Package: DataSpaceR (via r-universe)

August 27, 2026

Type Package

Title Interface to 'the CAVD DataSpace'

Version 1.0.2

Description Provides a convenient API interface to access immunological data within 'the CAVD DataSpace'(<https://dataspace.cavd.org>), a data sharing and discovery tool that facilitates exploration of HIV immunological data from pre-clinical and clinical HIV vaccine studies.

URL https://docs.ropensci.org/DataSpaceR/, https://github.com/ropensci/DataSpaceR

BugReports https://github.com/ropensci/DataSpaceR/issues

License GPL-3

Encoding UTF-8

Depends R (>= 4.3.0)

Imports utils, R6, Rlabkey (>= 3.4.5), curl, httr, assertthat, digest, jsonlite, data.table

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libssl-dev

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

Date/Publication 2026-08-27 16:52:32 UTC

RemoteUrl https://github.com/ropensci/DataSpaceR

RemoteRef main

RemoteSha 908f6f0aaff979a9a35d6ca803ce88245ca9790a

Contents

`DataSpaceR-package` *DataSpaceR*

Description

DataSpaceR provides a convenient API for accessing datasets within the DataSpace database.

Details

Uses the Rlabkey package to connect to DataSpace. Implements convenient methods for accessing datasets.

Author(s)

Ju Yeong Kim

See Also

`connectDS`

`checkNetrc` *Check netrc file*

Description

Check that there is a netrc file with a valid entry for the CAVD DataSpace.

Usage

```
checkNetrc(netrcFile = getNetrcPath(), onStaging = FALSE, verbose = TRUE)
```

Arguments

netrcFile	A character. File path to netrc file to check.
onStaging	A logical. Whether to check the staging server instead of the production server.
verbose	A logical. Whether to print the extra details for troubleshooting.

Value

The name of the netrc file

See Also

connectDS writeNetrc

Examples

```
## Not run:
checkNetrc()

## End(Not run)
```

---

connectDS *Create a connection to DataSpace*

Description

Constructor for DataSpaceConnection

Usage

```
connectDS(login = NULL, password = NULL, verbose = FALSE, onStaging = FALSE)
```

Arguments

login	A character. Optional argument. If there is no netrc file a temporary one can be written by passing login and password of an active DataSpace account.
password	A character. Optional. The password for the selected login.
verbose	A logical. Whether to print the extra details for troubleshooting.
onStaging	A logical. Whether to connect to the staging server instead of the production server.

Details

Instantiates an DataSpaceConnection. The constructor will try to take the values of the various labkey.* parameters from the global environment. If they don't exist, it will use default values. These are assigned to 'options', which are then used by the DataSpaceConnection class.

Value

an instance of `DataSpaceConnection`

See Also

`DataSpaceConnection`

Examples

```
## Not run:
con <- connectDS()

## End(Not run)
```

`DataSpaceConnection` *The DataSpaceConnection class*

Description

An R6 class for DataSpace browsing and fetching data in DataSpace.

Constructor

`connectDS`

Active bindings

`config` A list. Stores configuration of the connection object such as URL, path and username.

`availableStudies` A data.tabl of available studies.

`availableGroups` A data.table of available groups.

`availableMabs` A data.table of available mAbs.

`availableMabMixtures` A data.table. Metadata of available mAb mixtures.

`availableDonors` A data.table. Metadata about all mAb donors in the DataSpace.

`availableViruses` A data.table of metadata about all virsues in the DataSpace and virus name synonyms.

`availablePublications` A data.table of available publications metadata and available datasets.

`lanlMabMetadata` A data.table of mAb metadata from LANL of mAbs found in the object

`virusNameMappingTables` A list of data.tables containing virus name mappings.

`mabGridSummary` Defunct. Use ‘availableMabs‘.

`mabGrid` Defunct. Use ‘availableMabs‘.

`virusMetadata` Defunct. Use ‘virusNameMappingTables‘.

Methods

Public methods:

`DataSpaceConnection$new()`: Initialize a `DataSpaceConnection` object. See `connectDS`.

Usage:

```
DataSpaceConnection$new(
  login = NULL,
  password = NULL,
  verbose = FALSE,
  onStaging = FALSE
)
```

Arguments:

`login` A character. Optional argument. If there is no netrc file a temporary one can be written by passing login and password of an active DataSpace account.

`password` A character. Optional. The password for the selected login.

`verbose` A logical. Whether to print the extra details for troubleshooting.

