Package: RSirius (via r-universe)

September 23, 2026

Title R Package Client for SIRIUS Nightsky API

Version 6.5.4

Author person(``OpenAPI Generator community'', email = ``team@openapitools.org'', role = c(``aut'', ``cre''))

Maintainer OpenAPI Generator community <team@openapitools.org>

Description REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

URL https://github.com/GIT_USER_ID/GIT_REPO_ID

BugReports https://github.com/GIT_USER_ID/GIT_REPO_ID/issues

Depends R (>= 3.3)

Encoding UTF-8

License Unlicense

LazyData true

Suggests testthat

Imports jsonlite, httr2, R6, base64enc, stringr

RoxygenNote 7.2.1

Config/pak/sysreqs libicu-dev libssl-dev

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

Date/Publication 2026-09-23 16:46:19 UTC

RemoteUrl https://github.com/sirius-ms/sirius-client-openAPI

RemoteRef HEAD

RemoteSha 496dd29092d9c418b441c55a3bb7434f0099d2f9

RemoteSubdir client-api_r/generated

Contents

AccountCredentials . . . 4
AccountInfo . . . 8
ActuatorApi . . . 11
AlignedFeature . . . 13
AllowedFeatures . . . 18
AnnotatedMsMsData . . . 22
AnnotatedPeak . . . 25
AnnotatedSpectrum . . . 28
AnyValue . . . 33
ApiClient . . . 35
ApiResponse . . . 39
BackgroundComputationsStateEvent . . . 40
BasicSpectrum . . . 44
BinaryFingerprint . . . 48
BioTransformerParameters . . . 51
BioTransformerSequenceStep . . . 55
Canopus . . . 58
CanopusPrediction . . . 61
Compound . . . 65
CompoundClass . . . 69
CompoundClasses . . . 73
CompoundImport . . . 76
CompoundsApi . . . 79
ComputedSubtools . . . 90
ConnectionCheck . . . 93
ConnectionError . . . 97
ConsensusAnnotationsCSI . . . 100
ConsensusAnnotationsDeNovo . . . 104
DataImportEvent . . . 108
DBLink . . . 111
Deviation . . . 114
Feature . . . 117
FeatureAnnotations . . . 121
FeatureImport . . . 125
FeaturesApi . . . 129
FingerprintPrediction . . . 177
FormulaCandidate . . . 180
FragmentationTree . . . 185
FragmentNode . . . 188
GuiApi . . . 192
GuiInfo . . . 195
ImportResult . . . 198
Info . . . 202
InfoApi . . . 206
IsotopePatternAnnotation . . . 208
Job . . . 212

JobProgress . . . 216
JobsApi . . . 219
JobSubmission . . . 236
LcmsSubmissionParameters . . . 241
LicenseInfo . . . 246
LipidAnnotation . . . 249
LoginAndAccountApi . . . 253
LossEdge . . . 261
MsData . . . 264
MsNovelist . . . 268
PagedModelAlignedFeature . . . 271
PagedModelCompound . . . 274
PagedModelFeature . . . 277
PagedModelFormulaCandidate . . . 281
PagedModelJob . . . 284
PagedModelRun . . . 287
PagedModelSpectralLibraryMatch . . . 290
PagedModelStructureCandidateFormula . . . 293
PagedModelStructureCandidateScored . . . 296
PageMetadata . . . 300
ParentPeak . . . 303
PeakAnnotation . . . 306
PeakPair . . . 311
ProblemDetail . . . 314
ProjectChangeEvent . . . 318
ProjectInfo . . . 321
ProjectsApi . . . 325
QuantTable . . . 341
QuantTableOptField . . . 345
rsirius_api . . . 347
Run . . . 348
RunOptField . . . 352
RunsApi . . . 353
SearchableDatabase . . . 358
SearchableDatabaseParameters . . . 362
SearchableDatabasesApi . . . 365
SearchableField . . . 376
SearchableFieldType . . . 381
SecurityContext . . . 382
SecurityContextPrincipal . . . 385
SimplePeak . . . 388
Sirius . . . 392
SpectralLibraryMatch . . . 398
SpectralLibraryMatchSummary . . . 402
SpectralLibrarySearch . . . 406
SpectrumAnnotation . . . 410
StoredJobSubmission . . . 414
StructureCandidate . . . 418

StructureCandidateFormula 422
StructureCandidateScored 426
StructureDbSearch 431
Subscription 435
SubscriptionConsumables 439
Tag 442
Term 445
Timeout 449
UseHeuristic 452
WebServerNamespace 455
Zodiac 458
ZodiacAnalogueNodes 462
ZodiacEdgeFilterThresholds 465
ZodiacEpochs 469
ZodiacLibraryScoring 472

Index **476**

`AccountCredentials` *AccountCredentials*

Description

Simple object to hold account credentials, e.g. to perform login operations. If refreshToken is given, it is usually preferred over password based authentication. But in the end this is up to the respective web service.

AccountCredentials Class

Format

An `R6Class` generator object

Details

Create a new AccountCredentials

Public fields

`username` character [optional]

`password` character [optional]

`refreshToken` character [optional]

Methods

Public methods:

- AccountCredentials$new()
- AccountCredentials$toJSON()
- AccountCredentials$toList()
- AccountCredentials$toSimpleType()
- AccountCredentials$fromJSON()
- AccountCredentials$toJSONString()
- AccountCredentials$fromJSONString()
- AccountCredentials$validateJSON()
- AccountCredentials$toString()
- AccountCredentials$isValid()
- AccountCredentials$getInvalidFields()
- AccountCredentials$print()
- AccountCredentials$clone()

Method `new()`**:** Initialize a new AccountCredentials class.

Usage:

```
AccountCredentials$new(
  username = NULL,
  password = NULL,
  refreshToken = NULL,
  ...
)
```

Arguments:

`username` username

`password` password

`refreshToken` refreshToken

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AccountCredentials$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AccountCredentials$toList()
```

Returns: AccountCredentials as a base R list.

Examples:

```
# convert array of AccountCredentials (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AccountCredentials to a base R type

Usage:

```
AccountCredentials$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AccountCredentials

Usage:

```
AccountCredentials$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AccountCredentials

Method `toJSONString()`**:** To JSON String

Usage:

```
AccountCredentials$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AccountCredentials in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AccountCredentials

Usage:

```
AccountCredentials$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AccountCredentials

Method `validateJSON()`**:** Validate JSON input with respect to AccountCredentials and throw an exception if invalid

Usage:

```
AccountCredentials$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AccountCredentials$toString()
```

Returns: String representation of AccountCredentials

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AccountCredentials$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AccountCredentials$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AccountCredentials$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AccountCredentials$toList`
## ------------------------------------------------

# convert array of AccountCredentials (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

AccountInfo *AccountInfo*

Description

AccountInfo Class

AccountInfo Class

Format

An R6Class generator object

Details

Create a new AccountInfo

Public fields

`userID` character [optional]

`username` character [optional]

`userEmail` character [optional]

`gravatarURL` character [optional]

`subscriptions` list(Subscription) [optional]

`activeSubscriptionId` character [optional]

Methods

Public methods:

- `AccountInfo$new()`
- `AccountInfo$toJSON()`
- `AccountInfo$toList()`
- `AccountInfo$toSimpleType()`
- `AccountInfo$fromJSON()`
- `AccountInfo$toJSONString()`
- `AccountInfo$fromJSONString()`
- `AccountInfo$validateJSON()`
- `AccountInfo$toString()`
- `AccountInfo$isValid()`
- `AccountInfo$getInvalidFields()`
- `AccountInfo$print()`
- `AccountInfo$clone()`

Method `new()`**:** Initialize a new AccountInfo class.

Usage:

```
AccountInfo$new(
  userID = NULL,
  username = NULL,
  userEmail = NULL,
  gravatarURL = NULL,
  subscriptions = NULL,
  activeSubscriptionId = NULL,
  ...
)
```

Arguments:

`userID` userID

`username` username

`userEmail` userEmail

`gravatarURL` gravatarURL

`subscriptions` subscriptions

`activeSubscriptionId` activeSubscriptionId

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AccountInfo$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AccountInfo$toList()
```

Returns: AccountInfo as a base R list.

Examples:

```
# convert array of AccountInfo (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AccountInfo to a base R type

Usage:

```
AccountInfo$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AccountInfo

Usage:

```
AccountInfo$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AccountInfo

Method `toJSONString()`**:** To JSON String

Usage:

```
AccountInfo$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AccountInfo in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AccountInfo

Usage:

```
AccountInfo$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AccountInfo

Method `validateJSON()`**:** Validate JSON input with respect to AccountInfo and throw an exception if invalid

Usage:

```
AccountInfo$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AccountInfo$toString()
```

Returns: String representation of AccountInfo

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AccountInfo$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AccountInfo$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AccountInfo$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AccountInfo$toList`
## ------------------------------------------------

# convert array of AccountInfo (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ActuatorApi` *Actuator operations*

Description

ActuatorApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- ActuatorApi$new()
- ActuatorApi$Health()
- ActuatorApi$HealthWithHttpInfo()
- ActuatorApi$Shutdown()
- ActuatorApi$ShutdownWithHttpInfo()
- ActuatorApi$clone()

Method `new()`**:** Initialize a new ActuatorApi.

Usage:

```
ActuatorApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `Health()`**:** Actuator web endpoint 'health'

Usage:

```
ActuatorApi$Health(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: object

Method `HealthWithHttpInfo()`**:** Actuator web endpoint 'health'

Usage:

```
ActuatorApi$HealthWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (object) with additional information such as HTTP status code, headers

Method `Shutdown()`**:** Actuator web endpoint 'shutdown'

Usage:

```
ActuatorApi$Shutdown(...)
```

Arguments:

`...` Other optional arguments

Returns: void

Method `ShutdownWithHttpInfo()`**:** Actuator web endpoint 'shutdown'

Usage:

```
ActuatorApi$ShutdownWithHttpInfo(...)
```

Arguments:

... Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
####################  Health  ####################

library(RSirius)

#Actuator web endpoint 'health'
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$Health(data_file = "result.txt")
result <- api_instance$actuator_api$Health()
dput(result)


####################  Shutdown  ####################

library(RSirius)

#Actuator web endpoint 'shutdown'
api_instance <- rsirius_api$new()

api_instance$actuator_api$Shutdown()



## End(Not run)
```

`AlignedFeature` *AlignedFeature*

Description

The AlignedFeature contains the ID of a feature (aligned over runs) together with some read-only information that might be displayed in some summary view.

AlignedFeature Class

Format

An `R6Class` generator object

Details

Create a new AlignedFeature

Public fields

`alignedFeatureId` Unique identifier of the aligned feature within the project. character [optional]

`compoundId` Identifier of the compound the feature belongs to. Features that are different adducts or isotopologues of the same molecule share it. character [optional]

`name` Informative, human-readable name of the feature. character [optional]

`externalFeatureId` Externally provided FeatureId (e.g. by some preprocessing tool). This FeatureId is NOT used by SIRIUS but is stored to ease mapping information back to the source. character [optional]

`ionMass` Mass-to-charge ratio (m/z) of the precursor ion of the feature. numeric [optional]

`charge` Ion mode (charge) the feature has been measured in. integer

`detectedAdducts` Adducts that have been detected for the feature during preprocessing. Never empty: if no adduct could be detected, the unknown ion type matching the feature's charge ([M+?]+ or [M+?]-) is reported instead, so every feature is filterable by adduct. list(character)

`rtStartSeconds` Start of the retention time range the feature was detected in, in seconds. numeric [optional]

`rtEndSeconds` End of the retention time range the feature was detected in, in seconds. numeric [optional]

`rtApexSeconds` Retention time of the intensity apex of the feature, in seconds. numeric [optional]

`quality` Overall quality of the feature. NOT_APPLICABLE if no quality data is available character [optional]

`hasMs1` If true, the feature has at least one MS1 spectrum character [optional]

`hasMsMs` If true, the feature has at least one MS/MS spectrum character [optional]

`msData` MsData [optional]

`topAnnotations` FeatureAnnotations [optional]

`topAnnotationsDeNovo` FeatureAnnotations [optional]

`computing` Write lock for the feature. If the feature is locked no write operations are possible. True if any computation is modifying the feature or its results. character [optional]

`computedTools` ComputedSubtools [optional]

`qualities` Qualities per top level quality category. named list(character) [optional]

`tags` Key: tagName, value: tag named list(Tag) [optional]

Methods

Public methods:

- AlignedFeature$new()
- AlignedFeature$toJSON()
- AlignedFeature$toList()
- AlignedFeature$toSimpleType()
- AlignedFeature$fromJSON()
- AlignedFeature$toJSONString()
- AlignedFeature$fromJSONString()
- AlignedFeature$validateJSON()
- AlignedFeature$toString()
- AlignedFeature$isValid()
- AlignedFeature$getInvalidFields()
- AlignedFeature$print()
- AlignedFeature$clone()

Method `new()`**:** Initialize a new AlignedFeature class.

Usage:

```
AlignedFeature$new(
  charge,
  detectedAdducts,
  alignedFeatureId = NULL,
  compoundId = NULL,
  name = NULL,
  externalFeatureId = NULL,
  ionMass = NULL,
  rtStartSeconds = NULL,
  rtEndSeconds = NULL,
  rtApexSeconds = NULL,
  quality = NULL,
  hasMs1 = NULL,
  hasMsMs = NULL,
  msData = NULL,
  topAnnotations = NULL,
  topAnnotationsDeNovo = NULL,
  computing = NULL,
  computedTools = NULL,
  qualities = NULL,
  tags = NULL,
  ...
)
```

Arguments:

`charge` Ion mode (charge) the feature has been measured in.

`detectedAdducts` Adducts that have been detected for the feature during preprocessing. Never empty: if no adduct could be detected, the unknown ion type matching the feature's charge ([M+?]+ or [M+?]-) is reported instead, so every feature is filterable by adduct.

`alignedFeatureId` Unique identifier of the aligned feature within the project.

`compoundId` Identifier of the compound the feature belongs to. Features that are different adducts or isotopologues of the same molecule share it.

`name` Informative, human-readable name of the feature.

`externalFeatureId` Externally provided FeatureId (e.g. by some preprocessing tool). This FeatureId is NOT used by SIRIUS but is stored to ease mapping information back to the source.

`ionMass` Mass-to-charge ratio (m/z) of the precursor ion of the feature.

`rtStartSeconds` Start of the retention time range the feature was detected in, in seconds.

`rtEndSeconds` End of the retention time range the feature was detected in, in seconds.

`rtApexSeconds` Retention time of the intensity apex of the feature, in seconds.

`quality` Overall quality of the feature. NOT_APPLICABLE if no quality data is available

`hasMs1` If true, the feature has at least one MS1 spectrum

`hasMsMs` If true, the feature has at least one MS/MS spectrum

`msData` msData

`topAnnotations` topAnnotations

`topAnnotationsDeNovo` topAnnotationsDeNovo

`computing` Write lock for the feature. If the feature is locked no write operations are possible. True if any computation is modifying the feature or its results.

`computedTools` computedTools

`qualities` Qualities per top level quality category.

`tags` Key: tagName, value: tag

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AlignedFeature$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AlignedFeature$toList()
```

Returns: AlignedFeature as a base R list.

Examples:

```
# convert array of AlignedFeature (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AlignedFeature to a base R type

Usage:

```
AlignedFeature$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AlignedFeature

Usage:

```
AlignedFeature$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AlignedFeature

Method `toJSONString()`**:** To JSON String

Usage:

```
AlignedFeature$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AlignedFeature in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AlignedFeature

Usage:

```
AlignedFeature$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AlignedFeature

Method `validateJSON()`**:** Validate JSON input with respect to AlignedFeature and throw an exception if invalid

Usage:

```
AlignedFeature$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AlignedFeature$toString()
```

Returns: String representation of AlignedFeature

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AlignedFeature$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AlignedFeature$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AlignedFeature$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AlignedFeature$toList`
## ------------------------------------------------

# convert array of AlignedFeature (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`AllowedFeatures` *AllowedFeatures*

Description

AllowedFeatures Class

AllowedFeatures Class

Format

An `R6Class` generator object

Details

Create a new AllowedFeatures

Public fields

`cli` character [optional]

`api` character [optional]

`deNovo` character [optional]

`importMSRuns` character [optional]

`importPeakLists` character [optional]

`importCef` character [optional]

`transformationProducts` character [optional]

Methods

Public methods:

- `AllowedFeatures$new()`
- `AllowedFeatures$toJSON()`
- `AllowedFeatures$toList()`
- `AllowedFeatures$toSimpleType()`
- `AllowedFeatures$fromJSON()`
- `AllowedFeatures$toJSONString()`
- `AllowedFeatures$fromJSONString()`
- `AllowedFeatures$validateJSON()`
- `AllowedFeatures$toString()`
- `AllowedFeatures$isValid()`
- `AllowedFeatures$getInvalidFields()`
- `AllowedFeatures$print()`
- `AllowedFeatures$clone()`

Method `new()`**:** Initialize a new AllowedFeatures class.

Usage:

```
AllowedFeatures$new(
  cli = NULL,
  api = NULL,
  deNovo = NULL,
  importMSRuns = NULL,
  importPeakLists = NULL,
  importCef = NULL,
  transformationProducts = NULL,
  ...
)
```

Arguments:

`cli` cli

`api` api

`deNovo` deNovo

`importMSRuns` importMSRuns

`importPeakLists` importPeakLists

`importCef` importCef

`transformationProducts` transformationProducts

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AllowedFeatures$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AllowedFeatures$toList()
```

Returns: AllowedFeatures as a base R list.

Examples:

```
# convert array of AllowedFeatures (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AllowedFeatures to a base R type

Usage:

```
AllowedFeatures$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AllowedFeatures

Usage:

```
AllowedFeatures$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AllowedFeatures

Method `toJSONString()`**:** To JSON String

Usage:

```
AllowedFeatures$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AllowedFeatures in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AllowedFeatures

Usage:

```
AllowedFeatures$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AllowedFeatures

Method `validateJSON()`**:** Validate JSON input with respect to AllowedFeatures and throw an exception if invalid

Usage:

```
AllowedFeatures$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AllowedFeatures$toString()
```

Returns: String representation of AllowedFeatures

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AllowedFeatures$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AllowedFeatures$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AllowedFeatures$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AllowedFeatures$toList`
## ------------------------------------------------

# convert array of AllowedFeatures (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`AnnotatedMsMsData` *AnnotatedMsMsData*

Description

AnnotatedMsMsData Class

AnnotatedMsMsData Class

Format

An `R6Class` generator object

Details

Create a new AnnotatedMsMsData

Public fields

`mergedMs2` AnnotatedSpectrum

`ms2Spectra` list(AnnotatedSpectrum)

Methods

Public methods:

- `AnnotatedMsMsData$new()`
- `AnnotatedMsMsData$toJSON()`
- `AnnotatedMsMsData$toList()`
- `AnnotatedMsMsData$toSimpleType()`
- `AnnotatedMsMsData$fromJSON()`
- `AnnotatedMsMsData$toJSONString()`
- `AnnotatedMsMsData$fromJSONString()`
- `AnnotatedMsMsData$validateJSON()`

- AnnotatedMsMsData$toString()
- AnnotatedMsMsData$isValid()
- AnnotatedMsMsData$getInvalidFields()
- AnnotatedMsMsData$print()
- AnnotatedMsMsData$clone()

Method `new()`**:** Initialize a new AnnotatedMsMsData class.

Usage:

```
AnnotatedMsMsData$new(mergedMs2, ms2Spectra, ...)
```

Arguments:

`mergedMs2` mergedMs2

`ms2Spectra` ms2Spectra

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AnnotatedMsMsData$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AnnotatedMsMsData$toList()
```

Returns: AnnotatedMsMsData as a base R list.

Examples:

```
# convert array of AnnotatedMsMsData (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AnnotatedMsMsData to a base R type

Usage:

```
AnnotatedMsMsData$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AnnotatedMsMsData

Usage:

```
AnnotatedMsMsData$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AnnotatedMsMsData

Method `toJSONString()`**:** To JSON String

Usage:

```
AnnotatedMsMsData$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AnnotatedMsMsData in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AnnotatedMsMsData

Usage:

```
AnnotatedMsMsData$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AnnotatedMsMsData

Method `validateJSON()`**:** Validate JSON input with respect to AnnotatedMsMsData and throw an exception if invalid

Usage:

```
AnnotatedMsMsData$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AnnotatedMsMsData$toString()
```

Returns: String representation of AnnotatedMsMsData

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AnnotatedMsMsData$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AnnotatedMsMsData$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AnnotatedMsMsData$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AnnotatedMsMsData$toList`
## ------------------------------------------------

# convert array of AnnotatedMsMsData (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

AnnotatedPeak *AnnotatedPeak*

Description

AnnotatedPeak Class

AnnotatedPeak Class

Format

An R6Class generator object

Details

Create a new AnnotatedPeak

Public fields

`mz` numeric [optional]

`intensity` numeric [optional]

`peakAnnotation` PeakAnnotation [optional]

Methods

Public methods:

- `AnnotatedPeak$new()`
- `AnnotatedPeak$toJSON()`
- `AnnotatedPeak$toList()`
- `AnnotatedPeak$toSimpleType()`
- `AnnotatedPeak$fromJSON()`
- `AnnotatedPeak$toJSONString()`
- `AnnotatedPeak$fromJSONString()`

- AnnotatedPeak$validateJSON()
- AnnotatedPeak$toString()
- AnnotatedPeak$isValid()
- AnnotatedPeak$getInvalidFields()
- AnnotatedPeak$print()
- AnnotatedPeak$clone()

Method `new()`**:** Initialize a new AnnotatedPeak class.

Usage:

```
AnnotatedPeak$new(mz = NULL, intensity = NULL, peakAnnotation = NULL, ...)
```

Arguments:

`mz` mz

`intensity` intensity

`peakAnnotation` peakAnnotation

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AnnotatedPeak$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AnnotatedPeak$toList()
```

Returns: AnnotatedPeak as a base R list.

Examples:

```
# convert array of AnnotatedPeak (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AnnotatedPeak to a base R type

Usage:

```
AnnotatedPeak$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AnnotatedPeak

Usage:

```
AnnotatedPeak$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AnnotatedPeak

Method `toJSONString()`**:** To JSON String

Usage:

```
AnnotatedPeak$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AnnotatedPeak in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AnnotatedPeak

Usage:

```
AnnotatedPeak$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AnnotatedPeak

Method `validateJSON()`**:** Validate JSON input with respect to AnnotatedPeak and throw an exception if invalid

Usage:

```
AnnotatedPeak$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AnnotatedPeak$toString()
```

Returns: String representation of AnnotatedPeak

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AnnotatedPeak$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AnnotatedPeak$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AnnotatedPeak$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AnnotatedPeak$toList`
## ------------------------------------------------

# convert array of AnnotatedPeak (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`AnnotatedSpectrum` *AnnotatedSpectrum*

Description

Spectrum model with peak annotations based on the fragmentation tree and Epimetheus substructure annotations. Molecular formula and adduct of the spectrum are identical to the ones of the corresponding molecular formula candidate and FragmentationTree. Fragment molecular formulas and adducts correspond to the FragmentationTree's FragmentNodes

AnnotatedSpectrum Class

Format

An `R6Class` generator object

Details

Create a new AnnotatedSpectrum

Public fields

`name` Optional display name of this spectrum. character [optional]

`msLevel` MS level of the measured spectrum. Artificial spectra with no msLevel (e.g. Simulated Isotope patterns) use null or zero integer [optional]

`collisionEnergy` Collision energy used for MS/MS spectra Null for spectra where collision energy is not applicable character [optional]

`instrument` Instrument information. character [optional]

`precursorMz` Precursor m/z of the MS/MS spectrum Null for spectra where precursor m/z is not applicable numeric [optional]

`scanNumber` Scan number of the spectrum. Might be null for artificial spectra with no scan number (e.g. Simulated Isotope patterns or merged spectra) integer [optional]

`cosineQuery` True if spectrum is in cosine query normalized format. Such spectrum is compatible with SpectralLibraryMatch peak assignments to reference spectra. character

`precursorPeak` SimplePeak [optional]

`peaks` The peaks of this spectrum which might contain additional annotations such as molecular formulas. list(AnnotatedPeak)

`absIntensityFactor` Factor to convert relative intensities to absolute intensities. Might be null or 1 for spectra where absolute intensities are not available (e.g. artificial or merged spectra) <p> DEPRECATED: Spectra are always returned with raw intensities. Use provided normalization factors to normalize on the fly. numeric [optional]

`maxNormFactor` Factor to convert absolute intensities to MAX norm. numeric [optional]

`sumNormFactor` Factor to convert absolute intensities to SUM norm. numeric [optional]

`l2NormFactor` Factor to convert absolute intensities to L2 (Euclidean) norm. numeric [optional]

`firstPeakNormFactor` Factor to convert absolute intensities to normalize intensities by first peak intensity. numeric [optional]

`spectrumAnnotation` SpectrumAnnotation [optional]

Methods

Public methods:

- `AnnotatedSpectrum$new()`
- `AnnotatedSpectrum$toJSON()`
- `AnnotatedSpectrum$toList()`
- `AnnotatedSpectrum$toSimpleType()`
- `AnnotatedSpectrum$fromJSON()`
- `AnnotatedSpectrum$toJSONString()`
- `AnnotatedSpectrum$fromJSONString()`
- `AnnotatedSpectrum$validateJSON()`
- `AnnotatedSpectrum$toString()`
- `AnnotatedSpectrum$isValid()`
- `AnnotatedSpectrum$getInvalidFields()`
- `AnnotatedSpectrum$print()`

- AnnotatedSpectrum$clone()

Method `new()`**:** Initialize a new AnnotatedSpectrum class.

