

# Package: GwasDataImport (via r-universe)

September 11, 2024

**Title** Import EBI Data to OpenGWAS

**Version** 0.1.2

**Description** Determine the new EBI data not present in OpenGWAS.  
Download dataset and import metadata. Upload processed data and metadata to OpenGWAS.

**License** MIT + file LICENSE

**Depends** R (>= 3.6.0)

**Imports** biomaRt, CheckSumStats, data.table, dplyr, GenomeInfoDb,  
GenomicRanges, glue, gwasvcf, httr, ieugwasr, IRanges,  
jsonlite, magrittr, R6, rtracklayer, SummarizedExperiment,  
tidyr

**Suggests** covr, knitr, R.utils, rmarkdown, testthat

**VignetteBuilder** knitr

**Remotes** github::MRCIEU/CheckSumStats, github::MRCIEU/gwasvcf,  
github::mrcieu/ieugwasr

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Repository** <https://mrcieu.r-universe.dev>

**RemoteUrl** <https://github.com/MRCIEU/GwasDataImport>

**RemoteRef** HEAD

**RemoteSha** 4981140f4a5221c99c6d2ae1f0ccbe0af304b653

## Contents

being_processed . . . . .	2
create_build_reference . . . . .	2
create_marts . . . . .	3
Dataset . . . . .	3

determine_build . . . . .	10
determine_build_biomart . . . . .	10
determine_build_position . . . . .	11
determine_new_datasets . . . . .	11
EbiDataset . . . . .	12
ebi_datasets . . . . .	15
get_ftp_path . . . . .	15
get_positions . . . . .	16
liftover_gwas . . . . .	16
listftp . . . . .	17

## Index 18

---

being_processed	<i>being_processed</i>
-----------------	------------------------

---

### Description

List of EBI datasets that are currently being processed

### Usage

being\_processed(dat)

### Arguments

dat                      Output from determine\_new\_datasets

### Value

Updated dat

---

create_build_reference	<i>Create dataset of some hm3 SNPs and their build positions</i>
------------------------	--

---

### Description

Create dataset of some hm3 SNPs and their build positions

### Usage

create\_build\_reference()

### Value

saves build\_ref object

---

create_marts	<i>Create marts for b36-38</i>
--------------	--------------------------------

---

**Description**

Slow to create these so just make once and save

**Usage**

```
create_marts()
```

**Value**

Saves data to data/marts.rdata

---

Dataset	<i>Object that downloads, develops and uploads GWAS summary datasets for IEU OpenGWAS database</i>
---------	--

---

**Description**

Object that downloads, develops and uploads GWAS summary datasets for IEU OpenGWAS database

Object that downloads, develops and uploads GWAS summary datasets for IEU OpenGWAS database

**Public fields**

filename Path to raw GWAS summary dataset

igd\_id ID to use for upload. If NULL then the next available ID in batch ieu-b will be used automatically

wd Work directory in which to save processed files. Will be deleted upon completion

gwas\_out path to processed summary file

nsnp\_read Number of SNPs read initially

nsnp Number of SNPs retained after reading

metadata List of meta-data entries

metadata\_test List of outputs from tests of the effect allele, effect allele frequency columns and summary data using CheckSumStats