`onStaging` A logical. Whether to connect to the staging server instead of the production server.

Returns: A new 'DataSpaceConnection' object.

`DataSpaceConnection$print()`: Print the `DataSpaceConnection` object.

Usage:

```
DataSpaceConnection$print()
```

`DataSpaceConnection$getStudies()`: Create a 'DataSpaceStudies' object.

Usage:

```
DataSpaceConnection$getStudies(availableStudies = self$availableStudies)
```

Arguments:

`availableStudies` an 'availableStudies' object, or a vector of 'study_id' values.

`DataSpaceConnection$getGroups()`: Create a 'DataSpaceGroups' object.

Usage:

```
DataSpaceConnection$getGroups(availableGroups = self$availableGroups)
```

Arguments:

`availableGroups` an 'availableGroups' object, or a vector of 'group id' values.

`DataSpaceConnection$getMabs()`: Create a 'DataSpaceMabs' object.

Usage:

```
DataSpaceConnection$getMabs(
  availableMabs = self$availableMabs,
  includeMixtures = "yes"
)
```

Arguments:

`availableMabs` an 'availableMabs' or 'availableMabMixtures' object, or a vector of 'mab id' values. 'mab_id' values are inferred from 'availableMabMixtures' objects.

`includeMixtures` Whether or not to include mab mixtures. "yes", "no", or "only" are valid. The default, "yes", will return any available mAb mixtures for any mAb passed here.

`DataSpaceConnection$getDonors()`: Create a 'DataSpaceDonors' object.

Usage:

```
DataSpaceConnection$getDonors(availableDonors = self$availableDonors)
```

Arguments:

`availableDonors` an 'availableDonors' object, or a vector of 'donor_id' values.

`DataSpaceConnection$getDaash()`: Create a 'DataSpaceDaash' object.

Usage:

```
DataSpaceConnection$getDaash(availableDaash = NULL)
```

Arguments:

`availableDaash` an 'availableMabs', or 'availableDonors' object, or a vector of 'sequnce_id' values.

`DataSpaceConnection$downloadPublicationData()`: Download study related publication datasets.

Usage:

```
DataSpaceConnection$downloadPublicationData(
  availablePublications = NULL,
  downloadDir = tempdir()
)
```

Arguments:

`availablePublications` an 'availablePublications' object or a vector of 'publication_id' values.

downloadDir A character. Optional, specifies directory to download nonstandard datasets. Default is use to the R session temp directory

`DataSpaceConnection$loadLanlMabMetadata()`: Load any available mAb metadata from LANL.

Usage:

```
DataSpaceConnection$loadLanlMabMetadata()
```

`DataSpaceConnection$getStudy()`: Defunct. Use 'getStudies'.

Usage:

```
DataSpaceConnection$getStudy()
```

`DataSpaceConnection$getGroup()`: Defunct. Use 'getGroups'.

Usage:

```
DataSpaceConnection$getGroup()
```

`DataSpaceConnection$getMab()`: Defunct. Use 'getMabs'.

Usage:

```
DataSpaceConnection$getMab()
```

`DataSpaceConnection$filterMabGrid()`: Defunct. Use 'availableMabs'.

Usage:

```
DataSpaceConnection$filterMabGrid()
```

`DataSpaceConnection$resetMabGrid()`: Defunct. Use 'availableMabs'.

Usage:

```
DataSpaceConnection$resetMabGrid()
```

`DataSpaceConnection$refresh()`: Refresh the connection object to update available studies and groups.

Usage:

```
DataSpaceConnection$refresh()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

connectDS DataSpaceR-package

Examples

```
## Not run:
# Create a connection (Initiate a DataSpaceConnection object)
con <- connectDS()

# View available data

con$availableStudies
con$availableGroups
con$availablePublications
con$availableMabs
con$availableMabMixtures
con$availableDonors
con$availableViruses

# Pass an available object to a "get" method to get data

cvd408 <- con$availableStudies[study_id == "cvd408"] |>
  con$getStudies()

cd4Mabs <- con$availableMabs[grepl("CD4bs", mab_ab_binding_type)] |>
  con$getMabs()


