Package ‘GenomicDataCommons’

March 22, 2024

<table>
<thead>
<tr>
<th>Type</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>NIH / NCI Genomic Data Commons Access</td>
</tr>
<tr>
<td>Description</td>
<td>Programmatically access the NIH / NCI Genomic Data Commons RESTful service.</td>
</tr>
<tr>
<td>Version</td>
<td>1.26.0</td>
</tr>
<tr>
<td>Date</td>
<td>2022-04-27</td>
</tr>
<tr>
<td>License</td>
<td>Artistic-2.0</td>
</tr>
<tr>
<td>Depends</td>
<td>R (&gt;= 3.4.0), magrittr</td>
</tr>
<tr>
<td>Imports</td>
<td>stats, httr, xml2, jsonlite, utils, rlang, readr, GenomicRanges, IRanges, dplyr, rappdirs, tibble, tidyr</td>
</tr>
<tr>
<td>Suggests</td>
<td>BiocStyle, knitr, rmarkdown, DT, testthat, listviewer, ggplot2, GenomicAlignments, Rsamtools, BiocParallel, TxDb.Hsapiens.UCSC.hg38.knownGene, VariantAnnotation, maftools, R.utils, data.table</td>
</tr>
<tr>
<td>biocViews</td>
<td>DataImport, Sequencing</td>
</tr>
<tr>
<td>BugReports</td>
<td><a href="https://github.com/Bioconductor/GenomicDataCommons/issues/new">https://github.com/Bioconductor/GenomicDataCommons/issues/new</a></td>
</tr>
<tr>
<td>Encoding</td>
<td>UTF-8</td>
</tr>
<tr>
<td>VignetteBuilder</td>
<td>knitr</td>
</tr>
<tr>
<td>RoxygenNote</td>
<td>7.2.3</td>
</tr>
<tr>
<td>git_url</td>
<td><a href="https://git.bioconductor.org/packages/GenomicDataCommons">https://git.bioconductor.org/packages/GenomicDataCommons</a></td>
</tr>
<tr>
<td>git_branch</td>
<td>RELEASE_3_18</td>
</tr>
<tr>
<td>git_last_commit</td>
<td>c8ad9dd</td>
</tr>
<tr>
<td>git_last_commit_date</td>
<td>2023-10-24</td>
</tr>
<tr>
<td>Repository</td>
<td>Bioconductor 3.18</td>
</tr>
<tr>
<td>Date/Publication</td>
<td>2024-03-22</td>
</tr>
</tbody>
</table>
Author  Martin Morgan [aut],
        Sean Davis [aut, cre],
        Marcel Ramos [ctb]

Maintainer  Sean Davis <seandavi@gmail.com>

R topics documented:

- aggregations .................................................. 3
- available_expand ............................................ 4
- available_fields ............................................. 4
- available_values ............................................. 5
- count ......................................................... 6
- default_fields ................................................ 7
- endpoints ..................................................... 7
- entity_name ................................................... 8
- expand ......................................................... 9
- facet .......................................................... 10
- field_description ............................................. 11
- filtering ....................................................... 12
- GDC-deprecated ................................................ 13
- gcdata ......................................................... 15
- gdc_cache ..................................................... 16
- gdc_client ..................................................... 17
- gdc_clinical ................................................... 18
- gdc_token ..................................................... 19
- GenomicDataCommons ......................................... 20
- grep_fields ................................................... 20
- ids ............................................................ 21
- id_field ....................................................... 22
- make_filter ................................................... 23
- manifest ....................................................... 23
- mapping ........................................................ 25
- query .......................................................... 25
- readDNAcopy ................................................... 27
- readHTSeqFile ................................................ 28
- response ....................................................... 29
- results ........................................................ 30
- results_all ................................................... 31
- select .......................................................... 32
- slicing ........................................................ 33
- status .......................................................... 34
- transfer ....................................................... 35
- write_manifest ............................................... 36

Index  38
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

library(magrittr)
# Number of each file type
res = files() %>% facet(c('type','data_type')) %>% aggregations()
res$type
available_expand  
Return valid values for "expand"

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

```r
available_expand(entity)
```

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

```r
## 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


Examples

```r
head(available_expand('files'))
```
available_values

Usage

```r
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

```r
available_fields('projects')
projQuery = query('projects')
available_fields(projQuery)
```
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 provide count of records in a GDCQuery

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 GDCQuery 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

**Description**

S3 Generic to return default GDC fields

**Usage**

```r
default_fields(x)
```

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

```r
## 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**

```r
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

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

entity_name(qcases)
entity_name(qprojects)
```

---

**Description**

S3 generic to set GDCQuery expand parameter

**Usage**

```r
expand(x, expand)
```

```r
## 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**

```r
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

```r
# 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

S3 Generic that returns the field description text, if available

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 GDCQuery.

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')
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)
```

**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')

gget_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))
```
Description

The ‘legacy’ endpoint was deprecated as part of the GDC API release version ‘v3.28.0’. This means that legacy data is no longer available via the GDC API.

Usage

```r
legacy(x, legacy)
```

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

Arguments

- `x`: the objects on which to set fields
- `legacy`: logical(1) DEPRECATED: Whether or not to use the GDC legacy archives

Details

This includes the S3 generic and method for setting the legacy parameter (‘legacy’ and ‘legacy.GDCQuery’)

Value

A `GDCQuery` object with the `legacy` member altered.

