- c(head(readDNAStringSet(query[[1L]]), n=10),
           head(readDNAStringSet(query[[2L]]), n=10))

## Select the germline and C-region dbs to use with igblastn():
use_germline_db("_OGRDB.human.IGH+IGK+IGL.202605")
use_c_region_db("_IMGT.human.IGH+IGK+IGL.202605")

## ------------------------------------------------------------------------
## FIRST igblastn RUN: GET OUTPUT IN FORMAT 7
## ------------------------------------------------------------------------

parsed_out7 <- igblastn(query, outfmt=7)

## Note that the above is equivalent to:
out <- tempfile()
igblastn(query, outfmt=7, out=out)
parsed_out7b <- read_igblastn_fmt7_output(out)
stopifnot(identical(parsed_out7b, parsed_out7))

## and to:
out_lines <- igblastn(query, outfmt=7, parse.out=FALSE)
out_lines  # raw output
parsed_out7c <- parse_outfmt7(out_lines)
stopifnot(identical(parsed_out7c, parsed_out7))

## Now taking a closer look at the output...

## Output contains one record per query sequence:
```

```
length(parsed_out7$records)  # 20

## Each record can have 5 or 6 sections:
##   1. query_details
##   2. VDJ_rearrangement_summary
##   3. VDJ_junction_details
##   4. subregion_sequence_details (can be missing)
##   5. alignment_summary
##   6. hit_table

## Taking a close look at the first record:
rec1 <- parsed_out7$records[[1]]
rec1

qseqid(rec1)    # query sequence id associated with this record

rec1$hit_table  # data.frame with the standard columns

## ---------------------------------------------------------------------
## SECOND igblastn RUN: GET OUTPUT IN CUSTOMIZED FORMAT 7
## ---------------------------------------------------------------------

## For this second run we request a customized format 7 by supplying
## space delimited format specifiers. Use list_outfmt7_specifiers() to
## get the list of format specifiers supported by igblastn formatting
## option 7:
list_outfmt7_specifiers()
outfmt <- "7 qseqid sseqid pident nident length score"
parsed_out7 <- igblastn(query, outfmt=outfmt)

## Taking a close look at the first record:
rec1 <- parsed_out7$records[[1]]
rec1$hit_table  # data.frame with the requested columns (+ the
                # automatic "chaintype" column)
```

`reset_c_region_dbs` *Reset the cached C-region dbs*

Description

`reset_c_region_dbs()` will reset the cache of C-region dbs to a pristine state by:

1. Removing *all* the C-region dbs currently in **igblastr**'s persistent cache, including the built-in C-region dbs.
2. Recreating the predefined set of built-in C-region dbs.

This means that any non built-in C-region db present in the cache WILL BE LOST!

Also note that `reset_c_region_dbs()` will cancel the current selection if any C-region db was previously selected with `use_c_region_db()`.

Usage

```
reset_c_region_dbs(verbose=FALSE)
```

Arguments

verbose	Set to TRUE to have reset_c_region_dbs() display some details about its internal operations.

Value

Nothing (i.e. an invisible NULL).

See Also

- list_c_region_dbs to list the cached C-region dbs.
- use_c_region_db to select the cached C-region db to use with igblastn().
- install_IMGT_c_region_db to install a C-region db from IMGT.

Examples

```
## DANGER ZONE!
## Not run:
  reset_c_region_dbs(verbose=TRUE)

## End(Not run)
```

---

reset_germline_dbs *Reset the cached germline dbs*

Description

reset_germline_dbs() will reset the cache of germline dbs to a pristine state by:

1. Removing *all* the germline dbs currently in **igblastr**'s persistent cache, including the built-in germline dbs.
2. Recreating the predefined set of built-in germline dbs.

This means that any non built-in germline db present in the cache WILL BE LOST!

Also note that reset_germline_dbs() will cancel the current selection if any germline db was previously selected with use_germline_db().

Usage

