

# Package ‘GenomicDataCommons’

November 12, 2025

**Type** Package

**Title** NIH / NCI Genomic Data Commons Access

**Description** Programmatically access the NIH / NCI Genomic Data Commons RESTful service.

**Version** 1.34.1

**Date** 2025-11-07

**License** Artistic-2.0

**Depends** R (>= 4.1.0)

**Imports** stats, httr, xml2, jsonlite, utils, rlang, readr,  
GenomicRanges, IRanges, dplyr, rappdirs, tibble, tidyr

**Suggests** BiocStyle, knitr, rmarkdown, DT, testthat, listviewer,  
ggplot2, GenomicAlignments, Rsamtools, BiocParallel,  
TxDb.Hsapiens.UCSC.hg38.knownGene, VariantAnnotation, maftools,  
R.utils, data.table

**biocViews** DataImport, Sequencing

**URL** <https://bioconductor.org/packages/GenomicDataCommons>,  
<http://github.com/Bioconductor/GenomicDataCommons>,  
<http://bioconductor.github.io/GenomicDataCommons/>

**BugReports** <https://github.com/Bioconductor/GenomicDataCommons/issues/new>

**Encoding** UTF-8

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**git\_url** <https://git.bioconductor.org/packages/GenomicDataCommons>

**git\_branch** RELEASE\_3\_22

**git\_last\_commit** dc500e6

**git\_last\_commit\_date** 2025-11-07

**Repository** Bioconductor 3.22

**Date/Publication** 2025-11-11

**Author** Martin Morgan [aut],  
Sean Davis [aut, cre],  
Marcel Ramos [ctb]

**Maintainer** Sean Davis <seandavi@gmail.com>

## Contents

GenomicDataCommons-package . . . . .	2
aggregations . . . . .	3
available_expand . . . . .	4
available_fields . . . . .	5
available_values . . . . .	6
count . . . . .	6
default_fields . . . . .	7
endpoints . . . . .	8
entity_name . . . . .	8
expand . . . . .	9
facet . . . . .	10
field_description . . . . .	11
filtering . . . . .	11
gdcdata . . . . .	13
gdc_cache . . . . .	15
gdc_client . . . . .	16
gdc_clinical . . . . .	17
gdc_token . . . . .	18
grep_fields . . . . .	19
ids . . . . .	19
id_field . . . . .	20
make_filter . . . . .	21
manifest . . . . .	21
mapping . . . . .	23
query . . . . .	23
readDNACopy . . . . .	25
readHTSeqFile . . . . .	26
response . . . . .	26
results . . . . .	27
results_all . . . . .	28
select . . . . .	29
slicing . . . . .	29
status . . . . .	31
transfer . . . . .	32
write_manifest . . . . .	33
<b>Index</b>	<b>34</b>

---

GenomicDataCommons-package

*GenomicDataCommons: A package for interfacing with the NCI GDC*

---

## Description

Programmatically access the NIH / NCI Genomic Data Commons RESTful service.

**finding data**

- [query](#)
- [cases](#)
- [projects](#)
- [files](#)
- [annotations](#)
- [mapping](#)

**downloading data**

data

**Author(s)**

**Maintainer:** Sean Davis <seandavi@gmail.com>

Authors:

- Martin Morgan <martin.morgan@roswellpark.org>

Other contributors:

- Marcel Ramos <marcel.ramos@sph.cuny.edu> [contributor]

**See Also**

Useful links:

- <https://bioconductor.org/packages/GenomicDataCommons>
- <http://github.com/Bioconductor/GenomicDataCommons>
- <http://bioconductor.github.io/GenomicDataCommons/>
- Report bugs at <https://github.com/Bioconductor/GenomicDataCommons/issues/new>

---

aggregations

*aggregations*

---

**Description**

aggregations

**Usage**

```
aggregations(x)
```

```
## S3 method for class 'GDCQuery'  
aggregations(x)
```

```
## S3 method for class 'GDCResponse'  
aggregations(x)
```

**Arguments**

x                    a [GDCQuery](#) object

**Value**

a list of `data.frame` with one member for each requested facet. The data frames each have two columns, `key` and `doc_count`.

**Methods (by class)**

- `aggregations(GDCQuery)`:
- `aggregations(GDCResponse)`:

**Examples**

```
# Number of each file type
res = files() |> facet(c('type','data_type')) |> aggregations()
res$type
```

---

available_expand	<i>Return valid values for "expand"</i>
------------------	---

---

**Description**

The GDC allows a shorthand for specifying groups of fields to be returned by the metadata queries. These can be specified in a [select](#) method call to easily supply groups of fields.

