Package 'BaseSpaceR'

June 29, 2025

Title R SDK for BaseSpace RESTful API

Description A rich R interface to Illumina's BaseSpace cloud computing environment, enabling the fast development of data analysis and visualisation tools.

Version 1.52.0

Author Adrian Alexa

Maintainer Jared O'Connell <joconnell@illumina.com>

biocViews Infrastructure, DataRepresentation, ConnectTools, Software, DataImport, HighThroughputSequencing, Sequencing, Genetics

Depends R (>= 2.15.0), RCurl, RJSONIO

Imports methods

Suggests RUnit, IRanges, Rsamtools

License Apache License 2.0

Collate generics.R misc.R ServiceURI.R Error.R AppAuth.R AppSessionAuth.R Href.R Properties.R Response.R Users.R Runs.R Genomes.R Projects.R AppResults.R AppSessions.R Samples.R Files.R Files_extra.R Variants.R Coverage.R zzz.R

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

git_branch RELEASE_3_21

git_last_commit 57aab24

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-06-29

Contents

`BaseSpaceR-package` *R SDK for BaseSpace RESTful API*

Description

The `BaseSpaceR` package provides a rich R interface to Illumina's BaseSpace cloud computing environment, enabling the fast development of data analysis and visualisation tools.

Details

BaseSpace is Illumina's next-generation sequencing cloud computing environment designed with biologists in mind. Researcher can easily store, analyze, collaborate, and share genetic data (https://basespace.illumina.com).

BaseSpaceR is a SDK ofering methods and data structures for working with the data resources BaseSpace REST API exposes.

Features include:

- Persistent connection with the REST server.
- Support for the REST API query parameters.
- Vectorized operations in line with the R semantic. Allows for queries across multiple Projects, Samples, AppResults, Files, etc.
- S4 class system used to represent the BaseSpace data model.
- Portability on most platforms: Linux, Windows and Mac OS X.

Author(s)

Adrian Alexa

Maintainer: Adrian Alexa <aalexa@illumina.com>

References

BaseSpace https://basespace.illumina.com

BaseSpace API https://developer.basespace.illumina.com

See Also

AppAuth-class for details on how to establish a connection with the BaseSpace server.

Response-class for details on the structure of the objects and the interface used by the API.

Users-class, Genomes-class, Runs-class, Projects-class, Samples-class, Files-class **and** AppResults-class for details on various resources supported by the API.

---

aAuth *Sample* AppAuth *instance with 'browse global' scope*

Description

The aAuth is an instance of a AppAuth object. It can be used to browse some of the public resources available in BaseSpace without requireing user authentication.

Usage

```
data(aAuth)
```

Source

Build from a pre-generated access token. See AppAuth-class for code examples on how-to generate such an object.

Examples

```
data(aAuth)

## print the object
aAuth
```

AppAuth-class AppAuth *class*

Description

Class to manage client's communication and permissions with BaseSpace REST serve.

Methods

initializeAuth `signature(x = "AppAuth")`: ...

requestAccessToken `signature(x = "AppAuth")`: ...

hasAccess `signature(x = "AppAuth")`: ...

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

Runs, Projects, Samples, etc. for examples of objects using the AppAuth handler.

ServiceURI for a low-level object managing the REST calls.

Examples

```
showClass("AppAuth")

## load an AppAuth instance containing a pre-generated access token
data(aAuth)
aAuth

hasAccess(aAuth)

## new AppAuth instance using a pre-generated access token
my_access_token <- "eee44c28ba0e43a1badb85c5ce7bb94d"
myHandle <- AppAuth(access_token = my_access_token)
myHandle

Users(myHandle)


## using the OAuth v2 workflow
## Not run:
## paste your client_id and client_secret here
myAppClientId = ""
```

```
myAppClientSecret = ""

## instantiate a new AppAuth object
myHandle <- AppAuth(client_id = myAppClientId,
                    client_secret = myAppClientSecret,
                    scope = "browse global")

## Open the showed URI in a browser and perform the authentication.

requestAccessToken(myHandle)
hasAccess(myHandle)


## End(Not run)
```

AppResults-class AppResults *and* "AppResultsSummary" *objects*

Description

Classes and methods to handle the AppResults resource.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth and Projects.