metadata\_file Path to meta-data json file

datainfo List of GWAS file parameters

datainfo\_file Path to datainfo json file

params Initial column identifiers specified for raw dataset

metadata\_uploaded TRUE/FALSE of whether the metadata has been uploaded

gwasdata\_uploaded TRUE/FALSE of whether the gwas data has been uploaded

metadata\_upload\_status Response from server about upload process

gwasdata\_upload\_status Response from server about upload process

## Methods

### Public methods:

- `Dataset$new()`
- `Dataset$is_new_id()`
- `Dataset$delete_wd()`
- `Dataset$set_wd()`
- `Dataset$se_from_bp()`
- `Dataset$determine_columns()`
- `Dataset$format_dataset()`
- `Dataset$view_metadata_options()`
- `Dataset$get_gwasdata_fields()`
- `Dataset$get_metadata_fields()`
- `Dataset$collect_metadata()`
- `Dataset$check_meta_data()`
- `Dataset$write_metadata()`
- `Dataset$api_metadata_upload()`
- `Dataset$api_metadata_edit()`
- `Dataset$api_metadata_check()`
- `Dataset$api_metadata_delete()`
- `Dataset$api_gwasdata_upload()`
- `Dataset$api_gwasdata_check()`
- `Dataset$api_gwasdata_delete()`
- `Dataset$api_qc_status()`
- `Dataset$api_report()`
- `Dataset$api_gwas_release()`
- `Dataset$clone()`

**Method** `new()`: Initialise

*Usage:*

```
Dataset$new(filename = NULL, wd = tempdir(), igd_id = NULL)
```

*Arguments:*

`filename` Path to raw GWAS summary data file

`wd` Path to directory to use as the working directory. Will be deleted upon completion - best to keep as the default randomly generated temporary directory

`igd_id` Option to provide a specified ID for upload. If none provided then will use the next ieu-a batch ID

*Returns:* new ObtainEbiDataset object

**Method** `is_new_id()`: Check if the specified ID is unique within the database. It checks published GWASs and those currently being processed

*Usage:*

```
Dataset$is_new_id(id = self$igd_id)
```

*Arguments:*

id ID to check

**Method** delete\_wd(): Delete working directory

*Usage:*

```
Dataset$delete_wd()
```

**Method** set\_wd(): Set working directory (creates)

*Usage:*

```
Dataset$set_wd(wd)
```

*Arguments:*

wd working directory

**Method** se\_from\_bp(): Estimate standard error from beta and p-value

*Usage:*

```
Dataset$se_from_bp(beta, pval, minp = 1e-300)
```

*Arguments:*

beta Effect size

pval p-value

minp Minimum p-value cutoff default = 1e-300

**Method** determine\_columns(): Specify which columns in the dataset correspond to which fields.

*Usage:*

```
Dataset$determine_columns(params, nrows = 100, gwas_file = self$filename, ...)
```

*Arguments:*

params List of column identifiers. Identifiers can be numeric position or column header name.

Required columns are: c("chr\_col", "pos\_col", "ea\_col", "oa\_col", "beta\_col", "se\_col", "pval\_col", "rsid\_col"). Optional columns are: c("snp\_col", "eaf\_col", "oaf\_col", "ncase\_col", "imp\_z\_col", "imp\_info\_col", "ncontrol\_col").

nrows How many rows to read to check that parameters have been specified correctly

gwas\_file Filename to read

... Further arguments to pass to data.table::fread in order to correctly read the dataset

**Method** format\_dataset(): Process dataset ready for uploading. Determines build and lifts over to hg19/b37 if necessary.

*Usage:*

```
Dataset$format_dataset(
  gwas_file = self$filename,
  gwas_out = file.path(self$wd, "format.txt.gz"),
  params = self$params,
  metadata_test = self$metadata_test,
  ...
)
```

*Arguments:*

gwas\_file GWAS filename  
 gwas\_out Filename to save processed dataset to  
 params Column specifications (see `determine_columns` for more info)  
 metadata\_test List of outputs from tests of the effect allele, effect allele frequency columns  
 and summary data using `ChecksumStats`  
 ... Further arguments to pass to `data.table::fread` in order to correctly read the dataset

**Method** `view_metadata_options()`: View the specifications for available meta data fields, as taken from <http://gwas-api.mrcieu.ac.uk/docs>

*Usage:*

```
Dataset$view_metadata_options()
```

**Method** `get_gwasdata_fields()`: Get a list of GWAS data fields and whether or not they are required

*Usage:*

```
Dataset$get_gwasdata_fields()
```

*Returns:* data.frame

**Method** `get_metadata_fields()`: Get a list of metadata fields and whether or not they are required

*Usage:*

```
Dataset$get_metadata_fields()
```

*Returns:* data.frame

**Method** `collect_metadata()`: Input metadata

*Usage:*

```
Dataset$collect_metadata(metadata, igd_id = self$igd_id)
```

*Arguments:*

metadata List of meta-data fields and their values, see `view_metadata_options` for which fields  
 need to be inputted.  
 igd\_id ID to be used for uploading to the database