```
reset_germline_dbs(verbose=FALSE)
```

Arguments

verbose	Set to TRUE to have reset_germline_dbs() display some details about its internal operations.

Value

Nothing (i.e. an invisible NULL).

See Also

- list_germline_dbs to list the cached germline dbs.
- use_germline_db to select the cached germline db to use with igblastn().
- install_IMGT_germline_db to install a germline db from IMGT.
- install_custom_germline_db to install a germline db from user-supplied gene allele sequences.

Examples

```
## DANGER ZONE!
## Not run:
  reset_germline_dbs(verbose=TRUE)

## End(Not run)
```

---

summarizeMismatches *Summarize mismatches and indels between query and germline sequences*

Description

TODO

---

translate_codons *Extract and translate codons from a set of DNA sequences*

Description

The translate_codons() function extracts and translates codons from a set of DNA sequences.

Usage

```
translate_codons(dna, offset=0, with.init.codon=FALSE)

## Used internally by translate_codons():
extract_codons(dna, offset=0)
remove_gaps(dna, gap_letter=".")
```

Arguments

dna	A DNAStringSet (or DNAString) object containing the codons to translate. Note that if the sequences in dna contain gaps (represented by the . letter) then translate_codons() and extract_codons() will remove them (by calling remove_gaps() internally) before doing anything else.
offset	The number of nucleotides that precede the first codon to translate. This must be supplied as a numeric vector with one value per sequence in dna, or as a single value. If the latter, then the same offset is used for all sequences.
with.init.codon	Is the first codon to translate in each DNA sequence the initiation codon? By default, with.init.codon is set to FALSE, in which case translate_codons() assumes that the first codon to translate in each DNA sequence is *not* the initiation codon. See documentation of the no.init.codon argument in ?translate in the **Biostrings** package for more information.
gap_letter	The letter or symbol representing gaps. Note that IMGT and AIRR-community/OGRDB typically use dots (".") to represent gaps in germline V gene allele sequences.

Value

translate_codons() returns:

- An AAStringSet object with one amino acid sequence per input sequence if a DNAStringSet object was supplied.
- An AAString object if a DNAString object was supplied.

extract_codons() returns:

- A DNAStringSet object with one sequence per input sequence if a DNAStringSet object was supplied.
- A DNAString object if a DNAString object was supplied.

The sequences returned by extract_codons() are obtained by (1) removing all gaps (i.e. all . letters) from the input sequences, and (2) trimming the gap-free sequences as follow:

- On their 5' end, sequences are trimmed by the amount of nucleotides specified in offset.
- On their 3' end, sequences are trimmed by the smallest amount of nucleotides that makes the length of the trimmed sequence a multiple of 3. Note that this will always be 0, 1, or 2 nucleotides.

remove_gaps() returns the gap-free versions of the input sequences in an object of the same type as the input object.

See Also

- DNAStringSet and AAStringSet objects in the **Biostrings** package.
- The translate function in the **Biostrings** package upon which translate_codons() is based.
- list_germline_dbs to list the cached germline dbs.

Examples

```
## ---------------------------------------------------------------------
## translate_codons()
## ---------------------------------------------------------------------

## Gaps will be ignored:
dna1 <- DNAStringSet(c(".TTGTCCTTTAT..A", "GAAT.CATTTATC", "CTGTCGTTTATT"))
translate_codons(dna1)

## Handling of initiation codons (see '?translate' in the Biostrings
## package for how initiation codons are handled):
translate_codons(dna1, with.init.codon=TRUE)

## Load germline V gene allele sequences for human:
list_germline_dbs()
db_name <- "_OGRDB.human.IGH+IGK+IGL.202605"
V_alleles <- load_germline_sequences(db_name, region_types="V")
V_alleles  # DNAStringSet object

## Translate them:
V_aa <- translate_codons(V_alleles)
V_aa  # AAStringSet object

## Some human germline V gene allele sequences have a stop codon:
has_stop_codon <- grepl("*", as.character(V_aa), fixed=TRUE)
V_aa[has_stop_codon]