Usage:

```
AnnotatedSpectrum$new(
  cosineQuery,
  peaks,
  name = NULL,
  msLevel = NULL,
  collisionEnergy = NULL,
  instrument = NULL,
  precursorMz = NULL,
  scanNumber = NULL,
  precursorPeak = NULL,
  absIntensityFactor = NULL,
  maxNormFactor = NULL,
  sumNormFactor = NULL,
  l2NormFactor = NULL,
  firstPeakNormFactor = NULL,
  spectrumAnnotation = NULL,
  ...
)
```

Arguments:

`cosineQuery` True if spectrum is in cosine query normalized format. Such spectrum is compatible with SpectralLibraryMatch peak assignments to reference spectra.

`peaks` The peaks of this spectrum which might contain additional annotations such as molecular formulas.

`name` Optional display name of this spectrum.

`msLevel` MS level of the measured spectrum. Artificial spectra with no msLevel (e.g. Simulated Isotope patterns) use null or zero

`collisionEnergy` Collision energy used for MS/MS spectra Null for spectra where collision energy is not applicable

`instrument` Instrument information.

`precursorMz` Precursor m/z of the MS/MS spectrum Null for spectra where precursor m/z is not applicable

`scanNumber` Scan number of the spectrum. Might be null for artificial spectra with no scan number (e.g. Simulated Isotope patterns or merged spectra)

`precursorPeak` precursorPeak

`absIntensityFactor` Factor to convert relative intensities to absolute intensities. Might be null or 1 for spectra where absolute intensities are not available (e.g. artificial or merged spectra) <p> DEPRECATED: Spectra are always returned with raw intensities. Use provided normalization factors to normalize on the fly.

`maxNormFactor` Factor to convert absolute intensities to MAX norm.

`sumNormFactor` Factor to convert absolute intensities to SUM norm.

`l2NormFactor` Factor to convert absolute intensities to L2 (Euclidean) norm.

`firstPeakNormFactor` Factor to convert absolute intensities to normalize intensities by first peak intensity.

`spectrumAnnotation` spectrumAnnotation

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
AnnotatedSpectrum$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
AnnotatedSpectrum$toList()
```

Returns: AnnotatedSpectrum as a base R list.

Examples:

```
# convert array of AnnotatedSpectrum (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert AnnotatedSpectrum to a base R type

Usage:

```
AnnotatedSpectrum$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AnnotatedSpectrum

Usage:

```
AnnotatedSpectrum$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AnnotatedSpectrum

Method `toJSONString()`**:** To JSON String

Usage:

```
AnnotatedSpectrum$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: AnnotatedSpectrum in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AnnotatedSpectrum

Usage:

```
AnnotatedSpectrum$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of AnnotatedSpectrum

Method `validateJSON()`**:** Validate JSON input with respect to AnnotatedSpectrum and throw an exception if invalid

Usage:

```
AnnotatedSpectrum$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
AnnotatedSpectrum$toString()
```

Returns: String representation of AnnotatedSpectrum

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
AnnotatedSpectrum$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
AnnotatedSpectrum$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
AnnotatedSpectrum$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `AnnotatedSpectrum$toList`
## ------------------------------------------------

# convert array of AnnotatedSpectrum (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`AnyValue` *AnyValue*

Description

AnyValue Class

Format

An `R6Class` generator object

Public fields

`actual_instance` the object stored in this instance.

`actual_type` the type of the object stored in this instance.

`any_of` a list of object types defined in the anyOf schema.

Methods

Public methods:

- `AnyValue$new()`
- `AnyValue$fromJSONString()`
- `AnyValue$fromJSON()`
- `AnyValue$toJSONString()`
- `AnyValue$toJSON()`
- `AnyValue$validateJSON()`
- `AnyValue$toString()`
- `AnyValue$print()`
- `AnyValue$clone()`

Method `new()`**:** Initialize a new AnyValue.

Usage:

```
AnyValue$new(instance = NULL)
```

Arguments:

`instance` an instance of the object defined in the anyOf schemas: "character", "integer", "numeric"

Method `fromJSONString()`**:** Deserialize JSON string into an instance of AnyValue. An alias to the method 'fromJSON'.

Usage:

```
AnyValue$fromJSONString(input)
```

Arguments:

`input` The input JSON.

Returns: An instance of AnyValue.

Method `fromJSON()`**:** Deserialize JSON string into an instance of AnyValue.

Usage:

```
AnyValue$fromJSON(input)
```

Arguments:

`input` The input JSON.

Returns: An instance of AnyValue.

Method `toJSONString()`**:** Serialize AnyValue to JSON string.

Usage:

```
AnyValue$toJSONString()
```

Returns: JSON string representation of the AnyValue.

Method `toJSON()`**:** Serialize AnyValue to JSON.

Usage:

```
AnyValue$toJSON()
```

Returns: JSON representation of the AnyValue.

Method `validateJSON()`**:** Validate the input JSON with respect to AnyValue and throw exception if invalid.

Usage:

```
AnyValue$validateJSON(input)
```

Arguments:

`input` The input JSON.

Method `toString()`**:** Returns the string representation of the instance.

Usage:

```
AnyValue$toString()
```

Returns: The string representation of the instance.

Method `print()`**:** Print the object

Usage:

```
AnyValue$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ApiClient` *ApiClient*

Description

ApiClient Class

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

ApiClient Class

Generic API client for OpenAPI client library builds. OpenAPI generic API client. This client handles the client- server communication, and is invariant across implementations. Specifics of the methods and models for each application are generated from the OpenAPI Generator templates.

NOTE: This class is auto generated by OpenAPI Generator (https://openapi-generator.tech). Ref: https://openapi-generator.tech Do not edit the class manually.

Public fields

`base_path` Base url

`user_agent` Default user agent

`default_headers` Default headers

`username` Username for HTTP basic authentication

`password` Password for HTTP basic authentication

`api_keys` API keys

`bearer_token` Bearer token

`timeout` Default timeout in seconds

`retry_status_codes` vector of status codes to retry

`max_retry_attempts` maximum number of retries for the status codes

Methods

Public methods:

- ApiClient$new()
- ApiClient$CallApi()
- ApiClient$Execute()
- ApiClient$deserialize()
- ApiClient$deserializeObj()
- ApiClient$select_header()
- ApiClient$clone()

Method `new()`**:** Initialize a new ApiClient.

Usage:

```
ApiClient$new(
  base_path = NULL,
  user_agent = NULL,
  default_headers = NULL,
  username = NULL,
  password = NULL,
  api_keys = NULL,
  access_token = NULL,
  bearer_token = NULL,
  timeout = NULL,
  retry_status_codes = NULL,
  max_retry_attempts = NULL
)
```

Arguments:

`base_path` Base path.

`user_agent` User agent.

`default_headers` Default headers.

`username` User name.

`password` Password.

`api_keys` API keys.

`access_token` Access token.

`bearer_token` Bearer token.

`timeout` Timeout.

`retry_status_codes` Status codes for retry.

`max_retry_attempts` Maxmium number of retry.

Method `CallApi()`**:** Prepare to make an API call with the retry logic.

Usage:

```
ApiClient$CallApi(
  url,
  method,
  query_params,
```

```
  header_params,
  form_params,
  file_params,
  accepts,
  content_types,
  body,
  is_oauth = FALSE,
  oauth_scopes = NULL,
  stream_callback = NULL,
  ...
)
```

Arguments:

`url` URL.

`method` HTTP method.

`query_params` The query parameters.

`header_params` The header parameters.

`form_params` The form parameters.

`file_params` The form parameters to upload files.

`accepts` The HTTP accept headers.

`content_types` The HTTP content-type headers.

`body` The HTTP request body.

`is_oauth` True if the endpoints required OAuth authentication.

`oauth_scopes` OAuth scopes.

`stream_callback` Callback function to process the data stream.

`...` Other optional arguments.

Returns: HTTP response

Method `Execute()`**:** Make an API call

Usage:

```
ApiClient$Execute(
  req,
  method,
  query_params,
  header_params,
  form_params,
  file_params,
  accepts,
  content_types,
  body,
  is_oauth = FALSE,
  oauth_scopes = NULL,
  stream_callback = NULL,
  ...
)
```

Arguments:

`req` httr2 request.

`method` HTTP method.

`query_params` The query parameters.

`header_params` The header parameters.

`form_params` The form parameters.

`file_params` The form parameters for uploading files.

`accepts` The HTTP accept headers.

`content_types` The HTTP content-type headers.

`body` The HTTP request body.

`is_oauth` True if the endpoints required OAuth authentication.

`oauth_scopes` OAuth scopes.

`stream_callback` Callback function to process data stream.

`...` Other optional arguments.

Returns: HTTP response

Method `deserialize()`**:** Deserialize the content of API response to the given type.

Usage:

```
ApiClient$deserialize(raw_response, return_type, pkg_env)
```

Arguments:

`raw_response` Raw response.

`return_type` R return type.

`pkg_env` Package environment.

Returns: Deserialized object.

Method `deserializeObj()`**:** Deserialize the response from jsonlite object based on the given type. by handling complex and nested types by iterating recursively Example return_types will be like "array[integer]", "map(Pet)", "array[map(Tag)]", etc.

Usage:

```
ApiClient$deserializeObj(obj, return_type, pkg_env)
```

Arguments:

`obj` Response object.

`return_type` R return type.

`pkg_env` Package environment.

Returns: Deserialized object.

Method `select_header()`**:** Return a property header (for accept or content-type). If JSON-related MIME is found, return it. Otherwise, return the first one, if any.

Usage:

```
ApiClient$select_header(headers)
```

Arguments:

`headers` A list of headers

Returns: A header (e.g. 'application/json')

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`ApiResponse` *ApiResponse*

Description

ApiResponse Class

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`content` The deserialized response body.

`response` The raw response from the endpoint.

`status_code` The HTTP response status code.

`status_code_desc` The brief description of the HTTP response status code.

`headers` The HTTP response headers.

Methods

Public methods:

- `ApiResponse$new()`
- `ApiResponse$response_as_text()`
- `ApiResponse$clone()`

Method `new()`**:** Initialize a new ApiResponse class.

Usage:

```
ApiResponse$new(
  content = NULL,
  response = NULL,
  status_code = NULL,
  status_code_desc = NULL,
  headers = NULL
)
```

Arguments:

`content` The deserialized response body.

`response` The raw response from the endpoint.

`status_code` The HTTP response status code.

`status_code_desc` The brief description of the HTTP response status code.

`headers` The HTTP response headers.

Method `response_as_text()`**:** The response is stored as a raw vector. Use this to access the response after converting it to text. If the response is not coercible to text NA is returned.

Usage:

```
ApiResponse$response_as_text(from_encoding = "", to_encoding = "UTF-8")
```

Arguments:

`from_encoding` The encoding of the raw response.

`to_encoding` The target encoding of the return value.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`BackgroundComputationsStateEvent` *BackgroundComputationsStateEvent*

Description

BackgroundComputationsStateEvent Class

BackgroundComputationsStateEvent Class

Format

An `R6Class` generator object

Details

Create a new BackgroundComputationsStateEvent

Public fields

`affectedJobs` list(Job)

`numberOfJobs` integer

`numberOfRunningJobs` integer

`numberOfFinishedJobs` integer

Methods

Public methods:

- `BackgroundComputationsStateEvent$new()`
- `BackgroundComputationsStateEvent$toJSON()`
- `BackgroundComputationsStateEvent$toList()`
- `BackgroundComputationsStateEvent$toSimpleType()`
- `BackgroundComputationsStateEvent$fromJSON()`
- `BackgroundComputationsStateEvent$toJSONString()`
- `BackgroundComputationsStateEvent$fromJSONString()`
- `BackgroundComputationsStateEvent$validateJSON()`
- `BackgroundComputationsStateEvent$toString()`
- `BackgroundComputationsStateEvent$isValid()`
- `BackgroundComputationsStateEvent$getInvalidFields()`
- `BackgroundComputationsStateEvent$print()`
- `BackgroundComputationsStateEvent$clone()`

Method `new()`**:** Initialize a new BackgroundComputationsStateEvent class.

Usage:

```
BackgroundComputationsStateEvent$new(
  affectedJobs,
  numberOfJobs,
  numberOfRunningJobs,
  numberOfFinishedJobs,
  ...
)
```

Arguments:

`affectedJobs` affectedJobs

`numberOfJobs` numberOfJobs

`numberOfRunningJobs` numberOfRunningJobs

`numberOfFinishedJobs` numberOfFinishedJobs

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
BackgroundComputationsStateEvent$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
BackgroundComputationsStateEvent$toList()
```

Returns: BackgroundComputationsStateEvent as a base R list.

Examples:

```
# convert array of BackgroundComputationsStateEvent (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert BackgroundComputationsStateEvent to a base R type

Usage:

```
BackgroundComputationsStateEvent$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of BackgroundComputationsStateEvent

Usage:

```
BackgroundComputationsStateEvent$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BackgroundComputationsStateEvent

Method `toJSONString()`**:** To JSON String

Usage:

```
BackgroundComputationsStateEvent$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: BackgroundComputationsStateEvent in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of BackgroundComputationsStateEvent

Usage:

```
BackgroundComputationsStateEvent$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BackgroundComputationsStateEvent

Method `validateJSON()`**:** Validate JSON input with respect to BackgroundComputationsStateEvent and throw an exception if invalid

Usage:

```
BackgroundComputationsStateEvent$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
BackgroundComputationsStateEvent$toString()
```

Returns: String representation of BackgroundComputationsStateEvent

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
BackgroundComputationsStateEvent$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
BackgroundComputationsStateEvent$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
BackgroundComputationsStateEvent$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `BackgroundComputationsStateEvent$toList`
## ------------------------------------------------

# convert array of BackgroundComputationsStateEvent (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

BasicSpectrum *BasicSpectrum*

Description

BasicSpectrum Class

BasicSpectrum Class

Format

An R6Class generator object

Details

Create a new BasicSpectrum

Public fields

`name` Optional display name of this spectrum. character [optional]

`msLevel` MS level of the measured spectrum. Artificial spectra with no msLevel (e.g. Simulated Isotope patterns) use null or zero integer [optional]

`collisionEnergy` Collision energy used for MS/MS spectra Null for spectra where collision energy is not applicable character [optional]

`instrument` Instrument information. character [optional]

`precursorMz` Precursor m/z of the MS/MS spectrum Null for spectra where precursor m/z is not applicable numeric [optional]

`scanNumber` Scan number of the spectrum. Might be null for artificial spectra with no scan number (e.g. Simulated Isotope patterns or merged spectra) integer [optional]

`cosineQuery` True if spectrum is in cosine query normalized format. Such spectrum is compatible with SpectralLibraryMatch peak assignments to reference spectra. character

`precursorPeak` SimplePeak [optional]

`peaks` The peaks of this spectrum which might contain additional annotations such as molecular formulas. list(SimplePeak)

`absIntensityFactor` Factor to convert relative intensities to absolute intensities. Might be null or 1 for spectra where absolute intensities are not available (e.g. artificial or merged spectra) <p> DEPRECATED: Spectra are always returned with raw intensities. Use provided normalization factors to normalize on the fly. numeric [optional]

`maxNormFactor` Factor to convert absolute intensities to MAX norm. numeric [optional]

`sumNormFactor` Factor to convert absolute intensities to SUM norm. numeric [optional]

`l2NormFactor` Factor to convert absolute intensities to L2 (Euclidean) norm. numeric [optional]

`firstPeakNormFactor` Factor to convert absolute intensities to normalize intensities by first peak intensity. numeric [optional]

Methods

Public methods:

- BasicSpectrum$new()
- BasicSpectrum$toJSON()
- BasicSpectrum$toList()
- BasicSpectrum$toSimpleType()
- BasicSpectrum$fromJSON()
- BasicSpectrum$toJSONString()
- BasicSpectrum$fromJSONString()
- BasicSpectrum$validateJSON()
- BasicSpectrum$toString()
- BasicSpectrum$isValid()
- BasicSpectrum$getInvalidFields()
- BasicSpectrum$print()
- BasicSpectrum$clone()

Method `new()`**:** Initialize a new BasicSpectrum class.

Usage:

```
BasicSpectrum$new(
  cosineQuery,
  peaks,
  name = NULL,
  msLevel = NULL,
  collisionEnergy = NULL,
  instrument = NULL,
  precursorMz = NULL,
  scanNumber = NULL,
  precursorPeak = NULL,
  absIntensityFactor = NULL,
  maxNormFactor = NULL,
  sumNormFactor = NULL,
  l2NormFactor = NULL,
  firstPeakNormFactor = NULL,
  ...
)
```

Arguments:

`cosineQuery` True if spectrum is in cosine query normalized format. Such spectrum is compatible with SpectralLibraryMatch peak assignments to reference spectra.

`peaks` The peaks of this spectrum which might contain additional annotations such as molecular formulas.

`name` Optional display name of this spectrum.

`msLevel` MS level of the measured spectrum. Artificial spectra with no msLevel (e.g. Simulated Isotope patterns) use null or zero

`collisionEnergy` Collision energy used for MS/MS spectra Null for spectra where collision energy is not applicable

`instrument` Instrument information.

`precursorMz` Precursor m/z of the MS/MS spectrum Null for spectra where precursor m/z is not applicable

`scanNumber` Scan number of the spectrum. Might be null for artificial spectra with no scan number (e.g. Simulated Isotope patterns or merged spectra)

`precursorPeak` precursorPeak

`absIntensityFactor` Factor to convert relative intensities to absolute intensities. Might be null or 1 for spectra where absolute intensities are not available (e.g. artificial or merged spectra) <p> DEPRECATED: Spectra are always returned with raw intensities. Use provided normalization factors to normalize on the fly.

`maxNormFactor` Factor to convert absolute intensities to MAX norm.

`sumNormFactor` Factor to convert absolute intensities to SUM norm.

`l2NormFactor` Factor to convert absolute intensities to L2 (Euclidean) norm.

`firstPeakNormFactor` Factor to convert absolute intensities to normalize intensities by first peak intensity.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
BasicSpectrum$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
BasicSpectrum$toList()
```

Returns: BasicSpectrum as a base R list.

Examples:

```
# convert array of BasicSpectrum (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert BasicSpectrum to a base R type

Usage:

```
BasicSpectrum$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of BasicSpectrum

Usage:

```
BasicSpectrum$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BasicSpectrum

Method `toJSONString()`**:** To JSON String

Usage:

```
BasicSpectrum$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: BasicSpectrum in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of BasicSpectrum

Usage:

```
BasicSpectrum$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BasicSpectrum

Method `validateJSON()`**:** Validate JSON input with respect to BasicSpectrum and throw an exception if invalid

Usage:

```
BasicSpectrum$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
BasicSpectrum$toString()
```

Returns: String representation of BasicSpectrum

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
BasicSpectrum$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
BasicSpectrum$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
BasicSpectrum$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `BasicSpectrum$toList`
## ------------------------------------------------

# convert array of BasicSpectrum (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`BinaryFingerprint` *BinaryFingerprint*

Description

BinaryFingerprint Class

BinaryFingerprint Class

Format

An `R6Class` generator object

Details

Create a new BinaryFingerprint

Public fields

`bitsSet` Array that contains all RELATIVE indices (masked FP) of bits that are set (are 1) list(integer) [optional]

`length` Size of the fingerprint (masked fp), e.g. to reconstruct the binary array from the array of set bits integer [optional]

Methods

Public methods:

- BinaryFingerprint$new()
- BinaryFingerprint$toJSON()
- BinaryFingerprint$toList()
- BinaryFingerprint$toSimpleType()
- BinaryFingerprint$fromJSON()
- BinaryFingerprint$toJSONString()
- BinaryFingerprint$fromJSONString()
- BinaryFingerprint$validateJSON()
- BinaryFingerprint$toString()
- BinaryFingerprint$isValid()
- BinaryFingerprint$getInvalidFields()
- BinaryFingerprint$print()
- BinaryFingerprint$clone()

Method `new()`**:** Initialize a new BinaryFingerprint class.

Usage:

```
BinaryFingerprint$new(bitsSet = NULL, length = NULL, ...)
```

Arguments:

`bitsSet` Array that contains all RELATIVE indices (masked FP) of bits that are set (are 1)

`length` Size of the fingerprint (masked fp), e.g. to reconstruct the binary array from the array of set bits

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
BinaryFingerprint$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
BinaryFingerprint$toList()
```

Returns: BinaryFingerprint as a base R list.

Examples:

```
# convert array of BinaryFingerprint (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert BinaryFingerprint to a base R type

Usage:

```
BinaryFingerprint$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of BinaryFingerprint

Usage:

```
BinaryFingerprint$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BinaryFingerprint

Method `toJSONString()`**:** To JSON String

Usage:

```
BinaryFingerprint$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: BinaryFingerprint in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of BinaryFingerprint

Usage:

```
BinaryFingerprint$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BinaryFingerprint

Method `validateJSON()`**:** Validate JSON input with respect to BinaryFingerprint and throw an exception if invalid

Usage:

```
BinaryFingerprint$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
BinaryFingerprint$toString()
```

Returns: String representation of BinaryFingerprint

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
BinaryFingerprint$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
BinaryFingerprint$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
BinaryFingerprint$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `BinaryFingerprint$toList`
## ------------------------------------------------

# convert array of BinaryFingerprint (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`BioTransformerParameters` *BioTransformerParameters*

Description

BioTransformerParameters Class

BioTransformerParameters Class

Format

An `R6Class` generator object

Details

Create a new BioTransformerParameters

Public fields

`cyp450Mode` Specify the Phase I/Cyp450 mode for all provided BioTransformerSequenceSteps. Will only be applied to Steps that require the Cyp450 mode as parameter. Can be null in cases where only BioTransformerSequenceSteps are specified that do not need the Cyp450 mode. character

`p2Mode` Specify the Phase II mode for all provided BioTransformerSequenceSteps. Will only be applied to Steps that require the Phase II mode as parameter. Can be null in cases where only BioTransformerSequenceSteps are specified that do not need the Phase II mode. character

`useDB` Specify whether structures should additionally be retrieved from the HMDB database. character [optional]

`bioTransformerSequenceSteps` Specify BioTransformerSequenceSteps to be applied to input structures. MultiStep MetabolicTransformations can only be used as singletons (list size of one). list(BioTransformerSequenceStep)

Methods

Public methods:

- `BioTransformerParameters$new()`
- `BioTransformerParameters$toJSON()`
- `BioTransformerParameters$toList()`
- `BioTransformerParameters$toSimpleType()`
- `BioTransformerParameters$fromJSON()`
- `BioTransformerParameters$toJSONString()`
- `BioTransformerParameters$fromJSONString()`
- `BioTransformerParameters$validateJSON()`
- `BioTransformerParameters$toString()`
- `BioTransformerParameters$isValid()`
- `BioTransformerParameters$getInvalidFields()`
- `BioTransformerParameters$print()`
- `BioTransformerParameters$clone()`

Method `new()`**:** Initialize a new BioTransformerParameters class.

Usage:

```
BioTransformerParameters$new(
  cyp450Mode,
  p2Mode,
  bioTransformerSequenceSteps,
  useDB = TRUE,
  ...
)
```

Arguments:

`cyp450Mode` Specify the Phase I/Cyp450 mode for all provided BioTransformerSequenceSteps. Will only be applied to Steps that require the Cyp450 mode as parameter. Can be null in cases where only BioTransformerSequenceSteps are specified that do not need the Cyp450 mode.

`p2Mode` Specify the Phase II mode for all provided BioTransformerSequenceSteps. Will only be applied to Steps that require the Phase II mode as parameter. Can be null in cases where only BioTransformerSequenceSteps are specified that do not need the Phase II mode.

`bioTransformerSequenceSteps` Specify BioTransformerSequenceSteps to be applied to input structures. MultiStep MetabolicTransformations can only be used as singletons (list size of one).

`useDB` Specify whether structures should additionally be retrieved from the HMDB database.. Default to TRUE.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
BioTransformerParameters$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
BioTransformerParameters$toList()
```

Returns: BioTransformerParameters as a base R list.

Examples:

```
# convert array of BioTransformerParameters (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert BioTransformerParameters to a base R type

Usage:

```
BioTransformerParameters$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of BioTransformerParameters

Usage:

```
BioTransformerParameters$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BioTransformerParameters

Method `toJSONString()`**:** To JSON String

Usage:

```
BioTransformerParameters$toJSONString(...)
```

Arguments:

... Parameters passed to 'jsonlite::toJSON'

Returns: BioTransformerParameters in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of BioTransformerParameters

Usage:

```
BioTransformerParameters$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BioTransformerParameters

Method `validateJSON()`**:** Validate JSON input with respect to BioTransformerParameters and throw an exception if invalid

Usage:

```
BioTransformerParameters$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
BioTransformerParameters$toString()
```

Returns: String representation of BioTransformerParameters

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
BioTransformerParameters$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
BioTransformerParameters$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
BioTransformerParameters$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `BioTransformerParameters$toList`
## ------------------------------------------------

# convert array of BioTransformerParameters (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`BioTransformerSequenceStep` *BioTransformerSequenceStep*

Description

BioTransformerSequenceStep Class

BioTransformerSequenceStep Class

Format

An `R6Class` generator object

Details

Create a new BioTransformerSequenceStep

Public fields

`metabolicTransformation` character [optional]

`iterations` integer [optional]

Methods

Public methods:

- `BioTransformerSequenceStep$new()`
- `BioTransformerSequenceStep$toJSON()`
- `BioTransformerSequenceStep$toList()`
- `BioTransformerSequenceStep$toSimpleType()`
- `BioTransformerSequenceStep$fromJSON()`
- `BioTransformerSequenceStep$toJSONString()`
- `BioTransformerSequenceStep$fromJSONString()`

- BioTransformerSequenceStep$validateJSON()
- BioTransformerSequenceStep$toString()
- BioTransformerSequenceStep$isValid()
- BioTransformerSequenceStep$getInvalidFields()
- BioTransformerSequenceStep$print()
- BioTransformerSequenceStep$clone()

Method `new()`**:** Initialize a new BioTransformerSequenceStep class.