**Usage**

```
available_expand(entity)

## S3 method for class 'character'
available_expand(entity)

## S3 method for class 'GDCQuery'
available_expand(entity)
```

**Arguments**

entity                Either a [GDCQuery](#) object or a `character(1)` specifying a GDC entity ('cases', 'files', 'annotations', 'projects')

**Value**

A character vector

**See Also**

See [https://docs.gdc.cancer.gov/API/Users\\_Guide/Search\\_and\\_Retrieval/#expand](https://docs.gdc.cancer.gov/API/Users_Guide/Search_and_Retrieval/#expand) for details

**Examples**

```
head(available_expand('files'))
```

---

available_fields	<i>S3 Generic to return all GDC fields</i>
------------------	--

---

**Description**

S3 Generic to return all GDC fields

**Usage**

```
available_fields(x)

## S3 method for class 'GDCQuery'
available_fields(x)

## S3 method for class 'character'
available_fields(x)
```

**Arguments**

x                    A character(1) string ('cases','files','projects', 'annotations') or an subclass of [GDCQuery](#).

**Value**

a character vector of the default fields

**Methods (by class)**

- available\_fields(GDCQuery): GDCQuery method
- available\_fields(character): character method

**Examples**

```
available_fields('projects')
projQuery = query('projects')
available_fields(projQuery)
```

---

available_values	<i>Find common values for a GDC field</i>
------------------	---

---

**Description**

Find common values for a GDC field

**Usage**

```
available_values(entity, field)
```

**Arguments**

entity	character(1), a GDC entity ("cases", "files", "annotations", "projects")
field	character(1), a field that is present in the entity record

**Value**

character vector of the top 100 (or fewer) most frequent values for a the given field

**Examples**

```
available_values('files', 'cases.project.project_id')[1:5]
```

---

count	<i>provide count of records in a <a href="#">GDCQuery</a></i>
-------	---

---

**Description**

provide count of records in a [GDCQuery](#)

**Usage**

```
count(x, ...)

## S3 method for class 'GDCQuery'
count(x, ...)

## S3 method for class 'GDCResponse'
count(x, ...)
```

**Arguments**

x	a <a href="#">GDCQuery</a> object
...	passed to httr (good for passing config info, etc.)

**Value**

integer(1) representing the count of records that will be returned by the current query

**Methods (by class)**

- count(GDCQuery):
- count(GDCResponse):

**Examples**

```
# total number of projects
projects() |> count()

# total number of cases
cases() |> count()
```

---

default_fields	<i>S3 Generic to return default GDC fields</i>
----------------	--

---

**Description**

S3 Generic to return default GDC fields

**Usage**

```
default_fields(x)

## S3 method for class 'character'
default_fields(x)

## S3 method for class 'GDCQuery'
default_fields(x)
```

**Arguments**

x                    A character string ('cases', 'files', 'projects', 'annotations') or an subclass of [GDCQuery](#).

**Value**

a character vector of the default fields

**Methods (by class)**

- default\_fields(character): character method
- default\_fields(GDCQuery): GDCQuery method

**Examples**

```
default_fields('projects')
projQuery = query('projects')
default_fields(projQuery)
```

---

endpoints

*Endpoints and Parameters*


---

**Description**

endpoints() returns available endpoints.

**Usage**

```
endpoints()
```

```
parameters()
```

**Value**

endpoints() returns a character vector of possible endpoints.

parameters() returns a list of possible parameters and their default values.

**Examples**

```
endpoints()
parameters()
```

---

entity\_name

*Get the entity name from a GDCQuery object*


---

**Description**

An "entity" is simply one of the four metadata endpoints.

- cases
- projects
- files
- annotations

All [GDCQuery](#) objects will have an entity name. This S3 method is simply a utility accessor for those names.

**Usage**

```
entity_name(x)
```

```
## S3 method for class 'GDCQuery'
entity_name(x)
```

```
## S3 method for class 'GDCResults'
entity_name(x)
```

**Arguments**

x                    a [GDCQuery](#) object

**Value**

character(1) name of an associated entity; one of "cases", "files", "projects", "annotations".

**Examples**

```
qcases = cases()
qprojects = projects()

entity_name(qcases)
entity_name(qprojects)
```

---

expand	<i>Set the expand parameter</i>
--------	---------------------------------

---

**Description**

S3 generic to set GDCQuery expand parameter

**Usage**

```
expand(x, expand)

## S3 method for class 'GDCQuery'
expand(x, expand)
```

**Arguments**

x                    the objects on which to set fields  
 expand                a character vector specifying the fields

**Value**

A [GDCQuery](#) object, with the expand member altered.

**Methods (by class)**

- `expand(GDCQuery)`: set expand fields on a GDCQuery object

**Examples**

```
gProj = projects()
gProj$fields
head(available_fields(gProj))
default_fields(gProj)

gProj |>
  select(default_fields(gProj)[1:2]) |>
  response() |>
```

```
str(max_level=2)
```

---

facet

*Set facets for a [GDCQuery](#)*

---

### Description

Set facets for a [GDCQuery](#)

Get facets for a [GDCQuery](#)

### Usage

```
facet(x, facets)
```

```
get_facets(x)
```

```
## S3 method for class 'GDCQuery'  
get_facets(x)
```

### Arguments

x a [GDCQuery](#) object

facets a character vector of fields that will be used for forming aggregations (facets).  
Default is to set facets for all default fields. See [default\\_fields](#) for details

### Value

returns a [GDCQuery](#) object, with facets field updated.

