Package 'BiocAzul'

May 13, 2026

Title Programmatic Access to the Azul API

Version 1.1.3

Date 2026-05-03

Description Represents the OpenAPI v2 Azul API as an R object for performing requests. The infrastructure uses the AnVIL and rapiclient packages. Users can connect to either the AnVIL or Human Cell Atlas Data Explorers.

License Artistic-2.0

Depends R (>= 4.6.0), AnVIL

Imports AnVILPublish (>= 1.21.1), dplyr, httr, jsonlite, progress, rlang, methods, tidyr

Suggests BiocStyle, knitr, rmarkdown, tinytest

biocViews Software, Infrastructure, DataImport, ThirdPartyClient

VignetteBuilder knitr

URL https://github.com/Bioconductor/BiocAzul

BugReports https://github.com/Bioconductor/BiocAzul/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

git_url https://git.bioconductor.org/packages/BiocAzul

git_branch devel

git_last_commit 06bf71f

git_last_commit_date 2026-05-06

Repository Bioconductor 3.24

Date/Publication 2026-05-13

Author Marcel Ramos [aut, cre] (ORCID: <https://orcid.org/0000-0002-3242-0582>), NHGRI AnVIL Project [fnd] (GrantNo.: U24HG010263)

Maintainer Marcel Ramos <marcel.ramos@sph.cuny.edu>

Contents

`Azul` *The R Interface to the Human Cell Atlas Data Portal*

Description

The `Azul` class provides an interface to the Human Cell Atlas Data Portal API, allowing users to access and query the data portal programmatically. The class establishes a connection to the API and retrieves the OpenAPI specification, which defines the available endpoints and their parameters. Users can then use this connection to make requests to the API and retrieve data from the Human Cell Atlas Data Portal.

Usage

```
Azul(
  provider = c("hca", "anvil"),
  protocol = "https",
  api. = "/openapi.json",
  token = character()
)

## S4 method for signature 'Azul'
operations(x, ..., .deprecated = FALSE)
```

Arguments

`provider`	`character(1)` The data provider to connect to. Options include "hca" for the Human Cell Atlas and "anvil" for the AnVIL Data Explorer (default: "hca").
`protocol`	`character(1)` The internet protocol used to access the hostname (default: 'https')
`api.`	`character(1)` The directory location of the API protocol within the hostname (default: '/openapi.json').
`token`	`character(1)` The Authorization Bearer token e.g., "63eba81c-2591-4e15-9d1c-fb6e8e51e35d" or a path to text file.
`x`	An `Azul` instance or API representation as given by the `Azul()` function.
`...`	additional arguments passed to methods or, for `operations,Service-method`, to the internal `get_operation()` function.
`.deprecated`	optional `logical(1)` include deprecated operations?

Details

`operations`: List all the `operations` available with the Azul API object, e.g., `api$operation`

Value

An `Azul` object that can be used to interact with the Human Cell Atlas API

See Also

AnVIL::Service

Examples

```
showClass("Azul")

hca <- Azul(provider = "hca")
```

`Azul-utils` *Obtain a list of projects and their IDs from the specified catalog*

Description

This function queries the specified catalog for projects and returns a tibble with project names and their corresponding IDs. The `catalog` parameter allows you to specify which catalog to query, with options including "dcp58", "dcp59", and "lm10" (e.g., when using the Human Cell Atlas API).

Usage

```
projectTable(api, catalog)

listCatalogs(api)

availableFacets(api, catalog)

facetTable(api, facet, catalog)
```

Arguments

`api`	`Azul` object representing the connection to the API.
`catalog`	`character(1)` specifying the catalog to query. Options are given by `listCatalogs(api)`.
`facet`	`character(1)` a facet term for which to produce a table of tallies. The available facets can be obtained with `availableFacets()`.

Value

- `projectTable`: A tibble with three columns: `term`, `count`, and `projectId`. The `term` column contains the project names, the `count` column contains the number of occurrences of each project in the specified catalog, and the `projectId` column contains the unique identifiers for each project.
- `listCatalogs`: A character vector of catalog names that are available in the API.
- `availableFacets`: A character vector of facet names that are available for querying in the specified catalog.
- `facetTable`: A tibble with two columns: `term` and `count`. The `term` column contains the unique values of the specified facet, and the `count` column contains the number of occurrences of each term in the projects of the specified catalog.