Examples

```
showClass("AppAuth")
```

AppSessionAuth-class AppSessionAuth *class*

Description

Extension of the AppAuth class...

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

Runs, Projects, Samples, etc. for examples of objects using the AppAuth handler.

ServiceURI for a low-level object managing the REST calls.

Examples

```
showClass("AppAuth")

## using the OAuth v2 workflow
## Not run:
## paste your client_id and client_secret here
myAppClientId = ""
myAppClientSecret = ""

## instantiate a new AppAuth object
myHandle <- AppAuth(client_id = myAppClientId,
                    client_secret = myAppClientSecret,
                    scope = "browse global")

## Open the showed URI in a browser and perform the authentication.

requestAccessToken(myHandle)
hasAccess(myHandle)


## End(Not run)
```

AppSessions-class AppSessions *object*

Description

Classes and methods to handle the AppSessions resource.

Details

coming soon...

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth and Response.

Examples

```
showClass("AppAuth")
```

Coverage *Methods for accessing coverage data from BAM files*

Description

This methods are used to provide mean read coverage depth in a particular chromosomal region.

Usage

```
getCoverageStats(x, ...)
getCoverage(x, ...)
```

Arguments

x	An object of class AppAuth.
...	Adiditional arguments supported by the REST API. **id** File id (for a BAM file) within BaseSpace. **chrom** Character string given the chromosome name (UCSC).

Details

Coming soon...

Value

These methods return a list with a representation of coverage histogram.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth.

Examples

```
data(aAuth)

## get the Ids of some BAM files
##...


## You might require Read access to the AppResult
#initializeAuth(aAuth, scope = paste("browse global, read project", 12))
#requestAccessToken(aAuth)


#getCoverageStats(aAuth, id = bid, "phix")

#readcov <- getCoverage(aAuth, id = bid, "phix", StartPos = 1L, EndPos = 5e3L)[[1]]
#barplot(readcov$MeanCoverage, col = "lightblue1", border = NA)
#plot(readcov$MeanCoverage, col = "lightblue2", type = "l", lwd = 2)
```

Files-class Files *and* FilesSummary *objects*

Description

Classes and methods to handle the Files resource.

Details

The Files resource provides access to files stored in BaseSpace. A file should be seen as a data stream and associated attributes (date created, size, type, etc.).

Files are associated with specific Runs, Samples, or AppResults and the Files resource provides the interface for manipulating these files.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth, Runs, Samples and AppResults.

Examples

```
data(aAuth)


## get one AppResult
reseq <- AppResults(listAppResults(aAuth, projectId = 21383369, Limit = 1))

f <- listFiles(reseq)
TotalCount(f)
Name(f)


identical(f, listFiles(aAuth, appResultId = Id(reseq)))

## list only the BAM files
f <- listFiles(aAuth, appResultId = Id(reseq), Extensions = ".bam")
Name(f)
```

FilesMethods *High-level methods for* Files *objects and related*

Description

Methods to handle various files types.

Details

The Files resource provides access to files stored in BaseSpace. A file should be seen as a data stream and associated attributes (date created, size, type, etc.).

For known file types (.bam, .vcf, etc.) we offer wrappers to 'map' this objects to various Bioconductor objects.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth, AppResults and Files

Examples

```
data(aAuth)


## get one AppResult
reseq <- AppResults(listAppResults(aAuth, projectId = 21383369, Limit = 1))

f <- listFiles(reseq)
TotalCount(f)
Name(f)

## list only the BAM files
f <- listFiles(aAuth, appResultId = Id(reseq), Extensions = ".bam")
Name(f)

## construct the BAMFile
##...
```

Genomes-class Genomes *and* GenomesSummary *objects*

Description

Classes and methods to handle the Genomes resource.