### Examples

```
# create a new GDCQuery against the projects endpoint  
gProj = projects()  
  
# default facets are NULL  
get_facets(gProj)  
  
# set facets and save result  
gProjFacet = facet(gProj)  
  
# check facets  
get_facets(gProjFacet)  
  
# and get a response, noting that  
# the aggregations list member contains  
# tibbles for each facet  
str(response(gProjFacet, size=2), max.level=2)
```

---

field_description	<i>S3 Generic that returns the field description text, if available</i>
-------------------	---

---

**Description**

S3 Generic that returns the field description text, if available

**Usage**

```
field_description(entity, field)

## S3 method for class 'GDCQuery'
field_description(entity, field)

## S3 method for class 'character'
field_description(entity, field)
```

**Arguments**

entity	character(1) string ('cases','files','projects', 'annotations', etc.) or an subclass of <a href="#">GDCQuery</a> .
field	character(1), the name of the field that will be used to look up the description.

**Value**

character(1) descriptive text or character(0) if no description is available.

**Methods (by class)**

- field\_description(GDCQuery): GDCQuery method
- field\_description(character): character method

**Examples**

```
field_description('cases', 'annotations.category')
casesQuery = query('cases')
field_description(casesQuery, 'annotations.category')
field_description(cases(), 'annotations.category')
```

---

filtering	<i>Manipulating GDCQuery filters</i>
-----------	--------------------------------------

---

**Description**

Manipulating GDCQuery filters

The filter is simply a safe accessor for the filter element in [GDCQuery](#) objects.

The get\_filter is simply a safe accessor for the filter element in [GDCQuery](#) objects.

**Usage**

```

filter(x, expr)

## S3 method for class 'GDCQuery'
filter(x, expr)

get_filter(x)

## S3 method for class 'GDCQuery'
get_filter(x)

```

**Arguments**

**x** the object on which to set the filter list member

**expr** a filter expression in the form of the right hand side of a formula, where bare names (without quotes) are allowed if they are available fields associated with the GDCQuery object, x

**Value**

A [GDCQuery](#) object with the filter field replaced by specified filter expression

**Examples**

```

# make a GDCQuery object to start
#
# Projects
#
pQuery = projects()

# check for the default fields
# so that we can use one of them to build a filter
default_fields(pQuery)
pQuery = filter(pQuery, ~ project_id == 'TCGA-LUAC')
get_filter(pQuery)

#
# Files
#
fQuery = files()
default_fields(fQuery)

fQuery = filter(fQuery, ~ data_format == 'VCF')
# OR
# with recent GenomicDataCommons versions:
# no "~" needed
fQuery = filter(fQuery, data_format == 'VCF')

get_filter(fQuery)

fQuery = filter(fQuery, ~ data_format == 'VCF'
                & experimental_strategy == 'WXS'
                & type == 'simple_somatic_mutation')

files() |> filter(~ data_format == 'VCF'

```

```
& experimental_strategy=='WXS'
& type == 'simple_somatic_mutation') |> count()

files() |> filter( data_format == 'VCF'
                  & experimental_strategy=='WXS'
                  & type == 'simple_somatic_mutation') |> count()

# Filters may be chained for the
# equivalent query
#
# When chained, filters are combined with logical AND

files() |>
  filter(~ data_format == 'VCF') |>
  filter(~ experimental_strategy == 'WXS') |>
  filter(~ type == 'simple_somatic_mutation') |>
  count()

# OR

files() |>
  filter( data_format == 'VCF') |>
  filter( experimental_strategy == 'WXS') |>
  filter( type == 'simple_somatic_mutation') |>
  count()

# Use str() to get a cleaner picture
str(get_filter(fQuery))
```

---

gdcdata

*Download GDC files*

---

## Description

Download one or more files from GDC. Files are downloaded using the UUID and renamed to the file name on the remote system. By default, neither the uuid nor the file name on the remote system can exist.

## Usage

```
gdcdata(
  uuids,
  use_cached = TRUE,
  progress = interactive(),
  token = NULL,
  access_method = "api",
  transfer_args = character(),
  ...
)
```

**Arguments**

uuids	character() of GDC file UUIDs.
use_cached	logical(1) default TRUE indicating that, if found in the cache, the file will not be downloaded again. If FALSE, all supplied uuids will be re-downloaded.
progress	logical(1) default TRUE in interactive sessions, FALSE otherwise indicating whether a progress bar should be produced for each file download.
token	(optional) character(1) security token allowing access to restricted data. See <a href="https://gdc-docs.nci.nih.gov/API/Users_Guide/Authentication_and_Authorization/">https://gdc-docs.nci.nih.gov/API/Users_Guide/Authentication_and_Authorization/</a> .
access_method	character(1), either 'api' or 'client'. See details.
transfer_args	character(1), additional arguments to pass to the gdc-client command line. See <a href="#">gdc_client</a> and <a href="#">transfer_help</a> for details.
...	further arguments passed to files

**Details**

This function is appropriate for one or several files; for large downloads use [manifest](#) to create a manifest for and the GDC Data Transfer Tool.