## ---------------------------------------------------------------------
## extract_codons()
## ---------------------------------------------------------------------
extract_codons(dna1)
extract_codons(dna1, offset=1)

dna2 <- DNAStringSet(c("CCCAAAGGGTTT",
                       "CCAAAGGGTTT",
                       "CAAAGGGTTT",
                       "AAAGGGTTT"))
extract_codons(dna2)
extract_codons(dna2, offset=1)
extract_codons(dna2, offset=2)
extract_codons(dna2, offset=3)
extract_codons(dna2, offset=4)

extract_codons(dna2, offset=3:0)
extract_codons(dna2, offset=4:1)
extract_codons(dna2, offset=5:2)
extract_codons(dna2, offset=6:3)

## ---------------------------------------------------------------------
## remove_gaps()
## ---------------------------------------------------------------------
remove_gaps(dna1)
```

update_live_igdata *Update and manage IgBLAST auxiliary and internal data*

Description

A small set of low-level utility functions to update and manage IgBLAST *auxiliary data* and *internal data*.

Usage

```
update_live_igdata(check.only=FALSE)

igdata_info()

time_since_live_igdata_last_checked(units="days")

reset_live_igdata(subdirs=c("all", "internal_data", "optional_file"))
```

Arguments

check.only	By default, update_live_igdata() checks for new IgBLAST auxiliary or internal data files available at NCBI, and it installs them if any are found. Set check.only to TRUE to only do the check without installing anything.
units	See ?base::difftime for valid units.
subdirs	By default, reset_live_igdata() resets both internal_data/ and optional_file/ directories to their original states. Set subdirs to "internal_data" or "optional_file" to reset only a particular directory.

Details

Auxiliary data and internal data: A standard IgBLAST installation – like the one used by the **igblastr** package – typically includes *auxiliary data* and *internal data* that are normally found in directories internal_data/ and optional_file/, respectively. Both directories should be subdirectories of the *root directory* of the IgBLAST installation, that is, of the directory returned by get_igblast_root().

We sometimes refer to this data simply as the *IgBLAST data*.

NCBI updates: NCBI occasionally updates some of the files in the internal_data/ and optional_file/ directories between IgBLAST releases, and it is recommended to use the new files. They make the new files available at https://ftp.ncbi.nih.gov/blast/executables/igblast/release/patch/.

To download and install these new files, simply call update_live_igdata(). This will check for new IgBLAST auxiliary or internal data files available at NCBI, and install them if any are found.

You can restore the original files at any moment with reset_live_igdata().

Value

`update_live_igdata()` returns an invisible `TRUE` or `FALSE` that indicates whether updates were found (and applied) or not.

`igdata_info()` returns a named list containing information about the state of the "live" and "original" IgBLAST data.

`time_since_live_igdata_last_checked()` returns the time passed since the last run of `update_live_igdata()` in the specified units (days by default).

`update_live_igdata()` doesn't return anything (invisible `NULL`).

See Also

- intdata_utils to access IgBLAST *internal data*.
- auxdata_utils to access IgBLAST *auxiliary data*.
- The `igblastn` function to run the `igblastn` *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- `install_igblast` to perform an *internal* IgBLAST installation.
- `get_igblast_root` to get (or set) the IgBLAST installation used (or to be used) by the **igblastr** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

igblast_info()

## ------------------------------------------------------------------------
## Check for NCBI updates
## ------------------------------------------------------------------------

igdata_info()

ok <- update_live_igdata(check.only=TRUE)
ok  # TRUE if updates were found, FALSE if not

igdata_info()

## ------------------------------------------------------------------------
## "live" vs "original" IgBLAST data
## ------------------------------------------------------------------------

## By default, the "live IgBLAST data" gets accessed or returned:
get_auxdata_path("human")
live_human_auxdata <- load_auxdata("human")

## Access the original IgBLAST data:
get_auxdata_path("human", which="original")
orig_human_auxdata <- load_auxdata("human", which="original")
```

```
## "live" and "original" IgBLAST data can differ if the former was
## updated with update_live_igdata(). Otherwise, they'll be the same:
identical(live_human_auxdata, orig_human_auxdata)
```

---

use_c_region_db *Select cached C-region db to use with igblastn()*

Description

use_c_region_db() allows the user to select the cached C-region db to use with igblastn(). This choice will be remembered for the duration of the current R session but can be changed anytime.

load_c_region_sequences() allows the user to load the nucleotide sequences of the gene alleles stored in a cached C-region db.