## End(Not run)
```

DataSpaceDaash *The DataSpaceDAASH class*

Description

An R6 class for DataSpace DAASH data.

Constructor

`DataSpaceConnection$getDaash()`

Super class

`DataSpaceConnection` -> `DataSpaceDaash`

Active bindings

`mabMetadata` A data.table of mAbs with metadata found in the object.

`donorMetadata` A data.table of donors with metadata found in the object.

`daashMetadata` A data.table showing the donor and mAb metadata with CDS sequence_id values for the loaded DAASH dataset.

availableStructures A data.table showing the mAb structures available to download.

datasets A list of DAASH datastets loaded to the DAASH object.

variableDefinitions A data.table of variable definitions.

Methods

Public methods:

- DataSpaceDaash$new()
- DataSpaceDaash$print()
- DataSpaceDaash$getFastaFromSequences()
- DataSpaceDaash$downloadAntibodyStructures()
- DataSpaceDaash$refresh()
- DataSpaceDaash$clone()

DataSpaceDaash$new(): Initialize DataSpaceMabMetadata object. See DataSpaceConnection.

Usage:

```
DataSpaceDaash$new(availableDaash)
```

Arguments:

availableDaash availableDaash an 'availableMabs', or 'availableDonors' object, or a vector of 'sequnce_id' values.

config A list.

DataSpaceDaash$print(): Print the DataSpaceMab object summary.

Usage:

```
DataSpaceDaash$print()
```

DataSpaceDaash$getFastaFromSequences(): Return a fasta file for available daash sequences that have been loaded to the current object.

Usage:

```
DataSpaceDaash$getFastaFromSequences(
  sequenceType = "nt",
  originalHeaders = FALSE,
  path = NULL
)
```

Arguments:

sequenceType character the type of fasta file to return: nt = nucleotide, aa = amino acid.

originalHeaders boolean if the original fasta headers should be provided

path The path where to save the fasta files to. If using the default value, NULL, then a fasta file is returned as a character vector.

DataSpaceDaash$downloadAntibodyStructures(): Saves all antibody structures associated with the daash object's 'availableStuctures' object.

Usage:

```
DataSpaceDaash$downloadAntibodyStructures(path = tempdir(), mab_id = NULL)
```

Arguments:

`path` The directory to export fasta files to.

`mab_id` A subset of mab_ids to export. If using the default, NULL, all structures in availableStuctures are downloaded.

`DataSpaceDaash$refresh()`: Refresh the `DataSpaceMabMetadata` object to update datasets.

Usage:

```
DataSpaceDaash$refresh()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`connectDS` `DataSpaceConnection`

Examples

```
## Not run:
# Create a connection (Initiate a DataSpaceConnection object)
con <- connectDS()

# Get the daash object using either an availableMabs or
# availableDonors object.
daash <- con$availableMabs[mab_ab_binding_type %like% "CD4"] |>
  con$getDaash()

# To get lineage sequences, query donors, then pipe available
# donors to the connection getDaash object.
daash <- con$availableDonors[
  lineage_sequences_available == TRUE & mab_count < 10,
  ] |>
  con$getDaash()

# Inspect what datasets are available
names(daash$datasets)

# Inspect the `topCalls` dataset
daash$datasets$topCalls


## End(Not run)
```

DataSpaceDonors *The DataSpaceDonors class*

Description

An R6 class for DataSpace MAb Donor data.

Constructor

`DataSpaceConnection$getMab()`

Super class

`DataSpaceConnection` -> `DataSpaceDonors`

Active bindings

`mabMetadata` A data.table of mAbs with metadata found in the object.

`donorMetadata` A data.table of donors with metadata found in the object.

`datasets` A list of data.table objects containing the related data loaded.

`variableDefinitions` A data.table of variable definitions.

Methods

Public methods:

- `DataSpaceDonors$new()`
- `DataSpaceDonors$print()`
- `DataSpaceDonors$loadDaash()`
- `DataSpaceDonors$refresh()`
- `DataSpaceDonors$clone()`

`DataSpaceDonors$new()`: Initialize `DataSpaceMab` object. See `DataSpaceConnection`.

Usage:

```
DataSpaceDonors$new(donorIds)
```

Arguments:

`donorIds` a character vector of ‘donor_id‘ values.

`DataSpaceDonors$print()`: Print the `DataSpaceMab` object summary.

Usage:

```
DataSpaceDonors$print()
```

`DataSpaceDonors$loadDaash()`: Load DAASH data to the object.

Usage:

```
DataSpaceDonors$loadDaash()
```

`DataSpaceDonors$refresh()`: Refresh the 'DataSpaceDonors' object to update datasets.

