Package: apbCatalogR (via r-universe)

November 12, 2026

Title Look Up APB Result Fields by Meaning

Version 0.99.4

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 sets, 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, rhdf5, 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://bioc.r-universe.dev

Date/Publication 2026-11-12 14:42:02 UTC

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

RemoteRef HEAD

RemoteSha 8a4daab91ee6823ddcb8a0bb145f0a01722e81c1

Contents

[.Columns 2
[[.Slot 3
Catalog 3
catalog_table 5
catalog_version 5
Columns 6
DataCatalog 7
Level 9
LevelNames 10
ParsedLevels 11
read_parsed_levels 12
rule_coverage 13
sets 14
Slot 15
update_catalog 16

Index **18**

`[.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 catalog 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 `"intensity_norm_settings"`, 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`. `enforce` checks the set's requirements against the rule's levels once, when the catalog opens.

Value

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

Public fields

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

Active bindings

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

Methods

Public methods:

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

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

Usage:

```
Catalog$new(software, software_version, set_name, enforce = NULL)
```

Arguments:

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

`software_version` The software's version, such as `"2.2.0"`.

`set_name` The set, such as `"abundance_alpha"` or `"confidence_alpha"`; see `sets()`.

`enforce` `NULL`, `"required"` or `"recommended"`: signal one `MissingRequiredField` listing every field the set requires at that strength and the rule lacks; under `"required"`, missing recommended fields give one warning. An error when the set has no requirements.

`Catalog$fields()`: Every cataloged field the rule declares, one row per field, as `catalog_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$new("DIA-NN", "2.2.0", "abundance_alpha")
catalog$levels$peptidoform_ion$layers[["intensity"]]
catalog$levels$peptidoform_ion$layers$keys()
```

catalog_table *The catalog as one long table*

Description

The whole catalog, every set of the bundle, or one set: one row per entry and software version it applies to, as `apb-catalog table` prints it.

Usage

```
catalog_table(set_name = NULL)
```

Arguments

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

Value

A data frame with columns `set`, `source` (who writes the field: the quantification software, an app or `annotation:<convention>`), `version` (the version label; `*` for an app's or an annotation's entry, which applies to every version), `software_version_pattern` (`NA` for an entry of every software version), `level`, `slot` (`layers`, `var`, `obs` or `varm`), `table` (the `varm` table, else `NA`), `name` (the field's 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(catalog_table("confidence_alpha"))
```

catalog_version *The catalog in use*

Description

Reports the bundle lookups answer from: its version, whether it was shipped with the package or installed 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()
```

`Columns` *A var or obs slot, also read by several names*

Description

`obs[c("group")]` holds every column the key names, each under its own name, where `obs[["group"]]` signals `UnresolvedField` because there are several.

Value

The class generator, not exported: a Level's `var` and `obs` are its objects.

Super class

`Slot` -> `Columns`

Methods

Public methods:

- `Columns$new()`
- `Columns$get_all()`
- `Columns$clone()`

`Columns$new()`**:** Bind one var or obs slot of a level.

Usage:

```
Columns$new(level, location, lookup, read, named, combine)
```

Arguments:

`level` The level's name.

`location` `"var"` or `"obs"`.

`lookup` How the catalog answers keys: `find`, `find_all`, `offered`, `known` and `sets`, one lookup per set prefix.

`read` The value of a field the slot holds.

`named` The level's own names in this slot.

`combine` The fields read, a named list, as one value.

`Columns$get_all()`**:** Every field the names name, in order; an error for a name that is neither a key nor the level's own.

Usage:

```
Columns$get_all(names)
```

Arguments:

`names` Keys, or the level's own names.

Returns: A data frame; names without data.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`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`.

Composes the named sets alone: `apb_alpha`, the fields apb2 convert writes, answers only when named. A key routes to the sets that define it; two sets naming different fields for one key on one level is an error. With no set named, every name is the level's own. A named set that can answer nothing on the result, because its app has not run or the result records no sample annotation, is an error saying what to run. `enforce` checks the requirements once, here.

`DataCatalog$answering_sets(result)` returns the sets `DataCatalog$new(result, name)` opens, as a character vector: every set but one whose app has not run on the result or whose annotation it does not record. It signals no error for such a set, so a consumer can leave it out.

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 the first set named holds and who uses it; an error when no set was named.

Methods

Public methods:

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

`DataCatalog$new()`**:** Bind the named sets, and no others, to the levels of one result.

Usage:

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

Arguments:

`result` A ParsedLevels object.

`...` The sets, such as `"abundance_alpha"` and `"confidence_alpha"`; see `sets()`.

`enforce` `NULL`, `"required"` or `"recommended"`: signal one `MissingRequiredField` listing every field the named sets require at that strength and the result lacks; under `"required"`, missing recommended fields give one warning. An error when no named set has requirements.

`DataCatalog$fields()`**:** Every cataloged field of the result in every composed set, one row per field, as `catalog_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$new(result, "confidence_alpha")
dim(catalog$levels$peptidoform_ion$layers[["pep"]])
catalog$levels$peptidoform_ion$layers$name("pep")
```

`Level` *One level of a result, read through a catalog*

Description

anndataR's slots of one level: `X`, `layers`, `var`, `obs`, `obs_names` and `var_names`. `X` is the level's primary layer and `layers` its other layers, observations by variables; a key may name the primary layer, and then answers `X`. `cat$levels$protein$layers[["intensity"]]` looks the name up as one of the catalog's keys first, then as the level's own name in that slot, and signals an error when it is neither; a prefix makes the lookup exact, `"c:q_value"` the confidence set alone, `"v:q_value"` the level's own name alone. `obs[c("group", "Run")]` reads every column the names name as a data frame.

Value

The class generator, not exported: a DataCatalog's `levels` hold its objects, which the catalog creates.

Public fields

`name` The level's name.

`layers` The level's layers, read with `[[` or `$get()`; `$name()` names the field, `$keys()` lists the keys it offers.

`var` The level's per-feature columns, from `var` or a `varm` table, read with `[[`, `[` or `$get()`.

`obs` The level's per-sample columns, read with `[[` or `[`.

Active bindings

`X` The primary layer, observations by variables.

`obs_names` One name per observation: its key columns, joined by `|`.

`var_names` One name per variable: its key columns, joined by `|`.

Methods

Public methods:

- `Level$new()`
- `Level$clone()`

`Level$new()`: Bind one level to the catalog's lookups.

Usage:

```
Level$new(name, level, lookup)
```

Arguments:

`name` The level's name.

`level` The level, an anndataR `AnnData`.

`lookup` How the catalog answers keys: `find`, `find_all`, `offered`, `known` and `sets`, one lookup per set prefix.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`LevelNames` *One level of a software version's rule, holding names*

Description

The slots of a Level without data: `layers`, `var` and `obs` answer a key with the field's name, a `varm` column as `table/column`. There is no `X`, `obs_names`, `var_names` or own name: a name that is no key the level answers signals an error.

Value

The class generator, not exported: a Catalog's `levels` hold its objects, which the catalog creates.

Public fields

`name` The level's name.

`layers` The level's layer names, read with `[[`; `$keys()` lists the keys it offers.

`var` The level's per-feature column names, read with `[[` or `[`.

`obs` The level's per-sample column names, read with `[[` or `[`.

Methods

Public methods:

- `LevelNames$new()`
- `LevelNames$clone()`

`LevelNames$new()`: Bind one rule level to the catalog's lookups.

Usage:

```
LevelNames$new(name, lookup)
```

Arguments:

`name` The level's name.

`lookup` How the catalog answers keys: `find`, `find_all`, `offered`, `known` and `sets`, one lookup per set prefix.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`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` the `rhdf5` package, with which anndataR reads it.

Usage

```
read_parsed_levels(path)
```

Arguments

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

Value

A ParsedLevels object.

Input layout

apb2's convert command writes the result; each format records its format name and version, and a result of another format or version is refused:

- **Parquet**: a directory ending in .parquet whose manifest.json (format apb2-parsed-levels-parquet, version 9) names every level and table. Each level is a folder under levels/ with obs.parquet, var.parquet, one file per layer in layers/ (features by observations) and the tables of obsm/, varm/, obsp/ and varp/. The result's own tables are in annotation_tables/ and feature_relations/.
- **DuckDB**: one .duckdb file (format apb2-parsed-levels-duckdb, version 8) holding the same manifest in the table apb2_result_metadata; each table the manifest names is a database table.
- **h5ad**: one .h5ad file of a one-level result in anndata's layout (format apb2-parsed-levels, version 8), with apb2's storage descriptor in uns$apb. An .h5ad file other software wrote has none and is refused.