When `access_method` is "api", the GDC "data" endpoint is the transfer mechanism used. The alternative `access_method`, "client", will utilize the `gdc-client` transfer tool, which must be downloaded separately and available. See [gdc\\_client](#) for details on specifying the location of the `gdc-client` executable.

**Value**

a named vector with file uuids as the names and paths as the value

**See Also**

[manifest](#) for downloading large data.

**Examples**

```
# get some example file uuids
uuids <- files() |>
  filter(~ access == 'open' & file_size < 100000) |>
  results(size = 3) |>
  ids()

# and get the data, placing it into the gdc_cache() directory
gdcdata(uuids, use_cached=TRUE)
```

---

gdc_cache	<i>Work with gdc cache directory</i>
-----------	--------------------------------------

---

## Description

The GenomicDataCommons package will cache downloaded files to minimize network and allow for offline work. These functions are used to create a cache directory if one does not exist, set a global option, and query that option. The cache directory will default to the user "cache" directory according to specifications in [app\\_dir](#). However, the user may want to set this to another directory with more or higher performance storage.

## Usage

```
gdc_cache()  
  
gdc_set_cache(  
  directory = rappdirs::app_dir(appname = "GenomicDataCommons")$cache(),  
  verbose = TRUE,  
  create_without_asking = !interactive()  
)
```

## Arguments

directory	character(1) directory path, will be created recursively if not present.
verbose	logical(1) whether or not to message the location of the cache directory after creation.
create_without_asking	logical(1) specifying whether to allow the function to create the cache directory without asking the user first. In an interactive session, if the cache directory does not exist, the user will be prompted before creation.

## Details

The cache structure is currently just a directory with each file being represented by a path constructed as: CACHEDIR/UUID/FILENAME. The cached files can be manipulated using standard file system commands (removing, finding, etc.). In this sense, the cache system is minimalist in design.

## Value

character(1) directory path that serves as the base directory for GenomicDataCommons downloads.  
the created directory (invisibly)

## Functions

- `gdc_set_cache()`: (Re)set the GenomicDataCommons cache directory

## Examples

```
gdc_cache()
## Not run:
gdc_set_cache(getwd())

## End(Not run)
```

---

gdc\_client

*return gdc-client executable path*

---

## Description

This function is a convenience function to find and return the path to the GDC Data Transfer Tool executable assumed to be named 'gdc-client'. The assumption is that the appropriate version of the GDC Data Transfer Tool is a separate download available from <https://gdc.cancer.gov/access-data/gdc-data-transfer-tool> and as a backup from <https://github.com/NCI-GDC/gdc-client>.

## Usage

```
gdc_client()
```

## Details

The path is checked in the following order:

1. an R option("gdc\_client")
2. an environment variable GDC\_CLIENT
3. from the search PATH
4. in the current working directory

## Value

character(1) the path to the gdc-client executable.

## Examples

```
# this cannot run without first
# downloading the GDC Data Transfer Tool
gdc_client = try(gdc_client(),silent=TRUE)
```

---

`gdc_clinical`*Get clinical information from GDC*

---

## Description

The NCI GDC has a complex data model that allows various studies to supply numerous clinical and demographic data elements. However, across all projects that enter the GDC, there are similarities. This function returns four data.frames associated with `case_ids` from the GDC.

## Usage

```
gdc_clinical(case_ids, include_list_cols = FALSE)
```

## Arguments

`case_ids` a character() vector of `case_ids`, typically from "cases" query.

`include_list_cols` logical(1), whether to include list columns in the "main" data.frame. These list columns have values for aliquots, samples, etc. While these may be useful for some situations, they are generally not that useful as clinical annotations.

## Details

Note that these data.frames can, in general, have different numbers of rows (or even no rows at all). If one wishes to combine to produce a single data.frame, using the approach of left joining to the "main" data.frame will yield a useful combined data.frame. We do not do that directly given the potential for 1:many relationships. It is up to the user to determine what the best approach is for any given dataset.

## Value

A list of four data.frames:

1. main, representing basic case identification and metadata (update date, etc.)
2. diagnoses
3. exposures
4. demographic

## Examples

```
case_ids = cases() |> results(size=10) |> ids()
clinical_data = gdc_clinical(case_ids)

# overview of clinical results
class(clinical_data)
names(clinical_data)
sapply(clinical_data, class)
sapply(clinical_data, nrow)

# available data
head(clinical_data$main)
head(clinical_data$demographic)
```

```
head(clinical_data$diagnoses)
head(clinical_data$exposures)
```

---

gdc_token	<i>return a gdc token from file or environment</i>
-----------	--

---

## Description

The GDC requires an auth token for downloading data that are "controlled access". For example, BAM files for human datasets, germline variant calls, and SNP array raw data all are protected as "controlled access". For these files, a GDC access token is required. See the [https://docs.gdc.cancer.gov/Data\\_Portal/Users\\_Guide/Authentication\\_Tokens](https://docs.gdc.cancer.gov/Data_Portal/Users_Guide/Authentication_Tokens). Note that this function simply returns a string value. It is possible to keep the GDC token in a variable in R or to pass a string directly to the appropriate parameter. This function is simply a convenience function for alternative approaches to get a token from an environment variable or a file.