Usage:

```
BioTransformerSequenceStep$new(
  metabolicTransformation = NULL,
  iterations = NULL,
  ...
)
```

Arguments:

`metabolicTransformation` metabolicTransformation

`iterations` iterations

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
BioTransformerSequenceStep$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
BioTransformerSequenceStep$toList()
```

Returns: BioTransformerSequenceStep as a base R list.

Examples:

```
# convert array of BioTransformerSequenceStep (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert BioTransformerSequenceStep to a base R type

Usage:

```
BioTransformerSequenceStep$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of BioTransformerSequenceStep

Usage:

```
BioTransformerSequenceStep$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BioTransformerSequenceStep

Method `toJSONString()`**:** To JSON String

Usage:

```
BioTransformerSequenceStep$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: BioTransformerSequenceStep in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of BioTransformerSequenceStep

Usage:

```
BioTransformerSequenceStep$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of BioTransformerSequenceStep

Method `validateJSON()`**:** Validate JSON input with respect to BioTransformerSequenceStep and throw an exception if invalid

Usage:

```
BioTransformerSequenceStep$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
BioTransformerSequenceStep$toString()
```

Returns: String representation of BioTransformerSequenceStep

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
BioTransformerSequenceStep$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
BioTransformerSequenceStep$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
BioTransformerSequenceStep$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `BioTransformerSequenceStep$toList`
## ------------------------------------------------

# convert array of BioTransformerSequenceStep (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Canopus` *Canopus*

Description

User/developer friendly parameter subset for the CANOPUS tool CANOPUS is parameter-free, so this object is just a flag that CANOPUS should be executed. Needs results from the FingerprintPrediction tool.

Canopus Class

Format

An `R6Class` generator object

Details

Create a new Canopus

Public fields

`enabled` Indicates whether the tool is enabled. character [optional]

Methods

Public methods:

- Canopus$new()
- Canopus$toJSON()
- Canopus$toList()
- Canopus$toSimpleType()
- Canopus$fromJSON()
- Canopus$toJSONString()
- Canopus$fromJSONString()
- Canopus$validateJSON()
- Canopus$toString()
- Canopus$isValid()
- Canopus$getInvalidFields()
- Canopus$print()
- Canopus$clone()

Method `new()`**:** Initialize a new Canopus class.

Usage:

```
Canopus$new(enabled = NULL, ...)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Canopus$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Canopus$toList()
```

Returns: Canopus as a base R list.

Examples:

```
# convert array of Canopus (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Canopus to a base R type

Usage:

```
Canopus$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Canopus

Usage:

```
Canopus$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Canopus

Method `toJSONString()`**:** To JSON String

Usage:

```
Canopus$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Canopus in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Canopus

Usage:

```
Canopus$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Canopus

Method `validateJSON()`**:** Validate JSON input with respect to Canopus and throw an exception if invalid

Usage:

```
Canopus$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Canopus$toString()
```

Returns: String representation of Canopus

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Canopus$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Canopus$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Canopus$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Canopus$toList`
## ------------------------------------------------

# convert array of Canopus (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`CanopusPrediction` *CanopusPrediction*

Description

Container class that holds the CANOPUS compound class predictions for all predictable compound classes. This is the full CANOPUS result.

CanopusPrediction Class

Format

An `R6Class` generator object

Details

Create a new CanopusPrediction

Public fields

`classyFireClasses` All predicted ClassyFire classes list(CompoundClass) [optional]

`npcClasses` All predicted NPC classes list(CompoundClass) [optional]

Methods

Public methods:

- `CanopusPrediction$new()`
- `CanopusPrediction$toJSON()`
- `CanopusPrediction$toList()`
- `CanopusPrediction$toSimpleType()`
- `CanopusPrediction$fromJSON()`
- `CanopusPrediction$toJSONString()`
- `CanopusPrediction$fromJSONString()`
- `CanopusPrediction$validateJSON()`
- `CanopusPrediction$toString()`
- `CanopusPrediction$isValid()`
- `CanopusPrediction$getInvalidFields()`
- `CanopusPrediction$print()`
- `CanopusPrediction$clone()`

Method `new()`**:** Initialize a new CanopusPrediction class.

Usage:

```
CanopusPrediction$new(classyFireClasses = NULL, npcClasses = NULL, ...)
```

Arguments:

`classyFireClasses` All predicted ClassyFire classes

`npcClasses` All predicted NPC classes

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
CanopusPrediction$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
CanopusPrediction$toList()
```

Returns: CanopusPrediction as a base R list.

Examples:

```
# convert array of CanopusPrediction (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert CanopusPrediction to a base R type

Usage:

```
CanopusPrediction$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of CanopusPrediction

Usage:

```
CanopusPrediction$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CanopusPrediction

Method `toJSONString()`**:** To JSON String

Usage:

```
CanopusPrediction$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: CanopusPrediction in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of CanopusPrediction

Usage:

```
CanopusPrediction$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CanopusPrediction

Method `validateJSON()`**:** Validate JSON input with respect to CanopusPrediction and throw an exception if invalid

Usage:

```
CanopusPrediction$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
CanopusPrediction$toString()
```

Returns: String representation of CanopusPrediction

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
CanopusPrediction$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
CanopusPrediction$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
CanopusPrediction$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `CanopusPrediction$toList`
## ------------------------------------------------

# convert array of CanopusPrediction (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Compound` *Compound*

Description

Compound Class

Compound Class

Format

An `R6Class` generator object

Details

Create a new Compound

Public fields

`compoundId` Unique id of the compound entity. character [optional]

`name` Some (optional) human-readable name character [optional]

`rtStartSeconds` The merged/consensus retention time start (earliest rt) of the compound numeric [optional]

`rtEndSeconds` The merged/consensus retention time end (latest rt) of the compound numeric [optional]

`neutralMass` Neutral mass of the compound. Ion mass minus the mass of the assigned adduct of each feature of the compound should result in the same neutral mass numeric [optional]

`features` List of aligned features (adducts) that belong to the compound list(AlignedFeature) [optional]

`consensusAnnotations` ConsensusAnnotationsCSI [optional]

`consensusAnnotationsDeNovo` ConsensusAnnotationsDeNovo [optional]

`customAnnotations` ConsensusAnnotationsCSI [optional]

`tags` Key: tagName, value: tag named list(Tag) [optional]

Methods

Public methods:

- `Compound$new()`
- `Compound$toJSON()`
- `Compound$toList()`
- `Compound$toSimpleType()`
- `Compound$fromJSON()`
- `Compound$toJSONString()`
- `Compound$fromJSONString()`

- Compound$validateJSON()
- Compound$toString()
- Compound$isValid()
- Compound$getInvalidFields()
- Compound$print()
- Compound$clone()

Method `new()`**:** Initialize a new Compound class.

Usage:

```
Compound$new(
  compoundId = NULL,
  name = NULL,
  rtStartSeconds = NULL,
  rtEndSeconds = NULL,
  neutralMass = NULL,
  features = NULL,
  consensusAnnotations = NULL,
  consensusAnnotationsDeNovo = NULL,
  customAnnotations = NULL,
  tags = NULL,
  ...
)
```

Arguments:

`compoundId` Unique id of the compound entity.

`name` Some (optional) human-readable name

`rtStartSeconds` The merged/consensus retention time start (earliest rt) of the compound

`rtEndSeconds` The merged/consensus retention time end (latest rt) of the compound

`neutralMass` Neutral mass of the compound. Ion mass minus the mass of the assigned adduct of each feature of the compound should result in the same neutral mass

`features` List of aligned features (adducts) that belong to the compound

`consensusAnnotations` consensusAnnotations

`consensusAnnotationsDeNovo` consensusAnnotationsDeNovo

`customAnnotations` customAnnotations

`tags` Key: tagName, value: tag

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Compound$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Compound$toList()
```

Returns: Compound as a base R list.

Examples:

```
# convert array of Compound (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Compound to a base R type

Usage:

```
Compound$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Compound

Usage:

```
Compound$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Compound

Method `toJSONString()`**:** To JSON String

Usage:

```
Compound$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Compound in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Compound

Usage:

```
Compound$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Compound

Method `validateJSON()`**:** Validate JSON input with respect to Compound and throw an exception if invalid

Usage:

```
Compound$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Compound$toString()
```

Returns: String representation of Compound

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Compound$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Compound$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Compound$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Compound$toList`
## ------------------------------------------------

# convert array of Compound (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`CompoundClass` *CompoundClass*

Description

Predicted compound class with name, probability and id if available. (ClassyFire and NPC). This can be seen as the set of classes a feature most likely belongs to

CompoundClass Class

Format

An `R6Class` generator object

Details

Create a new CompoundClass

Public fields

`type` Specifies the classification ontology the CompoundClass belongs to. character [optional]

`level` Name of the level this compound class belongs to character [optional]

`levelIndex` Index of the level this compound class belongs to integer [optional]

`name` Name of the compound class. character [optional]

`description` Description of the compound class. character [optional]

`id` Unique id of the class. Might be undefined for certain classification ontologies. integer [optional]

`probability` Prediction probability. numeric [optional]

`index` Absolute index of this property in the predicted vector/embedding integer [optional]

`parentId` Unique id of the parent class. Might be undefined for certain classification ontologies. integer [optional]

`parentName` Name of the parent compound class. character [optional]

Methods

Public methods:

- `CompoundClass$new()`
- `CompoundClass$toJSON()`
- `CompoundClass$toList()`
- `CompoundClass$toSimpleType()`
- `CompoundClass$fromJSON()`
- `CompoundClass$toJSONString()`
- `CompoundClass$fromJSONString()`
- `CompoundClass$validateJSON()`

- CompoundClass$toString()
- CompoundClass$isValid()
- CompoundClass$getInvalidFields()
- CompoundClass$print()
- CompoundClass$clone()

Method `new()`**:** Initialize a new CompoundClass class.

Usage:

```
CompoundClass$new(
  type = NULL,
  level = NULL,
  levelIndex = NULL,
  name = NULL,
  description = NULL,
  id = NULL,
  probability = NULL,
  index = NULL,
  parentId = NULL,
  parentName = NULL,
  ...
)
```

Arguments:

`type` Specifies the classification ontology the CompoundClass belongs to.

`level` Name of the level this compound class belongs to

`levelIndex` Index of the level this compound class belongs to

`name` Name of the compound class.

`description` Description of the compound class.

`id` Unique id of the class. Might be undefined for certain classification ontologies.

`probability` Prediction probability.

`index` Absolute index of this property in the predicted vector/embedding

`parentId` Unique id of the parent class. Might be undefined for certain classification ontologies.

`parentName` Name of the parent compound class.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
CompoundClass$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
CompoundClass$toList()
```

Returns: CompoundClass as a base R list.

Examples:

```
# convert array of CompoundClass (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert CompoundClass to a base R type

Usage:

```
CompoundClass$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of CompoundClass

Usage:

```
CompoundClass$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CompoundClass

Method `toJSONString()`**:** To JSON String

Usage:

```
CompoundClass$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: CompoundClass in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of CompoundClass

Usage:

```
CompoundClass$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CompoundClass

Method `validateJSON()`**:** Validate JSON input with respect to CompoundClass and throw an exception if invalid

Usage:

```
CompoundClass$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
CompoundClass$toString()
```

Returns: String representation of CompoundClass

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
CompoundClass$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
CompoundClass$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
CompoundClass$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `CompoundClass$toList`
## ------------------------------------------------

# convert array of CompoundClass (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

CompoundClasses *CompoundClasses*

Description

Container class that holds the most likely compound class for different levels of each ontology for a certain Compound/Feature/FormulaCandidate/PredictedFingerprint.

CompoundClasses Class

Format

An `R6Class` generator object

Details

Create a new CompoundClasses

Public fields

`npcPathway` CompoundClass [optional]

`npcSuperclass` CompoundClass [optional]

`npcClass` CompoundClass [optional]

`classyFireLineage` Most likely ClassyFire lineage, ordered from least specific to most specific class classyFireLineage.get(classyFireLineage.size() - 1) gives the most specific ClassyFire compound class annotation list(CompoundClass) [optional]

`classyFireAlternatives` Alternative ClassyFire classes with high probability that do not fit into the lineage list(CompoundClass) [optional]

Methods

Public methods:

- `CompoundClasses$new()`
- `CompoundClasses$toJSON()`
- `CompoundClasses$toList()`
- `CompoundClasses$toSimpleType()`
- `CompoundClasses$fromJSON()`
- `CompoundClasses$toJSONString()`
- `CompoundClasses$fromJSONString()`
- `CompoundClasses$validateJSON()`
- `CompoundClasses$toString()`
- `CompoundClasses$isValid()`
- `CompoundClasses$getInvalidFields()`
- `CompoundClasses$print()`
- `CompoundClasses$clone()`

Method `new()`**:** Initialize a new CompoundClasses class.

Usage:

```
CompoundClasses$new(
  npcPathway = NULL,
  npcSuperclass = NULL,
  npcClass = NULL,
  classyFireLineage = NULL,
  classyFireAlternatives = NULL,
  ...
)
```

Arguments:

`npcPathway` npcPathway

`npcSuperclass` npcSuperclass

`npcClass` npcClass

`classyFireLineage` Most likely ClassyFire lineage, ordered from least specific to most specific class classyFireLineage.get(classyFireLineage.size() - 1) gives the most specific ClassyFire compound class annotation

`classyFireAlternatives` Alternative ClassyFire classes with high probability that do not fit into the lineage

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
CompoundClasses$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
CompoundClasses$toList()
```

Returns: CompoundClasses as a base R list.

Examples:

```
# convert array of CompoundClasses (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert CompoundClasses to a base R type

Usage:

```
CompoundClasses$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of CompoundClasses

Usage:

```
CompoundClasses$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CompoundClasses

Method `toJSONString()`**:** To JSON String

Usage:

```
CompoundClasses$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: CompoundClasses in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of CompoundClasses

Usage:

```
CompoundClasses$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CompoundClasses

Method `validateJSON()`**:** Validate JSON input with respect to CompoundClasses and throw an exception if invalid

Usage:

```
CompoundClasses$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
CompoundClasses$toString()
```

Returns: String representation of CompoundClasses

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
CompoundClasses$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
CompoundClasses$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
CompoundClasses$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `CompoundClasses$toList`
## ------------------------------------------------

# convert array of CompoundClasses (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`CompoundImport` *CompoundImport*

Description

CompoundImport Class

CompoundImport Class

Format

An `R6Class` generator object

Details

Create a new CompoundImport

Public fields

`name` Some (optional) human-readable name character [optional]

`features` The features this compound consists of. list(FeatureImport)

Methods

Public methods:

- CompoundImport$new()
- CompoundImport$toJSON()
- CompoundImport$toList()
- CompoundImport$toSimpleType()
- CompoundImport$fromJSON()
- CompoundImport$toJSONString()
- CompoundImport$fromJSONString()
- CompoundImport$validateJSON()
- CompoundImport$toString()
- CompoundImport$isValid()
- CompoundImport$getInvalidFields()
- CompoundImport$print()
- CompoundImport$clone()

Method `new()`**:** Initialize a new CompoundImport class.

Usage:

```
CompoundImport$new(features, name = NULL, ...)
```

Arguments:

`features` The features this compound consists of.

`name` Some (optional) human-readable name

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
CompoundImport$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
CompoundImport$toList()
```

Returns: CompoundImport as a base R list.

Examples:

```
# convert array of CompoundImport (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert CompoundImport to a base R type

Usage:

```
CompoundImport$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of CompoundImport

Usage:

```
CompoundImport$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CompoundImport

Method `toJSONString()`**:** To JSON String

Usage:

```
CompoundImport$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: CompoundImport in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of CompoundImport

Usage:

```
CompoundImport$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of CompoundImport

Method `validateJSON()`**:** Validate JSON input with respect to CompoundImport and throw an exception if invalid

Usage:

```
CompoundImport$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
CompoundImport$toString()
```

Returns: String representation of CompoundImport

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
CompoundImport$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
CompoundImport$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
CompoundImport$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `CompoundImport$toList`
## ------------------------------------------------

# convert array of CompoundImport (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`CompoundsApi` *Compounds operations*

Description

CompoundsApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `CompoundsApi$new()`
- `CompoundsApi$AddCompounds()`
- `CompoundsApi$AddCompoundsWithHttpInfo()`
- `CompoundsApi$DeleteCompound()`
- `CompoundsApi$DeleteCompoundWithHttpInfo()`
- `CompoundsApi$GetCompound()`
- `CompoundsApi$GetCompoundWithHttpInfo()`
- `CompoundsApi$GetCompoundQuantTable()`
- `CompoundsApi$GetCompoundQuantTableWithHttpInfo()`
- `CompoundsApi$GetCompounds()`
- `CompoundsApi$GetCompoundsWithHttpInfo()`
- `CompoundsApi$GetCompoundsPage()`
- `CompoundsApi$GetCompoundsPageWithHttpInfo()`
- `CompoundsApi$GetCompoundsSearchableFields()`
- `CompoundsApi$GetCompoundsSearchableFieldsWithHttpInfo()`
- `CompoundsApi$clone()`

Method `new()`**:** Initialize a new CompoundsApi.

Usage:

```
CompoundsApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `AddCompounds()`**:** Import compounds and their contained features.

Usage:

```
CompoundsApi$AddCompounds(
  project_id,
  compound_import,
  profile = NULL,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`compound_import` the compound data to be imported

`profile` (optional) profile describing the instrument used to measure the data. Used to merge spectra.

`opt_fields` (optional) set of optional fields to be included. Use 'none' to override defaults. (default value: ["none"])

`opt_fields_features` (optional) set of optional fields of the nested features to be included. Use 'none' to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[Compound]

Method `AddCompoundsWithHttpInfo()`**:** Import compounds and their contained features.

Usage:

```
CompoundsApi$AddCompoundsWithHttpInfo(
  project_id,
  compound_import,
  profile = NULL,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`compound_import` the compound data to be imported

`profile` (optional) profile describing the instrument used to measure the data. Used to merge spectra.

`opt_fields` (optional) set of optional fields to be included. Use 'none' to override defaults. (default value: ["none"])

`opt_fields_features` (optional) set of optional fields of the nested features to be included. Use 'none' to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[Compound]) with additional information such as HTTP status code, headers

Method `DeleteCompound()`**:** Delete compound (group of ion identities) with the given identifier (and the included features) from the specified project-space.

Usage:

```
CompoundsApi$DeleteCompound(project_id, compound_id, ...)
```

Arguments:

`project_id` project-space to delete from.

`compound_id` identifier of the compound to delete.

`...` Other optional arguments

Returns: void

Method `DeleteCompoundWithHttpInfo()`**:** Delete compound (group of ion identities) with the given identifier (and the included features) from the specified project-space.

Usage:

```
CompoundsApi$DeleteCompoundWithHttpInfo(project_id, compound_id, ...)
```

Arguments:

`project_id` project-space to delete from.

`compound_id` identifier of the compound to delete.

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `GetCompound()`**:** Get compound (group of ion identities) with the given identifier from the specified project-space.

Usage:

```
CompoundsApi$GetCompound(
  project_id,
  compound_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`compound_id` identifier of the compound (group of ion identities) to access.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`opt_fields_features` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Compound

Method `GetCompoundWithHttpInfo()`**:** Get compound (group of ion identities) with the given identifier from the specified project-space.

Usage:

```
CompoundsApi$GetCompoundWithHttpInfo(
  project_id,
  compound_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
```

```
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`compound_id` identifier of the compound (group of ion identities) to access.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`opt_fields_features` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Compound) with additional information such as HTTP status code, headers

Method `GetCompoundQuantTable()`**:** Returns the full quantification table of compounds

Usage:

```
CompoundsApi$GetCompoundQuantTable(
  project_id,
  search_query = NULL,
  type = "APEX_INTENSITY",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`search_query` (optional) Optional query in lucene syntax selecting compounds by id. Omit this parameter to quantify all compounds.

`type` (optional) quantification type. (default value: "APEX_INTENSITY")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: QuantTable

Method `GetCompoundQuantTableWithHttpInfo()`**:** Returns the full quantification table of compounds

Usage:

```
CompoundsApi$GetCompoundQuantTableWithHttpInfo(
  project_id,
  search_query = NULL,
  type = "APEX_INTENSITY",
```

```
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`search_query` (optional) Optional query in lucene syntax selecting compounds by id. Omit this parameter to quantify all compounds.

`type` (optional) quantification type. (default value: "APEX_INTENSITY")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (QuantTable) with additional information such as HTTP status code, headers

Method `GetCompounds()`: [DEPRECATED] List of all available compounds (group of ion identities) in the given project-space

Usage:

```
CompoundsApi$GetCompounds(
  project_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`opt_fields_features` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[Compound]

Method `GetCompoundsWithHttpInfo()`: [DEPRECATED] List of all available compounds (group of ion identities) in the given project-space

Usage:

```
CompoundsApi$GetCompoundsWithHttpInfo(
  project_id,
  ms_data_search_prepared = FALSE,
```

```
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`opt_fields_features` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[Compound]) with additional information such as HTTP status code, headers

Method `GetCompoundsPage()`**:** Page of available compounds (group of ion identities) in the given project-space

Usage:

```
CompoundsApi$GetCompoundsPage(
  project_id,
  search_query = NULL,
  page = 0,
  size = 20,
  sort = NULL,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`search_query` (optional) Optional search query in lucene syntax. Not yet supported for compounds; a non-empty query responds with 405 METHOD_NOT_ALLOWED. Omit this parameter to page over all compounds.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`opt_fields_features` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelCompound

Method `GetCompoundsPageWithHttpInfo()`**:** Page of available compounds (group of ion identities) in the given project-space

Usage:

```
CompoundsApi$GetCompoundsPageWithHttpInfo(
  project_id,
  search_query = NULL,
  page = 0,
  size = 20,
  sort = NULL,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  opt_fields_features = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`search_query` (optional) Optional search query in lucene syntax. Not yet supported for compounds; a non-empty query responds with 405 METHOD_NOT_ALLOWED. Omit this parameter to page over all compounds.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`opt_fields_features` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelCompound) with additional information such as HTTP status code, headers

Method `GetCompoundsSearchableFields()`**:** Get all fields that can be used in the searchQuery parameter of compound endpoints

Usage:

```
CompoundsApi$GetCompoundsSearchableFields(project_id, data_file = NULL, ...)
```

Arguments:

`project_id` project space to read the searchable compound fields from.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableField]

Method `GetCompoundsSearchableFieldsWithHttpInfo()`**:** Get all fields that can be used in the searchQuery parameter of compound endpoints

Usage:

```
CompoundsApi$GetCompoundsSearchableFieldsWithHttpInfo(
  project_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to read the searchable compound fields from.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableField]) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  AddCompounds  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to import into.
var_compound_import <- c(CompoundImport$new(c(FeatureImport$new(123, 123, "name_example", "externalFeatureId_exa
var_profile <- "profile_example" # character | profile describing the instrument used to measure the data. Used to m
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' to override defau
var_opt_fields_features <- c("none") # array[character] | set of optional fields of the nested features to be inclu

#Import compounds and their contained features.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$AddCompounds(var_project_id, var_compound_import, profile = var_profile, opt_fields = var
result <- api_instance$compounds_api$AddCompounds(var_project_id, var_compound_import, profile = var_profile, opt
```

```
dput(result)


####################  DeleteCompound  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to delete from.
var_compound_id <- "compound_id_example" # character | identifier of the compound to delete.

#Delete compound (group of ion identities) with the given identifier (and the included features) from the  specified
api_instance <- rsirius_api$new()

api_instance$compounds_api$DeleteCompound(var_project_id, var_compound_id)


####################  GetCompound  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_compound_id <- "compound_id_example" # character | identifier of the compound (group of ion identities) to acce
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override
var_opt_fields_features <- c("none") # array[character] |  (Optional)

#Get compound (group of ion identities) with the given identifier from the specified project-space.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCompound(var_project_id, var_compound_id, ms_data_search_prepared = var_ms_data_searc
result <- api_instance$compounds_api$GetCompound(var_project_id, var_compound_id, ms_data_search_prepared = var_
dput(result)


####################  GetCompoundQuantTable  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_search_query <- "search_query_example" # character | Optional query in lucene syntax selecting compounds by id.
var_type <- "APEX_INTENSITY" # character | quantification type. (Optional)
var_opt_fields <- c(QuantTableOptField$new()) # array[QuantTableOptField] |  (Optional)

#Returns the full quantification table of compounds
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCompoundQuantTable(var_project_id, search_query = var_search_query, type = var_type, o
result <- api_instance$compounds_api$GetCompoundQuantTable(var_project_id, search_query = var_search_query, type
dput(result)


####################  GetCompounds  ####################

library(RSirius)
```

```
var_project_id <- "project_id_example" # character | project-space to read from.
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override
var_opt_fields_features <- c("none") # array[character] |  (Optional)

#[DEPRECATED] List of all available compounds (group of ion identities) in the given project-space
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCompounds(var_project_id, ms_data_search_prepared = var_ms_data_search_prepared, opt_
result <- api_instance$compounds_api$GetCompounds(var_project_id, ms_data_search_prepared = var_ms_data_search_p
dput(result)


####################  GetCompoundsPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_search_query <- "search_query_example" # character | Optional search query in lucene syntax. Not yet supported
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override
var_opt_fields_features <- c("none") # array[character] |  (Optional)

#Page of available compounds (group of ion identities) in the given project-space
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCompoundsPage(var_project_id, search_query = var_search_query, page = var_page, size =
result <- api_instance$compounds_api$GetCompoundsPage(var_project_id, search_query = var_search_query, page = var
dput(result)


####################  GetCompoundsSearchableFields  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project space to read the searchable compound fields from.

#Get all fields that can be used in the searchQuery parameter of compound endpoints
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCompoundsSearchableFields(var_project_iddata_file = "result.txt")
result <- api_instance$compounds_api$GetCompoundsSearchableFields(var_project_id)
dput(result)



## End(Not run)
```

ComputedSubtools *ComputedSubtools*

Description

Specifies which tools have been executed for the feature. Can be used to estimate which results can be expected. Null if it was not requested and non-null otherwise.