Usage

```
use_c_region_db(db_name=NULL, verbose=FALSE)

load_c_region_sequences(db_name)
```

Arguments

db_name	For use_c_region_db(): NULL or a single string specifying the name of the cached C-region db to use. Use list_c_region_dbs() get the list of all cached C-region dbs. If set to NULL (the default), then use_c_region_db() returns the name of the cached C-region db that is currently in use, if any. Otherwise it returns the empty string (""). Note that the current selection can be cancelled with use_c_region_db(""). For load_c_region_sequences(): A single string specifying the name of the cached C-region db from which to load the allele sequences. Use list_c_region_dbs() to get the list of all cached C-region dbs.
verbose	If set to TRUE, then use_c_region_db() will display some information about its internal operations.

Value

When called with no argument, use_c_region_db() returns a single string containing the name of the cached C-region db currently used by igblastn() if any, or the empty string ("") if igblastn() is not using any C-region db.

When called with the db_name argument, use_c_region_db(db_name) returns db_name invisibly.

load_c_region_sequences() returns the allele sequences from the specified C-region db in a named DNAStringSet object.

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- list_c_region_dbs to list the cached C-region dbs.
- use_germline_db to select the cached germline db to use with igblastn().
- DNAStringSet objects in the **Biostrings** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

list_c_region_dbs()  # all cached C-region dbs

## Select C-region db to use with igblastn():
db_name <- "_IMGT.human.IGH+IGK+IGL.202605"
use_c_region_db(db_name)

## Get current selection:
use_c_region_db()    # get current selection

## Cancel current selection:
use_c_region_db("")
use_c_region_db()

## Load C-region sequences:
load_c_region_sequences(db_name)
load_c_region_sequences("_IMGT.mouse.IGH+IGK+IGL.202605")
```

---

use_germline_dbs *Select cached germline db to use with igblastn()*

Description

use_germline_db() allows the user to select the cached germline db to use with igblastn(). This choice will be remembered for the duration of the current R session but can be changed anytime.

load_germline_sequences() allows the user to load the nucleotide sequences of the gene alleles stored in a cached germline db.

Usage

```
use_germline_db(db_name=NULL, verbose=FALSE)

load_germline_sequences(db_name, region_types=NULL)
```

Arguments

db_name	For use_germline_db(): NULL or a single string specifying the name of the cached germline db to use. Use list_germline_dbs() to get the list of all cached germline dbs. If set to NULL (the default), then use_germline_db() returns the name of the cached germline db that is currently in use, if any. Otherwise it raises an error. For load_germline_sequences(): A single string specifying the name of the cached germline db from which to load the V, D, and/or J allele sequences. Use list_germline_dbs() to get the list of all cached germline dbs.
verbose	If set to TRUE, then use_germline_db() will display some information about its internal operations.
region_types	The types of regions (V, D, and/or J) to load from the database. Specified as a single string (e.g. "DJ") or as a character vector of single-letter elements (e.g. c("D", "J")). By default (i.e. when region_types is NULL), all the regions are returned.

Value

When called with no argument, use_germline_db() returns a single string containing the name of the cached germline db currently used by igblastn() if any, or it raises an error if no germline db has been selected yet.

When called with the db_name argument, use_germline_db(db_name) returns db_name invisibly.

load_germline_sequences() returns the allele sequences from the specified germline db in a named DNAStringSet object.

See Also

- The igblastn function to run the igblastn *standalone executable* included in IgBLAST from R. This is the main function in the **igblastr** package.
- list_germline_dbs to list the cached germline dbs.
- use_c_region_db to select the cached C-region db to use with igblastn().
- DNAStringSet objects in the **Biostrings** package.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
if (!has_igblast()) install_igblast()

list_germline_dbs()  # all cached germline dbs

## Select germline db to use with igblastn():
db_name <- "_OGRDB.human.IGH+IGK+IGL.202605"
use_germline_db(db_name)

## Get current selection:
```

```
use_germline_db()

## Load germline gene allele sequences:
load_germline_sequences(db_name)
load_germline_sequences(db_name, region_types="D")
load_germline_sequences(db_name, region_types="DJ")
```