## Usage

```
gdc_token()
```

## Details

This function will resolve locations of the GDC token in the following order:

- from the environment variable, GDC\_TOKEN, expected to contain the token downloaded from the GDC as a string
- using `readLines` to read a file named in the environment variable, GDC\_TOKEN\_FILE
- using `readLines` to read from a file called `.gdc_token` in the user's home directory

If all of these fail, this function will return an error.

## Value

`character(1)` (invisibly, to protect against inadvertently printing) the GDC token.

## References

[https://docs.gdc.cancer.gov/Data\\_Portal/Users\\_Guide/Cart/#gdc-authentication-tokens](https://docs.gdc.cancer.gov/Data_Portal/Users_Guide/Cart/#gdc-authentication-tokens)

## Examples

```
# This will not run before a GDC token
# is in place.
token = try(gdc_token(), silent=TRUE)
```

---

```
grep_fields          Find matching field names
```

---

**Description**

This utility function allows quick text-based search of available fields for using [grep](#)

**Usage**

```
grep_fields(entity, pattern, ..., value = TRUE)
```

**Arguments**

entity	one of the available gdc entities ('files','cases',...) against which to gather available fields for matching
pattern	A regular expression that will be used in a call to <a href="#">grep</a>
...	passed on to grep
value	logical(1) whether to return values as opposed to indices (passed along to grep)

**Value**

character() vector of field names matching pattern

**Examples**

```
grep_fields('files','analysis')
```

---

```
ids          Get the ids associated with a GDC query or response
```

---

**Description**

The GDC assigns ids (in the form of uuids) to objects in its database. Those ids can be used for relationships, searching on the website, and as unique ids. All

**Usage**

```
ids(x)

## S3 method for class 'GDCManifest'
ids(x)

## S3 method for class 'GDCQuery'
ids(x)

## S3 method for class 'GDCResults'
ids(x)

## S3 method for class 'GDCResponse'
ids(x)
```

**Arguments**

x                    A [GDCQuery](#) or [GDCResponse](#) object

**Value**

a character vector of all the entity ids

**Examples**

```
# use with a GDC query, in this case for "cases"
ids(cases() |> filter(~ project.project_id == "TCGA-CHOL"))
# also works for responses
ids(response(files()))
# and results
ids(results(cases()))
```

---

id_field	<i>get the name of the id field</i>
----------	-------------------------------------

---

**Description**

In many places in the `GenomicDataCommons` package, the entity ids are stored in a column or a vector with a specific name that corresponds to the field name at the GDC. The format is the entity name (singular) "\_id". This generic simply returns that name from a given object.

**Usage**

```
id_field(x)

## S3 method for class 'GDCQuery'
id_field(x)

## S3 method for class 'GDCResults'
id_field(x)
```

**Arguments**

x                    An object representing the query or results of an entity from the GDC ("cases", "files", "annotations", "projects")

**Value**

character(1) such as "case\_id", "file\_id", etc.

**Methods (by class)**

- `id_field(GDCQuery)`: `GDCQuery` method
- `id_field(GDCResults)`: `GDCResults` method

**Examples**

```
id_field(cases())
```

---

make\_filter

*Create NCI GDC filters for limiting GDC query results*

---

**Description**

Searching the NCI GDC allows for complex filtering based on logical operations and simple comparisons. This function facilitates writing such filter expressions in R-like syntax with R code evaluation.

**Usage**

```
make_filter(expr, available_fields)
```

**Arguments**

`expr` a lazy-wrapped expression or a formula RHS equivalent

`available_fields` a character vector of the additional names that will be injected into the filter evaluation environment

**Details**

If used with `available_fields`, "bare" fields that are named in the `available_fields` character vector can be used in the filter expression without quotes.

**Value**

a list that represents an R version of the JSON that will ultimately be used in an NCI GDC search or other query.

---

manifest

*Prepare GDC manifest file for bulk download*

---

**Description**

The manifest function/method creates a manifest of files to be downloaded using the GDC Data Transfer Tool. There are methods for creating manifest data frames from [GDCQuery](#) objects that contain file information ("cases" and "files" queries).

## Usage

```
manifest(x, from = 0, size = count(x), ...)  
  
## S3 method for class 'gdc_files'  
manifest(x, from = 0, size = count(x), ...)  
  
## S3 method for class 'GDCfilesResponse'  
manifest(x, from = 0, size = count(x), ...)  
  
## S3 method for class 'GDCcasesResponse'  
manifest(x, from = 0, size = count(x), ...)
```

## Arguments

x	An <a href="#">GDCQuery</a> object of subclass "gdc_files" or "gdc_cases".
from	Record number from which to start when returning the manifest.
size	The total number of records to return. Default will return the usually desirable full set of records.
...	passed to <a href="#">PUT</a> .