Functions

- `legacy(GDCQuery)`: set legacy field on a GDCQuery object

See Also

https://docs.gdc.cancer.gov/API/Release_Notes/API_Release_Notes/#v3280

Examples

```r
qcases <- query("cases")

legacy(qcases, legacy = FALSE)
```
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

```r
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 par should be produced for each file download.
- **token** (optional) character(1) security token allowing access to restricted data. See https://gdc-docs.nci.nih.gov/API/Users_Guide/Authentication_and_Authorization/.
- **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 gdc_client and transfer_help for details.
- ... further arguments passed to files, particularly useful when requesting legacy=TRUE

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.
gdc_cache

Value

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

See Also

manifest for downloading large data.

Examples

```r
# get some example file uids
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  Work with gdc cache directory

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

```r
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.
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

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

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

```r
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
gdc_clinical

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

```r
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**

*return a gdc token from file or environment*

**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/#gdc-authentication-tokens](https://docs.gdc.cancer.gov/Data_Portal/Users_Guide/Authentication/#gdc-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


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)
**ids**

Get the ids associated with a GDC query or response

### 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 `grep` 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

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

---

**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

```r
dis(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

get the name of the id field

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**

```r
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 `GDCQuery` 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 `PUT`.

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**

```r
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**

```r
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,
  legacy = FALSE,
  expand = NULL,
  fields = default_fields(entity)
)

cases(...)

files(...)

projects(...)

annotations(...)

ssms(...)

ssm_occurrences(...)

cnvs(...)

cnv_occurrences(...)

genes(...)

Arguments

entity character vector, including one of the entities in .gdc_entities

filters a filter list, typically created using make_filter, or added to an existing GDCQuery object using filter.

facets a character vector of facets for counting common values. See available_fields. In general, one will not specify this parameter but will use facet instead.

legacy logical(1) DEPRECATED; whether to use the "legacy" archive or not.

expand a character vector of "expands" to include in returned data. See available_expand

fields a character vector of fields to return. See available_fields. In general, one will not specify fields directly, but instead use select

... passed through to query

Value

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

• filters
• facets
functions

- 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

```r
qcases = query("cases")
# equivalent to:
qcases = cases()
```

---

**readDNAcopy**

*Read DNAcopy results into GRanges object*

**description**

Read DNAcopy results into GRanges object

**usage**

`readDNAcopy(fname, ...)`

**arguments**

- `fname` The path to a DNAcopy-like file.
- `...` passed to `read_tsv`

**value**

a `GRanges` object
readHTSeqFile

Read a single htseq-counts result file.

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.
...

Value

a two-column data frame

Examples

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

dat = readHTSeqFile(fname)
head(dat)
response  

Fetch GDCQuery metadata from GDC

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]]
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)
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  

S3 generic to set GDCQuery fields

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

Query GDC for data slices

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(),
  legacy = FALSE
)

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 https://docs.gdc.cancer.gov/Data/Data_Security/Data_Security/#authentication-tokens.
legacy logical(1) DEPRECATED; whether or not to use the "legacy" archive, containing older, non-harmonized data. Slicing of unharmonized legacy BAM files is not supported. See https://docs.gdc.cancer.gov/API/Users_Guide/BAM_Slicing/.

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) = 'x'

# 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
transfer

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/, 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 transfer_help 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 https://gdc-docs.nci.nih.gov/API/Users_Guide/Authentication_and_Authorization/. 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 (>= 1.3.0).
- transfer_help():

Examples

```r
## 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_manfifest  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

```r
write_manfifest(manifest, destfile = tempfile())
```
Arguments

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

Value

character(1) the destination file name.

Examples

```r
mf = files() %>% manifest(size=10)
write_manifest(mf)
```
Index

* internal
  endpoints, 7
aggregations, 3
annotations, 20
annotations (query), 25
app_dir, 16
available_expand, 4, 26
available_fields, 4, 26
available_values, 5
cases, 20
cases (query), 25
cnv_occurrences (query), 25
cnvs (query), 25
count, 6
default_fields, 7, 10
endpoints, 7
entity_name, 8
expand, 9
facet, 10, 26
field_description, 11
files, 20
files (query), 25
filter, 26
filter (filtering), 12
filtering, 12
fromJSON, 29
GDC-deprecated, 13
gdc_cache, 16
gdc_client, 15, 17
gdc_client_version_validate (transfer), 35
gdc_clinical, 18
gdc_set_cache (gdc_cache), 16
gdc_token, 19
gdcdatal, 15
GDCQuery, 3–12, 14, 21, 23, 24, 29–32
GDCQuery (query), 25
GDCResponse, 21
GDCResponse (response), 29
genes (query), 25
GenomicDataCommons, 20
get_facets (facet), 10
get_filter (filtering), 12
GRanges, 27
grep, 20, 21
grep_fields, 20
id_field, 22
ids, 21
legacy (GDC-deprecated), 13
make_filter, 23, 26
manifest, 15, 16, 23, 36, 37
mapping, 20, 25
parameters (endpoints), 7
projects, 20
projects (query), 25
PUT, 24
query, 20, 25, 26
read_tsv, 27, 28
readDNAcopy, 27
readHTSeqFile, 28
response, 29, 30
response_all (response), 29
results, 30
results_all, 31
select, 4, 26, 32
slicing, 33
ssm_occurrences (query), 25
ssms (query), 25
status, 34
INDEX

tibble, 24
transfer, 35, 36
transfer_help, 15, 35
transfer_help(transfer), 35
write_manifest, 36