`V_alleles-inspect` *Basic inspection of V allele sequences*

Description

A small set of utilities for (very) basic inspection of V allele sequences.

Note that all of them require access to the *internal data* associated with the V alleles to inspect. See `intdata` argument below for how to obtain this data.

Usage

```
translate_V_alleles(V_alleles, intdata, V_segment=NULL)
V_allele_has_stop_codon(V_alleles, intdata)
```

Arguments

`V_alleles`	A DNAStringSet object containing germline V gene allele sequences.
`intdata`	A data.frame as returned by `load_intdata(..., for.aa=FALSE)` that contains annotations for the alleles in `V_alleles`.
`V_segment`	The name of a V gene segment. This can be set to `"fwr1"`, `"cdr1"`, `"fwr2"`, `"cdr2"`, or `"fwr3"`. By default `translate_V_alleles()` will translate the entire coding frame in each allele. Otherwise, it will translate the specified segment only.

Value

`translate_V_alleles()` returns a named character vector with 1 amino acid sequence per supplied allele. The vector contains an `NA` for any allele that is not annotated in `intdata` or for which the required information is `NA`. The names on it are the names of the supplied alleles.

`V_allele_has_stop_codon()` returns a named logical vector with 1 value per supplied allele. The vector contains an `NA` for any allele that is not annotated in `intdata` or for which `intdata$coding_frame_start` has an `NA`. The names on it are the names of the supplied alleles.

See Also

- intdata_utils to access IgBLAST *internal data*.
- auxdata_utils to access IgBLAST *auxiliary data*.
- J_alleles_inspect for basic inspection of J allele sequences.
- `update_live_igdata` for more information about "live" and "original" IgBLAST data.

- DNAStringSet objects in the **Biostrings** package.
- The translate_codons function upon which translate_V_alleles() is based.
- allele2gene to go from germline gene allele names to germline gene names.
- IgBLAST is described at https://pubmed.ncbi.nlm.nih.gov/23671333/.

Examples

```
## Let's inspect the V allele sequences stored in
## germline db _OGRDB.human.IGH+IGK+IGL.202605.

## Load the V allele sequences:
db_name <- "_OGRDB.human.IGH+IGK+IGL.202605"
V_alleles <- load_germline_sequences(db_name, region_types="V")
V_alleles  # DNAStringSet object

## Load the corresponding internal data:
intdata <- load_intdata(db_name)

## Translate the V allele sequences:
V_aa <- translate_V_alleles(V_alleles, intdata)
head(V_aa)

## Translate the FWR2 portion of the V allele sequences:
fwr2_aa <- translate_V_alleles(V_alleles, intdata, V_segment="fwr2")
head(fwr2_aa)

## Surprisingly, 13 V alleles in _OGRDB.human.IGH+IGK+IGL.202605
## contain the stop codon:
has_stop_codon <- grepl("*", V_aa, fixed=TRUE)
table(has_stop_codon)

## Display the V allele sequences with stop codon:
V_aa[has_stop_codon]       # protein
V_alleles[has_stop_codon]  # nucleotides
```

