Package ‘laminr’

January 8, 2025

Title Client for 'LaminDB'

Version 0.3.1

Description Interact with 'LaminDB'. 'LaminDB' is an open-source data framework for biology. This package allows you to query and download data from 'LaminDB' instances.

License Apache License (>= 2)

URL https://laminr.lamin.ai, https://github.com/laminlabs/laminr

BugReports https://github.com/laminlabs/laminr/issues

Depends R (>= 4.0.0)

Imports cli, httr, jsonlite, purrr, R.utils, R6, reticulate, rlang, tibble

Suggests anndata, knitr, nanoparquet, quarto, readr, rstudioapi, rsvg, s3 (>= 1.1.0), Seurat, testthat (>= 3.0.0), withr, yaml

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Robrecht Cannoodt [aut, cre] (<https://orcid.org/0000-0003-3641-729X>),
Luke Zappia [aut] (<https://orcid.org/0000-0001-7744-8565>),
Data Intuitive [aut],
Lamin Labs [aut, cph]

Maintainer Robrecht Cannoodt <robrecht@data-intuitive.com>

Repository CRAN

Date/Publication 2025-01-08 11:30:07 UTC

Contents

`connect` *Connect to instance*

Description

Note that prior to connecting to an instance, you need to authenticate with `lamin login`. If no slug is provided, the default instance is loaded, which is set by running `lamin connect <slug>`.

Usage

```
connect(slug = NULL)
```

Arguments

`slug`	The instance slug `account_handle/instance_name` or URL. If the instance is owned by you, it suffices to pass the instance name. If no slug is provided, the default instance is loaded.

Examples

```
## Not run:
# first run 'lamin login' to authenticate
instance <- connect("laminlabs/cellxgene")
instance

## End(Not run)
```

`Field` *Field*

Description

A field in a registry.

Active bindings

`type` (`character(1)`)
: The type of the field.

`through` (`list()` or `NULL`)
: The through value of the field.

`field_name` (`character(1)`)
: The field name.

`registry_name` (`character(1)`)
: The registry name.

`column_name` (`character(1)`)
: The column name.

`module_name` (`character(1)`)
: The module name.

`is_link_table` (`logical(1)`)
: Whether the field is a link table.

`relation_type` (`character(1)` or `NULL`)
: The relation type. Can be one of: "one-to-many", "many-to-one", "many-to-many".

`related_field_name` (`character(1)` or `NULL`)
: The related field name.

`related_registry_name` (`character(1)` or `NULL`)
: The related registry name.

`related_module_name` (`character(1)` or `NULL`)
: The related module name.

Methods

Public methods:

- `Field$new()`
- `Field$print()`
- `Field$to_string()`

Method `new()`**:** Creates an instance of this R6 class. This class should not be instantiated directly, but rather by connecting to a LaminDB instance using the `connect()` function.

Usage:

```
Field$new(
  type,
  through,
  field_name,
  registry_name,
  column_name,
  module_name,
  is_link_table,
  relation_type,
  related_field_name,
  related_registry_name,
  related_module_name
)
```

Arguments:

`type` The type of the field. Can be one of: "IntegerField", "JSONField", "OneToOneField", "SmallIntegerField", "BigIntegerField", "AutoField", "BigAutoField", "BooleanField", "TextField", "DateTimeField", "ManyToManyField", "CharField", "ForeignKey"

`through` If the relation type is one-to-many, many-to-one, or many-to-many, This value will be a named list with keys 'left_key', 'right_key', 'link_table_name'.

`field_name` The name of the field in the registry. Example: `"name"`.

`registry_name` The name of the registry. Example: `"user"`.

`column_name` The name of the column in the database. Example: `"name"`.

`module_name` The name of the module. Example: `"core"`.

`is_link_table` Whether the field is a link table.

`relation_type` The type of relation. Can be NULL or one of: "one-to-one", "many-to-one", "many-to-many".

`related_field_name` The name of the related field in the related registry. Example: `"name"`.

`related_registry_name` The name of the related registry. Example: `"user"`.

`related_module_name` The name of the related module. Example: `"core"`.

Method `print()`: Print a `Field`

Usage:

```
Field$print(style = TRUE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Method `to_string()`: Create a string representation of a `Field`

Usage:

```
Field$to_string(style = FALSE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A `cli::cli_ansi_string` if `style = TRUE` or a character vector

`install_lamindb` *Install LaminDB*

Description

Create a Python environment containing **lamindb** or install **lamindb** into an existing environment.

