Package: apbCatalogR (via r-universe)

November 3, 2026

Title Look Up APB Result Fields by Meaning

Version 0.99.0

Description Looks up the fields of AnnData Proteomics Bridge (APB) results by meaning, such as a PEP or an unnormalized intensity, without naming vendor columns. Reimplements the apb-catalog API on the bundle apb-catalog releases: reviewed catalogues, apb2's rule levels and expected answers. Reads APB results, as h5ad, Parquet or DuckDB, into anndataR objects.

License MIT + file LICENSE

URL https://anndata-omics-bridge.github.io/apbCatalogR/, https://github.com/anndata-omics-bridge/apbCatalogR

BugReports https://github.com/anndata-omics-bridge/apbCatalogR/issues

biocViews Proteomics, MassSpectrometry, DataImport, Infrastructure

Depends R (>= 4.5.0)

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests altdoc, arrow, BiocStyle, covr, cyclocomp, DBI, devtools, duckdb, knitr, lintr, rcmdcheck, rmarkdown, roxygen2, testthat (>= 3.0.0), withr

Config/testthat/edition 3

VignetteBuilder knitr

Imports anndataR (>= 1.2.2), jsonlite, Matrix, R6, stats, tools, utils

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libpng-dev python3

Repository https://biocstaging.r-universe.dev

Date/Publication 2026-11-03 19:10:46 UTC

RemoteUrl https://github.com/BiocStaging/apbCatalogR

RemoteRef HEAD

RemoteSha f015f97f567dbf228a46c9e5191278b79e11d511

Contents

[.Columns 2
[[.Slot 3
Catalog 3
catalog_version 5
catalogue_table 5
catalogues 6
DataCatalog 6
ParsedLevels 8
read_parsed_levels 9
rule_coverage 10
update_catalog 11

Index **12**

`[.Columns` *Read a var or obs slot by several keys*

Description

Read a var or obs slot by several keys

Usage

```
## S3 method for class 'Columns'
x[i, ...]
```

Arguments

`x`	A var or obs slot of a level.
`i`	Keys, or the level's own names.
`...`	Ignored.

Value

A data frame of every column the names name; the names without data.

[[.Slot *Read a slot by key*

Description

Read a slot by key

Usage

```
## S3 method for class 'Slot'
x[[i, ...]]
```

Arguments

`x`	A Slot.
`i`	A key, or the level's own name.
`...`	Ignored.

Value

A matrix for layers, a vector for var and obs; a name without data.

Catalog *Which column one software version reports for each meaning*

Description

Answers from the catalogue alone, without data. The rule levels whose version pattern admits the version give the levels; the set's entries for that pattern give each level's fields. `levels` have the slots of a DataCatalog's levels, holding names: a slot answers one of the set's keys, such as `"normalized_intensity"`, with the field's name, and its `$get()` takes further qualifiers, such as `unit`. Lookups that several fields answer, or that an unreviewed level cannot, signal `UnresolvedField`.

Value

The class generator; `Catalog$new()` returns a `Catalog`.

Public fields

`levels` One level per rule level, with a DataCatalog's slots holding names.

Active bindings

`description` What this catalogue set holds and who uses it.

Methods

Public methods:

- `Catalog$new()`
- `Catalog$fields()`
- `Catalog$print()`
- `Catalog$clone()`

`Catalog$new()`: Bind a catalogue set to the rule levels of one software version.

Usage:

```
Catalog$new(software, version, catalog, enforce = NULL)
```

Arguments:

`software` The software name, such as `"DIA-NN"`.

`version` The software version, such as `"2.2.0"`.

`catalog` The catalogue set, such as `"abundance"` or `"confidence"`; see `catalogues()`.

`enforce` `NULL`, `"MUST"` or `"SHOULD"`: signal `MissingRequiredField` when a field the set requires at that strength is missing.

`Catalog$fields()`: Every catalogued field the rule declares, one row per field, as `catalogue_table()` shows them.

Usage:

```
Catalog$fields()
```

Returns: A data frame, with `present` true for each row.

`Catalog$print()`: Print the set and the levels.

Usage:

```
Catalog$print(...)
```

Arguments:

`...` Ignored.