Index

∗ **manip**
igblastn, 37
read_igblastn_AIRR_output, 82
read_igblastn_fmt7_output, 84
∗ **misc**
IGBLAST_ROOT, 36
∗ **utilities**
allele2gene, 3
augment_germline_db, 4
auxdata-IO, 8
auxdata-utils, 9
combine_germline_dbs, 12
compute_auxdata, 13
compute_V_gene_delineations, 20
download_IMGT_germline_sequences, 24
download_OGRDB_germline_json, 26
download_OGRDB_germline_sequences, 30
get_igblast_root, 33
igblast_info, 34
igblastr_usage_report, 44
igbrowser, 45
install_custom_germline_db, 46
install_igblast, 54
install_IMGT_germline_db, 55
intdata-utils, 59
J_alleles-inspect, 62
list_c_region_dbs, 68
list_germline_dbs, 69
ndm_data-IO, 72
OAS-utils, 74
parse_imgt_fasta_headers, 77
percent_mutation, 80
read_igblastn_AIRR_output, 82
read_igblastn_fmt7_output, 84
reset_c_region_dbs, 86
reset_germline_dbs, 87
summarizeMismatches, 88
translate_codons, 88
update_live_igdata, 91
use_c_region_db, 93
use_germline_dbs, 94
V_alleles-inspect, 96

AAString, *89*
AAStringSet, *17*, *64*, *89*
allele2gene, 3, *97*
augment_germline_db, 4
augment_germline_db_D *(*augment_germline_db*)*, 4
augment_germline_db_J *(*augment_germline_db*)*, 4
augment_germline_db_V *(*augment_germline_db*)*, 4
auxdata *(*auxdata-utils*)*, 9
auxdata-IO, 8
auxdata-utils, 9
auxdata_IO *(*auxdata-IO*)*, 8
auxdata_utils, *9*, *28*, *36*, *61*, *64*, *92*, *96*
auxdata_utils *(*auxdata-utils*)*, 9

bcr_browser *(*igbrowser*)*, 45
BLAST_USAGE_REPORT *(*igblastr_usage_report*)*, 44
browseURL, *46*

combine_c_region_dbs *(*combine_germline_dbs*)*, 12
combine_germline_dbs, *4, 5*, 12
compute_auxdata, *9–11*, 13, *14*, *22*, *28*, *63*
compute_germline_db_auxdata, *63*
compute_germline_db_auxdata *(*auxdata-utils*)*, 9
compute_imgt_intdata *(*compute_V_gene_delineations*)*, 20
compute_V_gene_delineations, *17*, 20, *28*, *61*, *73*, *78*

DataFrame, *63*
difftime, *91*
DNAString, *89*
DNAStringSet, *5*, *14*, *17*, *21*, *22*, *37*, *41*, *63*, *64*, *75*, *76*, *89*, *93–97*
download.file, *25*, *28*, *32*, *55*, *56*, *74*
download_IMGT_germline_sequences, 24, *32*, *50*, *56*, *78*
download_OGRDB_germline_json, 26, *32*
download_OGRDB_germline_sequences, *26*, *28*, 30, *50*
download_paired_OAS_units *(*OAS-utils*)*, 74

extract_auxdata_from_ogrdb_json, *9*
extract_auxdata_from_ogrdb_json *(*download_OGRDB_germline_json*)*, 26
extract_codons *(*translate_codons*)*, 88
extract_intdata_from_ogrdb_json, *73*
extract_intdata_from_ogrdb_json *(*download_OGRDB_germline_json*)*, 26
extract_metadata_from_OAS_units *(*OAS-utils*)*, 74
extract_sequences_from_paired_OAS_df *(*OAS-utils*)*, 74
extract_sequences_from_paired_OAS_units *(*OAS-utils*)*, 74

fmt7-utils *(*read_igblastn_fmt7_output*)*, 84
fmt7_utils *(*read_igblastn_fmt7_output*)*, 84