Usage

```
install_lamindb(
  ...,
  envname = "r-lamindb",
  extra_packages = NULL,
  new_env = identical(envname, "r-lamindb")
)
```

Arguments

`...`	Additional arguments passed to `reticulate::py_install()`
`envname`	String giving the name of the environment to install packages into
`extra_packages`	A vector giving the names of additional Python packages to install
`new_env`	Whether to remove any existing `virtualenv` with the same name before creating a new one with the requested packages

Details

See `vignette("setup", package = "laminr")` for further details on setting up a Python environment

Value

`NULL`, invisibly

Examples

```
## Not run:
install_lamindb()

# Add additional packages to the environment
install_lamindb(extra_packages = c("bionty", "wetlab"))

# Install into a different environment
install_lamindb(envvname = "your-env")

## End(Not run)
```

`Instance` *Instance*

Description

Connect to a LaminDB instance using the `connect()` function. The instance object provides access to the modules and registries of the instance.

Details

Note that by connecting to an instance via `connect()`, you receive a "richer" version of the Instance class documented here, providing direct access to all core registries and additional modules. See the vignette on "Package Architecture" for more information: `vignette("architecture", package = "laminr")`.

Active bindings

`is_default (logical(1))`
Whether this is the default instance.

Methods

Public methods:

- `Instance$new()`
- `Instance$get_modules()`
- `Instance$get_module()`
- `Instance$get_module_names()`
- `Instance$get_settings()`
- `Instance$get_api()`
- `Instance$get_py_lamin()`
- `Instance$track()`
- `Instance$finish()`
- `Instance$print()`
- `Instance$to_string()`

Method `new()`**:** Creates an instance of this R6 class. This class should not be instantiated directly, but rather by connecting to a LaminDB instance using the `connect()` function.

Usage:

```
Instance$new(settings, api, schema, is_default, py_lamin)
```

Arguments:

`settings` The settings for the instance
`api` The API for the instance
`schema` The schema for the instance
`is_default` Logical, whether this is the default instance

`py_lamin` A Python `lamindb` module object

Method `get_modules()`**:** Get the modules for the instance.

Usage:

```
Instance$get_modules()
```

Returns: A list of Module objects.

Method `get_module()`**:** Get a module by name.

Usage:

```
Instance$get_module(module_name)
```

Arguments:

`module_name` The name of the module.

Returns: The Module object.

Method `get_module_names()`**:** Get the names of the modules. Example: `c("core", "bionty")`.

Usage:

```
Instance$get_module_names()
```

Returns: A character vector of module names.

Method `get_settings()`**:** Get instance settings.
Note: This method is intended for internal use only and may be removed in the future.

Usage:

```
Instance$get_settings()
```

Returns: The settings for the instance.

Method `get_api()`**:** Get instance API.
Note: This method is intended for internal use only and may be removed in the future.

Usage:

```
Instance$get_api()
```

Returns: The API for the instance.

Method `get_py_lamin()`**:** Get the Python lamindb module

Usage:

```
Instance$get_py_lamin(check = FALSE, what = "This functionality")
```

Arguments:

`check` Logical, whether to perform checks
`what` What the python module is being requested for, used in check messages

Returns: Python lamindb module.

Method `track()`**:** Start a run with tracked data lineage

Usage:

```
Instance$track(transform = NULL, path = NULL)
```

Arguments:

`transform` UID specifying the data transformation

`path` Path to the R script or document to track

Details: Calling `track()` with `transform = NULL` with return a UID, providing that UID with the same path with start a run

Method `finish()`**:** Finish a tracked run

Usage:

```
Instance$finish()
```

Method `print()`**:** Print an `Instance`

Usage:

```
Instance$print(style = TRUE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Method `to_string()`**:** Create a string representation of an `Instance`

Usage:

```
Instance$to_string(style = FALSE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A `cli::cli_ansi_string` if `style = TRUE` or a character vector

Examples

```
## Not run:
# Connect to an instance
db <- connect("laminlabs/cellxgene")

# fetch an artifact
artifact <- db$Artifact$get("MkRm3eUKPwfnAyZMWD9v")

# describe the artifact
artifact$describe()

# view field
artifact$id

# load dataset
artifact$load()

## End(Not run)
```

`lamin_connect` *Set the default LaminDB instance*

Description

Set the default LaminDB instance by calling `lamin connect` on the command line

Usage

```
lamin_connect(slug)
```

Arguments

`slug`	Slug giving the instance to connect to (`<owner>/<name>`)

Examples

```
## Not run:
lamin_connect("laminlabs/cellxgene")

## End(Not run)
```

`lamin_login` *Login to LaminDB*

Description

Login as a LaminDB user

Usage

```
lamin_login(user = NULL, api_key = NULL)
```

Arguments

`user`	Handle for the user to login as
`api_key`	API key for a user

Details

Setting `user` will run `lamin login <user>`. Setting `api_key` will set the `LAMIN_API_KEY` environment variable tempoarily with `withr::with_envvar()` and run `lamin login`. If neither `user` or `api_key` are set `lamin login` will be run if `LAMIN_API_KEY` is set.