**Method** `check_meta_data()`: Check that the reported effect allele and effect allele frequency columns are correct.

*Usage:*

```

Dataset$check_meta_data(
  gwas_file = self$filename,
  params = self$params,
  metadata = self$metadata
)

```

*Arguments:*

gwas\_file Filename to read

params column names from x\$determine\_columns(). Required columns are: c("snp\_col", "ea\_col", "oa\_col", "eaf\_col" )  
metadata metadata from x\$collect\_metadata()

**Method write\_metadata():** Write meta data to json file

*Usage:*

```
Dataset$write_metadata(  
  metadata = self$metadata,  
  datainfo = self$datainfo,  
  outdir = self$wd  
)
```

*Arguments:*

metadata List of meta data fields and their values  
datainfo List of data column parameters  
outdir Output directory to write json files

**Method api\_metadata\_upload():** Upload meta data to API

*Usage:*

```
Dataset$api_metadata_upload(  
  metadata = self$metadata,  
  metadata_test = self$metadata_test,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

metadata List of meta data fields and their values  
metadata\_test List of outputs from tests of the effect allele, effect allele frequency columns  
and summary data using CheckSumStats  
access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method api\_metadata\_edit():** Upload meta data to API

*Usage:*

```
Dataset$api_metadata_edit(  
  metadata = self$metadata,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

metadata List of meta data fields and their values  
access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method api\_metadata\_check():** View meta-data

*Usage:*

```
Dataset$api_metadata_check(  
  id = self$igd_id,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

id ID to check

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_metadata_delete()`: Delete a dataset. This deletes the metadata AND any uploaded GWAS data (and related processing files)

*Usage:*

```
Dataset$api_metadata_delete(
  id = self$igd_id,
  access_token = ieugwasr::check_access_token()
)
```

*Arguments:*

id ID to delete

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_gwasdata_upload()`: Upload gwas dataset

*Usage:*

```
Dataset$api_gwasdata_upload(
  datainfo = self$datainfo,
  gwasfile = self$gwas_out,
  metadata_test = self$metadata_test,
  access_token = ieugwasr::check_access_token()
)
```

*Arguments:*

datainfo List of data column parameters

gwasfile Path to processed gwasfile

metadata\_test List of outputs from tests of the effect allele, effect allele frequency columns and summary data using CheckSumStats

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_gwasdata_check()`: Check status of API processing pipeline

*Usage:*

```
Dataset$api_gwasdata_check(
  id = self$igd_id,
  access_token = ieugwasr::check_access_token()
)
```

*Arguments:*

id ID to check

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_gwasdata_delete()`: Delete a dataset. This deletes the metadata AND any uploaded GWAS data (and related processing files)

*Usage:*



```
Dataset$api_gwasdata_delete(  
  id = self$igd_id,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

id ID to delete

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_qc_status()`: Check the status of the GWAS QC processing pipeline

*Usage:*

```
Dataset$api_qc_status(  
  id = self$igd_id,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

id ID to delete

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_report()`: View the html report for a processed dataset

*Usage:*

```
Dataset$api_report(  
  id = self$igd_id,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

id ID of report to view

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `api_gwas_release()`: Release a dataset

*Usage:*

```
Dataset$api_gwas_release(  
  comments = NULL,  
  passed_qc = "True",  
  id = self$igd_id,  
  access_token = ieugwasr::check_access_token()  
)
```

*Arguments:*

comments Optional comments to provide when uploading

passed\_qc True or False

id ID to release

access\_token Google OAuth2.0 token. See ieugwasr documentation for more info

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
Dataset$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

determine_build	<i>Determine build based on reference dataset</i>
-----------------	---

---

### Description

Determine build based on reference dataset

### Usage

```
determine_build(rsid, chr, pos, build = c(37, 38, 36), fallback = "position")
```

### Arguments

rsid	rsid
chr	chr
pos	pos
build	Builds to try e.g. c(37,38,36)
fallback	Whether to try "position" (fast) or "biomart" (more accurate if you have rsids) based approaches instead