get_auxdata_path *(*auxdata-utils*)*, 9
get_igblast_auxiliary_data *(*auxdata-utils*)*, 9
get_igblast_root, 33, *35*, *36*, *91*, *92*
get_intdata_path *(*intdata-utils*)*, 59

has_igblast *(*igblast_info*)*, 34

igblast_build *(*igblast_info*)*, 34
igblast_info, 34, *34*, *37*, *41*, *55*
IGBLAST_ROOT, *34*, 36, *36*, *55*
igblastn, *5*, *9–13*, *17*, *22*, *34*, *36*, 37, *37*, *45–48*, *50*, *55–57*, *61*, *68–71*, *73*, *76*, *81–83*, *85*, *92–95*
igblastn_help *(*igblastn*)*, 37
igblastn_version *(*igblast_info*)*, 34
igblastr_usage_report, *41*, 44
igbrowser, *41*, 45
igdata_info *(*update_live_igdata*)*, 91
IMGT_is_up *(*download_IMGT_germline_sequences*)*, 24
install_custom_germline_db, *26*, *32*, *41*, 46, *56*, *57*, *71*, *88*
install_igblast, *34*, *36*, *37*, *41*, 54, *92*
install_IMGT_c_region_db, *87*
install_IMGT_c_region_db *(*install_IMGT_germline_db*)*, 55
install_IMGT_germline_db, *13*, *26*, *41*, *50*, 55, *71*, *88*
intdata *(*intdata-utils*)*, 59
intdata-utils, 59
intdata_utils, *4*, *11*, *28*, *36*, *48*, *56*, *64*, *72*, *73*, *92*, *96*
intdata_utils *(*intdata-utils*)*, 59
IRangesList, *21*

J_allele_has_stop_codon *(*J_alleles-inspect*)*, 62
J_alleles-inspect, 62
J_alleles_inspect, *96*
J_alleles_inspect *(*J_alleles-inspect*)*, 62

list_c_region_dbs, *13*, 68, *71*, *87*, *93*, *94*
list_germline_dbs, *5*, *10*, *11*, *13*, *39*, *48*, *50*, *56*, *57*, *60*, *61*, *63*, 69, *69*, *88*, *89*, *95*
list_igblast_organisms, *10*, *11*, *38*, *41*, *49*, *60*, *61*, *63*
list_igblast_organisms *(*igblast_info*)*, 34
list_IMGT_organisms *(*download_IMGT_germline_sequences*)*, 24
list_IMGT_releases, *57*
list_IMGT_releases *(*download_IMGT_germline_sequences*)*, 24
list_outfmt7_specifiers, *38*, *41*
list_outfmt7_specifiers *(*read_igblastn_fmt7_output*)*, 84
list_paired_OAS_studies *(*OAS-utils*)*, 74
list_paired_OAS_units *(*OAS-utils*)*, 74

live_igdata (update_live_igdata), 91
load_auxdata, *13*, *17*, *28*, *49*, *63*
load_auxdata (auxdata-utils), 9
load_c_region_db (use_c_region_db), 93
load_c_region_sequences, *4*
load_c_region_sequences (use_c_region_db), 93
load_germline_db (use_germline_dbs), 94
load_germline_sequences, *4*, *17*, *50*
load_germline_sequences (use_germline_dbs), 94
load_igblast_auxiliary_data (auxdata-utils), 9
load_intdata, *21*, *22*, *28*, *50*, *96*
load_intdata (intdata-utils), 59

makeblastdb_version (igblast_info), 34
mcols, *21*

ndm_data-IO, 72
ndm_data_IO (ndm_data-IO), 72

OAS-utils, 74
OAS_is_up (OAS-utils), 74
OAS_utils (OAS-utils), 74
outfmt7-utils (read_igblastn_fmt7_output), 84
outfmt7_utils (read_igblastn_fmt7_output), 84