## Value

A [tibble](#), also of type "gdc\_manifest", with five columns:

- id
- filename
- md5
- size
- state

## Methods (by class)

- manifest(gdc\_files):
- manifest(GDCfilesResponse):
- manifest(GDCcasesResponse):

## Examples

```
gFiles = files()  
shortManifest = gFiles |> manifest(size=10)  
head(shortManifest,n=3)
```

---

`mapping`*Query GDC for available endpoint fields*

---

**Description**

Query GDC for available endpoint fields

**Usage**

```
mapping(endpoint)
```

**Arguments**

`endpoint` `character(1)` corresponding to endpoints for which users may specify additional or alternative fields. Endpoints include “projects”, “cases”, “files”, and “annotations”.

**Value**

A data frame describing the field (field name), full (full data model name), type (data type), and four additional columns describing the "set" to which the fields belong—“default”, “expand”, “multi”, and “nested”.

**Examples**

```
map <- mapping("projects")
head(map)
# get only the "default" fields
subset(map, defaults)
# And get just the text names of the "default" fields
subset(map, defaults)$field
```

---

`query`*Start a query of GDC metadata*

---

**Description**

The basis for all functionality in this package starts with constructing a query in R. The `GDCQuery` object contains the filters, facets, and other parameters that define the returned results. A token is required for accessing certain datasets.

**Usage**

```
query(
  entity,
  filters = NULL,
  facets = NULL,
  expand = NULL,
  fields = default_fields(entity),
```

```

    ...
)
cases(...)
files(...)
projects(...)
annotations(...)
ssms(...)
ssm_occurrences(...)
cnvs(...)
cnv_occurrences(...)
genes(...)

```

### Arguments

<code>entity</code>	character vector, including one of the entities in <code>.gdc_entities</code>
<code>filters</code>	a filter list, typically created using <code>make_filter</code> , or added to an existing GDCQuery object using <code>filter</code> .
<code>facets</code>	a character vector of facets for counting common values. See <code>available_fields</code> . In general, one will not specify this parameter but will use <code>facet</code> instead.
<code>expand</code>	a character vector of "expands" to include in returned data. See <code>available_expand</code>
<code>fields</code>	a character vector of fields to return. See <code>available_fields</code> . In general, one will not specify fields directly, but instead use <code>select</code>
<code>...</code>	passed through to <code>query</code>

### Value

An S3 object, the GDCQuery object. This is a list with the following members.

- `filters`
- `facets`
- `fields`
- `expand`
- `archive`
- `token`

### Functions

- `cases()`: convenience constructor for a GDCQuery for cases
- `files()`: convenience constructor for a GDCQuery for files
- `projects()`: convenience constructor for a GDCQuery for projects
- `annotations()`: convenience constructor for a GDCQuery for annotations

- `ssms()`: convenience constructor for a GDCQuery for ssms
- `ssm_occurrences()`: convenience constructor for a GDCQuery for ssm\_occurrences
- `cnvs()`: convenience constructor for a GDCQuery for cnvs
- `cnv_occurrences()`: convenience constructor for a GDCQuery for cnv\_occurrences
- `genes()`: convenience constructor for a GDCQuery for genes

### Examples

```
qcases = query('cases')
# equivalent to:
qcases = cases()
```

---

readDNACopy	<i>Read DNACopy results into GRanges object</i>
-------------	---

---

### Description

Read DNACopy results into GRanges object

### Usage

```
readDNACopy(fname, ...)
```

### Arguments

fname	The path to a DNACopy-like file.
...	passed to <a href="#">read_tsv</a>

### Value

a [GRanges](#) object

### Examples

```
fname = system.file(package='GenomicDataCommons',
                    'extdata/dnacopy.tsv.gz')
dnac = readDNACopy(fname)
class(dnac)
length(dnac)
```

---

readHTSeqFile	<i>Read a single htseq-counts result file.</i>
---------------	--

---

### Description

The htseq package is used extensively to count reads relative to regions (see <http://www-huber.embl.de/HTSeq/doc/counting.html>). The output of htseq-count is a simple two-column table that includes features in column 1 and counts in column 2. This function simply reads in the data from one such file and assigns column names.

### Usage

```
readHTSeqFile(fname, samplename = "sample", ...)
```

### Arguments

fname	character(1), the path of the htseq-count file.
samplename	character(1), the name of the sample. This will become the name of the second column on the resulting data.frame, making for easier merging if necessary.
...	passed to <a href="#">read_tsv</a>

### Value

a two-column data frame

### Examples

```
fname = system.file(package='GenomicDataCommons',
                    'extdata/example.htseq.counts.gz')
dat = readHTSeqFile(fname)
head(dat)
```

---

response	<i>Fetch <a href="#">GDCQuery</a> metadata from GDC</i>
----------	---

---

### Description

Fetch [GDCQuery](#) metadata from GDC

### Usage

```
response(x, ...)

## S3 method for class 'GDCQuery'
response(x, from = 0, size = 10, ..., response_handler = jsonlite::fromJSON)

response_all(x, ...)
```

**Arguments**

x a [GDCQuery](#) object  
 ... passed to `httr` (good for passing config info, etc.)  
 from integer index from which to start returning data  
 size number of records to return  
 response\_handler a function that processes JSON (as text) and returns an R object. Default is [fromJSON](#).