Examples

```
hca <- Azul(provider = "hca")

hca_cat <- listCatalogs(hca) |>
    head(n = 1L)
projectTable(hca, catalog = hca_cat)

anvil <- Azul(provider = "anvil")

anvil_cat <- listCatalogs(anvil) |>
    head(n = 1L)
projectTable(anvil, catalog = anvil_cat)
availableFacets(hca, catalog = hca_cat)

availableFacets(anvil, catalog = anvil_cat)
facetTable(hca, "genusSpecies", hca_cat)

facetTable(anvil, "donors.organism_type", anvil_cat)
```

`importToTerra` *Import Data from the Human Cell Atlas Data Portal or the AnVIL Data Repository to a Terra Workspace*

Description

This function is intended to facilitate the import of data from the Human Cell Atlas Data Portal or the AnVIL Data Repository. It is designed to create a manifest for the specified filters and format, and then import the data into a Terra workspace. The function takes advantage of the `Azul` class to interact with the Human Cell Atlas API and the `AnVIL` package to manage the Terra workspace and import job.

Usage

```
importToTerra(api, namespace, name, filters, catalog, format = "terra.pfb")
```

Arguments

`api`	`Azul` object representing the connection to the API.
`namespace`	`character(1)` AnVIL workspace namespace as returned by, e.g., `AnVILGCP::avworkspace_namesp` where the data will be imported.
`name`	`character(1)` AnVIL workspace name as returned by, e.g., `AnVILGCP::avworkspace_name()`, where the data will be imported.
`filters`	`list()` A list of filters to apply when preparing the manifest. The filters should be structured according to the requirements of the Human Cell Atlas API, and will be converted to JSON format before being sent in the request to the API. See the details section for more information on the expected structure of the filters.
`catalog`	`character(1)` specifying the catalog to query. Options are given by `listCatalogs(api)`.
`format`	`character(1)` The format of the manifest to be prepared. Currently, only "terra.pfb" is supported, which prepares the manifest in a format suitable for import into Terra.

Details

The `filters` parameter should be a list that specifies the criteria for selecting the data to be imported.

Each filter consists of a field name, a relation (relational operator), and an array of field values. The available relations are "is", "within", "contains", and "intersects". Multiple filters are combined using "and" logic. An entity must match all filters to be included in the response. How multiple field values within a single filter are combined depends on the relation.

For the "is" relation, multiple values are combined using "or" logic. For example, `list(fileFormat = list(is = c("fastq", "fastq.gz")))` selects entities where the file format is either "fastq" or "fastq.gz". For the "within", "intersects", and "contains" relations, the field values must come in nested pairs specifying upper and lower bounds, and multiple pairs are combined using "and" logic. For example, `list(donorCount = list(within = matrix(c(1, 5, 5, 10), 2L, 2L, TRUE)))` selects entities whose donor organism count falls within both ranges, i.e., is exactly 5. The accessions field supports filtering for a specific accession and/or namespace within a project. For example,

```
list(
    accessions = list(
        is = data.frame(namespace = "array_express")
    )
)
```

will filter for projects that have an `array_express` accession. Similarly,

```
list(
    accessions = list(
        is = data.frame(accession = "ERP112843")
    )
)
```

will filter for projects that have the accession `ERP112843` while

```
list(
    accessions = list(
        is = data.frame(
            namespace = "array_express",
            accession = "E-AAAA-000"
        )
    )
)
```

will filter for projects that match both values. The `organismAge` field is special in that it contains two property keys: `value` and `unit`. For example,

```
list(
    organismAge = list(
        is = data.frame(value = "20", unit = "year")
    )
)
```

Both keys are required. `list(organismAge = list(is = NA))` selects entities that have no organism age.

Value

Called for the side effect of importing data into the specified Terra workspace. If successful, returns a list with the following elements:

- `jobId`: The unique identifier for the import job created in Terra.
- `status`: The final status of the import job, which should be "Done" if the import was successful.
- `elapsed`: The total time taken for the import job to complete, in seconds.

See Also

`makeFilter()`

Examples

```
hca <- Azul(provider = "hca")
hca_cat <- listCatalogs(hca) |>
    head(n = 1)
importToTerra(
    hca,
    namespace = "anvil-namespace",
    name = "my-anvil-workspace",
    catalog = hca_cat,
    filters = list(
        projectId = list(is = "74b6d569-3b11-42ef-b6b1-a0454522b4a0")
    )
)
```

---

makeFilter *Convert a formula to a filter list*

Description

Convert a formula to a filter list

Usage

```
makeFilter(expr)
```

Arguments

`expr`	A one-sided formula, e.g. `~ projectId == "abc"`

Value

A named list suitable for JSON serialization

Examples

```
makeFilter(~ projectId == "abc" & organism == "Homo sapiens")
makeFilter(~ donorCount %within% list(c(1, 5), c(5, 10)))
```

Index