Usage:

```
DataSpaceDonors$refresh()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

connectDS DataSpaceConnection

Examples

```
## Not run:
# Create a connection (Initiate a DataSpaceConnection object)
con <- connectDS()

# Print available donors to the console
con$availableDonors

# Query the available donors object and pass that to `getDonors` to get a DataSpaceDonors object
donors <- con$availableDonors[lineage_sequences_available == TRUE & donor_clade == "B",] |>
  con$getDonors()

# Load DAASH data to the object
donors$loadDaash()


## End(Not run)
```

DataSpaceGroups *The DataSpaceGroups class*

Description

An R6 class for DataSpace Groups data.

Constructor

`DataSpaceConnection$getGroups()`

Super class

DataSpaceConnection -> DataSpaceGroups

Active bindings

`availableDatasets` A data.table of datasets available in the object.

`datasets` A list of data.table objects containing the availableDatasets that were loaded.

`variableDefinitions` A data.table of variable definitions.

Methods

Public methods:

- `DataSpaceGroups$new()`
- `DataSpaceGroups$print()`
- `DataSpaceGroups$refresh()`
- `DataSpaceGroups$clone()`

`DataSpaceGroups$new()`: Initialize 'DataSpaceGroups' class. See `DataSpaceConnection`.

Usage:

```
DataSpaceGroups$new(groupIds = NULL)
```

Arguments:

`groupIds` A character vecotor of 'group_id' values. as URL, path and username.

`DataSpaceGroups$print()`: Print `DataSpaceStudy` class.

Usage:

```
DataSpaceGroups$print()
```

`DataSpaceGroups$refresh()`: Refresh loaded integrated datasets, and information of what datasets are available.

Usage:

```
DataSpaceGroups$refresh()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`connectDS` `DataSpaceConnection`

Examples

```
## Not run:
# Create a connection (Initiate a DataSpaceConnection object)
con <- connectDS()

# Get group by `group_id` or pass a filtered `availableGroups` object.
groups <- con$getGroups(c(266, 267))
groups <- con$availableGroups[label == "NYVAC durability comparison"] |>
  con$getGroups()

# Retrieving group assay data for cvd408 from
# DataSpace is done automatically when the groups object is created.
groups$datasets$BAMA

# Get variable information of the assay dataset
groups$datasetDescription$BAMA


## End(Not run)
```

`DataSpaceMabs` *The DataSpaceMab class*

Description

An R6 class for DataSpace MAb data.

Constructor

`DataSpaceConnection$getMab()`

Super class

`DataSpaceConnection` -> `DataSpaceMabs`

Active bindings

`mabMetadata` A data.table of mAbs with metadata found in the object.

`donorMetadata` A data.table of donors with metadata found in the object.

`mabMixMetadata` A data.table. A table of mAb mixtures with metadata found in this DataSpaceMab instance.

`mabMix` A data.table. A mapping table of mab_mix_id to mab_id. with metadata found in this DataSpaceMab instance.

`datasets` A list of data.table objects containing the mab related that were loaded.

`variableDefinitions` A data.table of variable definitions.

Methods

Public methods:

- DataSpaceMabs$new()
- DataSpaceMabs$print()
- DataSpaceMabs$loadDaash()
- DataSpaceMabs$refresh()
- DataSpaceMabs$clone()

`DataSpaceMabs$new()`: Initialize `DataSpaceMab` object. See `DataSpaceConnection`.

Usage:

```
DataSpaceMabs$new(mabIds, includeMixtures)
```

Arguments:

`mabIds` A character vector of ‘mab_id‘ values.

`includeMixtures` Whether or not to include mab mixtures. "yes", "no", or "only" are valid.

`DataSpaceMabs$print()`: Print the `DataSpaceMab` object summary.

Usage:

```
DataSpaceMabs$print()
```

`DataSpaceMabs$loadDaash()`: Load any available DAASH datasets.

Usage:

```
DataSpaceMabs$loadDaash()
```

`DataSpaceMabs$refresh()`: Refresh the `DataSpaceMab` object to update datasets.