ComputedSubtools Class

Format

An `R6Class` generator object

Details

Create a new ComputedSubtools

Public fields

`librarySearch` character [optional]

`formulaSearch` character [optional]

`zodiac` character [optional]

`fingerprint` character [optional]

`canopus` character [optional]

`structureSearch` character [optional]

`deNovoSearch` character [optional]

Methods

Public methods:

- `ComputedSubtools$new()`
- `ComputedSubtools$toJSON()`
- `ComputedSubtools$toList()`
- `ComputedSubtools$toSimpleType()`
- `ComputedSubtools$fromJSON()`
- `ComputedSubtools$toJSONString()`
- `ComputedSubtools$fromJSONString()`
- `ComputedSubtools$validateJSON()`
- `ComputedSubtools$toString()`
- `ComputedSubtools$isValid()`
- `ComputedSubtools$getInvalidFields()`
- `ComputedSubtools$print()`
- `ComputedSubtools$clone()`

Method `new()`**:** Initialize a new ComputedSubtools class.

Usage:

```
ComputedSubtools$new(
  librarySearch = NULL,
  formulaSearch = NULL,
  zodiac = NULL,
  fingerprint = NULL,
  canopus = NULL,
  structureSearch = NULL,
  deNovoSearch = NULL,
  ...
)
```

Arguments:

`librarySearch` librarySearch

`formulaSearch` formulaSearch

`zodiac` zodiac

`fingerprint` fingerprint

`canopus` canopus

`structureSearch` structureSearch

`deNovoSearch` deNovoSearch

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ComputedSubtools$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ComputedSubtools$toList()
```

Returns: ComputedSubtools as a base R list.

Examples:

```
# convert array of ComputedSubtools (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ComputedSubtools to a base R type

Usage:

```
ComputedSubtools$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ComputedSubtools

Usage:

```
ComputedSubtools$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ComputedSubtools

Method `toJSONString()`**:** To JSON String

Usage:

```
ComputedSubtools$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ComputedSubtools in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ComputedSubtools

Usage:

```
ComputedSubtools$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ComputedSubtools

Method `validateJSON()`**:** Validate JSON input with respect to ComputedSubtools and throw an exception if invalid

Usage:

```
ComputedSubtools$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ComputedSubtools$toString()
```

Returns: String representation of ComputedSubtools

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ComputedSubtools$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ComputedSubtools$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ComputedSubtools$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ComputedSubtools$toList`
## ------------------------------------------------

# convert array of ComputedSubtools (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ConnectionCheck` *ConnectionCheck*

Description

ConnectionCheck Class

ConnectionCheck Class

Format

An `R6Class` generator object

Details

Create a new ConnectionCheck

Public fields

`licenseInfo` LicenseInfo

`errors` List of errors ordered by significance. First error should be reported and addressed first. Following errors might just be follow-up errors list(ConnectionError)

Methods

Public methods:

- `ConnectionCheck$new()`
- `ConnectionCheck$toJSON()`
- `ConnectionCheck$toList()`
- `ConnectionCheck$toSimpleType()`
- `ConnectionCheck$fromJSON()`
- `ConnectionCheck$toJSONString()`
- `ConnectionCheck$fromJSONString()`
- `ConnectionCheck$validateJSON()`
- `ConnectionCheck$toString()`
- `ConnectionCheck$isValid()`
- `ConnectionCheck$getInvalidFields()`
- `ConnectionCheck$print()`
- `ConnectionCheck$clone()`

Method `new()`**:** Initialize a new ConnectionCheck class.

Usage:

```
ConnectionCheck$new(licenseInfo, errors, ...)
```

Arguments:

`licenseInfo` licenseInfo

`errors` List of errors ordered by significance. First error should be reported and addressed first. Following errors might just be follow-up errors

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ConnectionCheck$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ConnectionCheck$toList()
```

Returns: ConnectionCheck as a base R list.

Examples:

```
# convert array of ConnectionCheck (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ConnectionCheck to a base R type

Usage:

```
ConnectionCheck$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ConnectionCheck

Usage:

```
ConnectionCheck$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConnectionCheck

Method `toJSONString()`**:** To JSON String

Usage:

```
ConnectionCheck$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ConnectionCheck in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ConnectionCheck

Usage:

```
ConnectionCheck$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConnectionCheck

Method `validateJSON()`**:** Validate JSON input with respect to ConnectionCheck and throw an exception if invalid

Usage:

```
ConnectionCheck$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ConnectionCheck$toString()
```

Returns: String representation of ConnectionCheck

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ConnectionCheck$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ConnectionCheck$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ConnectionCheck$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ConnectionCheck$toList`
## ------------------------------------------------

# convert array of ConnectionCheck (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

ConnectionError *ConnectionError*

Description

ConnectionError Class

ConnectionError Class

Format

An R6Class generator object

Details

Create a new ConnectionError

Public fields

errorType character

errorKlass character

siriusErrorCode integer

siriusMessage character

serverResponseErrorCode integer [optional]

serverResponseErrorMessage character [optional]

error character [optional]

warning character [optional]

Methods

Public methods:

- ConnectionError$new()
- ConnectionError$toJSON()
- ConnectionError$toList()
- ConnectionError$toSimpleType()
- ConnectionError$fromJSON()
- ConnectionError$toJSONString()
- ConnectionError$fromJSONString()
- ConnectionError$validateJSON()
- ConnectionError$toString()
- ConnectionError$isValid()
- ConnectionError$getInvalidFields()
- ConnectionError$print()
- ConnectionError$clone()

Method `new()`**:** Initialize a new ConnectionError class.

Usage:

```
ConnectionError$new(
  errorType,
  errorKlass,
  siriusErrorCode,
  siriusMessage,
  serverResponseErrorCode = NULL,
  serverResponseErrorMessage = NULL,
  error = NULL,
  warning = NULL,
  ...
)
```

Arguments:

`errorType` errorType

`errorKlass` errorKlass

`siriusErrorCode` siriusErrorCode

`siriusMessage` siriusMessage

`serverResponseErrorCode` serverResponseErrorCode

`serverResponseErrorMessage` serverResponseErrorMessage

`error` error

`warning` warning

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ConnectionError$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ConnectionError$toList()
```

Returns: ConnectionError as a base R list.

Examples:

```
# convert array of ConnectionError (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ConnectionError to a base R type

Usage:

```
ConnectionError$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ConnectionError

Usage:

```
ConnectionError$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConnectionError

Method `toJSONString()`**:** To JSON String

Usage:

```
ConnectionError$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ConnectionError in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ConnectionError

Usage:

```
ConnectionError$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConnectionError

Method `validateJSON()`**:** Validate JSON input with respect to ConnectionError and throw an exception if invalid

Usage:

```
ConnectionError$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ConnectionError$toString()
```

Returns: String representation of ConnectionError

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ConnectionError$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ConnectionError$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ConnectionError$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ConnectionError$toList`
## ------------------------------------------------

# convert array of ConnectionError (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ConsensusAnnotationsCSI` *ConsensusAnnotationsCSI*

Description

ConsensusAnnotationsCSI Class

ConsensusAnnotationsCSI Class

Format

An `R6Class` generator object

Details

Create a new ConsensusAnnotationsCSI

Public fields

`selectionCriterion` Null if this is a custom selection character [optional]

`csiFingerIdStructure` StructureCandidate [optional]

`confidenceExactMatch` Confidence value that represents the certainty that reported consensus structure is exactly the measured one If multiple features support this consensus structure the maximum confidence is reported numeric [optional]

`confidenceApproxMatch` Confidence value that represents the certainty that the exact consensus structure or a very similar structure (e.g. measured by Maximum Common Edge Subgraph Distance) is the measured one. If multiple features support this consensus structure the maximum confidence is reported numeric [optional]

`molecularFormula` Molecular formula of the consensus annotation Might be null if no consensus formula is available. character [optional]

`compoundClasses` CompoundClasses [optional]

`supportingFeatureIds` FeatureIds where the topAnnotation supports this annotation. list(character) [optional]

Methods

Public methods:

- `ConsensusAnnotationsCSI$new()`
- `ConsensusAnnotationsCSI$toJSON()`
- `ConsensusAnnotationsCSI$toList()`
- `ConsensusAnnotationsCSI$toSimpleType()`
- `ConsensusAnnotationsCSI$fromJSON()`
- `ConsensusAnnotationsCSI$toJSONString()`
- `ConsensusAnnotationsCSI$fromJSONString()`
- `ConsensusAnnotationsCSI$validateJSON()`
- `ConsensusAnnotationsCSI$toString()`
- `ConsensusAnnotationsCSI$isValid()`
- `ConsensusAnnotationsCSI$getInvalidFields()`
- `ConsensusAnnotationsCSI$print()`
- `ConsensusAnnotationsCSI$clone()`

Method `new()`**:** Initialize a new ConsensusAnnotationsCSI class.

Usage:

```
ConsensusAnnotationsCSI$new(
  selectionCriterion = NULL,
  csiFingerIdStructure = NULL,
  confidenceExactMatch = NULL,
  confidenceApproxMatch = NULL,
  molecularFormula = NULL,
  compoundClasses = NULL,
  supportingFeatureIds = NULL,
  ...
)
```

Arguments:

`selectionCriterion` Null if this is a custom selection

`csiFingerIdStructure` csiFingerIdStructure

`confidenceExactMatch` Confidence value that represents the certainty that reported consensus structure is exactly the measured one If multiple features support this consensus structure the maximum confidence is reported

`confidenceApproxMatch` Confidence value that represents the certainty that the exact consensus structure or a very similar structure (e.g. measured by Maximum Common Edge Subgraph Distance) is the measured one. If multiple features support this consensus structure the maximum confidence is reported

`molecularFormula` Molecular formula of the consensus annotation Might be null if no consensus formula is available.

`compoundClasses` compoundClasses

`supportingFeatureIds` FeatureIds where the topAnnotation supports this annotation.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ConsensusAnnotationsCSI$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ConsensusAnnotationsCSI$toList()
```

Returns: ConsensusAnnotationsCSI as a base R list.

Examples:

```
# convert array of ConsensusAnnotationsCSI (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ConsensusAnnotationsCSI to a base R type

Usage:

```
ConsensusAnnotationsCSI$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ConsensusAnnotationsCSI

Usage:

```
ConsensusAnnotationsCSI$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConsensusAnnotationsCSI

Method `toJSONString()`**:** To JSON String

Usage:

```
ConsensusAnnotationsCSI$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ConsensusAnnotationsCSI in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ConsensusAnnotationsCSI

Usage:

```
ConsensusAnnotationsCSI$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConsensusAnnotationsCSI

Method `validateJSON()`**:** Validate JSON input with respect to ConsensusAnnotationsCSI and throw an exception if invalid

Usage:

```
ConsensusAnnotationsCSI$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ConsensusAnnotationsCSI$toString()
```

Returns: String representation of ConsensusAnnotationsCSI

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ConsensusAnnotationsCSI$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ConsensusAnnotationsCSI$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ConsensusAnnotationsCSI$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ConsensusAnnotationsCSI$toList`
## ------------------------------------------------

# convert array of ConsensusAnnotationsCSI (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ConsensusAnnotationsDeNovo` *ConsensusAnnotationsDeNovo*

Description

ConsensusAnnotationsDeNovo Class

ConsensusAnnotationsDeNovo Class

Format

An `R6Class` generator object

Details

Create a new ConsensusAnnotationsDeNovo

Public fields

`selectionCriterion` Criterion that was used to select the consensus annotation. character [optional]

`molecularFormula` Molecular formula of the consensus annotation Might be null if no consensus formula is available. character [optional]

`compoundClasses` CompoundClasses [optional]

`supportingFeatureIds` FeatureIds where the topAnnotation supports this annotation. list(character) [optional]

Methods

Public methods:

- ConsensusAnnotationsDeNovo$new()
- ConsensusAnnotationsDeNovo$toJSON()
- ConsensusAnnotationsDeNovo$toList()
- ConsensusAnnotationsDeNovo$toSimpleType()
- ConsensusAnnotationsDeNovo$fromJSON()
- ConsensusAnnotationsDeNovo$toJSONString()
- ConsensusAnnotationsDeNovo$fromJSONString()
- ConsensusAnnotationsDeNovo$validateJSON()
- ConsensusAnnotationsDeNovo$toString()
- ConsensusAnnotationsDeNovo$isValid()
- ConsensusAnnotationsDeNovo$getInvalidFields()
- ConsensusAnnotationsDeNovo$print()
- ConsensusAnnotationsDeNovo$clone()

Method `new()`**:** Initialize a new ConsensusAnnotationsDeNovo class.

Usage:

```
ConsensusAnnotationsDeNovo$new(
  selectionCriterion = NULL,
  molecularFormula = NULL,
  compoundClasses = NULL,
  supportingFeatureIds = NULL,
  ...
)
```

Arguments:

`selectionCriterion` Criterion that was used to select the consensus annotation.

`molecularFormula` Molecular formula of the consensus annotation Might be null if no consensus formula is available.

`compoundClasses` compoundClasses

`supportingFeatureIds` FeatureIds where the topAnnotation supports this annotation.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ConsensusAnnotationsDeNovo$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ConsensusAnnotationsDeNovo$toList()
```

Returns: ConsensusAnnotationsDeNovo as a base R list.

Examples:

```
# convert array of ConsensusAnnotationsDeNovo (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ConsensusAnnotationsDeNovo to a base R type

Usage:

```
ConsensusAnnotationsDeNovo$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ConsensusAnnotationsDeNovo

Usage:

```
ConsensusAnnotationsDeNovo$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConsensusAnnotationsDeNovo

Method `toJSONString()`**:** To JSON String

Usage:

```
ConsensusAnnotationsDeNovo$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ConsensusAnnotationsDeNovo in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ConsensusAnnotationsDeNovo

Usage:

```
ConsensusAnnotationsDeNovo$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ConsensusAnnotationsDeNovo

Method `validateJSON()`**:** Validate JSON input with respect to ConsensusAnnotationsDeNovo and throw an exception if invalid

Usage:

```
ConsensusAnnotationsDeNovo$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ConsensusAnnotationsDeNovo$toString()
```

Returns: String representation of ConsensusAnnotationsDeNovo

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ConsensusAnnotationsDeNovo$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ConsensusAnnotationsDeNovo$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ConsensusAnnotationsDeNovo$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ConsensusAnnotationsDeNovo$toList`
## ------------------------------------------------

# convert array of ConsensusAnnotationsDeNovo (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

DataImportEvent *DataImportEvent*

Description

DataImportEvent Class

DataImportEvent Class

Format

An `R6Class` generator object

Details

Create a new DataImportEvent

Public fields

`importJobId` character [optional]

`importedCompoundIds` list(character)

`importedFeatureIds` list(character)

Methods

Public methods:

- `DataImportEvent$new()`
- `DataImportEvent$toJSON()`
- `DataImportEvent$toList()`
- `DataImportEvent$toSimpleType()`
- `DataImportEvent$fromJSON()`
- `DataImportEvent$toJSONString()`
- `DataImportEvent$fromJSONString()`
- `DataImportEvent$validateJSON()`
- `DataImportEvent$toString()`
- `DataImportEvent$isValid()`
- `DataImportEvent$getInvalidFields()`
- `DataImportEvent$print()`
- `DataImportEvent$clone()`

Method `new()`**:** Initialize a new DataImportEvent class.

Usage:

```
DataImportEvent$new(
  importedCompoundIds,
  importedFeatureIds,
  importJobId = NULL,
  ...
)
```

Arguments:

`importedCompoundIds` importedCompoundIds

`importedFeatureIds` importedFeatureIds

`importJobId` importJobId

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
DataImportEvent$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
DataImportEvent$toList()
```

Returns: DataImportEvent as a base R list.

Examples:

```
# convert array of DataImportEvent (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert DataImportEvent to a base R type

Usage:

```
DataImportEvent$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of DataImportEvent

Usage:

```
DataImportEvent$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of DataImportEvent

Method `toJSONString()`**:** To JSON String

Usage:

```
DataImportEvent$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: DataImportEvent in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of DataImportEvent

Usage:

```
DataImportEvent$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of DataImportEvent

Method `validateJSON()`**:** Validate JSON input with respect to DataImportEvent and throw an exception if invalid

Usage:

```
DataImportEvent$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
DataImportEvent$toString()
```

Returns: String representation of DataImportEvent

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
DataImportEvent$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
DataImportEvent$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
DataImportEvent$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `DataImportEvent$toList`
## ------------------------------------------------

# convert array of DataImportEvent (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

DBLink *DBLink*

Description

DBLink Class

DBLink Class

Format

An R6Class generator object

Details

Create a new DBLink

Public fields

`name` character

`id` character [optional]

Methods

Public methods:

- DBLink$new()
- DBLink$toJSON()
- DBLink$toList()
- DBLink$toSimpleType()
- DBLink$fromJSON()
- DBLink$toJSONString()
- DBLink$fromJSONString()
- DBLink$validateJSON()

- DBLink$toString()
- DBLink$isValid()
- DBLink$getInvalidFields()
- DBLink$print()
- DBLink$clone()

Method `new()`**:** Initialize a new DBLink class.

Usage:

```
DBLink$new(name, id = NULL, ...)
```

Arguments:

`name` name

`id` id

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
DBLink$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
DBLink$toList()
```

Returns: DBLink as a base R list.

Examples:

```
# convert array of DBLink (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert DBLink to a base R type

Usage:

```
DBLink$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of DBLink

Usage:

```
DBLink$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of DBLink

Method `toJSONString()`**:** To JSON String

Usage:

```
DBLink$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: DBLink in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of DBLink

Usage:

```
DBLink$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of DBLink

Method `validateJSON()`**:** Validate JSON input with respect to DBLink and throw an exception if invalid

Usage:

```
DBLink$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
DBLink$toString()
```

Returns: String representation of DBLink

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
DBLink$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
DBLink$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
DBLink$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `DBLink$toList`
## ------------------------------------------------

# convert array of DBLink (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Deviation` *Deviation*

Description

Deviation Class

Deviation Class

Format

An `R6Class` generator object

Details

Create a new Deviation

Public fields

`ppm` numeric [optional]

`absolute` numeric [optional]

Methods

Public methods:

- `Deviation$new()`
- `Deviation$toJSON()`
- `Deviation$toList()`
- `Deviation$toSimpleType()`
- `Deviation$fromJSON()`
- `Deviation$toJSONString()`
- `Deviation$fromJSONString()`
- `Deviation$validateJSON()`

- Deviation$toString()
- Deviation$isValid()
- Deviation$getInvalidFields()
- Deviation$print()
- Deviation$clone()

Method `new()`**:** Initialize a new Deviation class.

Usage:

```
Deviation$new(ppm = NULL, absolute = NULL, ...)
```

Arguments:

`ppm` ppm

`absolute` absolute

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Deviation$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Deviation$toList()
```

Returns: Deviation as a base R list.

Examples:

```
# convert array of Deviation (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Deviation to a base R type

Usage:

```
Deviation$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Deviation

Usage:

```
Deviation$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Deviation

Method `toJSONString()`**:** To JSON String

Usage:

```
Deviation$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Deviation in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Deviation

Usage:

```
Deviation$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Deviation

Method `validateJSON()`**:** Validate JSON input with respect to Deviation and throw an exception if invalid

Usage:

```
Deviation$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Deviation$toString()
```

Returns: String representation of Deviation

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Deviation$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Deviation$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Deviation$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Deviation$toList`
## ------------------------------------------------

# convert array of Deviation (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Feature` *Feature*

Description

A feature as it has been detected in one single run, in contrast to an AlignedFeature which combines the features of the same compound over all runs it was detected in. <p> It provides the properties that are specific to the run it was detected in, such as its position on the retention time axis of that run and its quantity within that run.

Feature Class

Format

An `R6Class` generator object

Details

Create a new Feature

Public fields

`featureId` Identifier character [optional]

`alignedFeatureId` ID of the AlignedFeature this feature belongs to character [optional]

`runId` ID of the run this feature belongs to character [optional]

`averageMz` Average m/z over the whole feature numeric [optional]

`apexMz` m/z at the apex of the feature, the m/z this feature was measured at in its run numeric [optional]

`rtStartSeconds` Start of the feature on the retention time axis in seconds numeric [optional]

`rtEndSeconds` End of the feature on the retention time axis in seconds numeric [optional]

`rtApexSeconds` Apex of the feature on the retention time axis in seconds numeric [optional]

`rtFwhmSeconds` Full width at half maximum of the feature on the retention time axis in seconds numeric [optional]

`apexIntensity` Feature quantity measured as the intensity of the apex of the feature numeric [optional]

`areaUnderCurve` Feature quantity measured as the area under the curve of the whole feature numeric [optional]

Methods

Public methods:

- `Feature$new()`
- `Feature$toJSON()`
- `Feature$toList()`
- `Feature$toSimpleType()`
- `Feature$fromJSON()`
- `Feature$toJSONString()`
- `Feature$fromJSONString()`
- `Feature$validateJSON()`
- `Feature$toString()`
- `Feature$isValid()`
- `Feature$getInvalidFields()`
- `Feature$print()`
- `Feature$clone()`

Method `new()`**:** Initialize a new Feature class.

Usage:

```
Feature$new(
  featureId = NULL,
  alignedFeatureId = NULL,
  runId = NULL,
  averageMz = NULL,
  apexMz = NULL,
  rtStartSeconds = NULL,
  rtEndSeconds = NULL,
  rtApexSeconds = NULL,
  rtFwhmSeconds = NULL,
  apexIntensity = NULL,
  areaUnderCurve = NULL,
  ...
)
```

Arguments:

`featureId` Identifier

`alignedFeatureId` ID of the AlignedFeature this feature belongs to

`runId` ID of the run this feature belongs to

`averageMz` Average m/z over the whole feature

`apexMz` m/z at the apex of the feature, the m/z this feature was measured at in its run

`rtStartSeconds` Start of the feature on the retention time axis in seconds

`rtEndSeconds` End of the feature on the retention time axis in seconds

`rtApexSeconds` Apex of the feature on the retention time axis in seconds

`rtFwhmSeconds` Full width at half maximum of the feature on the retention time axis in seconds

`apexIntensity` Feature quantity measured as the intensity of the apex of the feature

`areaUnderCurve` Feature quantity measured as the area under the curve of the whole feature

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Feature$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Feature$toList()
```

Returns: Feature as a base R list.

Examples:

```
# convert array of Feature (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Feature to a base R type

Usage:

```
Feature$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Feature

Usage:

```
Feature$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Feature

Method `toJSONString()`**:** To JSON String

Usage:

```
Feature$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Feature in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Feature

Usage:

```
Feature$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Feature

Method `validateJSON()`**:** Validate JSON input with respect to Feature and throw an exception if invalid

Usage:

```
Feature$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Feature$toString()
```

Returns: String representation of Feature

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Feature$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Feature$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Feature$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Feature$toList`
## ------------------------------------------------

# convert array of Feature (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`FeatureAnnotations` *FeatureAnnotations*

Description

Summary of the results of a feature (aligned over runs). Can be added to a AlignedFeature. The different annotation fields within this summary object are null if the corresponding feature does not contain the represented results. If fields are non-null the corresponding result has been computed but might still be empty.

FeatureAnnotations Class

Format

An `R6Class` generator object

Details

Create a new FeatureAnnotations

Public fields

`formulaAnnotation` FormulaCandidate [optional]

`structureAnnotation` StructureCandidateScored [optional]

`compoundClassAnnotation` CompoundClasses [optional]

`confidenceExactMatch` Confidence Score that represents the confidence whether the top hit is correct. numeric [optional]

`confidenceApproxMatch` Confidence Score that represents the confidence whether the top hit or a very similar hit (estimated by MCES distance) is correct. numeric [optional]

`expansiveSearchState` Result that shows if structure annotation was expanded by using PubChem as fallback and if so, which confidence mode was used (as per input parameter) character [optional]

`specifiedDatabases` List of databases that have been specified for structure db search. Null if no structure db search has been performed. list(character) [optional]

`expandedDatabases` List of databases that have been used to expand search space during expansive search. Null if no structure db search has been performed. list(character) [optional]

Methods

Public methods:

- `FeatureAnnotations$new()`
- `FeatureAnnotations$toJSON()`
- `FeatureAnnotations$toList()`
- `FeatureAnnotations$toSimpleType()`
- `FeatureAnnotations$fromJSON()`
- `FeatureAnnotations$toJSONString()`
- `FeatureAnnotations$fromJSONString()`
- `FeatureAnnotations$validateJSON()`
- `FeatureAnnotations$toString()`
- `FeatureAnnotations$isValid()`
- `FeatureAnnotations$getInvalidFields()`
- `FeatureAnnotations$print()`
- `FeatureAnnotations$clone()`

Method `new()`**:** Initialize a new FeatureAnnotations class.