**Value**

A `GDCResponse` object which is a list with the following members:

- results
- query
- aggregations
- pages

**Examples**

```

# basic class stuff
gCases = cases()
resp = response(gCases)
class(resp)
names(resp)

# And results from query
resp$results[[1]]

```

---

 results

*results*


---

**Description**

results

**Usage**

```

results(x, ...)

## S3 method for class 'GDCQuery'
results(x, ...)

## S3 method for class 'GDCResponse'
results(x, ...)

```

**Arguments**

x a [GDCQuery](#) object  
 ... passed on to [response](#)

**Value**

A (typically nested) list of GDC records

**Methods (by class)**

- results(GDCQuery):
- results(GDCResponse):

**Examples**

```
qcases = cases() |> results()
length(qcases)
```

---

results_all	<i>results_all</i>
-------------	--------------------

---

**Description**

results\_all

**Usage**

```
results_all(x)
```

```
## S3 method for class 'GDCQuery'
results_all(x)
```

```
## S3 method for class 'GDCResponse'
results_all(x)
```

**Arguments**

x                    a [GDCQuery](#) object

**Value**

A (typically nested) list of GDC records

**Methods (by class)**

- results\_all(GDCQuery):
- results\_all(GDCResponse):

**Examples**

```
# details of all available projects
projResults = projects() |> results_all()
length(projResults)
count(projects())
```

---

select	<i>S3 generic to set GDCQuery fields</i>
--------	--

---

**Description**

S3 generic to set GDCQuery fields

**Usage**

```
select(x, fields)

## S3 method for class 'GDCQuery'
select(x, fields)
```

**Arguments**

x	the objects on which to set fields
fields	a character vector specifying the fields

**Value**

A [GDCQuery](#) object, with the fields member altered.

**Methods (by class)**

- `select(GDCQuery)`: set fields on a GDCQuery object

**Examples**

```
gProj = projects()
gProj$fields
head(available_fields(gProj))
default_fields(gProj)

gProj |>
  select(default_fields(gProj)[1:2]) |>
  response() |>
  str(max_level=2)
```

---

slicing	<i>Query GDC for data slices</i>
---------	----------------------------------

---

**Description**

This function returns a BAM file representing reads overlapping regions specified either as chromosomal regions or as gencode gene symbols.

**Usage**

```
slicing(
  uuid,
  regions,
  symbols,
  destination = file.path(tempdir(), paste0(uuid, ".bam")),
  overwrite = FALSE,
  progress = interactive(),
  token = gdc_token()
)
```

**Arguments**

uuid	character(1) identifying the BAM file resource
regions	character() vector describing chromosomal regions, e.g., c("chr1", "chr2:10000", "chr3:10000-20000") (all of chromosome 1, chromosome 2 from position 10000 to the end, chromosome 3 from 10000 to 20000).
symbols	character() vector of gencode gene symbols, e.g., c("BRCA1", "PTEN")
destination	character(1) default tempfile() file path for BAM file slice
overwrite	logical(1) default FALSE can destination be overwritten?
progress	logical(1) default interactive() should a progress bar be used?
token	character(1) security token allowing access to restricted data. Almost all BAM data is restricted, so a token is usually required. See <a href="https://docs.gdc.cancer.gov/Data/Data_Security/Data_Security/#authentication-tokens">https://docs.gdc.cancer.gov/Data/Data_Security/Data_Security/#authentication-tokens</a> .

**Details**

This function uses the Genomic Data Commons "slicing" API to get portions of a BAM file specified either using "regions" or using HGNC gene symbols.

**Value**

character(1) destination to the downloaded BAM file

**Examples**

```
## Not run:
slicing("df80679e-c4d3-487b-934c-fcc782e5d46e",
  regions="chr17:75000000-76000000",
  token=gdc_token())

# Get 10 BAM files.
bamfiles = files() |>
  filter(data_format=='BAM') |>
  results(size=10) |> ids()

# Current alignments at the GDC are to GRCh38
library('TxDb.Hsapiens.UCSC.hg38.knownGene')
all_genes = genes(TxDb.Hsapiens.UCSC.hg38.knownGene)

first3genes = all_genes[1:3]
# remove strand info
```

```
strand(first3genes) = '*'

# We can get our regions easily now
as.character(first3genes)

# Use parallel downloads to speed processing
library(BiocParallel)
register(MulticoreParam())

fnames = bplapply(bamfiles, slicing, overwrite = TRUE,
                 regions=as.character(first3genes))

# 10 BAM files
fnames

library(GenomicAlignments)
lapply(unlist(fnames), readGAlignments)

## End(Not run)
```

---

status

*Query the GDC for current status*

---

## Description

Query the GDC for current status

## Usage

```
status(version = NULL)
```

## Arguments

version (optional) character(1) version of GDC

## Value

List describing current status.