Usage:

```
DataSpaceMabs$refresh()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`connectDS` `DataSpaceConnection`

Examples

```
## Not run:
# Create a connection (Initiate a DataSpaceConnection object)
con <- connectDS()

# Inspect available mabs, then pass subset to the `getMabs` method.
vrc01 <- con$availableMabs[mab_name_std == "VRC01"] |>
```

```
  con$getMabs()

# Inspect the `NABMAb` assay data.
vrc01$datasets$NABMAb

# Load DAASH data from mab object
vrc01$loadDaash()

# Inspect DAASH datasets
vrc01$datasets$daash |> names()


## End(Not run)
```

`DataSpaceStudies` *The DataSpaceStudies class*

Description

An R6 class for DataSpace Study data.

Constructor

`DataSpaceConnection$getStudies()`

Super class

`DataSpaceConnection` -> `DataSpaceStudies`

Active bindings

`studies` A character vector of 'study_id' values found in the object.

`availableDatasets` A table of datasets available in the `DataSpaceStudies` object.

`datasets` A list of data.table objects containing the availableDatasets that were loaded.

`variableDefinitions` A list of data.table objects containing the data dictionaries of the integrated data loaded.

`treatmentArm` A data.table. The table of treatment arm information for the connected study. Not available for all study connection.

`studyInfo` A list. Stores the information about the study.

Methods

Public methods:

- `DataSpaceStudies$new()`
- `DataSpaceStudies$print()`

- DataSpaceStudies$loadAvailableDatasets()
- DataSpaceStudies$refresh()
- DataSpaceStudies$clone()

`DataSpaceStudies$new()`: Initialize `DataSpaceStudy` class. See `DataSpaceConnection`.

Usage:

```
DataSpaceStudies$new(studyIds)
```

Arguments:

`studyIds` A character. Name of the study to retrieve. as URL, path and username.

`DataSpaceStudies$print()`: Print `DataSpaceStudy` class.

Usage:

```
DataSpaceStudies$print()
```

`DataSpaceStudies$loadAvailableDatasets()`: Load datasets to the studies object from an availableDatasets object.

Usage:

```
DataSpaceStudies$loadAvailableDatasets(
  availableDatasets = self$availableDatasets,
  downloadDir = tempdir()
)
```

Arguments:

`availableDatasets` An ‘availableDatasets‘ object or vector of ‘study_id‘ values.

`downloadDir` Optional, a character path specifying a directory to download. nonstandard datasets. The default is the working temp directory.

`DataSpaceStudies$refresh()`: Refresh the study object to update available datasets and treatment info.

Usage:

```
DataSpaceStudies$refresh()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

`connectDS` `DataSpaceConnection`

Examples

```
## Not run:
# Create a connection (Initiate a DataSpaceConnection object)
con <- connectDS()

# Get group by `study_id` or pass a filtered `availableStudies` object.
studies <- con$getStudies(c("vtn505", "cvd408"))
studies <- con$getStudies(
  con$availableStudies[grepl("BAMA", data_availability) & species == "Human"]
)

# Load BAMA to the studies object.
studies$loadAssayDatasets("BAMA")
studies$datasets$BAMA

# Inspect variable information of the BAMA dataset
studies$datasetDescriptions$BAMA

# Inspect treatment arm information for all studies in study object
studies$treatmentArm


## End(Not run)
```

---

getNetrcPath *Get a default netrc file path*

Description

Get a default netrc file path

Usage

```
getNetrcPath()
```

Value

A character vector containing the default netrc file path

Examples

```
## Not run:
getNetrcPath()

## End(Not run)
```

`writeNetrc` *Write a netrc file*

Description

Write a netrc file that is valid for accessing DataSpace.

Usage

```
writeNetrc(
  login,
  password,
  netrcFile = NULL,
  onStaging = FALSE,
  overwrite = FALSE
)
```

Arguments

`login`	A character. Email address used for logging in on DataSpace.
`password`	A character. Password associated with the login.
`netrcFile`	A character. Credentials will be written into that file. If left NULL, netrc will be written into a temporary file.
`onStaging`	A logical. Whether to connect to the staging server instead of the production server.
`overwrite`	A logical. Whether to overwrite the existing netrc file.

Details

The database is accessed with the user's credentials. A netrc file storing login and password information is required. See here for instruction on how to register and set DataSpace credential. By default `curl` will look for the file in your home directory.

Value

A character vector containing the netrc file path

See Also

`connectDS` `checkNetrc`

Examples

```
## Not run:
# First, create an account in the DataSpace App and read the terms of use
# Next, create a netrc file using writeNetrc()
writeNetrc(
  login = "dataspaceuser@email.com",
  password = "yourSecretPassword"
)
# Specify `netrcFile = getNetrcPath()` to write netrc in the default path

## End(Not run)
```

Index