Usage:

```
FeatureAnnotations$new(
  formulaAnnotation = NULL,
  structureAnnotation = NULL,
  compoundClassAnnotation = NULL,
  confidenceExactMatch = NULL,
  confidenceApproxMatch = NULL,
  expansiveSearchState = NULL,
  specifiedDatabases = NULL,
  expandedDatabases = NULL,
  ...
)
```

Arguments:

`formulaAnnotation` formulaAnnotation

`structureAnnotation` structureAnnotation

`compoundClassAnnotation` compoundClassAnnotation

`confidenceExactMatch` Confidence Score that represents the confidence whether the top hit is correct.

`confidenceApproxMatch` Confidence Score that represents the confidence whether the top hit or a very similar hit (estimated by MCES distance) is correct.

`expansiveSearchState` Result that shows if structure annotation was expanded by using PubChem as fallback and if so, which confidence mode was used (as per input parameter)

`specifiedDatabases` List of databases that have been specified for structure db search. Null if no structure db search has been performed.

`expandedDatabases` List of databases that have been used to expand search space during expansive search. Null if no structure db search has been performed.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
FeatureAnnotations$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
FeatureAnnotations$toList()
```

Returns: FeatureAnnotations as a base R list.

Examples:

```
# convert array of FeatureAnnotations (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert FeatureAnnotations to a base R type

Usage:

```
FeatureAnnotations$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of FeatureAnnotations

Usage:

```
FeatureAnnotations$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FeatureAnnotations

Method `toJSONString()`**:** To JSON String

Usage:

```
FeatureAnnotations$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: FeatureAnnotations in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of FeatureAnnotations

Usage:

```
FeatureAnnotations$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FeatureAnnotations

Method `validateJSON()`**:** Validate JSON input with respect to FeatureAnnotations and throw an exception if invalid

Usage:

```
FeatureAnnotations$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
FeatureAnnotations$toString()
```

Returns: String representation of FeatureAnnotations

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
FeatureAnnotations$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
FeatureAnnotations$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
FeatureAnnotations$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `FeatureAnnotations$toList`
## ------------------------------------------------

# convert array of FeatureAnnotations (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`FeatureImport` *FeatureImport*

Description

Represents an (aligned) feature to be imported into a SIRIUS project. At least one of the Mass Spec data sources (e.g. mergedMs1, ms1Spectra, ms2Spectra) needs to be given. Otherwise, the import will fail.

FeatureImport Class

Format

An `R6Class` generator object

Details

Create a new FeatureImport

Public fields

`name` character [optional]

`externalFeatureId` Externally provided FeatureId (by some preprocessing tool). This FeatureId is NOT used by SIRIUS but is stored to ease mapping information back to the source. character [optional]

`ionMass` numeric

`charge` integer

`detectedAdducts` Detected adducts of this feature. Can be NULL or empty if no adducts are known. list(character) [optional]

`rtStartSeconds` numeric [optional]

`rtEndSeconds` numeric [optional]

`rtApexSeconds` numeric [optional]

`dataQuality` A optional feature quality flag that can be used to filter features to be shown in the gui or to be considered for further analysis. character [optional]

`mergedMs1` BasicSpectrum [optional]

`ms1Spectra` List of MS1Spectra belonging to this feature. These spectra will be merged and only a representative mergedMs1 spectrum will be stored in SIRIUS. At least one of these spectra should contain the isotope pattern of the precursor ion. Note: Will be ignored if 'mergedMs1' is given. list(BasicSpectrum) [optional]

`ms2Spectra` List of MS/MS spectra that belong to this feature. list(BasicSpectrum) [optional]

Methods

Public methods:

- `FeatureImport$new()`
- `FeatureImport$toJSON()`
- `FeatureImport$toList()`
- `FeatureImport$toSimpleType()`
- `FeatureImport$fromJSON()`
- `FeatureImport$toJSONString()`
- `FeatureImport$fromJSONString()`
- `FeatureImport$validateJSON()`
- `FeatureImport$toString()`
- `FeatureImport$isValid()`
- `FeatureImport$getInvalidFields()`
- `FeatureImport$print()`
- `FeatureImport$clone()`

Method `new()`**:** Initialize a new FeatureImport class.

Usage:

```
FeatureImport$new(
  ionMass,
  charge,
  name = NULL,
  externalFeatureId = NULL,
  detectedAdducts = NULL,
  rtStartSeconds = NULL,
  rtEndSeconds = NULL,
  rtApexSeconds = NULL,
  dataQuality = NULL,
  mergedMs1 = NULL,
  ms1Spectra = NULL,
  ms2Spectra = NULL,
  ...
)
```

Arguments:

`ionMass` ionMass

`charge` charge

`name` name

`externalFeatureId` Externally provided FeatureId (by some preprocessing tool). This FeatureId is NOT used by SIRIUS but is stored to ease mapping information back to the source.

`detectedAdducts` Detected adducts of this feature. Can be NULL or empty if no adducts are known.

`rtStartSeconds` rtStartSeconds

`rtEndSeconds` rtEndSeconds

`rtApexSeconds` rtApexSeconds

`dataQuality` A optional feature quality flag that can be used to filter features to be shown in the gui or to be considered for further analysis.

`mergedMs1` mergedMs1

`ms1Spectra` List of MS1Spectra belonging to this feature. These spectra will be merged and only a representative mergedMs1 spectrum will be stored in SIRIUS. At least one of these spectra should contain the isotope pattern of the precursor ion. Note: Will be ignored if 'mergedMs1' is given.

`ms2Spectra` List of MS/MS spectra that belong to this feature.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
FeatureImport$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
FeatureImport$toList()
```

Returns: FeatureImport as a base R list.

Examples:

```
# convert array of FeatureImport (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert FeatureImport to a base R type

Usage:

```
FeatureImport$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of FeatureImport

Usage:

```
FeatureImport$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FeatureImport

Method `toJSONString()`**:** To JSON String

Usage:

```
FeatureImport$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: FeatureImport in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of FeatureImport

Usage:

```
FeatureImport$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FeatureImport

Method `validateJSON()`**:** Validate JSON input with respect to FeatureImport and throw an exception if invalid

Usage:

```
FeatureImport$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
FeatureImport$toString()
```

Returns: String representation of FeatureImport

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
FeatureImport$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
FeatureImport$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
FeatureImport$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `FeatureImport$toList`
## ------------------------------------------------

# convert array of FeatureImport (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`FeaturesApi` *Features operations*

Description

FeaturesApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- FeaturesApi$new()
- FeaturesApi$AddAlignedFeatures()
- FeaturesApi$AddAlignedFeaturesWithHttpInfo()
- FeaturesApi$DeleteAlignedFeature()
- FeaturesApi$DeleteAlignedFeatureWithHttpInfo()
- FeaturesApi$DeleteAlignedFeatures()
- FeaturesApi$DeleteAlignedFeaturesWithHttpInfo()
- FeaturesApi$DeleteAlignedFeaturesByQuery()
- FeaturesApi$DeleteAlignedFeaturesByQueryWithHttpInfo()
- FeaturesApi$GetAlignedFeature()
- FeaturesApi$GetAlignedFeatureWithHttpInfo()
- FeaturesApi$GetAlignedFeatures()
- FeaturesApi$GetAlignedFeaturesWithHttpInfo()
- FeaturesApi$GetAlignedFeaturesPage()
- FeaturesApi$GetAlignedFeaturesPageWithHttpInfo()
- FeaturesApi$GetAlignedFeaturesSearchableFields()
- FeaturesApi$GetAlignedFeaturesSearchableFieldsWithHttpInfo()
- FeaturesApi$GetBestMatchingCompoundClasses()
- FeaturesApi$GetBestMatchingCompoundClassesWithHttpInfo()
- FeaturesApi$GetCanopusPrediction()
- FeaturesApi$GetCanopusPredictionWithHttpInfo()
- FeaturesApi$GetDeNovoStructureCandidates()
- FeaturesApi$GetDeNovoStructureCandidatesWithHttpInfo()
- FeaturesApi$GetDeNovoStructureCandidatesByFormula()
- FeaturesApi$GetDeNovoStructureCandidatesByFormulaWithHttpInfo()
- FeaturesApi$GetDeNovoStructureCandidatesByFormulaPage()
- FeaturesApi$GetDeNovoStructureCandidatesByFormulaPageWithHttpInfo()
- FeaturesApi$GetDeNovoStructureCandidatesPage()
- FeaturesApi$GetDeNovoStructureCandidatesPageWithHttpInfo()
- FeaturesApi$GetFeatureQuantTable()
- FeaturesApi$GetFeatureQuantTableWithHttpInfo()
- FeaturesApi$GetFeatures()
- FeaturesApi$GetFeaturesWithHttpInfo()
- FeaturesApi$GetFeaturesPage()
- FeaturesApi$GetFeaturesPageWithHttpInfo()
- FeaturesApi$GetFingerprintPrediction()
- FeaturesApi$GetFingerprintPredictionWithHttpInfo()
- FeaturesApi$GetFormulaAnnotatedMsMsData()
- FeaturesApi$GetFormulaAnnotatedMsMsDataWithHttpInfo()
- FeaturesApi$GetFormulaAnnotatedSpectrum()

- FeaturesApi$GetFormulaAnnotatedSpectrumWithHttpInfo()
- FeaturesApi$GetFormulaCandidate()
- FeaturesApi$GetFormulaCandidateWithHttpInfo()
- FeaturesApi$GetFormulaCandidates()
- FeaturesApi$GetFormulaCandidatesWithHttpInfo()
- FeaturesApi$GetFormulaCandidatesPage()
- FeaturesApi$GetFormulaCandidatesPageWithHttpInfo()
- FeaturesApi$GetFragTree()
- FeaturesApi$GetFragTreeWithHttpInfo()
- FeaturesApi$GetIsotopePatternAnnotation()
- FeaturesApi$GetIsotopePatternAnnotationWithHttpInfo()
- FeaturesApi$GetLipidAnnotation()
- FeaturesApi$GetLipidAnnotationWithHttpInfo()
- FeaturesApi$GetMsData()
- FeaturesApi$GetMsDataWithHttpInfo()
- FeaturesApi$GetSpectralLibraryMatch()
- FeaturesApi$GetSpectralLibraryMatchWithHttpInfo()
- FeaturesApi$GetSpectralLibraryMatches()
- FeaturesApi$GetSpectralLibraryMatchesWithHttpInfo()
- FeaturesApi$GetSpectralLibraryMatchesPage()
- FeaturesApi$GetSpectralLibraryMatchesPageWithHttpInfo()
- FeaturesApi$GetSpectralLibraryMatchesSummary()
- FeaturesApi$GetSpectralLibraryMatchesSummaryWithHttpInfo()
- FeaturesApi$GetStructureCandidates()
- FeaturesApi$GetStructureCandidatesWithHttpInfo()
- FeaturesApi$GetStructureCandidatesByFormula()
- FeaturesApi$GetStructureCandidatesByFormulaWithHttpInfo()
- FeaturesApi$GetStructureCandidatesByFormulaPage()
- FeaturesApi$GetStructureCandidatesByFormulaPageWithHttpInfo()
- FeaturesApi$GetStructureCandidatesPage()
- FeaturesApi$GetStructureCandidatesPageWithHttpInfo()
- FeaturesApi$clone()

Method `new()`**:** Initialize a new FeaturesApi.

Usage:

```
FeaturesApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `AddAlignedFeatures()`**:** Import (aligned) features into the project.

Usage:

```
FeaturesApi$AddAlignedFeatures(
  project_id,
  feature_import,
  profile = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`feature_import` the feature data to be imported

`profile` (optional) profile describing the instrument used to measure the data. Used to merge spectra.

`opt_fields` (optional) set of optional fields to be included. Use 'none' to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[AlignedFeature]

Method `AddAlignedFeaturesWithHttpInfo()`**:** Import (aligned) features into the project.

Usage:

```
FeaturesApi$AddAlignedFeaturesWithHttpInfo(
  project_id,
  feature_import,
  profile = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`feature_import` the feature data to be imported

`profile` (optional) profile describing the instrument used to measure the data. Used to merge spectra.

`opt_fields` (optional) set of optional fields to be included. Use 'none' to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[AlignedFeature]) with additional information such as HTTP status code, headers

Method `DeleteAlignedFeature()`**:** Delete feature (aligned over runs) with the given identifier from the specified project-space.

Usage:

```
FeaturesApi$DeleteAlignedFeature(project_id, aligned_feature_id, ...)
```

Arguments:

`project_id` project-space to delete from.

`aligned_feature_id` identifier of feature (aligned over runs) to delete.

`...` Other optional arguments

Returns: void

Method `DeleteAlignedFeatureWithHttpInfo()`**:** Delete feature (aligned over runs) with the given identifier from the specified project-space.

Usage:

```
FeaturesApi$DeleteAlignedFeatureWithHttpInfo(
  project_id,
  aligned_feature_id,
  ...
)
```

Arguments:

`project_id` project-space to delete from.

`aligned_feature_id` identifier of feature (aligned over runs) to delete.

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `DeleteAlignedFeatures()`**:** Delete feature (aligned over runs) with the given identifier from the specified project-space.

Usage:

```
FeaturesApi$DeleteAlignedFeatures(project_id, request_body, ...)
```

Arguments:

`project_id` project-space to delete from.

`request_body`

`...` Other optional arguments

Returns: void

Method `DeleteAlignedFeaturesWithHttpInfo()`**:** Delete feature (aligned over runs) with the given identifier from the specified project-space.

Usage:

```
FeaturesApi$DeleteAlignedFeaturesWithHttpInfo(project_id, request_body, ...)
```

Arguments:

`project_id` project-space to delete from.

`request_body`

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `DeleteAlignedFeaturesByQuery()`**:** Delete all features (aligned over runs) that match the given lucene search query from the specified project-space.

Usage:

```
FeaturesApi$DeleteAlignedFeaturesByQuery(project_id, search_query, ...)
```

Arguments:

`project_id` project-space to delete from.

`search_query` tag/text/range query in lucene syntax; must be non-empty (a blank query would match every feature).

`...` Other optional arguments

Returns: void

Method `DeleteAlignedFeaturesByQueryWithHttpInfo()`**:** Delete all features (aligned over runs) that match the given lucene search query from the specified project-space.

Usage:

```
FeaturesApi$DeleteAlignedFeaturesByQueryWithHttpInfo(
  project_id,
  search_query,
  ...
)
```

Arguments:

`project_id` project-space to delete from.

`search_query` tag/text/range query in lucene syntax; must be non-empty (a blank query would match every feature).

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `GetAlignedFeature()`**:** Get feature (aligned over runs) with the given identifier from the specified project-space.

Usage:

```
FeaturesApi$GetAlignedFeature(
  project_id,
  aligned_feature_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` identifier of feature (aligned over runs) to access.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: AlignedFeature

Method `GetAlignedFeatureWithHttpInfo()`**:** Get feature (aligned over runs) with the given identifier from the specified project-space.

Usage:

```
FeaturesApi$GetAlignedFeatureWithHttpInfo(
  project_id,
  aligned_feature_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` identifier of feature (aligned over runs) to access.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (AlignedFeature) with additional information such as HTTP status code, headers

Method `GetAlignedFeatures()`**:** [DEPRECATED] Get all available features (aligned over runs) in the given project-space

Usage:

```
FeaturesApi$GetAlignedFeatures(
  project_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[AlignedFeature]

Method `GetAlignedFeaturesWithHttpInfo()`: [DEPRECATED] Get all available features (aligned over runs) in the given project-space

Usage:

```
FeaturesApi$GetAlignedFeaturesWithHttpInfo(
  project_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[AlignedFeature]) with additional information such as HTTP status code, headers

Method `GetAlignedFeaturesPage()`: Get features (aligned over runs) in the given project-space

Usage:

```
FeaturesApi$GetAlignedFeaturesPage(
  project_id,
  page = 0,
  size = 20,
  sort = NULL,
  search_query = NULL,
  ms_data_search_prepared = FALSE,
  opt_fields = list("computedTools", "qualities", "tags"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to get features (aligned over runs) from.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`search_query` (optional) Optional search query in lucene syntax. Omit this parameter to page over all features.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["computedTools","qualities","tags"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelAlignedFeature

Method `GetAlignedFeaturesPageWithHttpInfo()`**:** Get features (aligned over runs) in the given project-space

Usage:

```
FeaturesApi$GetAlignedFeaturesPageWithHttpInfo(
  project_id,
  page = 0,
  size = 20,
  sort = NULL,
  search_query = NULL,
  ms_data_search_prepared = FALSE,
  opt_fields = list("computedTools", "qualities", "tags"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to get features (aligned over runs) from.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`search_query` (optional) Optional search query in lucene syntax. Omit this parameter to page over all features.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["computedTools","qualities","tags"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelAlignedFeature) with additional information such as HTTP status code, headers

Method `GetAlignedFeaturesSearchableFields()`**:** Get all fields that can be used in the searchQuery parameter of feature (aligned over runs) endpoints

Usage:

```
FeaturesApi$GetAlignedFeaturesSearchableFields(
  project_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to read the searchable feature fields from.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableField]

Method `GetAlignedFeaturesSearchableFieldsWithHttpInfo()`**:** Get all fields that can be used in the searchQuery parameter of feature (aligned over runs) endpoints

Usage:

```
FeaturesApi$GetAlignedFeaturesSearchableFieldsWithHttpInfo(
  project_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to read the searchable feature fields from.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableField]) with additional information such as HTTP status code, headers

Method `GetBestMatchingCompoundClasses()`**:** Return Best matching compound classes for given formulaId

Usage:

```
FeaturesApi$GetBestMatchingCompoundClasses(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: CompoundClasses

Method `GetBestMatchingCompoundClassesWithHttpInfo()`: Return Best matching compound classes for given formulaId

Usage:

```
FeaturesApi$GetBestMatchingCompoundClassesWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (CompoundClasses) with additional information such as HTTP status code, headers

Method `GetCanopusPrediction()`: All predicted compound classes (CANOPUS) from ClassyFire and NPC and their probabilities,

Usage:

```
FeaturesApi$GetCanopusPrediction(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: CanopusPrediction

Method `GetCanopusPredictionWithHttpInfo()`: All predicted compound classes (CANOPUS) from ClassyFire and NPC and their probabilities,

Usage:

```
FeaturesApi$GetCanopusPredictionWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
```

```
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (CanopusPrediction) with additional information such as HTTP status code, headers

Method `GetDeNovoStructureCandidates()`**:** [DEPRECATED] List of de novo structure candidates (e

Usage:

```
FeaturesApi$GetDeNovoStructureCandidates(
  project_id,
  aligned_feature_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[StructureCandidateFormula]

Method `GetDeNovoStructureCandidatesWithHttpInfo()`**:** [DEPRECATED] List of de novo structure candidates (e

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesWithHttpInfo(
  project_id,
  aligned_feature_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[StructureCandidateFormula]) with additional information such as HTTP status code, headers

Method `GetDeNovoStructureCandidatesByFormula()`: [DEPRECATED] List of de novo structure candidates (e

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesByFormula(
  project_id,
  aligned_feature_id,
  formula_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[StructureCandidateScored]

Method `GetDeNovoStructureCandidatesByFormulaWithHttpInfo()`: [DEPRECATED] List of de novo structure candidates (e

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesByFormulaWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[StructureCandidateScored]) with additional information such as HTTP status code, headers

Method `GetDeNovoStructureCandidatesByFormulaPage()`**:** Page of de novo structure candidates (e.g. generated by MsNovelist) ranked by CSI:FingerID score for the given 'formulaId' with minimal information. StructureCandidates can be enriched with molecular fingerprint.

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesByFormulaPage(
  project_id,
  aligned_feature_id,
  formula_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelStructureCandidateScored

Method `GetDeNovoStructureCandidatesByFormulaPageWithHttpInfo()`**:** Page of de novo structure candidates (e.g. generated by MsNovelist) ranked by CSI:FingerID score for the given 'formulaId' with minimal information. StructureCandidates can be enriched with molecular fingerprint.

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesByFormulaPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  page = 0,
  size = 20,
```

```
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelStructureCandidateScored) with additional information such as HTTP status code, headers

Method `GetDeNovoStructureCandidatesPage()`**:** Page of de novo structure candidates (e.g. generated by MsNovelist) ranked by CSI:FingerID score for the given 'alignedFeatureId' with minimal information. StructureCandidates can be enriched with molecular fingerprint.

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesPage(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

... Other optional arguments

Returns: PagedModelStructureCandidateFormula

Method `GetDeNovoStructureCandidatesPageWithHttpInfo()`: Page of de novo structure candidates (e.g. generated by MsNovelist) ranked by CSI:FingerID score for the given 'alignedFeatureId' with minimal information. StructureCandidates can be enriched with molecular fingerprint.

Usage:

```
FeaturesApi$GetDeNovoStructureCandidatesPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

... Other optional arguments

Returns: API response (PagedModelStructureCandidateFormula) with additional information such as HTTP status code, headers

Method `GetFeatureQuantTable()`: Returns the full quantification table of features

Usage:

```
FeaturesApi$GetFeatureQuantTable(
  project_id,
  search_query = NULL,
  type = "APEX_INTENSITY",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`search_query` (optional) Optional search query in lucene syntax. Omit this parameter to quantify all features.

`type` (optional) quantification type. (default value: "APEX_INTENSITY")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: QuantTable

Method `GetFeatureQuantTableWithHttpInfo()`**:** Returns the full quantification table of features

Usage:

```
FeaturesApi$GetFeatureQuantTableWithHttpInfo(
  project_id,
  search_query = NULL,
  type = "APEX_INTENSITY",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`search_query` (optional) Optional search query in lucene syntax. Omit this parameter to quantify all features.

`type` (optional) quantification type. (default value: "APEX_INTENSITY")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (QuantTable) with additional information such as HTTP status code, headers

Method `GetFeatures()`**:** [DEPRECATED] Get all features the given feature (aligned over runs) was aligned from

Usage:

```
FeaturesApi$GetFeatures(project_id, aligned_feature_id, data_file = NULL, ...)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) whose features to read.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[Feature]

Method `GetFeaturesWithHttpInfo()`**:** [DEPRECATED] Get all features the given feature (aligned over runs) was aligned from

Usage:

```
FeaturesApi$GetFeaturesWithHttpInfo(
  project_id,
  aligned_feature_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) whose features to read.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[Feature]) with additional information such as HTTP status code, headers

Method `GetFeaturesPage()`**:** Page of features the given feature (aligned over runs) was aligned from, one per run it has been detected in

Usage:

```
FeaturesApi$GetFeaturesPage(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) whose features to read.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelFeature

Method `GetFeaturesPageWithHttpInfo()`**:** Page of features the given feature (aligned over runs) was aligned from, one per run it has been detected in

Usage:

```
FeaturesApi$GetFeaturesPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
```

```
  sort = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) whose features to read.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelFeature) with additional information such as HTTP status code, headers

Method `GetFingerprintPrediction()`**:** Returns predicted fingerprint (CSI:FingerID) for the given formula result identifier (formulaId)

Usage:

```
FeaturesApi$GetFingerprintPrediction(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[numeric]

Method `GetFingerprintPredictionWithHttpInfo()`**:** Returns predicted fingerprint (CSI:FingerID) for the given formula result identifier (formulaId)

Usage:

```
FeaturesApi$GetFingerprintPredictionWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[numeric]) with additional information such as HTTP status code, headers

Method `GetFormulaAnnotatedMsMsData()`**:** Returns MS/MS Spectrum annotated with fragments and losses for provided formulaId

Usage:

```
FeaturesApi$GetFormulaAnnotatedMsMsData(
  project_id,
  aligned_feature_id,
  formula_id,
  ms_data_search_prepared = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: AnnotatedMsMsData

Method `GetFormulaAnnotatedMsMsDataWithHttpInfo()`**:** Returns MS/MS Spectrum annotated with fragments and losses for provided formulaId

Usage:

```
FeaturesApi$GetFormulaAnnotatedMsMsDataWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  ms_data_search_prepared = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (AnnotatedMsMsData) with additional information such as HTTP status code, headers

Method `GetFormulaAnnotatedSpectrum()`**:** Returns a fragmentation spectrum (e

Usage:

```
FeaturesApi$GetFormulaAnnotatedSpectrum(
  project_id,
  aligned_feature_id,
  formula_id,
  spectrum_index = -1,
  search_prepared = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`spectrum_index` (optional) index of the spectrum to be annotated. Merged MS/MS will be used if spectrumIndex < 0 (default) (default value: -1)

`search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: AnnotatedSpectrum

Method `GetFormulaAnnotatedSpectrumWithHttpInfo()`**:** Returns a fragmentation spectrum (e

Usage:

```
FeaturesApi$GetFormulaAnnotatedSpectrumWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  spectrum_index = -1,
  search_prepared = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`spectrum_index` (optional) index of the spectrum to be annotated. Merged MS/MS will be used if spectrumIndex < 0 (default) (default value: -1)

`search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (AnnotatedSpectrum) with additional information such as HTTP status code, headers

Method `GetFormulaCandidate()`**:** FormulaResultContainers for the given 'formulaId' with minimal information.

Usage:

```
FeaturesApi$GetFormulaCandidate(
  project_id,
  aligned_feature_id,
  formula_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: FormulaCandidate

Method `GetFormulaCandidateWithHttpInfo()`**:** FormulaResultContainers for the given 'formulaId' with minimal information.