### Value

build if detected, or dataframe of matches if not

---

determine_build_biomart	<i>Determines which build a set of SNPs are on</i>
-------------------------	--

---

### Description

Determines which build a set of SNPs are on

### Usage

```
determine_build_biomart(rsid, chr, pos, build = c(37, 38, 36))
```

**Arguments**

rsid	rsid
chr	chr
pos	pos
build	Builds to try e.g. c(37,38,36)

**Value**

build if detected, or dataframe of matches if not

---

determine\_build\_position  
*Determine build just from positions*

---

**Description**

A bit sketchy but computationally fast - just assumes that there will be at least 50x more position matches in the true build than either of the others.

**Usage**

```
determine_build_position(pos, build = c(37, 38, 36))
```

**Arguments**

pos	Vector of positions
build	c(37,38,36)

**Value**

build or if not determined then dataframe

---

determine\_new\_datasets  
*determine\_new\_datasets*

---

**Description**

Figure out which datasets are not present in the database

**Usage**

```
determine_new_datasets(
  ebi_ftp_url = options()$ebi_ftp_url,
  blacklist = NULL,
  exclude_multi_datasets = TRUE
)
```

**Arguments**

ebi\_ftp\_url      FTP url default=options()\$ebi\_ftp\_url  
 blacklist        List of EBI datasets to ignore default=NULL  
 exclude\_multi\_datasets  
                   If a EBI ID has more than one dataset then should it be ignored

**Value**

data frame

---

EbiDataset

*Object that downloads, develops and uploads EBI dataset*

---

**Description**

Object that downloads, develops and uploads EBI dataset

Object that downloads, develops and uploads EBI dataset

**Super class**

[GwasDataImport::Dataset](#) -> EbiDataset

**Public fields**

ebi\_id EBI ID to look for

traitname Name of trait

ftp\_path Path to files in EBI FTP

or\_flag TRUE/FALSE if had to convert OR to beta

gwas\_out1 Path to first look at EBI dataset

**Methods****Public methods:**

- `EbiDataset$new()`
- `EbiDataset$download_dataset()`
- `EbiDataset$format_ebi_dataset()`
- `EbiDataset$organise_metadata()`
- `EbiDataset$pipeline()`
- `EbiDataset$clone()`

**Method new():** Initialise object*Usage:*

```
EbiDataset$new(
  ebi_id,
  wd = tempdir(),
  ftp_path = NULL,
  igd_id = paste0("ebi-a-", ebi_id),
  traitname = NULL
)
```

*Arguments:*

`ebi_id` e.g. GCST005522

`wd` Directory in which to download and develop dataset. Default=tempdir(). Deleted automatically upon object removal

`ftp_path` Pre-specified path to data. Default=NULL

`igd_id` Defaults to "ebi-a-<ebi\_id>"

`traitname` Option to provide traitname of dataset

*Returns:* A new EbiDataset object

**Method download\_dataset():** Download*Usage:*

```
EbiDataset$download_dataset(
  ftp_path = self$ftp_path,
  ftp_url = options()$ebi_ftp_url,
  outdir = self$wd
)
```

*Arguments:*

`ftp_path` Pre-specified path to data. Default=self\$ftp\_path

`ftp_url` Default=options()\$ebi\_ftp\_url

`outdir` Default=self\$wd

**Method format\_ebi\_dataset():** organise data before formatting. This is slow but doesn't really matter*Usage:*

```
EbiDataset$format_ebi_dataset(
  filename = self$filename,
  output = file.path(self$wd, "step1.txt.gz")
)
```

*Arguments:*

filename Filename of GWAS dataset  
output Where to save formatted dataset

**Method** `organise_metadata()`: Download and parse metadata

*Usage:*

```
EbiDataset$organise_metadata(
  ebi_id = self$ebi_id,
  or_flag = self$or_flag,
  igd_id = self$igd_id,
  units = NULL,
  sex = "NA",
  category = "NA",
  subcategory = "NA",
  build = "HG19/GRCh37",
  group_name = "public",
  traitname = self$traitname
)
```