Browsing

`listGenomes(x, ...)` lists all the available genomes, returning only a small summary for each genome.

x is an `AppAuth` object.

... Parameters supported by the REST API, specified as `name = value`. For example, `listGenomes(aAuth, Offset = 5, Limit = 2)`

Constructor

`Genomes()`: Instantiates an empty `Genomes` object. Same as `new("Genomes")`.

`Genomes(x, id)`: `x` is an `AppAuth` object. `id` is either and integer or a character string storing an integer. `id` can have length larger that 1.

`Genomes(x)`: `x` is an `GenomesSummary` object.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

`AppAuth`, `Response`.

Examples

```
data(aAuth)

## list all available genomes
g <- listGenomes(aAuth)
g$SpeciesName

## using the REST API query parameters
listGenomes(aAuth, Limit = 2)
g <- listGenomes(aAuth, Offset = 5, Limit = 2, SortBy = "Build")
g

## get the details for the listed geneomes
Genomes(g)

## get the genomes based on their ID
Genomes(aAuth, id = 4)
## if the ID is missing thean NULL is returned for that particular ID
Genomes(aAuth, id = c(4, 1, 110))
```

Private methods *Internal methods*

Description

This page is used as a placeholder for private methods in order to pass the R checks.

Details

Internal use.

Author(s)

Adrian Alexa

Projects-class `Projects` *and* `ProjectsSummary` *objects*

Description

Classes and methods to handle the Projects resource.

Details

The Projects resource provides a logical grouping of the Samples resource and the AppResults resource for a given user.

Browsing

`listProjects(x, ...)` lists all the available projects visable to the user, returning only a small summary for each project.

x is an `AppAuth` object.

... Parameters supported by the REST API, specified as `name = value`. For example, `listProjects(x, Limit = 2)`

Constructor

`Projects()`: Instantiates an empty `Projects` object. Same as `new("Projects")`.

`Projects(x, id)`: `x` is an `AppAuth` object. `id` is either and integer or a character string storing an integer. `id` can have length larger that 1.

`Projetcs(x)`: `x` is an `ProjectsSummary` object.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth, Response, Users and Runs.

Examples

```
data(aAuth)

p <- listProjects(aAuth)
p

Projects(aAuth, id = c(2, 12, 1012))
Projects(p)


## Make a new project ...
createProject(aAuth, name = "My Project X")


## We need 'write global' access to be able to create a new project
## Not run:
initializeAuth(aAuth, scope = "write global")
requestAccessToken(aAuth)

createProject(aAuth, name = "My Project X")

## End(Not run)
```

Response-class Item, Collection *and* Response *objects*

Description

These virtual classes provide the building blocks for the containers return by quering various BaseSpace resource. They are modeled after the Response container defined by the REST API.

Conceptually there are two response types exposed by BaseSpace, an individual response and a collection response. The former, modeled by the Item S4 class, is used when querying an individual item/instance within a resource. The later is used for listing the items/instances available for a given resource and is modeled by the Collection S4 class.

Item Accessors

The `Item` class models a simple unordered set of key/value pairs. There is a core set of keys, for which access methods are defined and which are inherited by any child class.

In the following `x` is an `Item` object.

`$`: The `$` operator can be used to access the Item elements. When `x` is an Item object this is equivalet to `@`. But one can think at it as user level operator. Unlike `@` the replace method is not implemented for `$`.

`Id(x)`: Id of the resource. Character string, though it will always be an integer.

`Name(x)`: Name of the selected resource.

`Href(x)`: Location of the resource in the API. The first component of the URI is the version of the REST API.

`DateCreated(x)`: When this resource was created. Character string. It can be converted to a `Date` instace by `as.Date(DateCreated(x))`.

`UserOwnedBy(x)`: Information about the User who owns this resource. At this moment this is a `list` object, but it might be replace with an object at a future point.