`Catalog$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
catalog <- Catalog$new("DIA-NN", "2.2.0", "abundance")
catalog$levels$peptidoform_ion$layers[["intensity"]]
catalog$levels$peptidoform_ion$layers$keys()
```

catalog_version *The catalog in use*

Description

Reports the bundle lookups answer from: its version, whether it was shipped with the package or downloaded by `update_catalog()`, and the apb2 version its rule levels come from.

Usage

```
catalog_version()
```

Value

A list with `version`, `origin` (`"shipped"` or `"cache"`), `path` and `apb2_version`.

Examples

```
catalog_version()
```

catalogue_table *The catalogue sets as one long table*

Description

One row per entry and software version it applies to, as `apb-catalog table` prints it.

Usage

```
catalogue_table(catalogue = NULL)
```

Arguments

`catalogue`	One set's name; `NULL` lists every set of the bundle.

Value

A data frame with columns `set`, `source` (the conversion software, an app or `annotation:<convention>`), `software_version_pattern` (`NA` for an entry of every software version), `level`, `location`, `table`, `name` (`prefix:<prefix>` for a field found by column-name prefix), `key`, `kind`, one `qualifier.<name>` column per other qualifier, `requirement` and `entry_id`.

Examples

```
head(catalogue_table("confidence"))
```

catalogues *The catalogue sets*

Description

Every catalogue set of the bundle with its purpose, its users and its number of entries.

Usage

```
catalogues()
```

Value

A data frame with columns `catalogue`, `purpose`, `used_by` (a list column), `has_requirements` and `entries`.

Examples

```
catalogues()
```

DataCatalog *What one result's fields mean to one consumer*

Description

Answers on a result read with `read_parsed_levels()` and hands the data back directly: `levels` give each level's slots, looked up by one of the set's keys, such as `"q_value"` or Sage's `"quantification_q_value"`. Ambiguous or unreviewed answers signal `UnresolvedField`.

Value

The class generator; `DataCatalog$new()` returns a `DataCatalog`.

Public fields

`levels` One Level per level of the result, with anndataR's slots looked up by key.

Active bindings

`description` What this catalogue set holds and who uses it.

Methods

Public methods:

- DataCatalog$new()
- DataCatalog$fields()
- DataCatalog$print()
- DataCatalog$clone()

`DataCatalog$new()`: Bind catalogue sets to the levels of one result; `apb`, APB's own fields, is always in.

Usage:

```
DataCatalog$new(result, ..., enforce = NULL)
```

Arguments:

`result` A ParsedLevels object.

`...` The catalogue sets, such as `"abundance"` and `"confidence"`; see `catalogues()`.

`enforce` `NULL`, `"MUST"` or `"SHOULD"`: signal `MissingRequiredField` when a field a set requires at that strength is missing.

`DataCatalog$fields()`: Every catalogued field of the result in every composed set, one row per field, as `catalogue_table()` shows them.

Usage:

```
DataCatalog$fields()
```

Returns: A data frame with `present` and, for a field apb2 left out because it held one value, that `constant`.

`DataCatalog$print()`: Print the set and the levels.

Usage:

```
DataCatalog$print(...)
```

Arguments:

`...` Ignored.

`DataCatalog$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
results <- tempfile()
archive <- system.file(
    "extdata", "test-bundle", "results.tar.gz",
    package = "apbCatalogR"
)
utils::untar(archive, exdir = results)
```

```
result <- read_parsed_levels(
    file.path(results, "maxquant_wide-peptidoform_ion.parquet")
)
catalog <- DataCatalog$new(result, "confidence")
dim(catalog$levels$peptidoform_ion$layers[["pep"]])
catalog$levels$peptidoform_ion$layers$name("pep")
```

`ParsedLevels` *One APB result: its levels on one observation axis*

Description

The R counterpart of apb2's `ParsedLevels`, as `read_parsed_levels()` returns it.

Each level is an anndataR `AnnData` laid out as apb2's `.h5ad` writer lays it out: the primary layer in `X`, the other layers in `layers`, obs and var frames named by their key columns, and the level's APB part in `uns$apb`: `parse` (with `provenance$rule_json`), `roles`, `storage` and post-parse records.

Value

The class generator; `ParsedLevels$new()` returns a `ParsedLevels` object.

Public fields

`levels` A named list of anndataR `AnnData`, one per level.