*Arguments:*

ebi\_id Default=self\$ebi\_id  
or\_flag Default=self\$or\_flag  
igd\_id Default=NULL  
units Default=NULL  
sex Default="NA"  
category Default="NA"  
subcategory Default="NA"  
build Default="HG19/GRCh37"  
group\_name Default="public"  
traitname Default=self\$traitname

**Method** `pipeline()`: Once initialised this function will string together everything i.e. downloading, processing and uploading

*Usage:*

```
EbiDataset$pipeline()
```

**Method** `clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
EbiDataset$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

---

ebi_datasets	<i>ebi_datasets</i>
--------------	---------------------

---

**Description**

Convert output from listftp into something that is easier to read

**Usage**

```
ebi_datasets(ebi_ftp_url = options()$ebi_ftp_url)
```

**Arguments**

ebi\_ftp\_url     EBI FTP default=options()\$ebi\_ftp\_url

**Value**

data frame

---

get_ftp_path	<i>Get harmonised file for specific EBI ID</i>
--------------	--

---

**Description**

Get harmonised file for specific EBI ID

**Usage**

```
get_ftp_path(ebi_id, ebi_ftp_url = options()$ebi_ftp_url)
```

**Arguments**

ebi\_id            EBI ID e.g. GCST000879  
ebi\_ftp\_url     EBI FTP default=options()\$ebi\_ftp\_url

**Value**

ftp path (excluding server)

---

get_positions	<i>Lookup positions for given rsids in particular build</i>
---------------	---

---

**Description**

Lookup positions for given rsids in particular build

**Usage**

```
get_positions(
  rsid,
  build = 37,
  method = c("opengwas", "biomart")[1],
  splitsize = 50000
)
```

**Arguments**

rsid	rsid
build	build (36, 37 default or 38)
method	"opengwas" (fastest) or "biomart"
splitsize	Default 50000

**Value**

data frame

---

liftover_gwas	<i>Liftover GWAS positions</i>
---------------	--------------------------------

---

**Description**

Determine GWAS build and liftover to required build

**Usage**

```
liftover_gwas(
  dat,
  build = c(37, 38, 36),
  to = 37,
  chr_col = "chr",
  pos_col = "pos",
  snp_col = "snp",
  ea_col = "ea",
  oa_col = "oa",
  build_fallback = "position"
)
```



**Arguments**

dat	Data frame with chr, pos, snp name, effect allele, non-effect allele columns
build	The possible builds to check data against Default = c(37,38,26)
to	Which build to lift over to. Default=37
chr_col	Name of chromosome column name. Required
pos_col	Name of position column name. Required
snp_col	Name of SNP column name. Optional. Uses less certain method of matching if not available
ea_col	Name of effect allele column name. Optional. Might lead to duplicated rows if not presented
oa_col	Name of other allele column name. Optional. Might lead to duplicated rows if not presented
build_fallback	Whether to try "position" (fast) or "biomart" (more accurate if you have rsids) based approaches instead

**Value**

Data frame

---

listftp	<i>listftp</i>
---------	----------------

---

**Description**

List all files on the EBI FTP server

**Usage**

```
listftp(url = options()$ebi_ftp_url, recursive = TRUE)
```

**Arguments**

url	FTP url to look up
recursive	If false then just the top directory, otherwise list recursively

**Value**

Vector of paths

# Index

being\_processed, [2](#)

create\_build\_reference, [2](#)

create\_marts, [3](#)

Dataset, [3](#)

determine\_build, [10](#)

determine\_build\_biomart, [10](#)

determine\_build\_position, [11](#)

determine\_new\_datasets, [11](#)

ebi\_datasets, [15](#)

EbiDataset, [12](#)

get\_ftp\_path, [15](#)

get\_positions, [16](#)

GwasDataImport::Dataset, [12](#)

liftover\_gwas, [16](#)

listftp, [17](#)