`Status(x)`: The status of the resource. Can be of any type and it will be defined by the classes extended Item.

`HrefBaseSpaceUI(x)`: The location of this project in BaseSpace. Character string giving the complete URL within the BaseSpace dashboard.

Collection Accessors

The `Collection` class models an ordered set of `Item` objects and a set of predefined attributes. The interface provided by the `Item` class is implemented by this class. However, since we deal with an ordered set of objects, the methods and the access methods, return a vector of the same length as the size of the collection.

In the following `x` is a `Collection` object.

All accesor implemented by the item class are implemented by Collection class. However here the return value is a vector. If `x` has 2 elements, then `Id(x)` will be a vector with 2 elements. The same stands for the general accessor `$`.

`length(x)`: Returns the number of elements in Collection `x`.

`Items(x)`: List of `Item` objects.

`TotalCount(x)`: The total number of items in the collection as reported by the queried resource.

`Offset(x)`: The starting point the collection was read from.

`Limit(x)`: The maximum number of items returned. Ranges from 0 to 1024.

`SortDir(x)`: The way the collection is sorted, either ascending or descending.

`SortBy(x)`: The field to use to sort the collection.

Subsetting

Comming soon ...

Methods

`length(x)`: Returns the number of elements in Collection `x`.

`show(x)`: Prints the object.

`as.list(x)`: R `list` representation of the object.

Author(s)

Adrian Alexa

Examples

```
showClass("Item")
showClass("Collection")
showClass("Response")
```

ResponseStatus-class *Class* "ResponseStatus"

Description

The ResponseStatus class is used internally to manage the status messages returned by the REST service.

Objects from the Class

Objects can be created by calls of the form `ResponseStatus()`. `ResponseStatus()` creates an `ResponseStatus` instances...

Methods

success `signature(x = "ServiceURI")`: Returns `TRUE` if the HTTP status is 2xx.

show `signature(object = "ServiceURI")`: ...

Author(s)

Adrian Alexa

Examples

```
showClass("ResponseStatus")
```

Runs-class Runs *and* RunsSummary *objects*

Description

Classes and methods to handle the Runs resource.

Details

The Runs resource contains the raw data produced by the instruments, the base calls, together with run metrics, instrument health data, and other information used for data processing and analysis.

Browsing

`listRuns(x, ...)` lists all the available runs visable to the user, returning only a small summary for each run.

x is an AppAuth object.

... Parameters supported by the REST API, specified as `name = value`. For example, `listRuns(x, Limit = 2, Statuses = "Failed")`

Constructor

`Runs()`: Instantiates an empty Runs object. Same as `new("Runs")`.

`Runs(x, id)`: `x` is an AppAuth object. `id` is either and integer or a character string storing an integer. `id` can have length larger that 1.

`Runs(x)`: `x` is an RunsSummary object.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth, Response, and Users.

Examples

```
data(aAuth)

r <- listRuns(aAuth)
r

listRuns(aAuth, Statuses = "Failed") # no faild runs
```

```
listRuns(aAuth, Statuses = "Complete")
listRuns(aAuth, SortBy = "Id", SortDir="Desc")

Runs(r)[[1]]


Runs(aAuth, id = 101102)
Runs(r)

Runs(aAuth, id = c(Id(r), "11111")) # the third element must be NULL
```

`Samples-class` `Samples` *and* `SamplesSummary` *objects*

Description

Classes and methods to handle the Samples resource.

Details

In general samples are the result of demultiplexing and are thought as the holding the input data for an App. One example of data within a Samples resource are the FASTQ files.

Browsing

`listSamples(x, projectId, ...)` lists all the available samples associated with a particular project. It returns a small summary for each existing sample.

x is an `AppAuth` object.

projectId the ID of the project we want to explore.

... Parameters supported by the REST API, specified as `name = value`. For example, `listSamples(x, projectId = "1", Limit = 2)`

`listSamples(x, ...)`

x is an `Projects` object.

... Parameters supported by the REST API.