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 set*

Description

One row per cataloged or required field of each rule level the set reviews; a set without vendor entries 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 required field is unsupported, warn when a recommended one is, pass otherwise; an unsupported optional field is listed but passes.

Usage

```
rule_coverage(set_name)
```

Arguments

set_name	The set's name.

Value

A data frame with columns software (the quantification software of the rule), version (the label), software_version_pattern, level, slot, table (the varm table, else NA), name (the field's name, NA when unsupported), key, kind, one qualifier.<name> column per other qualifier of the set, as in catalog_table(), producer (the app, NA for the quantification software's fields), requirement, needs (what to run before the field exists) and verdict (NA for a set without requirements).

Examples

```
head(rule_coverage("abundance_alpha"))
```

---

sets *The catalog's sets*

Description

Every set of the bundle with its prefix, its purpose, its users and its number of entries. The prefix asks the set alone, as "c:q_value".

Usage

```
sets()
```

Value

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

Examples

```
sets()
```

`Slot` *One slot of a level, read by key first*

Description

`[[` reads a key, else the level's own name; `$keys()` lists the catalog's keys only. `$get()` returns `NULL` where `[[` signals an error, `$name()` the field's name; both take the qualifiers a key leaves open. A set's prefix, `"c:q_value"`, asks that set alone and never the level's own name; `"v:q_value"` asks the level's own name alone. The design follows the Python package cf_xarray, whose accessor finds a variable by its CF-convention standard name before its own name.

Value

The class generator, not exported: a Level's `layers` is its object.

Methods

Public methods:

- `Slot$new()`
- `Slot$get()`
- `Slot$name()`
- `Slot$keys()`
- `Slot$clone()`

`Slot$new()`**:** Bind one slot of a level.

Usage:

```
Slot$new(level, location, lookup, read, named)
```

Arguments:

`level` The level's name.

`location` `"layers"`, `"var"` or `"obs"`.

`lookup` How the catalog answers keys: `find`, `find_all`, `offered`, `known` and `sets`, one lookup per set prefix.

`read` The value of a field the slot holds.

`named` The level's own names in this slot.

`Slot$get()`**:** The field a key names, narrowed by further qualifiers, else the level's own field of that name.

Usage:

```
Slot$get(key, ...)
```

Arguments:

`key` A key, or the level's own name.

`...` Further qualifiers by name, such as `unit = "minute"`.

Returns: A matrix for layers, a vector for var and obs, a name without data; `NULL` when it is neither.

`Slot$name()`: The name of the field `get` reads; an error when the name is neither a key the level answers nor one of its own names.

Usage:

```
Slot$name(key, ...)
```

Arguments:

`key` A key, or the level's own name.

`...` Further qualifiers by name, such as `unit = "minute"`.

Returns: The field's name, such as `"PEP"`; `table/column` for a column of a `varm` table.

`Slot$keys()`: The keys the level offers in this slot.

Usage:

```
Slot$keys()
```

Returns: A character vector.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`update_catalog` *Install a catalog bundle*

Description

Installs a bundle archive into the user cache, so lookups use it without a package update. apb-catalog's `make bundle` writes the archive, as `build/apb-catalog-bundle.tar.gz`. Later sessions keep using the newest bundle, shipped or installed. Nothing is downloaded.

Usage

```
update_catalog(path)
```

Arguments

`path`	Path to the bundle archive, a `.tar.gz` file.

Value

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

Examples

```
if (interactive()) {
    update_catalog("apb-catalog/build/apb-catalog-bundle.tar.gz")
}
```

Index

[.Columns, 2
[[.Slot, 3

Catalog, 3, *10*
catalog_table, 5
catalog_table(), *4*, *8*, *14*
catalog_version, 5
catalog_version(), *16*
Columns, 6

DataCatalog, *3*, 7, *9*

Level, *6, 7*, 9, *10*, *15*
LevelNames, *3*, 10

ParsedLevels, *8*, 11, *12, 13*

read_parsed_levels, 12
read_parsed_levels(), *7*, *11*
rule_coverage, 13

sets, 14
sets(), *4*, *8*
Slot, *3*, *6*, 15

update_catalog, 16
update_catalog(), *5*