Usage:

```
FeaturesApi$GetFormulaCandidateWithHttpInfo(
  project_id,
```

```
  aligned_feature_id,
  formula_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (FormulaCandidate) with additional information such as HTTP status code, headers

Method `GetFormulaCandidates()`**:** [DEPRECATED] List of FormulaResultContainers available for this feature with minimal information

Usage:

```
FeaturesApi$GetFormulaCandidates(
  project_id,
  aligned_feature_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[FormulaCandidate]

Method `GetFormulaCandidatesWithHttpInfo()`**:** [DEPRECATED] List of FormulaResultContainers available for this feature with minimal information

Usage:

```
FeaturesApi$GetFormulaCandidatesWithHttpInfo(
  project_id,
  aligned_feature_id,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[FormulaCandidate]) with additional information such as HTTP status code, headers

Method `GetFormulaCandidatesPage()`**:** Page of FormulaResultContainers available for this feature with minimal information.

Usage:

```
FeaturesApi$GetFormulaCandidatesPage(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelFormulaCandidate

Method `GetFormulaCandidatesPageWithHttpInfo()`**:** Page of FormulaResultContainers available for this feature with minimal information.

Usage:

```
FeaturesApi$GetFormulaCandidatesPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  ms_data_search_prepared = FALSE,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelFormulaCandidate) with additional information such as HTTP status code, headers

Method `GetFragTree()`**:** Returns fragmentation tree (SIRIUS) for the given formula result identifier

Usage:

```
FeaturesApi$GetFragTree(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: FragmentationTree

Method `GetFragTreeWithHttpInfo()`**:** Returns fragmentation tree (SIRIUS) for the given formula result identifier

Usage:

```
FeaturesApi$GetFragTreeWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (FragmentationTree) with additional information such as HTTP status code, headers

Method `GetIsotopePatternAnnotation()`**:** Returns Isotope pattern information for given formulaId

Usage:

```
FeaturesApi$GetIsotopePatternAnnotation(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: IsotopePatternAnnotation

Method `GetIsotopePatternAnnotationWithHttpInfo()`**:** Returns Isotope pattern information for given formulaId

Usage:

```
FeaturesApi$GetIsotopePatternAnnotationWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (IsotopePatternAnnotation) with additional information such as HTTP status code, headers

Method `GetLipidAnnotation()`**:** Returns Lipid annotation (ElGordo) for the given formulaId

Usage:

```
FeaturesApi$GetLipidAnnotation(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: LipidAnnotation

Method `GetLipidAnnotationWithHttpInfo()`**:** Returns Lipid annotation (ElGordo) for the given formulaId

Usage:

```
FeaturesApi$GetLipidAnnotationWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (LipidAnnotation) with additional information such as HTTP status code, headers

Method `GetMsData()`**:** Mass Spec data (input data) for the given 'alignedFeatureId' .

Usage:

```
FeaturesApi$GetMsData(
  project_id,
  aligned_feature_id,
  ms_data_search_prepared = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the Mass Spec data belongs to.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: MsData

Method `GetMsDataWithHttpInfo()`**:** Mass Spec data (input data) for the given 'alignedFeatureId' .

Usage:

```
FeaturesApi$GetMsDataWithHttpInfo(
  project_id,
  aligned_feature_id,
  ms_data_search_prepared = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the Mass Spec data belongs to.

`ms_data_search_prepared` (optional) Returns all fragment spectra in a preprocessed form as used for fast Cosine/Modified Cosine computation. Gives you spectra compatible with SpectralLibraryMatch peak assignments and reference spectra. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (MsData) with additional information such as HTTP status code, headers

Method `GetSpectralLibraryMatch()`: Spectral library match for the given 'alignedFeatureId'.

Usage:

```
FeaturesApi$GetSpectralLibraryMatch(
  project_id,
  aligned_feature_id,
  match_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`match_id` id of the library match to be returned.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: SpectralLibraryMatch

Method `GetSpectralLibraryMatchWithHttpInfo()`: Spectral library match for the given 'alignedFeatureId'.

Usage:

```
FeaturesApi$GetSpectralLibraryMatchWithHttpInfo(
  project_id,
  aligned_feature_id,
  match_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`match_id` id of the library match to be returned.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (SpectralLibraryMatch) with additional information such as HTTP status code, headers

Method `GetSpectralLibraryMatches()`: [DEPRECATED] List of spectral library matches for the given 'alignedFeatureId'

Usage:

```
FeaturesApi$GetSpectralLibraryMatches(
  project_id,
  aligned_feature_id,
  min_shared_peaks = 1,
  min_similarity = 0.2,
  inchi_key = "",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`min_shared_peaks` (optional) No description (default value: 1)

`min_similarity` (optional) No description (default value: 0.2)

`inchi_key` (optional) No description (default value: "")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SpectralLibraryMatch]

Method `GetSpectralLibraryMatchesWithHttpInfo()`: [DEPRECATED] List of spectral library matches for the given 'alignedFeatureId'

Usage:

```
FeaturesApi$GetSpectralLibraryMatchesWithHttpInfo(
  project_id,
  aligned_feature_id,
  min_shared_peaks = 1,
  min_similarity = 0.2,
  inchi_key = "",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`min_shared_peaks` (optional) No description (default value: 1)

`min_similarity` (optional) No description (default value: 0.2)

`inchi_key` (optional) No description (default value: "")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SpectralLibraryMatch]) with additional information such as HTTP status code, headers

Method `GetSpectralLibraryMatchesPage()`**:** Page of spectral library matches for the given 'alignedFeatureId'.

Usage:

```
FeaturesApi$GetSpectralLibraryMatchesPage(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  min_shared_peaks = 1,
  min_similarity = 0.2,
  inchi_key = "",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`min_shared_peaks` (optional) No description (default value: 1)

`min_similarity` (optional) No description (default value: 0.2)

`inchi_key` (optional) No description (default value: "")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelSpectralLibraryMatch

Method `GetSpectralLibraryMatchesPageWithHttpInfo()`**:** Page of spectral library matches for the given 'alignedFeatureId'.

Usage:

```
FeaturesApi$GetSpectralLibraryMatchesPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
```

```
  min_shared_peaks = 1,
  min_similarity = 0.2,
  inchi_key = "",
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`min_shared_peaks` (optional) No description (default value: 1)

`min_similarity` (optional) No description (default value: 0.2)

`inchi_key` (optional) No description (default value: "")

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelSpectralLibraryMatch) with additional information such as HTTP status code, headers

Method `GetSpectralLibraryMatchesSummary()`**:** Summarize matched reference spectra for the given 'alignedFeatureId'.

Usage:

```
FeaturesApi$GetSpectralLibraryMatchesSummary(
  project_id,
  aligned_feature_id,
  min_shared_peaks = 1,
  min_similarity = 0.2,
  inchi_key = "",
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`min_shared_peaks` (optional) min threshold of shared peaks. (default value: 1)

`min_similarity` (optional) min spectral similarity threshold. (default value: 0.2)

`inchi_key` (optional) 2D inchi key of the compound in the structure database. (default value: "")

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: SpectralLibraryMatchSummary

Method `GetSpectralLibraryMatchesSummaryWithHttpInfo()`: Summarize matched reference spectra for the given 'alignedFeatureId'.

Usage:

```
FeaturesApi$GetSpectralLibraryMatchesSummaryWithHttpInfo(
  project_id,
  aligned_feature_id,
  min_shared_peaks = 1,
  min_similarity = 0.2,
  inchi_key = "",
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`min_shared_peaks` (optional) min threshold of shared peaks. (default value: 1)

`min_similarity` (optional) min spectral similarity threshold. (default value: 0.2)

`inchi_key` (optional) 2D inchi key of the compound in the structure database. (default value: "")

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (SpectralLibraryMatchSummary) with additional information such as HTTP status code, headers

Method `GetStructureCandidates()`: [DEPRECATED] List of structure database search candidates ranked by CSI:FingerID score for the given 'alignedFeatureId' with minimal information

Usage:

```
FeaturesApi$GetStructureCandidates(
  project_id,
  aligned_feature_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[StructureCandidateFormula]

Method `GetStructureCandidatesWithHttpInfo()`: [DEPRECATED] List of structure database search candidates ranked by CSI:FingerID score for the given 'alignedFeatureId' with minimal information

Usage:

```
FeaturesApi$GetStructureCandidatesWithHttpInfo(
  project_id,
  aligned_feature_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[StructureCandidateFormula]) with additional information such as HTTP status code, headers

Method `GetStructureCandidatesByFormula()`: [DEPRECATED] List of CSI:FingerID structure database search candidates for the given 'formulaId' with minimal information

Usage:

```
FeaturesApi$GetStructureCandidatesByFormula(
  project_id,
  aligned_feature_id,
  formula_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[StructureCandidateScored]

Method `GetStructureCandidatesByFormulaWithHttpInfo()`: [DEPRECATED] List of CSI:FingerID structure database search candidates for the given 'formulaId' with minimal information

Usage:

```
FeaturesApi$GetStructureCandidatesByFormulaWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[StructureCandidateScored]) with additional information such as HTTP status code, headers

Method `GetStructureCandidatesByFormulaPage()`**:** Page of CSI:FingerID structure database search candidates for the given 'formulaId' with minimal information.

Usage:

```
FeaturesApi$GetStructureCandidatesByFormulaPage(
  project_id,
  aligned_feature_id,
  formula_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelStructureCandidateScored

Method `GetStructureCandidatesByFormulaPageWithHttpInfo()`**:** Page of CSI:FingerID structure database search candidates for the given 'formulaId' with minimal information.

Usage:

```
FeaturesApi$GetStructureCandidatesByFormulaPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  formula_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the formula result belongs to.

`formula_id` identifier of the requested formula result

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelStructureCandidateScored) with additional information such as HTTP status code, headers

Method `GetStructureCandidatesPage()`**:** Page of structure database search candidates ranked by CSI:FingerID score for the given 'alignedFeatureId' with minimal information.

Usage:

```
FeaturesApi$GetStructureCandidatesPage(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelStructureCandidateFormula

Method `GetStructureCandidatesPageWithHttpInfo()`**:** Page of structure database search candidates ranked by CSI:FingerID score for the given 'alignedFeatureId' with minimal information.

Usage:

```
FeaturesApi$GetStructureCandidatesPageWithHttpInfo(
  project_id,
  aligned_feature_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`aligned_feature_id` feature (aligned over runs) the structure candidates belong to.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelStructureCandidateFormula) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  AddAlignedFeatures  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to import into.
var_feature_import <- c(FeatureImport$new(123, 123, "name_example", "externalFeatureId_example", c("detectedAddu
var_profile <- "profile_example" # character | profile describing the instrument used to measure the data. Used to m
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' to override defau

#Import (aligned) features into the project.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$AddAlignedFeatures(var_project_id, var_feature_import, profile = var_profile, opt_fields
result <- api_instance$features_api$AddAlignedFeatures(var_project_id, var_feature_import, profile = var_profile
dput(result)


####################  DeleteAlignedFeature  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to delete from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | identifier of feature (aligned over runs) to d

#Delete feature (aligned over runs) with the given identifier from the specified project-space.
api_instance <- rsirius_api$new()

api_instance$features_api$DeleteAlignedFeature(var_project_id, var_aligned_feature_id)


####################  DeleteAlignedFeatures  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to delete from.
var_request_body <- c("property_example") # array[character] |

#Delete feature (aligned over runs) with the given identifier from the specified project-space.
api_instance <- rsirius_api$new()

api_instance$features_api$DeleteAlignedFeatures(var_project_id, var_request_body)


####################  DeleteAlignedFeaturesByQuery  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to delete from.
var_search_query <- "search_query_example" # character | tag/text/range query in lucene syntax; must be non-empty (

#Delete all features (aligned over runs) that match the given lucene search query from the specified  project-space.
api_instance <- rsirius_api$new()
```

```
api_instance$features_api$DeleteAlignedFeaturesByQuery(var_project_id, var_search_query)


####################  GetAlignedFeature  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | identifier of feature (aligned over runs) to a
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Get feature (aligned over runs) with the given identifier from the specified project-space.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetAlignedFeature(var_project_id, var_aligned_feature_id, ms_data_search_prepared = var_
result <- api_instance$features_api$GetAlignedFeature(var_project_id, var_aligned_feature_id, ms_data_search_pre
dput(result)


####################  GetAlignedFeatures  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#[DEPRECATED] Get all available features (aligned over runs) in the given project-space
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetAlignedFeatures(var_project_id, ms_data_search_prepared = var_ms_data_search_prepare
result <- api_instance$features_api$GetAlignedFeatures(var_project_id, ms_data_search_prepared = var_ms_data_sea
dput(result)


####################  GetAlignedFeaturesPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project space to get features (aligned over runs) from.
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_search_query <- "search_query_example" # character | Optional search query in lucene syntax. Omit this paramete
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Get features (aligned over runs) in the given project-space
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetAlignedFeaturesPage(var_project_id, page = var_page, size = var_size, sort = var_sort,
result <- api_instance$features_api$GetAlignedFeaturesPage(var_project_id, page = var_page, size = var_size, sort
```

```
dput(result)


####################  GetAlignedFeaturesSearchableFields  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project space to read the searchable feature fields from.

#Get all fields that can be used in the searchQuery parameter of feature (aligned over runs) endpoints
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetAlignedFeaturesSearchableFields(var_project_iddata_file = "result.txt")
result <- api_instance$features_api$GetAlignedFeaturesSearchableFields(var_project_id)
dput(result)


####################  GetBestMatchingCompoundClasses  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result

#Return Best matching compound classes for given formulaId
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetBestMatchingCompoundClasses(var_project_id, var_aligned_feature_id, var_formula_idda
result <- api_instance$features_api$GetBestMatchingCompoundClasses(var_project_id, var_aligned_feature_id, var_f
dput(result)


####################  GetCanopusPrediction  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result

#All predicted compound classes (CANOPUS) from ClassyFire and NPC and their probabilities,
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCanopusPrediction(var_project_id, var_aligned_feature_id, var_formula_iddata_file = "
result <- api_instance$features_api$GetCanopusPrediction(var_project_id, var_aligned_feature_id, var_formula_id)
dput(result)


####################  GetDeNovoStructureCandidates  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
```

```
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#[DEPRECATED] List of de novo structure candidates (e
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDeNovoStructureCandidates(var_project_id, var_aligned_feature_id, opt_fields = var_op
result <- api_instance$features_api$GetDeNovoStructureCandidates(var_project_id, var_aligned_feature_id, opt_fie
dput(result)


####################  GetDeNovoStructureCandidatesByFormula  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#[DEPRECATED] List of de novo structure candidates (e
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDeNovoStructureCandidatesByFormula(var_project_id, var_aligned_feature_id, var_formu
result <- api_instance$features_api$GetDeNovoStructureCandidatesByFormula(var_project_id, var_aligned_feature_i
dput(result)


####################  GetDeNovoStructureCandidatesByFormulaPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Page of de novo structure candidates (e.g. generated by MsNovelist) ranked by CSI:FingerID score for the given 'for
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDeNovoStructureCandidatesByFormulaPage(var_project_id, var_aligned_feature_id, var_f
result <- api_instance$features_api$GetDeNovoStructureCandidatesByFormulaPage(var_project_id, var_aligned_featu
dput(result)


####################  GetDeNovoStructureCandidatesPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
```

```
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Page of de novo structure candidates (e.g. generated by MsNovelist) ranked by CSI:FingerID score for the given 'ali
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDeNovoStructureCandidatesPage(var_project_id, var_aligned_feature_id, page = var_page
result <- api_instance$features_api$GetDeNovoStructureCandidatesPage(var_project_id, var_aligned_feature_id, pag
dput(result)


####################  GetFeatureQuantTable  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_search_query <- "search_query_example" # character | Optional search query in lucene syntax. Omit this paramete
var_type <- "APEX_INTENSITY" # character | quantification type. (Optional)
var_opt_fields <- c(QuantTableOptField$new()) # array[QuantTableOptField] |  (Optional)

#Returns the full quantification table of features
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFeatureQuantTable(var_project_id, search_query = var_search_query, type = var_type, op
result <- api_instance$features_api$GetFeatureQuantTable(var_project_id, search_query = var_search_query, type =
dput(result)


####################  GetFeatures  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) whose features to

#[DEPRECATED] Get all features the given feature (aligned over runs) was aligned from
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFeatures(var_project_id, var_aligned_feature_iddata_file = "result.txt")
result <- api_instance$features_api$GetFeatures(var_project_id, var_aligned_feature_id)
dput(result)


####################  GetFeaturesPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) whose features to
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
```

```
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so

#Page of features the given feature (aligned over runs) was aligned from, one per run it has been detected in
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFeaturesPage(var_project_id, var_aligned_feature_id, page = var_page, size = var_size,
result <- api_instance$features_api$GetFeaturesPage(var_project_id, var_aligned_feature_id, page = var_page, size
dput(result)


####################  GetFingerprintPrediction  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result

#Returns predicted fingerprint (CSI:FingerID) for the given formula result identifier (formulaId)
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFingerprintPrediction(var_project_id, var_aligned_feature_id, var_formula_iddata_fil
result <- api_instance$features_api$GetFingerprintPrediction(var_project_id, var_aligned_feature_id, var_formula
dput(result)


####################  GetFormulaAnnotatedMsMsData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f

#Returns MS/MS Spectrum annotated with fragments and losses for provided formulaId
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFormulaAnnotatedMsMsData(var_project_id, var_aligned_feature_id, var_formula_id, ms_
result <- api_instance$features_api$GetFormulaAnnotatedMsMsData(var_project_id, var_aligned_feature_id, var_form
dput(result)


####################  GetFormulaAnnotatedSpectrum  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_spectrum_index <- -1 # integer | index of the spectrum to be annotated. Merged MS/MS will be used if spectrumInde
var_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for fast
```

```
#Returns a fragmentation spectrum (e
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFormulaAnnotatedSpectrum(var_project_id, var_aligned_feature_id, var_formula_id, spe
result <- api_instance$features_api$GetFormulaAnnotatedSpectrum(var_project_id, var_aligned_feature_id, var_form
dput(result)


####################  GetFormulaCandidate  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#FormulaResultContainers for the given 'formulaId' with minimal information.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFormulaCandidate(var_project_id, var_aligned_feature_id, var_formula_id, ms_data_sea
result <- api_instance$features_api$GetFormulaCandidate(var_project_id, var_aligned_feature_id, var_formula_id,
dput(result)


####################  GetFormulaCandidates  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#[DEPRECATED] List of FormulaResultContainers available for this feature with minimal information
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFormulaCandidates(var_project_id, var_aligned_feature_id, ms_data_search_prepared = v
result <- api_instance$features_api$GetFormulaCandidates(var_project_id, var_aligned_feature_id, ms_data_search_
dput(result)


####################  GetFormulaCandidatesPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
```

```
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Page of FormulaResultContainers available for this feature with minimal information.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFormulaCandidatesPage(var_project_id, var_aligned_feature_id, page = var_page, size =
result <- api_instance$features_api$GetFormulaCandidatesPage(var_project_id, var_aligned_feature_id, page = var_
dput(result)


####################  GetFragTree  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result

#Returns fragmentation tree (SIRIUS) for the given formula result identifier
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFragTree(var_project_id, var_aligned_feature_id, var_formula_iddata_file = "result.tx
result <- api_instance$features_api$GetFragTree(var_project_id, var_aligned_feature_id, var_formula_id)
dput(result)


####################  GetIsotopePatternAnnotation  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result

#Returns Isotope pattern information for given formulaId
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetIsotopePatternAnnotation(var_project_id, var_aligned_feature_id, var_formula_iddata_
result <- api_instance$features_api$GetIsotopePatternAnnotation(var_project_id, var_aligned_feature_id, var_form
dput(result)


####################  GetLipidAnnotation  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result

#Returns Lipid annotation (ElGordo) for the given formulaId
api_instance <- rsirius_api$new()
```

```
# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetLipidAnnotation(var_project_id, var_aligned_feature_id, var_formula_iddata_file = "re
result <- api_instance$features_api$GetLipidAnnotation(var_project_id, var_aligned_feature_id, var_formula_id)
dput(result)


####################  GetMsData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the Mass Spec data
var_ms_data_search_prepared <- FALSE # character | Returns all fragment spectra in a preprocessed form as used for f

#Mass Spec data (input data) for the given 'alignedFeatureId' .
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetMsData(var_project_id, var_aligned_feature_id, ms_data_search_prepared = var_ms_data_
result <- api_instance$features_api$GetMsData(var_project_id, var_aligned_feature_id, ms_data_search_prepared = 
dput(result)


####################  GetSpectralLibraryMatch  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_match_id <- "match_id_example" # character | id of the library match to be returned.
var_opt_fields <- c("none") # array[character] |  (Optional)

#Spectral library match for the given 'alignedFeatureId'.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetSpectralLibraryMatch(var_project_id, var_aligned_feature_id, var_match_id, opt_field
result <- api_instance$features_api$GetSpectralLibraryMatch(var_project_id, var_aligned_feature_id, var_match_id
dput(result)


####################  GetSpectralLibraryMatches  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_min_shared_peaks <- 1 # integer |  (Optional)
var_min_similarity <- 0.2 # numeric |  (Optional)
var_inchi_key <- "" # character |  (Optional)
var_opt_fields <- c("none") # array[character] |  (Optional)

#[DEPRECATED] List of spectral library matches for the given 'alignedFeatureId'
api_instance <- rsirius_api$new()
```

```
# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetSpectralLibraryMatches(var_project_id, var_aligned_feature_id, min_shared_peaks = var
result <- api_instance$features_api$GetSpectralLibraryMatches(var_project_id, var_aligned_feature_id, min_shared
dput(result)


####################  GetSpectralLibraryMatchesPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_min_shared_peaks <- 1 # integer |  (Optional)
var_min_similarity <- 0.2 # numeric |  (Optional)
var_inchi_key <- "" # character |  (Optional)
var_opt_fields <- c("none") # array[character] |  (Optional)

#Page of spectral library matches for the given 'alignedFeatureId'.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetSpectralLibraryMatchesPage(var_project_id, var_aligned_feature_id, page = var_page, s
result <- api_instance$features_api$GetSpectralLibraryMatchesPage(var_project_id, var_aligned_feature_id, page =
dput(result)


####################  GetSpectralLibraryMatchesSummary  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_min_shared_peaks <- 1 # integer | min threshold of shared peaks. (Optional)
var_min_similarity <- 0.2 # numeric | min spectral similarity threshold. (Optional)
var_inchi_key <- "" # character | 2D inchi key of the compound in the structure database. (Optional)

#Summarize matched reference spectra for the given 'alignedFeatureId'.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetSpectralLibraryMatchesSummary(var_project_id, var_aligned_feature_id, min_shared_pea
result <- api_instance$features_api$GetSpectralLibraryMatchesSummary(var_project_id, var_aligned_feature_id, mi
dput(result)


####################  GetStructureCandidates  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override
```

```
#[DEPRECATED] List of structure database search candidates ranked by CSI:FingerID score for the given 'alignedFeatu
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetStructureCandidates(var_project_id, var_aligned_feature_id, opt_fields = var_opt_fiel
result <- api_instance$features_api$GetStructureCandidates(var_project_id, var_aligned_feature_id, opt_fields = 
dput(result)


####################  GetStructureCandidatesByFormula  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override 

#[DEPRECATED] List of CSI:FingerID structure database search candidates for the given 'formulaId' with minimal info
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetStructureCandidatesByFormula(var_project_id, var_aligned_feature_id, var_formula_id,
result <- api_instance$features_api$GetStructureCandidatesByFormula(var_project_id, var_aligned_feature_id, var_
dput(result)


####################  GetStructureCandidatesByFormulaPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the formula resul
var_formula_id <- "formula_id_example" # character | identifier of the requested formula result
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override 

#Page of CSI:FingerID structure database search candidates for the given 'formulaId' with minimal information.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetStructureCandidatesByFormulaPage(var_project_id, var_aligned_feature_id, var_formula
result <- api_instance$features_api$GetStructureCandidatesByFormulaPage(var_project_id, var_aligned_feature_id,
dput(result)


####################  GetStructureCandidatesPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_aligned_feature_id <- "aligned_feature_id_example" # character | feature (aligned over runs) the structure can
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
```

```
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Page of structure database search candidates ranked by CSI:FingerID score for the given 'alignedFeatureId' with mi
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetStructureCandidatesPage(var_project_id, var_aligned_feature_id, page = var_page, size
result <- api_instance$features_api$GetStructureCandidatesPage(var_project_id, var_aligned_feature_id, page = va
dput(result)



## End(Not run)
```

`FingerprintPrediction` *FingerprintPrediction*

Description

User/developer friendly parameter subset for the CSI:FingerID Fingerprint tool Needs results from the Formula identification tool (Sirius).