Constructor

`Samples()`: Instantiates an empty `Samples` object. Same as `new("Samples")`.

`Samples(x, id)`: `x` is an `AppAuth` object. `id` is either and integer or a character string storing an integer. `id` can have length larger that 1.

`Samples(x)`: `x` is an `SamplesSummary` object.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth and Projects.

Examples

```
data(aAuth)

## list all the available projects and select one
p <- Projects(listProjects(aAuth, Limit = 1), simplify = TRUE)
p

## list the samples available in this project
allS <- listSamples(aAuth, projectId = Id(p))

##  we can call listSamples() directly using 'p'
identical(allS, listSamples(p))


oneS <- listSamples(aAuth, projectId = Id(p), Limit = 1)
oneS
Samples(oneS) # list with one Samples object
Samples(oneS, simplify = TRUE) # Samples object
```

ServiceURI-class *Class* "ServiceURI"

Description

The ServiceURI class is a general container for storing the URI of a REST based Web service.

Objects from the Class

Objects can be created by calls of the form `ServiceURI()`. `ServiceURI()` creates an `ServiceURI` instances out of an URL and an API resource.

Slots

`url`: Object of class "`character`" ~~

`version`: Object of class "`character`" ~~

Methods

show `signature(object = "ServiceURI")`: ...

uri `signature(x = "ServiceURI")`: ...

Author(s)

Adrian Alexa

Examples

```
showClass("ServiceURI")
```

`Users-class` `Users` *objects and resources*

Description

Class to manage the Users resource.

Details

The Users resource allows the client to get basic information about the user that is currently using the application.

To query this resource we use the `Users()` method.

Constructor

`Users()`: Instantiates an empty `Users` object. Same as `new("Users")`.

`Users(x)`: `x` is an AppAuth object. The function returns a `Response` object of class `Users`.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

`AppAuth`, `Response`.

Examples

```
data(aAuth)

## Empty instance
Users()

## Quering the Users resource using the AppAuth handler
u <- Users(aAuth)
## Printing the object shows the response elements
u
```

```
## Accesors
Id(u)
Name(u)
## Using the general '$' accesor, same interface as 'list'
u$Id
u$Email # there is no accesor method Email(), so '$' is useful here!
u$fakeElement # returns NULL (to keep the same semantic as 'list')


## Quering the resource unsing a Response object
Users(u) # u is of class Users which extends Response


## Specifying a user ID. ID can be specify either as an integer or as a string
Users(aAuth, id = 1463464) # must work if given as an integer(even of mode numeric)
Users(aAuth, id = "1463464") # must work

## This should fail since is not the current user
tryCatch(Users(aAuth, id = "660666"), error = function(e) cat("No access to this user data!\n"))
```

`Variants` *Methods for accessing varinant data*

Description

Manipulating variant data.

Usage

```
getVariantSet(x, ...)
getVariants(x, ...)
```

Arguments

`x`	An object of class `AppAuth`.
`...`	Adiditional arguments supported by the REST API.
	id File id (for a BAM file) within BaseSpace.
	chrom Character string given the chromosome name (UCSC).

Details

Coming soon...

Value

These methods return a list with a representation of variant data.

Author(s)

Adrian Alexa

References

https://developer.basespace.illumina.com/docs/content/documentation/rest-api/api-reference

See Also

AppAuth.

Examples

```
data(aAuth)

## get the ids of VCF files (within an AppResult? )
reseq <- listAppResults(aAuth, projectId = 21383369, Limit = 1)
AppResults(reseq)

vcfs <- listFiles(AppResults(reseq), Extensions = ".vcf")
Name(vcfs)
Id(vcfs)
vcfs

## Not run:
## variant metadata
getVariantSet(aAuth, vid)

## get the variants
getVariants(aAuth, vid, chrom = "chr")

v <- getVariants(aAuth, vid, chrom = "chr", EndPos = 1000000L, Limit = 5)
v

## End(Not run)
```

Index