## Examples

```
status()
```

transfer

*Bulk data download***Description**

The GDC maintains a special tool, [https://docs.gdc.cancer.gov/Data\\_Transfer\\_Tool/Users\\_Guide/Getting\\_Started/](https://docs.gdc.cancer.gov/Data_Transfer_Tool/Users_Guide/Getting_Started/), that enables high-performance, potentially parallel, and resumable downloads. The Data Transfer Tool is an external program that requires separate download. Due to recent changes in the GDC API, the transfer function now validates the version of the ‘gdc-client’ to ensure reliable downloads.

**Usage**

```
transfer(uuids, args = character(), token = NULL, overwrite = FALSE)
```

```
gdc_client_version_validate(valid_version = .GDC_COMPATIBLE_VERSION)
```

```
transfer_help()
```

**Arguments**

uuids	character() vector of GDC file UUIDs
args	character() vector specifying command-line arguments to be passed to gdc-client. See <a href="#">transfer_help</a> for possible values. The arguments --manifest, --dir, and --token-file are determined by manifest, destination_dir, and token, respectively, and should NOT be provided as elements of args.
token	character(1) containing security token allowing access to restricted data. See <a href="https://gdc-docs.nci.nih.gov/API/Users_Guide/Authentication_and_Authorization/">https://gdc-docs.nci.nih.gov/API/Users_Guide/Authentication_and_Authorization/</a> . Note that the GDC transfer tool requires a file for data transfer. Therefore, this token will be written to a temporary file (with appropriate permissions set).
overwrite	logical(1) default FALSE indicating whether existing files with identical name should be over-written.
valid_version	character(1) The last known version that works for the current data release for which to validate against, not typically changed by the end-user.

**Value**

character(1) directory path to which the files were downloaded.

**Functions**

- `gdc_client_version_validate()`: If you are using the ‘client’ option, your ‘gdc-client’ should be up-to-date ( $\geq 1.3.0$ ).
- `transfer_help()`:

## Examples

```
## Not run:
uuids = files() |>
  filter(access == "open") |>
  results() |>
  ids()
file_paths <- transfer(uuids)
file_paths
names(file_paths)
# and with authentication
# REQUIRES gdc_token
# destination <- transfer(uuids,token=gdc_token())

## End(Not run)
```

---

write\_manifest

*write a manifest data.frame to disk*

---

## Description

The `manifest` method creates a data.frame that represents the data for a manifest file needed by the GDC Data Transfer Tool. While the file format is nothing special, this is a simple helper function to write a manifest data.frame to disk. It returns the path to which the file is written, so it can be used "in-line" in a call to `transfer`.

## Usage

```
write_manifest(manifest, destfile = tempfile())
```

## Arguments

manifest	A data.frame with five columns, typically created by a call to <code>manifest</code>
destfile	The filename for saving the manifest.

## Value

character(1) the destination file name.

## Examples

```
mf = files() |> manifest(size=10)
write_manifest(mf)
```

# Index

## \* **internal**

- endpoints, 8
- aggregations, 3
- annotations, 3
- annotations (query), 23
- app\_dir, 15
- available\_expand, 4, 24
- available\_fields, 5, 24
- available\_values, 6
- cases, 3
- cases (query), 23
- cnv\_occurrences (query), 23
- cnvs (query), 23
- count, 6
- default\_fields, 7, 10
- endpoints, 8
- entity\_name, 8
- expand, 9
- facet, 10, 24
- field\_description, 11
- files, 3
- files (query), 23
- filter, 24
- filter (filtering), 11
- filtering, 11
- fromJSON, 27
- gdc\_cache, 15
- gdc\_client, 14, 16
- gdc\_client\_version\_validate (transfer), 32
- gdc\_clinical, 17
- gdc\_set\_cache (gdc\_cache), 15
- gdc\_token, 18
- gdcdata, 13
- GDCQuery, 4–12, 20–22, 26–29
- GDCQuery (query), 23
- GDCResponse, 20
- GDCResponse (response), 26
- genes (query), 23
- GenomicDataCommons
  - (GenomicDataCommons-package), 2
- GenomicDataCommons-package, 2
- get\_facets (facet), 10
- get\_filter (filtering), 11
- GRanges, 25
- grep, 19
- grep\_fields, 19
- id\_field, 20
- ids, 19
- make\_filter, 21, 24
- manifest, 14, 21, 33
- mapping, 3, 23
- parameters (endpoints), 8
- projects, 3
- projects (query), 23
- PUT, 22
- query, 3, 23, 24
- read\_tsv, 25, 26
- readDNACopy, 25
- readHTSeqFile, 26
- response, 26, 27
- response\_all (response), 26
- results, 27
- results\_all, 28
- select, 4, 24, 29
- slicing, 29
- ssm\_occurrences (query), 23
- ssms (query), 23
- status, 31
- tibble, 22
- transfer, 32, 33
- transfer\_help, 14, 32
- transfer\_help (transfer), 32
- write\_manifest, 33