`uns` The root parse record.

`metadata` The root post-parse records, such as `sdrf` or `fasta`.

`hierarchy` How the levels nest, or `NULL`.

`annotation_tables` A named list of data frames; each carries its metadata as the attribute `apb`.

`feature_relations` A named list of data frames of `source`, `row`, `target`, `column` and `value`; each carries its metadata as the attribute `apb`.

Methods

Public methods:

- `ParsedLevels$new()`
- `ParsedLevels$print()`
- `ParsedLevels$clone()`

`ParsedLevels$new()`: Collect the levels and records of one result.

Usage:

```
ParsedLevels$new(
  levels,
  uns = list(),
  metadata = list(),
  hierarchy = NULL,
```

```
  annotation_tables = list(),
  feature_relations = list()
)
```

Arguments:

`levels` A named list of anndataR `AnnData`.

`uns` The root parse record.

`metadata` The root post-parse records.

`hierarchy` How the levels nest, or `NULL`.

`annotation_tables` A named list of data frames.

`feature_relations` A named list of data frames.

`ParsedLevels$print()`**:** Print the levels and their shapes.

Usage:

```
ParsedLevels$print(...)
```

Arguments:

`...` Ignored.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
results <- tempfile()
archive <- system.file(
    "extdata", "test-bundle", "results.tar.gz",
    package = "apbCatalogR"
)
utils::untar(archive, exdir = results)
result <- read_parsed_levels(file.path(results, "edge-two-levels.parquet"))
names(result$levels)
result$levels$protein
```

`read_parsed_levels` *Read an APB result*

Description

Reads a result apb2 wrote into a ParsedLevels object, one anndataR `AnnData` per level, the same for every format: a `.h5ad` file, a `.duckdb` file, or the directory apb2 writes for a `.parquet` result. Reading Parquet needs the `arrow` package, DuckDB the `duckdb` and `DBI` packages; `.h5ad` needs nothing beyond anndataR.

Usage

```
read_parsed_levels(path)
```

Arguments

path	The result, such as converted.h5ad, converted.duckdb or the directory converted.parquet.

Value

A ParsedLevels object.

Examples

```
results <- tempfile()
archive <- system.file(
    "extdata", "test-bundle", "results.tar.gz",
    package = "apbCatalogR"
)
utils::untar(archive, exdir = results)
read_parsed_levels(file.path(results, "edge-two-levels.parquet"))
```

---

rule_coverage *Check every rule against one catalogue set*

Description

One row per catalogued or required field of each rule level the set reviews; a set without vendor catalogues is checked against every level the bundle's entries name for each version of versions.csv. A set with requirements judges each level: fail when a MUST field is unsupported, warn when a SHOULD or MAY field is, pass otherwise.

Usage

```
rule_coverage(catalogue)
```

Arguments

catalogue	The set's name.

Value

A data frame with columns software, version (the label), software_version_pattern, level, key, kind, qualifiers (the other qualifiers as name=value), origin, producer (the app, NA for the conversion software's fields), requirement, location, table, column (NA when unsupported), needs (what to run before the field exists) and verdict (NA for a set without requirements).

Examples

```
head(rule_coverage("abundance"))
```

`update_catalog` *Download a newer catalog*

Description

Downloads the bundle of an apb-catalog release from GitHub into the user cache, so lookups use it without a package update. Later sessions keep using the newest bundle, shipped or downloaded. It works from the next apb-catalog release, the first to ship the bundle.

Usage

```
update_catalog(version = NULL)
```

Arguments

`version`	The apb-catalog release, such as `"0.2.0"`; `NULL` downloads the latest release.

Value

The `catalog_version()` now in use, invisibly.

Examples

```
if (interactive()) {
  update_catalog()
}
```

Index

[.Columns, 2
[[.Slot, 3

Catalog, 3
catalog_version, 5
catalog_version(), *11*
catalogue_table, 5
catalogue_table(), *4*, *7*
catalogues, 6
catalogues(), *4*, *7*

DataCatalog, *3*, 6

Level, *6*

ParsedLevels, *7*, 8, *9*, *10*

read_parsed_levels, 9
read_parsed_levels(), *6*, *8*
rule_coverage, 10

Slot, *3*

update_catalog, 11
update_catalog(), *5*