`Module` *Module*

Description

A LaminDB module containing one or more registries.

Active bindings

`name` (`character(1)`)
Get the name of the module.

Methods

Public methods:

- `Module$new()`
- `Module$get_registries()`
- `Module$get_registry()`
- `Module$get_registry_names()`
- `Module$print()`
- `Module$to_string()`

Method `new()`**:** Creates an instance of this R6 class. This class should not be instantiated directly, but rather by connecting to a LaminDB instance using the `connect()` function.

Usage:

```
Module$new(instance, api, module_name, module_schema)
```

Arguments:

`instance` The instance the module belongs to.
`api` The API for the instance.
`module_name` The name of the module.
`module_schema` The schema of the module.

Method `get_registries()`**:** Get the registries in the module.

Usage:

```
Module$get_registries()
```

Returns: A list of Registry objects.

Method `get_registry()`**:** Get a registry by name.

Usage:

```
Module$get_registry(registry_name)
```

Arguments:

`registry_name` The name of the registry.

Returns: A Registry object.

Method `get_registry_names()`**:** Get the names of the registries in the module. E.g. `c("User", "Artifact")`.

Usage:

```
Module$get_registry_names()
```

Returns: A character vector of registry names.

Method `print()`**:** Print a `Module`

Usage:

```
Module$print(style = TRUE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes.

Method `to_string()`**:** Create a string representation of a `Module`

Usage:

```
Module$to_string(style = FALSE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A `cli::cli_ansi_string` if `style` = `TRUE` or a character vector

`Record` *Record*

Description

A record from a registry.

Methods

Public methods:

- `Record$new()`
- `Record$delete()`
- `Record$print()`
- `Record$to_string()`

Method `new()`**:** Creates an instance of this R6 class. This class should not be instantiated directly, but rather by connecting to a LaminDB instance using the `connect()` function.

Usage:

```
Record$new(instance, registry, api, data)
```

Arguments:

`instance` The instance the record belongs to.

`registry` The registry the record belongs to.

`api` The API for the instance.

`data` The data for the record.

Method `delete()`**:** Delete a `Record`

Usage:

```
Record$delete(verbose = FALSE)
```

Arguments:

`verbose` Whether to print details of the API call

Returns: `TRUE` invisibly if the deletion is successful

Method `print()`**:** Print a `Record`

Usage:

```
Record$print(style = TRUE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Method `to_string()`**:** Create a string representation of a `Record`

Usage:

```
Record$to_string(style = FALSE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A `cli::cli_ansi_string` if `style` = `TRUE` or a character vector

`Registry` *Registry*

Description

A registry in a module.

Active bindings

`module` (Module)
The instance the registry belongs to.

`name` `(character(1))`
The API for the instance.

`class_name` `(character(1))`
The class name for the registry.

`is_link_table` `(logical(1))`
Whether the registry is a link table.

Methods

Public methods:

- `Registry$new()`
- `Registry$get()`
- `Registry$df()`
- `Registry$from_df()`
- `Registry$from_path()`
- `Registry$from_anndata()`
- `Registry$get_fields()`
- `Registry$get_field()`
- `Registry$get_field_names()`
- `Registry$get_record_class()`
- `Registry$get_temporary_record_class()`
- `Registry$print()`
- `Registry$to_string()`

Method `new()`**:** Creates an instance of this R6 class. This class should not be instantiated directly, but rather by connecting to a LaminDB instance using the `connect()` function.

Usage:

```
Registry$new(instance, module, api, registry_name, registry_schema)
```

Arguments:

`instance` The instance the registry belongs to.

`module` The module the registry belongs to.

`api` The API for the instance.

`registry_name` The name of the registry.

`registry_schema` The schema for the registry.

Method `get()`**:** Get a record by ID or UID.

Usage:

```
Registry$get(id_or_uid, include_foreign_keys = FALSE, verbose = FALSE)
```

Arguments:

`id_or_uid` The ID or UID of the record.

`include_foreign_keys` Logical, whether to include foreign keys in the record.

`verbose` Logical, whether to print verbose output.