FingerprintPrediction Class

Format

An `R6Class` generator object

Details

Create a new FingerprintPrediction

Public fields

`enabled` Indicates whether the tool is enabled. character [optional]

`useScoreThreshold` If true, an adaptive soft threshold will be applied to only compute Fingerprints for promising formula candidates Enabling is highly recommended. character [optional]

`alwaysPredictHighRefMatches` If true, fingerprints, compound classes and structures will be predicted for formula candidates whose reference spectrum similarity is above Sirius.minReferenceMatchScoreToInject, no matter which score threshold rules would otherwise apply. If NULL default value will be used. character [optional]

Methods

Public methods:

- `FingerprintPrediction$new()`
- `FingerprintPrediction$toJSON()`

- FingerprintPrediction$toList()
- FingerprintPrediction$toSimpleType()
- FingerprintPrediction$fromJSON()
- FingerprintPrediction$toJSONString()
- FingerprintPrediction$fromJSONString()
- FingerprintPrediction$validateJSON()
- FingerprintPrediction$toString()
- FingerprintPrediction$isValid()
- FingerprintPrediction$getInvalidFields()
- FingerprintPrediction$print()
- FingerprintPrediction$clone()

Method `new()`**:** Initialize a new FingerprintPrediction class.

Usage:

```
FingerprintPrediction$new(
  enabled = NULL,
  useScoreThreshold = NULL,
  alwaysPredictHighRefMatches = NULL,
  ...
)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`useScoreThreshold` If true, an adaptive soft threshold will be applied to only compute Fingerprints for promising formula candidates Enabling is highly recommended.

`alwaysPredictHighRefMatches` If true, fingerprints, compound classes and structures will be predicted for formula candidates whose reference spectrum similarity is above Sirius.minReferenceMatchScoreToInje no matter which score threshold rules would otherwise apply. If NULL default value will be used.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
FingerprintPrediction$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
FingerprintPrediction$toList()
```

Returns: FingerprintPrediction as a base R list.

Examples:

```
# convert array of FingerprintPrediction (x) to a data frame
\dontrun{
library(purrr)
```

```
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert FingerprintPrediction to a base R type

Usage:

```
FingerprintPrediction$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of FingerprintPrediction

Usage:

```
FingerprintPrediction$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FingerprintPrediction

Method `toJSONString()`**:** To JSON String

Usage:

```
FingerprintPrediction$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: FingerprintPrediction in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of FingerprintPrediction

Usage:

```
FingerprintPrediction$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FingerprintPrediction

Method `validateJSON()`**:** Validate JSON input with respect to FingerprintPrediction and throw an exception if invalid

Usage:

```
FingerprintPrediction$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
FingerprintPrediction$toString()
```

Returns: String representation of FingerprintPrediction

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
FingerprintPrediction$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
FingerprintPrediction$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
FingerprintPrediction$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `FingerprintPrediction$toList`
## ------------------------------------------------

# convert array of FingerprintPrediction (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`FormulaCandidate` *FormulaCandidate*

Description

Molecular formula candidate that holds a unique identifier (molecular formula + adduct). It can be extended with optional scoring metrics and the raw results such as fragmentation trees and simulated isotope pattern.

FormulaCandidate Class

Format

An R6Class generator object

Details

Create a new FormulaCandidate

Public fields

`formulaId` Unique identifier of this formula candidate character [optional]

`molecularFormula` Molecular formula of this formula candidate. character [optional]

`adduct` Adduct of this formula candidate character [optional]

`rank` integer [optional]

`siriusScoreNormalized` Normalized Sirius Score of the formula candidate. If NULL result is not available numeric [optional]

`siriusScore` Sirius Score (isotope + tree score) of the formula candidate. If NULL result is not available numeric [optional]

`isotopeScore` numeric [optional]

`treeScore` numeric [optional]

`zodiacScore` Zodiac Score of the formula candidate. If NULL result is not available numeric [optional]

`numOfExplainedPeaks` integer [optional]

`numOfExplainablePeaks` integer [optional]

`totalExplainedIntensity` numeric [optional]

`medianMassDeviation` Deviation [optional]

`fragmentationTree` FragmentationTree [optional]

`annotatedSpectrum` AnnotatedSpectrum [optional]

`isotopePatternAnnotation` IsotopePatternAnnotation [optional]

`lipidAnnotation` LipidAnnotation [optional]

`predictedFingerprint` Probabilistic molecular fingerprint predicted by CSI:FingerID list(numeric) [optional]

`compoundClasses` CompoundClasses [optional]

`canopusPrediction` CanopusPrediction [optional]

Methods

Public methods:

- `FormulaCandidate$new()`
- `FormulaCandidate$toJSON()`
- `FormulaCandidate$toList()`
- `FormulaCandidate$toSimpleType()`
- `FormulaCandidate$fromJSON()`

- FormulaCandidate$toJSONString()
- FormulaCandidate$fromJSONString()
- FormulaCandidate$validateJSON()
- FormulaCandidate$toString()
- FormulaCandidate$isValid()
- FormulaCandidate$getInvalidFields()
- FormulaCandidate$print()
- FormulaCandidate$clone()

Method `new()`**:** Initialize a new FormulaCandidate class.

Usage:

```
FormulaCandidate$new(
  formulaId = NULL,
  molecularFormula = NULL,
  adduct = NULL,
  rank = NULL,
  siriusScoreNormalized = NULL,
  siriusScore = NULL,
  isotopeScore = NULL,
  treeScore = NULL,
  zodiacScore = NULL,
  numOfExplainedPeaks = NULL,
  numOfExplainablePeaks = NULL,
  totalExplainedIntensity = NULL,
  medianMassDeviation = NULL,
  fragmentationTree = NULL,
  annotatedSpectrum = NULL,
  isotopePatternAnnotation = NULL,
  lipidAnnotation = NULL,
  predictedFingerprint = NULL,
  compoundClasses = NULL,
  canopusPrediction = NULL,
  ...
)
```

Arguments:

`formulaId` Unique identifier of this formula candidate

`molecularFormula` Molecular formula of this formula candidate.

`adduct` Adduct of this formula candidate

`rank` rank

`siriusScoreNormalized` Normalized Sirius Score of the formula candidate. If NULL result is not available

`siriusScore` Sirius Score (isotope + tree score) of the formula candidate. If NULL result is not available

`isotopeScore` isotopeScore

`treeScore` treeScore

`zodiacScore` Zodiac Score of the formula candidate. If NULL result is not available

`numOfExplainedPeaks` numOfExplainedPeaks

`numOfExplainablePeaks` numOfExplainablePeaks

`totalExplainedIntensity` totalExplainedIntensity

`medianMassDeviation` medianMassDeviation

`fragmentationTree` fragmentationTree

`annotatedSpectrum` annotatedSpectrum

`isotopePatternAnnotation` isotopePatternAnnotation

`lipidAnnotation` lipidAnnotation

`predictedFingerprint` Probabilistic molecular fingerprint predicted by CSI:FingerID

`compoundClasses` compoundClasses

`canopusPrediction` canopusPrediction

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
FormulaCandidate$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
FormulaCandidate$toList()
```

Returns: FormulaCandidate as a base R list.

Examples:

```
# convert array of FormulaCandidate (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert FormulaCandidate to a base R type

Usage:

```
FormulaCandidate$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of FormulaCandidate

Usage:

```
FormulaCandidate$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FormulaCandidate

Method `toJSONString()`**:** To JSON String

Usage:

```
FormulaCandidate$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: FormulaCandidate in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of FormulaCandidate

Usage:

```
FormulaCandidate$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FormulaCandidate

Method `validateJSON()`**:** Validate JSON input with respect to FormulaCandidate and throw an exception if invalid

Usage:

```
FormulaCandidate$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
FormulaCandidate$toString()
```

Returns: String representation of FormulaCandidate

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
FormulaCandidate$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
FormulaCandidate$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
FormulaCandidate$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `FormulaCandidate$toList`
## ------------------------------------------------

# convert array of FormulaCandidate (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`FragmentationTree` *FragmentationTree*

Description

Simple and easy serializable fragmentation tree model with annotated fragments/nodes and losses/edges Root fragment has index 0; Molecular formula and adduct are identical to the ones of the corresponding molecular formula candidate and SpectrumAnnotation

FragmentationTree Class

Format

An `R6Class` generator object

Details

Create a new FragmentationTree

Public fields

`fragments` list(FragmentNode) [optional]

`losses` list(LossEdge) [optional]

`treeScore` numeric [optional]

`molecularFormula` character [optional]

`adduct` character [optional]

Methods

Public methods:

- `FragmentationTree$new()`
- `FragmentationTree$toJSON()`
- `FragmentationTree$toList()`

- FragmentationTree$toSimpleType()
- FragmentationTree$fromJSON()
- FragmentationTree$toJSONString()
- FragmentationTree$fromJSONString()
- FragmentationTree$validateJSON()
- FragmentationTree$toString()
- FragmentationTree$isValid()
- FragmentationTree$getInvalidFields()
- FragmentationTree$print()
- FragmentationTree$clone()

Method `new()`**:** Initialize a new FragmentationTree class.

Usage:

```
FragmentationTree$new(
  fragments = NULL,
  losses = NULL,
  treeScore = NULL,
  molecularFormula = NULL,
  adduct = NULL,
  ...
)
```

Arguments:

`fragments` fragments

`losses` losses

`treeScore` treeScore

`molecularFormula` molecularFormula

`adduct` adduct

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
FragmentationTree$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
FragmentationTree$toList()
```

Returns: FragmentationTree as a base R list.

Examples:

```
# convert array of FragmentationTree (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
```

```
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert FragmentationTree to a base R type

Usage:

```
FragmentationTree$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of FragmentationTree

Usage:

```
FragmentationTree$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FragmentationTree

Method `toJSONString()`**:** To JSON String

Usage:

```
FragmentationTree$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: FragmentationTree in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of FragmentationTree

Usage:

```
FragmentationTree$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FragmentationTree

Method `validateJSON()`**:** Validate JSON input with respect to FragmentationTree and throw an exception if invalid

Usage:

```
FragmentationTree$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
FragmentationTree$toString()
```

Returns: String representation of FragmentationTree

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
FragmentationTree$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
FragmentationTree$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
FragmentationTree$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `FragmentationTree$toList`
## ------------------------------------------------

# convert array of FragmentationTree (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`FragmentNode` *FragmentNode*

Description

FragmentNode Class

FragmentNode Class

Format

An `R6Class` generator object

Details

Create a new FragmentNode

Public fields

`fragmentId` integer [optional]

`molecularFormula` neutral molecular formula of the fragment without adduct character [optional]

`adduct` This combines the ionization plus adduct of the fragment. In contrast to the <code>FragmentationTree</code>'s adduct, this adduct may not include any in-source loss. character [optional]

`massDeviationDa` numeric [optional]

`massDeviationPpm` numeric [optional]

`score` numeric [optional]

`intensity` numeric [optional]

`mz` numeric [optional]

Methods

Public methods:

- `FragmentNode$new()`
- `FragmentNode$toJSON()`
- `FragmentNode$toList()`
- `FragmentNode$toSimpleType()`
- `FragmentNode$fromJSON()`
- `FragmentNode$toJSONString()`
- `FragmentNode$fromJSONString()`
- `FragmentNode$validateJSON()`
- `FragmentNode$toString()`
- `FragmentNode$isValid()`
- `FragmentNode$getInvalidFields()`
- `FragmentNode$print()`
- `FragmentNode$clone()`

Method `new()`**:** Initialize a new FragmentNode class.

Usage:

```
FragmentNode$new(
  fragmentId = NULL,
  molecularFormula = NULL,
  adduct = NULL,
  massDeviationDa = NULL,
  massDeviationPpm = NULL,
  score = NULL,
  intensity = NULL,
  mz = NULL,
  ...
)
```

Arguments:

`fragmentId` fragmentId

`molecularFormula` neutral molecular formula of the fragment without adduct

`adduct` This combines the ionization plus adduct of the fragment. In contrast to the <code>FragmentationTree</code>'s adduct, this adduct may not include any in-source loss.

`massDeviationDa` massDeviationDa

`massDeviationPpm` massDeviationPpm

`score` score

`intensity` intensity

`mz` mz

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
FragmentNode$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
FragmentNode$toList()
```

Returns: FragmentNode as a base R list.

Examples:

```
# convert array of FragmentNode (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert FragmentNode to a base R type

Usage:

```
FragmentNode$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of FragmentNode

Usage:

```
FragmentNode$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FragmentNode

Method `toJSONString()`**:** To JSON String

Usage:

```
FragmentNode$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: FragmentNode in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of FragmentNode

Usage:

```
FragmentNode$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of FragmentNode

Method `validateJSON()`**:** Validate JSON input with respect to FragmentNode and throw an exception if invalid

Usage:

```
FragmentNode$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
FragmentNode$toString()
```

Returns: String representation of FragmentNode

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
FragmentNode$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
FragmentNode$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
FragmentNode$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `FragmentNode$toList`
## ------------------------------------------------

# convert array of FragmentNode (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

GuiApi *Gui operations*

Description

GuiApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `GuiApi$new()`
- `GuiApi$CloseGui()`
- `GuiApi$CloseGuiWithHttpInfo()`
- `GuiApi$GetGuis()`
- `GuiApi$GetGuisWithHttpInfo()`
- `GuiApi$OpenGui()`

- GuiApi$OpenGuiWithHttpInfo()
- GuiApi$clone()

Method `new()`**:** Initialize a new GuiApi.

Usage:

```
GuiApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `CloseGui()`**:** Close GUI instance of given project-space if available.

Usage:

```
GuiApi$CloseGui(project_id, close_project = NULL, data_file = NULL, ...)
```

Arguments:

`project_id` project-space the GUI instance is connected to.

`close_project` (optional) No description

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `CloseGuiWithHttpInfo()`**:** Close GUI instance of given project-space if available.

Usage:

```
GuiApi$CloseGuiWithHttpInfo(
  project_id,
  close_project = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space the GUI instance is connected to.

`close_project` (optional) No description

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `GetGuis()`**:** Get list of currently running gui windows, managed by this SIRIUS instance.

Usage:

```
GuiApi$GetGuis(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[GuiInfo]

Method `GetGuisWithHttpInfo()`**:** Get list of currently running gui windows, managed by this SIRIUS instance.

Usage:

```
GuiApi$GetGuisWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[GuiInfo]) with additional information such as HTTP status code, headers

Method `OpenGui()`**:** Open GUI instance on specified project-space and bring the GUI window to foreground.

Usage:

```
GuiApi$OpenGui(project_id, ...)
```

Arguments:

`project_id` project-space the GUI instance will connect to.

`...` Other optional arguments

Returns: void

Method `OpenGuiWithHttpInfo()`**:** Open GUI instance on specified project-space and bring the GUI window to foreground.

Usage:

```
GuiApi$OpenGuiWithHttpInfo(project_id, ...)
```

Arguments:

`project_id` project-space the GUI instance will connect to.

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  CloseGui  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space the GUI instance is connected to.
var_close_project <- "close_project_example" # character |  (Optional)

#Close GUI instance of given project-space if available.
```

```
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$CloseGui(var_project_id, close_project = var_close_projectdata_file = "result.txt")
result <- api_instance$gui_api$CloseGui(var_project_id, close_project = var_close_project)
dput(result)


####################  GetGuis  ####################

library(RSirius)

#Get list of currently running gui windows, managed by this SIRIUS instance.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetGuis(data_file = "result.txt")
result <- api_instance$gui_api$GetGuis()
dput(result)


####################  OpenGui  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space the GUI instance will connect to.

#Open GUI instance on specified project-space and bring the GUI window to foreground.
api_instance <- rsirius_api$new()

api_instance$gui_api$OpenGui(var_project_id)



## End(Not run)
```

GuiInfo *GuiInfo*

Description

GuiInfo Class

GuiInfo Class

Format

An R6Class generator object

Details

Create a new GuiInfo

Public fields

`projectId` The project this instance is running on character [optional]

Methods

Public methods:

- `GuiInfo$new()`
- `GuiInfo$toJSON()`
- `GuiInfo$toList()`
- `GuiInfo$toSimpleType()`
- `GuiInfo$fromJSON()`
- `GuiInfo$toJSONString()`
- `GuiInfo$fromJSONString()`
- `GuiInfo$validateJSON()`
- `GuiInfo$toString()`
- `GuiInfo$isValid()`
- `GuiInfo$getInvalidFields()`
- `GuiInfo$print()`
- `GuiInfo$clone()`

Method `new()`**:** Initialize a new GuiInfo class.

Usage:

```
GuiInfo$new(projectId = NULL, ...)
```

Arguments:

`projectId` The project this instance is running on

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
GuiInfo$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
GuiInfo$toList()
```

Returns: GuiInfo as a base R list.

Examples:

```
# convert array of GuiInfo (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert GuiInfo to a base R type

Usage:

```
GuiInfo$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of GuiInfo

Usage:

```
GuiInfo$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of GuiInfo

Method `toJSONString()`**:** To JSON String

Usage:

```
GuiInfo$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: GuiInfo in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of GuiInfo

Usage:

```
GuiInfo$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of GuiInfo

Method `validateJSON()`**:** Validate JSON input with respect to GuiInfo and throw an exception if invalid

Usage:

```
GuiInfo$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
GuiInfo$toString()
```

Returns: String representation of GuiInfo

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
GuiInfo$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
GuiInfo$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
GuiInfo$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `GuiInfo$toList`
## ------------------------------------------------

# convert array of GuiInfo (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ImportResult` *ImportResult*

Description

ImportResult Class

ImportResult Class

Format

An `R6Class` generator object

Details

Create a new ImportResult

Public fields

`affectedCompoundIds` List of compoundIds that have been imported. list(character)

`affectedAlignedFeatureIds` List of alignedFeatureIds that have been imported. list(character)

Methods

Public methods:

- `ImportResult$new()`
- `ImportResult$toJSON()`
- `ImportResult$toList()`
- `ImportResult$toSimpleType()`
- `ImportResult$fromJSON()`
- `ImportResult$toJSONString()`
- `ImportResult$fromJSONString()`
- `ImportResult$validateJSON()`
- `ImportResult$toString()`
- `ImportResult$isValid()`
- `ImportResult$getInvalidFields()`
- `ImportResult$print()`
- `ImportResult$clone()`

Method `new()`**:** Initialize a new ImportResult class.

Usage:

```
ImportResult$new(affectedCompoundIds, affectedAlignedFeatureIds, ...)
```

Arguments:

`affectedCompoundIds` List of compoundIds that have been imported.

`affectedAlignedFeatureIds` List of alignedFeatureIds that have been imported.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ImportResult$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ImportResult$toList()
```

Returns: ImportResult as a base R list.

Examples:

```
# convert array of ImportResult (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ImportResult to a base R type

Usage:

```
ImportResult$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ImportResult

Usage:

```
ImportResult$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ImportResult

Method `toJSONString()`**:** To JSON String

Usage:

```
ImportResult$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ImportResult in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ImportResult

Usage:

```
ImportResult$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ImportResult

Method `validateJSON()`**:** Validate JSON input with respect to ImportResult and throw an exception if invalid

Usage:

```
ImportResult$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ImportResult$toString()
```

Returns: String representation of ImportResult

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ImportResult$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ImportResult$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ImportResult$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ImportResult$toList`
## ------------------------------------------------

# convert array of ImportResult (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

Info *Info*

Description

Info Class

Info Class

Format

An R6Class generator object

Details

Create a new Info

Public fields

`nightSkyApiVersion` API version of the SIRIUS Nightsky API character [optional]

`siriusVersion` Version of the SIRIUS application character [optional]

`latestSiriusVersion` Latest available Version of the SIRIUS application character [optional]

`latestSiriusLink` Link to the latest available Version of the SIRIUS application character [optional]

`updateAvailable` True if a newer SIRIUS version is available. character [optional]

`siriusLibVersion` Version of the SIRIUS libraries character [optional]

`fingerIdLibVersion` Version of the CSI:FingerID libraries character [optional]

`chemDbVersion` Version of the Chemical Database available via SIRIUS web services character [optional]

`fingerIdModelVersion` Version of the Machine learning models used for Fingerprint, Compound Class and Structure Prediction Not available if web service is not reachable. character [optional]

`fingerprintId` Version of the Molecular Fingerprint used by SIRIUS character [optional]

`availableILPSolvers` Set of solvers that are configured correctly and can be loaded list(character)

`supportedILPSolvers` Set of ILP solvers that are supported, and their version information named list(character)

Methods

Public methods:

- `Info$new()`
- `Info$toJSON()`
- `Info$toList()`
- `Info$toSimpleType()`

- Info$fromJSON()
- Info$toJSONString()
- Info$fromJSONString()
- Info$validateJSON()
- Info$toString()
- Info$isValid()
- Info$getInvalidFields()
- Info$print()
- Info$clone()

Method `new()`**:** Initialize a new Info class.

Usage:

```
Info$new(
  availableILPSolvers,
  supportedILPSolvers,
  nightSkyApiVersion = NULL,
  siriusVersion = NULL,
  latestSiriusVersion = NULL,
  latestSiriusLink = NULL,
  updateAvailable = NULL,
  siriusLibVersion = NULL,
  fingerIdLibVersion = NULL,
  chemDbVersion = NULL,
  fingerIdModelVersion = NULL,
  fingerprintId = NULL,
  ...
)
```

Arguments:

`availableILPSolvers` Set of solvers that are configured correctly and can be loaded

`supportedILPSolvers` Set of ILP solvers that are supported, and their version information

`nightSkyApiVersion` API version of the SIRIUS Nightsky API

`siriusVersion` Version of the SIRIUS application

`latestSiriusVersion` Latest available Version of the SIRIUS application

`latestSiriusLink` Link to the latest available Version of the SIRIUS application

`updateAvailable` True if a newer SIRIUS version is available.

`siriusLibVersion` Version of the SIRIUS libraries

`fingerIdLibVersion` Version of the CSI:FingerID libraries

`chemDbVersion` Version of the Chemical Database available via SIRIUS web services

`fingerIdModelVersion` Version of the Machine learning models used for Fingerprint, Compound Class and Structure Prediction Not available if web service is not reachable.

`fingerprintId` Version of the Molecular Fingerprint used by SIRIUS

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Info$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Info$toList()
```

Returns: Info as a base R list.

Examples:

```
# convert array of Info (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Info to a base R type

Usage:

```
Info$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Info

Usage:

```
Info$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Info

Method `toJSONString()`**:** To JSON String

Usage:

```
Info$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Info in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Info

Usage:

```
Info$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Info

Method `validateJSON()`**:** Validate JSON input with respect to Info and throw an exception if invalid

Usage:

```
Info$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Info$toString()
```

Returns: String representation of Info

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Info$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Info$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Info$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Info$toList`
## ------------------------------------------------

# convert array of Info (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`InfoApi` *Info operations*

Description

InfoApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `InfoApi$new()`
- `InfoApi$GetConnectionCheck()`
- `InfoApi$GetConnectionCheckWithHttpInfo()`
- `InfoApi$GetInfo()`
- `InfoApi$GetInfoWithHttpInfo()`
- `InfoApi$clone()`

Method `new()`**:** Initialize a new InfoApi.

Usage:

```
InfoApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `GetConnectionCheck()`**:**

Usage:

```
InfoApi$GetConnectionCheck(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

... Other optional arguments

Returns: ConnectionCheck

Method `GetConnectionCheckWithHttpInfo()`**:**

Usage:

```
InfoApi$GetConnectionCheckWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

... Other optional arguments

Returns: API response (ConnectionCheck) with additional information such as HTTP status code, headers

Method `GetInfo()`**:**

Usage:

```
InfoApi$GetInfo(server_info = TRUE, update_info = TRUE, data_file = NULL, ...)
```

Arguments:

`server_info` (optional) No description (default value: TRUE)

`update_info` (optional) No description (default value: TRUE)

`data_file` (optional) name of the data file to save the result

... Other optional arguments

Returns: Info

Method `GetInfoWithHttpInfo()`**:**

Usage:

```
InfoApi$GetInfoWithHttpInfo(
  server_info = TRUE,
  update_info = TRUE,
  data_file = NULL,
  ...
)
```

Arguments:

`server_info` (optional) No description (default value: TRUE)

`update_info` (optional) No description (default value: TRUE)

`data_file` (optional) name of the data file to save the result

... Other optional arguments

Returns: API response (Info) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  GetConnectionCheck  ####################

library(RSirius)

api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetConnectionCheck(data_file = "result.txt")
result <- api_instance$info_api$GetConnectionCheck()
dput(result)


####################  GetInfo  ####################

library(RSirius)
var_server_info <- TRUE # character |  (Optional)
var_update_info <- TRUE # character |  (Optional)

api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetInfo(server_info = var_server_info, update_info = var_update_infodata_file = "result.t
result <- api_instance$info_api$GetInfo(server_info = var_server_info, update_info = var_update_info)
dput(result)



## End(Not run)
```

`IsotopePatternAnnotation`
IsotopePatternAnnotation

Description

IsotopePatternAnnotation Class

IsotopePatternAnnotation Class

Format

An `R6Class` generator object

Details

Create a new IsotopePatternAnnotation

Public fields

`isotopePattern` BasicSpectrum [optional]

`simulatedPattern` BasicSpectrum [optional]

Methods

Public methods:

- `IsotopePatternAnnotation$new()`
- `IsotopePatternAnnotation$toJSON()`
- `IsotopePatternAnnotation$toList()`
- `IsotopePatternAnnotation$toSimpleType()`
- `IsotopePatternAnnotation$fromJSON()`
- `IsotopePatternAnnotation$toJSONString()`
- `IsotopePatternAnnotation$fromJSONString()`
- `IsotopePatternAnnotation$validateJSON()`
- `IsotopePatternAnnotation$toString()`
- `IsotopePatternAnnotation$isValid()`
- `IsotopePatternAnnotation$getInvalidFields()`
- `IsotopePatternAnnotation$print()`
- `IsotopePatternAnnotation$clone()`

Method `new()`**:** Initialize a new IsotopePatternAnnotation class.