parse_imgt_fasta_headers, 77
parse_outfmt7 (read_igblastn_fmt7_output), 84
percent_mutation, *41*, 80
print.alignment_summary (read_igblastn_fmt7_output), 84
print.auxdata_md5sum_df (update_live_igdata), 91
print.c_region_dbs_df (list_c_region_dbs), 68
print.fmt7footer (read_igblastn_fmt7_output), 84
print.fmt7record (read_igblastn_fmt7_output), 84
print.germline_dbs_df (list_germline_dbs), 69
print.hit_table (read_igblastn_fmt7_output), 84
print.igblast_info (igblast_info), 34
print.igblastn_raw_output (igblastn), 37
print.igdata_info (update_live_igdata), 91
print.outfmt7_specifiers (read_igblastn_fmt7_output), 84
print.query_details (read_igblastn_fmt7_output), 84
print.subregion_sequence_details (read_igblastn_fmt7_output), 84
print.VDJ_junction_details (read_igblastn_fmt7_output), 84
print.VDJ_rearrangement_summary (read_igblastn_fmt7_output), 84
print_J_alleles, *17*
print_J_alleles (J_alleles-inspect), 62
PWM, *16*

qseqid (read_igblastn_fmt7_output), 84

read.table, *74*
read_auxdata, *10*, *16*
read_auxdata (auxdata-IO), 8
read_igblastn_AIRR_output, 82, *85*
read_igblastn_fmt7_output, *40*, *83*, 84
read_ndm_data, *21*, *60*
read_ndm_data (ndm_data-IO), 72
read_OAS_csv (OAS-utils), 74
read_OAS_csv_metadata (OAS-utils), 74
remove_gaps (translate_codons), 88
reset_c_region_dbs, *69*, 86
reset_germline_dbs, *72*, 87
reset_live_igdata (update_live_igdata), 91
rm_c_region_db (list_c_region_dbs), 68
rm_germline_db (list_germline_dbs), 69
Rprofile, *44*

same_alleles_annot (ndm_data-IO), 72
set_igblast_root (get_igblast_root), 33
show_intdata_disagreements (intdata-utils), 59
solve_cdr3_ends_using_fwr4_aa_comparisons (compute_auxdata), 13
solve_cdr3_ends_using_fwr4_dna_comparisons (compute_auxdata), 13
solve_cdr3_ends_using_fwr4_dna_PWM (compute_auxdata), 13
summarizeMismatches, 88

summary.query_details
(read_igblastn_fmt7_output), 84

tabulate_deletions
(summarizeMismatches), 88
tabulate_insertions
(summarizeMismatches), 88
tabulate_mismatches
(summarizeMismatches), 88
tibble, *39–41*, *45, 46*, *74–76*, *81, 82*
time_since_live_igdata_last_checked
(update_live_igdata), 91
translate, *89*
translate_codons, *64*, 88, *97*
translate_fwr4 (J_alleles-inspect), 62
translate_J_alleles
(J_alleles-inspect), 62
translate_V_alleles
(V_alleles-inspect), 96

update_live_igdata, *10, 11*, *60, 61*, *64*, 91, *96*
Usage_report (igblastr_usage_report), 44
usage_report (igblastr_usage_report), 44
Usage_reporting
(igblastr_usage_report), 44
usage_reporting
(igblastr_usage_report), 44
use_c_region_db, *13*, *38*, *41*, *69*, *86, 87*, 93, *95*
use_germline_db, *13*, *38, 39*, *41*, *50*, *57*, *71*, *87, 88*, *94*
use_germline_db (use_germline_dbs), 94
use_germline_dbs, 94

V_allele_has_stop_codon
(V_alleles-inspect), 96
V_alleles-inspect, 96
V_alleles_inspect, *64*
V_alleles_inspect (V_alleles-inspect), 96
V_genes_with_varying_fwrcdr_boundaries
(intdata-utils), 59
validate_and_complete_auxdata_with_ref
(compute_auxdata), 13
validate_ndm_rows (ndm_data-IO), 72

write_auxdata, *28*
write_auxdata (auxdata-IO), 8
write_ndm_data, *28*
write_ndm_data (ndm_data-IO), 72