Returns: A Record object.

Method `df()`**:** Get a data frame summarising records in the registry

Usage:

```
Registry$df(limit = 100, verbose = FALSE)
```

Arguments:

`limit` Maximum number of records to return

`verbose` Boolean, whether to print progress messages

Returns: A data.frame containing the available records

Method `from_df()`**:** Create a record from a data frame

Usage:

```
Registry$from_df(dataframe, key = NULL, description = NULL, run = NULL)
```

Arguments:

`dataframe` The `data.frame` to create a record from

`key` A relative path within the default storage

`description` A string describing the record

`run` A `Run` object that creates the record

Details: Creating records is only possible for the default instance, requires the Python `lamindb` module and is only implemented for the core `Artifact` registry.

Returns: A `TemporaryRecord` object containing the new record. This is not saved to the database until `temp_record$save()` is called.

Method `from_path()`**:** Create a record from a path

Usage:

```
Registry$from_path(path, key = NULL, description = NULL, run = NULL)
```

Arguments:

`path` Path to create a record from

`key` A relative path within the default storage

`description` A string describing the record

`run` A `Run` object that creates the record

Details: Creating records is only possible for the default instance, requires the Python `lamindb` module and is only implemented for the core `Artifact` registry.

Returns: A `TemporaryRecord` object containing the new record. This is not saved to the database until `temp_record$save()` is called.

Method `from_anndata()`**:** Create a record from an `AnnData`

Usage:

```
Registry$from_anndata(adata, key = NULL, description = NULL, run = NULL)
```

Arguments:

`adata` The anndata::AnnData object to create a record from

`key` A relative path within the default storage

`description` A string describing the record

`run` A `Run` object that creates the record

Details: Creating records is only possible for the default instance, requires the Python `lamindb` module and is only implemented for the core `Artifact` registry.

Returns: A `TemporaryRecord` object containing the new record. This is not saved to the database until `temp_record$save()` is called.

Method `get_fields()`**:** Get the fields in the registry.

Usage:

```
Registry$get_fields()
```

Returns: A list of Field objects.

Method `get_field()`**:** Get a field by name.

Usage:

```
Registry$get_field(field_name)
```

Arguments:

`field_name` The name of the field.

Returns: A Field object.

Method `get_field_names()`**:** Get the field names in the registry.

Usage:

```
Registry$get_field_names()
```

Returns: A character vector of field names.

Method `get_record_class()`**:** Get the record class for the registry.
Note: This method is intended for internal use only and may be removed in the future.

Usage:

```
Registry$get_record_class()
```

Returns: A Record class.

Method `get_temporary_record_class()`**:** Get the temporary record class for the registry.
Note: This method is intended for internal use only and may be removed in the future.

Usage:

```
Registry$get_temporary_record_class()
```

Returns: A `TemporaryRecord` class.

Method `print()`**:** Print a `Registry`

Usage:

```
Registry$print(style = TRUE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A character vector

Method `to_string()`**:** Create a string representation of a `Registry`

Usage:

```
Registry$to_string(style = FALSE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A `cli::cli_ansi_string` if `style` = `TRUE` or a character vector

`RelatedRecords` *RelatedRecords*

Description

A container for accessing records with a one-to-many or many-to-many relationship.

Methods

Public methods:

- `RelatedRecords$new()`
- `RelatedRecords$df()`
- `RelatedRecords$print()`
- `RelatedRecords$to_string()`

Method `new()`**:** Creates an instance of this R6 class. This class should not be instantiated directly, but rather by connecting to a LaminDB instance using the `connect()` function.

Usage:

```
RelatedRecords$new(instance, registry, field, related_to, api)
```

Arguments:

`instance` The instance the records list belongs to.
`registry` The registry the records list belongs to.
`field` The field associated with the records list.
`related_to` ID or UID of the parent that records are related to.
`api` The API for the instance.

Method `df()`**:** Get a data frame summarising records in the registry

Usage:

```
RelatedRecords$df(limit = 100, verbose = FALSE)
```

Arguments:

`limit` Maximum number of records to return
`verbose` Boolean, whether to print progress messages

Returns: A data.frame containing the available records

Method `print()`**:** Print a `RelatedRecords`

Usage:

```
RelatedRecords$print(style = TRUE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Method `to_string()`**:** Create a string representation of a `RelatedRecords`

Usage:

```
RelatedRecords$to_string(style = FALSE)
```

Arguments:

`style` Logical, whether the output is styled using ANSI codes

Returns: A `cli::cli_ansi_string` if `style` = `TRUE` or a character vector

Index