Usage:

```
IsotopePatternAnnotation$new(
  isotopePattern = NULL,
  simulatedPattern = NULL,
  ...
)
```

Arguments:

`isotopePattern` isotopePattern

`simulatedPattern` simulatedPattern

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
IsotopePatternAnnotation$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
IsotopePatternAnnotation$toList()
```

Returns: IsotopePatternAnnotation as a base R list.

Examples:

```
# convert array of IsotopePatternAnnotation (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert IsotopePatternAnnotation to a base R type

Usage:

```
IsotopePatternAnnotation$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of IsotopePatternAnnotation

Usage:

```
IsotopePatternAnnotation$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of IsotopePatternAnnotation

Method `toJSONString()`**:** To JSON String

Usage:

```
IsotopePatternAnnotation$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: IsotopePatternAnnotation in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of IsotopePatternAnnotation

Usage:

```
IsotopePatternAnnotation$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of IsotopePatternAnnotation

Method `validateJSON()`**:** Validate JSON input with respect to IsotopePatternAnnotation and throw an exception if invalid

Usage:

```
IsotopePatternAnnotation$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
IsotopePatternAnnotation$toString()
```

Returns: String representation of IsotopePatternAnnotation

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
IsotopePatternAnnotation$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
IsotopePatternAnnotation$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
IsotopePatternAnnotation$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `IsotopePatternAnnotation$toList`
## ------------------------------------------------

# convert array of IsotopePatternAnnotation (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

Job *Job*

Description

Identifier created by the SIRIUS Nightsky API for a newly created Job. Object can be enriched with Job status/progress information (<code>JobProgress</code>) and/or Job command information. This is a return value of the API. So nullable values can also be NOT_REQUIRED to allow for easy removal.

Job Class

Format

An `R6Class` generator object

Details

Create a new Job

Public fields

`id` Unique identifier to access the job via the API character [optional]

`command` Command string of the executed Task character [optional]

`progress` JobProgress [optional]

`affectedCompoundIds` List of compoundIds that are affected by this job. This list will also contain compoundIds where not all features of the compound are affected by the job. If this job is creating compounds (e.g. data import jobs) this value will be NULL until the job has finished list(character) [optional]

`affectedAlignedFeatureIds` List of alignedFeatureIds that are affected by this job. If this job is creating features (e.g. data import jobs) this value will be NULL until the job has finished list(character) [optional]

`jobEffect` Effect this job has. The affected ids are added, removed or modified. Null if job does not affect features/compounds Not available/null if affected Ids are not requested character [optional]

Methods

Public methods:

- `Job$new()`
- `Job$toJSON()`
- `Job$toList()`
- `Job$toSimpleType()`
- `Job$fromJSON()`
- `Job$toJSONString()`

- Job$fromJSONString()
- Job$validateJSON()
- Job$toString()
- Job$isValid()
- Job$getInvalidFields()
- Job$print()
- Job$clone()

Method `new()`**:** Initialize a new Job class.

Usage:

```
Job$new(
  id = NULL,
  command = NULL,
  progress = NULL,
  affectedCompoundIds = NULL,
  affectedAlignedFeatureIds = NULL,
  jobEffect = NULL,
  ...
)
```

Arguments:

`id` Unique identifier to access the job via the API

`command` Command string of the executed Task

`progress` progress

`affectedCompoundIds` List of compoundIds that are affected by this job. This list will also contain compoundIds where not all features of the compound are affected by the job. If this job is creating compounds (e.g. data import jobs) this value will be NULL until the job has finished

`affectedAlignedFeatureIds` List of alignedFeatureIds that are affected by this job. If this job is creating features (e.g. data import jobs) this value will be NULL until the job has finished

`jobEffect` Effect this job has. The affected ids are added, removed or modified. Null if job does not affect features/compounds Not available/null if affected Ids are not requested

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Job$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Job$toList()
```

Returns: Job as a base R list.

Examples:

```
# convert array of Job (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Job to a base R type

Usage:

```
Job$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Job

Usage:

```
Job$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Job

Method `toJSONString()`**:** To JSON String

Usage:

```
Job$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Job in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Job

Usage:

```
Job$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Job

Method `validateJSON()`**:** Validate JSON input with respect to Job and throw an exception if invalid

Usage:

```
Job$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Job$toString()
```

Returns: String representation of Job

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Job$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Job$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Job$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Job$toList`
## ------------------------------------------------

# convert array of Job (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

JobProgress *JobProgress*

Description

Progress information of a computation job that has already been submitted to SIRIUS. If currentProgress == maxProgress, the job is finished and should change to state DONE soon. If a job is DONE, all results can be accessed via the project-space API.

JobProgress Class

Format

An `R6Class` generator object

Details

Create a new JobProgress

Public fields

`indeterminate` Is the progress indeterminate or not character [optional]

`state` Current state of the Jobs in the SIRIUS internal Job scheduler WAITING: Waiting for submission to ExecutorService (e.g. due to dependent jobs) READY: Ready for submission but not yet enqueued for submission to ExecutorService. QUEUED: Enqueued for submission to ExecutorService. SUBMITTED: Submitted and waiting to be executed. RUNNING: Job is running. CANCELED: Jobs is finished due to cancellation by user or dependent jobs. FAILED: Job is finished but failed. DONE: Job finished successfully. character [optional]

`currentProgress` Current progress value of the job. integer [optional]

`maxProgress` Progress value to reach (might also change during execution) integer [optional]

`message` Progress information and warnings. character [optional]

`errorMessage` Error message if the job did not finish successfully. character [optional]

Methods

Public methods:

- `JobProgress$new()`
- `JobProgress$toJSON()`
- `JobProgress$toList()`
- `JobProgress$toSimpleType()`
- `JobProgress$fromJSON()`
- `JobProgress$toJSONString()`
- `JobProgress$fromJSONString()`
- `JobProgress$validateJSON()`
- `JobProgress$toString()`

- JobProgress$isValid()
- JobProgress$getInvalidFields()
- JobProgress$print()
- JobProgress$clone()

Method `new()`**:** Initialize a new JobProgress class.

Usage:

```
JobProgress$new(
  indeterminate = NULL,
  state = NULL,
  currentProgress = NULL,
  maxProgress = NULL,
  message = NULL,
  errorMessage = NULL,
  ...
)
```

Arguments:

`indeterminate` Is the progress indeterminate or not

`state` Current state of the Jobs in the SIRIUS internal Job scheduler WAITING: Waiting for submission to ExecutorService (e.g. due to dependent jobs) READY: Ready for submission but not yet enqueued for submission to ExecutorService. QUEUED: Enqueued for submission to ExecutorService. SUBMITTED: Submitted and waiting to be executed. RUNNING: Job is running. CANCELED: Jobs is finished due to cancellation by user or dependent jobs. FAILED: Job is finished but failed. DONE: Job finished successfully.

`currentProgress` Current progress value of the job.

`maxProgress` Progress value to reach (might also change during execution)

`message` Progress information and warnings.

`errorMessage` Error message if the job did not finish successfully.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
JobProgress$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
JobProgress$toList()
```

Returns: JobProgress as a base R list.

Examples:

```
# convert array of JobProgress (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
```

```
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert JobProgress to a base R type

Usage:

```
JobProgress$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of JobProgress

Usage:

```
JobProgress$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of JobProgress

Method `toJSONString()`**:** To JSON String

Usage:

```
JobProgress$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: JobProgress in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of JobProgress

Usage:

```
JobProgress$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of JobProgress

Method `validateJSON()`**:** Validate JSON input with respect to JobProgress and throw an exception if invalid

Usage:

```
JobProgress$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
JobProgress$toString()
```

Returns: String representation of JobProgress

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
JobProgress$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
JobProgress$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
JobProgress$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `JobProgress$toList`
## ------------------------------------------------

# convert array of JobProgress (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`JobsApi` *Jobs operations*

Description

JobsApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- JobsApi$new()
- JobsApi$DeleteJob()
- JobsApi$DeleteJobWithHttpInfo()
- JobsApi$DeleteJobConfig()
- JobsApi$DeleteJobConfigWithHttpInfo()
- JobsApi$DeleteJobs()
- JobsApi$DeleteJobsWithHttpInfo()
- JobsApi$GetCommand()
- JobsApi$GetCommandWithHttpInfo()
- JobsApi$GetDefaultJobConfig()
- JobsApi$GetDefaultJobConfigWithHttpInfo()
- JobsApi$GetJob()
- JobsApi$GetJobWithHttpInfo()
- JobsApi$GetJobConfig()
- JobsApi$GetJobConfigWithHttpInfo()
- JobsApi$GetJobConfigNames()
- JobsApi$GetJobConfigNamesWithHttpInfo()
- JobsApi$GetJobConfigs()
- JobsApi$GetJobConfigsWithHttpInfo()
- JobsApi$GetJobs()
- JobsApi$GetJobsWithHttpInfo()
- JobsApi$GetJobsPage()
- JobsApi$GetJobsPageWithHttpInfo()
- JobsApi$HasJobs()
- JobsApi$HasJobsWithHttpInfo()
- JobsApi$SaveJobConfig()
- JobsApi$SaveJobConfigWithHttpInfo()
- JobsApi$StartJob()
- JobsApi$StartJobWithHttpInfo()

- JobsApi$StartJobFromConfig()
- JobsApi$StartJobFromConfigWithHttpInfo()
- JobsApi$clone()

Method `new()`**:** Initialize a new JobsApi.

Usage:

```
JobsApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `DeleteJob()`**:** Delete job.

Usage:

```
JobsApi$DeleteJob(
  project_id,
  job_id,
  cancel_if_running = TRUE,
  await_deletion = TRUE,
  ...
)
```

Arguments:

`project_id` project-space to delete job from

`job_id` of the job to be deleted

`cancel_if_running` (optional) If true, job will be canceled if it is not finished. Otherwise, deletion will fail for running jobs or request will block until job has finished. (default value: TRUE)

`await_deletion` (optional) If true, request will block until deletion succeeded or failed. If the job is still running the request will wait until the job has finished. (default value: TRUE)

`...` Other optional arguments

Returns: void

Method `DeleteJobWithHttpInfo()`**:** Delete job.

Usage:

```
JobsApi$DeleteJobWithHttpInfo(
  project_id,
  job_id,
  cancel_if_running = TRUE,
  await_deletion = TRUE,
  ...
)
```

Arguments:

`project_id` project-space to delete job from

`job_id` of the job to be deleted

`cancel_if_running` (optional) If true, job will be canceled if it is not finished. Otherwise, deletion will fail for running jobs or request will block until job has finished. (default value: TRUE)

`await_deletion` (optional) If true, request will block until deletion succeeded or failed. If the job is still running the request will wait until the job has finished. (default value: TRUE)

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `DeleteJobConfig()`**:** Delete job configuration with given name.

Usage:

```
JobsApi$DeleteJobConfig(name, ...)
```

Arguments:

`name` name of the job-config to delete

`...` Other optional arguments

Returns: void

Method `DeleteJobConfigWithHttpInfo()`**:** Delete job configuration with given name.

Usage:

```
JobsApi$DeleteJobConfigWithHttpInfo(name, ...)
```

Arguments:

`name` name of the job-config to delete

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `DeleteJobs()`**:** Delete ALL jobs.

Usage:

```
JobsApi$DeleteJobs(
  project_id,
  cancel_if_running = TRUE,
  await_deletion = TRUE,
  ...
)
```

Arguments:

`project_id` project-space to delete jobs from

`cancel_if_running` (optional) If true, job will be canceled if it is not finished. Otherwise, deletion will fail for running jobs or request will block until job has finished. (default value: TRUE)

`await_deletion` (optional) If true, request will block until deletion succeeded or failed. If the job is still running the request will wait until the job has finished. (default value: TRUE)

`...` Other optional arguments

Returns: void

Method `DeleteJobsWithHttpInfo()`**:** Delete ALL jobs.

Usage:

```
JobsApi$DeleteJobsWithHttpInfo(
  project_id,
  cancel_if_running = TRUE,
  await_deletion = TRUE,
  ...
)
```

Arguments:

`project_id` project-space to delete jobs from

`cancel_if_running` (optional) If true, job will be canceled if it is not finished. Otherwise, deletion will fail for running jobs or request will block until job has finished. (default value: TRUE)

`await_deletion` (optional) If true, request will block until deletion succeeded or failed. If the job is still running the request will wait until the job has finished. (default value: TRUE)

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `GetCommand()`**:** Get a CLI command for the given job configuration.

Usage:

```
JobsApi$GetCommand(job_submission, data_file = NULL, ...)
```

Arguments:

`job_submission`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[character]

Method `GetCommandWithHttpInfo()`**:** Get a CLI command for the given job configuration.

Usage:

```
JobsApi$GetCommandWithHttpInfo(job_submission, data_file = NULL, ...)
```

Arguments:

`job_submission`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[character]) with additional information such as HTTP status code, headers

Method `GetDefaultJobConfig()`**:** Request default job configuration

Usage:

```
JobsApi$GetDefaultJobConfig(
  include_config_map = FALSE,
  move_parameters_to_config_map = FALSE,
  include_custom_dbs_for_structure_search = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_config_map` (optional) if true, generic configmap with-defaults will be included (default value: FALSE)

`move_parameters_to_config_map` (optional) if true, object-based parameters will be converted to and added to the generic configMap parameters (default value: FALSE)

`include_custom_dbs_for_structure_search` (optional) if true, default database selection of structure db search and spectral library search also contains all available custom databases. If no custom databases are selected, spectral library search is disabled by default. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: JobSubmission

Method `GetDefaultJobConfigWithHttpInfo()`**:** Request default job configuration

Usage:

```
JobsApi$GetDefaultJobConfigWithHttpInfo(
  include_config_map = FALSE,
  move_parameters_to_config_map = FALSE,
  include_custom_dbs_for_structure_search = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_config_map` (optional) if true, generic configmap with-defaults will be included (default value: FALSE)

`move_parameters_to_config_map` (optional) if true, object-based parameters will be converted to and added to the generic configMap parameters (default value: FALSE)

`include_custom_dbs_for_structure_search` (optional) if true, default database selection of structure db search and spectral library search also contains all available custom databases. If no custom databases are selected, spectral library search is disabled by default. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (JobSubmission) with additional information such as HTTP status code, headers

Method `GetJob()`**:** Get job information and its current state and progress (if available).

Usage:

```
JobsApi$GetJob(
  project_id,
  job_id,
  opt_fields = list("progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`job_id` id of the job to be returned

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Job

Method `GetJobWithHttpInfo()`**:** Get job information and its current state and progress (if available).

Usage:

```
JobsApi$GetJobWithHttpInfo(
  project_id,
  job_id,
  opt_fields = list("progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`job_id` id of the job to be returned

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Job) with additional information such as HTTP status code, headers

Method `GetJobConfig()`**:** Request job configuration with given name.

Usage:

```
JobsApi$GetJobConfig(
  name,
  move_parameters_to_config_map = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`name` name of the job-config to return

`move_parameters_to_config_map` (optional) if true, object-based parameters will be converted to and added to the generic configMap parameters (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: StoredJobSubmission

Method `GetJobConfigWithHttpInfo()`**:** Request job configuration with given name.

Usage:

```
JobsApi$GetJobConfigWithHttpInfo(
  name,
  move_parameters_to_config_map = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`name` name of the job-config to return

`move_parameters_to_config_map` (optional) if true, object-based parameters will be converted to and added to the generic configMap parameters (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (StoredJobSubmission) with additional information such as HTTP status code, headers

Method `GetJobConfigNames()`**:** [DEPRECATED] Get all (non-default) job configuration names

Usage:

```
JobsApi$GetJobConfigNames(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[character]

Method `GetJobConfigNamesWithHttpInfo()`**:** [DEPRECATED] Get all (non-default) job configuration names

Usage:

```
JobsApi$GetJobConfigNamesWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[character]) with additional information such as HTTP status code, headers

Method `GetJobConfigs()`**:** Request all available job configurations

Usage:

```
JobsApi$GetJobConfigs(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[StoredJobSubmission]

Method `GetJobConfigsWithHttpInfo()`**:** Request all available job configurations

Usage:

```
JobsApi$GetJobConfigsWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[StoredJobSubmission]) with additional information such as HTTP status code, headers

Method `GetJobs()`**:** [DEPRECATED] Get list of all available jobs with information such as current state and progress (if available)

Usage:

```
JobsApi$GetJobs(project_id, opt_fields = list("none"), data_file = NULL, ...)
```

Arguments:

`project_id` project-space to run jobs on

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[Job]

Method `GetJobsWithHttpInfo()`**:** [DEPRECATED] Get list of all available jobs with information such as current state and progress (if available)

Usage:

```
JobsApi$GetJobsWithHttpInfo(
  project_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[Job]) with additional information such as HTTP status code, headers

Method `GetJobsPage()`**:** Get Page of jobs with information such as current state and progress (if available).

Usage:

```
JobsApi$GetJobsPage(
  project_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelJob

Method `GetJobsPageWithHttpInfo()`**:** Get Page of jobs with information such as current state and progress (if available).

Usage:

```
JobsApi$GetJobsPageWithHttpInfo(
  project_id,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelJob) with additional information such as HTTP status code, headers

Method `HasJobs()`**:**

Usage:

```
JobsApi$HasJobs(project_id, include_finished = FALSE, data_file = NULL, ...)
```

Arguments:

`project_id`

`include_finished` (optional) No description (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `HasJobsWithHttpInfo()`**:**

Usage:

```
JobsApi$HasJobsWithHttpInfo(
  project_id,
  include_finished = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id`

`include_finished` (optional) No description (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `SaveJobConfig()`**:** Add new job configuration with given name.

Usage:

```
JobsApi$SaveJobConfig(
  name,
  job_submission,
  override_existing = FALSE,
  move_parameters_to_config_map = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`name` name of the job-config to add

`job_submission` to add

`override_existing` (optional) No description (default value: FALSE)

`move_parameters_to_config_map` (optional) if true, object-based parameters will be converted to and added to the generic configMap parameters in the return object (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: StoredJobSubmission

Method `SaveJobConfigWithHttpInfo()`**:** Add new job configuration with given name.

Usage:

```
JobsApi$SaveJobConfigWithHttpInfo(
  name,
  job_submission,
  override_existing = FALSE,
  move_parameters_to_config_map = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`name` name of the job-config to add

`job_submission` to add

`override_existing` (optional) No description (default value: FALSE)

`move_parameters_to_config_map` (optional) if true, object-based parameters will be converted to and added to the generic configMap parameters in the return object (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (StoredJobSubmission) with additional information such as HTTP status code, headers

Method `StartJob()`**:** Start computation for given compounds and with given parameters.

Usage:

```
JobsApi$StartJob(
  project_id,
  job_submission,
  opt_fields = list("command", "progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`job_submission` configuration of the job to be submitted

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["command","progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Job

Method `StartJobWithHttpInfo()`**:** Start computation for given compounds and with given parameters.

Usage:

```
JobsApi$StartJobWithHttpInfo(
  project_id,
  job_submission,
  opt_fields = list("command", "progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`job_submission` configuration of the job to be submitted

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["command","progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Job) with additional information such as HTTP status code, headers

Method `StartJobFromConfig()`**:** Start computation for given compounds and with parameters from a stored job-config.

Usage:

```
JobsApi$StartJobFromConfig(
  project_id,
  job_config_name,
  request_body,
  recompute = NULL,
  opt_fields = list("command", "progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`job_config_name` name of the config to be used

`request_body` List of alignedFeatureIds to be computed

`recompute` (optional) enable or disable recompute. If null the stored value will be used.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["command","progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Job

Method `StartJobFromConfigWithHttpInfo()`**:** Start computation for given compounds and with parameters from a stored job-config.

Usage:

```
JobsApi$StartJobFromConfigWithHttpInfo(
  project_id,
  job_config_name,
  request_body,
  recompute = NULL,
  opt_fields = list("command", "progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to run jobs on

`job_config_name` name of the config to be used

`request_body` List of alignedFeatureIds to be computed

`recompute` (optional) enable or disable recompute. If null the stored value will be used.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["command","progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Job) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  DeleteJob  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to delete job from
var_job_id <- "job_id_example" # character | of the job to be deleted
var_cancel_if_running <- TRUE # character | If true, job will be canceled if it is not finished. Otherwise,
var_await_deletion <- TRUE # character | If true, request will block until deletion succeeded or failed.

#Delete job.
api_instance <- rsirius_api$new()

api_instance$jobs_api$DeleteJob(var_project_id, var_job_id, cancel_if_running = var_cancel_if_running, await_del


####################  DeleteJobConfig  ####################

library(RSirius)
```

```
var_name <- "name_example" # character | name of the job-config to delete

#Delete job configuration with given name.
api_instance <- rsirius_api$new()

api_instance$jobs_api$DeleteJobConfig(var_name)


####################  DeleteJobs  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to delete jobs from
var_cancel_if_running <- TRUE # character | If true, job will be canceled if it is not finished. Otherwise,
var_await_deletion <- TRUE # character | If true, request will block until deletion succeeded or failed.

#Delete ALL jobs.
api_instance <- rsirius_api$new()

api_instance$jobs_api$DeleteJobs(var_project_id, cancel_if_running = var_cancel_if_running, await_deletion = var


####################  GetCommand  ####################

library(RSirius)
var_job_submission <- JobSubmission$new(c("fallbackAdducts_example"), c("enforcedAdducts_example"), c("detectabl

#Get a CLI command for the given job configuration.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCommand(var_job_submissiondata_file = "result.txt")
result <- api_instance$jobs_api$GetCommand(var_job_submission)
dput(result)


####################  GetDefaultJobConfig  ####################

library(RSirius)
var_include_config_map <- FALSE # character | if true, generic configmap with-defaults will be included (Optional)
var_move_parameters_to_config_map <- FALSE # character | if true, object-based parameters will be converted to and
var_include_custom_dbs_for_structure_search <- FALSE # character | if true, default database selection of structur

#Request default job configuration
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDefaultJobConfig(include_config_map = var_include_config_map, move_parameters_to_con
result <- api_instance$jobs_api$GetDefaultJobConfig(include_config_map = var_include_config_map, move_parameters
dput(result)


####################  GetJob  ####################
```

```
library(RSirius)
var_project_id <- "project_id_example" # character | project-space to run jobs on
var_job_id <- "job_id_example" # character | id of the job to be returned
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Get job information and its current state and progress (if available).
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetJob(var_project_id, var_job_id, opt_fields = var_opt_fieldsdata_file = "result.txt")
result <- api_instance$jobs_api$GetJob(var_project_id, var_job_id, opt_fields = var_opt_fields)
dput(result)


####################  GetJobConfig  ####################

library(RSirius)
var_name <- "name_example" # character | name of the job-config to return
var_move_parameters_to_config_map <- FALSE # character | if true, object-based parameters will be converted to and

#Request job configuration with given name.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetJobConfig(var_name, move_parameters_to_config_map = var_move_parameters_to_config_ma
result <- api_instance$jobs_api$GetJobConfig(var_name, move_parameters_to_config_map = var_move_parameters_to_co
dput(result)


####################  GetJobConfigNames  ####################

library(RSirius)

#[DEPRECATED] Get all (non-default) job configuration names
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetJobConfigNames(data_file = "result.txt")
result <- api_instance$jobs_api$GetJobConfigNames()
dput(result)


####################  GetJobConfigs  ####################

library(RSirius)

#Request all available job configurations
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetJobConfigs(data_file = "result.txt")
result <- api_instance$jobs_api$GetJobConfigs()
dput(result)
```

```
####################  GetJobs  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to run jobs on
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#[DEPRECATED] Get list of all available jobs with information such as current state and progress (if available)
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetJobs(var_project_id, opt_fields = var_opt_fieldsdata_file = "result.txt")
result <- api_instance$jobs_api$GetJobs(var_project_id, opt_fields = var_opt_fields)
dput(result)


####################  GetJobsPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to run jobs on
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Get Page of jobs with information such as current state and progress (if available).
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetJobsPage(var_project_id, page = var_page, size = var_size, sort = var_sort, opt_fields
result <- api_instance$jobs_api$GetJobsPage(var_project_id, page = var_page, size = var_size, sort = var_sort, opt
dput(result)


####################  HasJobs  ####################

library(RSirius)
var_project_id <- "project_id_example" # character |
var_include_finished <- FALSE # character |  (Optional)

api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$HasJobs(var_project_id, include_finished = var_include_finisheddata_file = "result.txt")
result <- api_instance$jobs_api$HasJobs(var_project_id, include_finished = var_include_finished)
dput(result)


####################  SaveJobConfig  ####################

library(RSirius)
var_name <- "name_example" # character | name of the job-config to add
```

```
var_job_submission <- JobSubmission$new(c("fallbackAdducts_example"), c("enforcedAdducts_example"), c("detectabl
var_override_existing <- FALSE # character |  (Optional)
var_move_parameters_to_config_map <- FALSE # character | if true, object-based parameters will be converted to and

#Add new job configuration with given name.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$SaveJobConfig(var_name, var_job_submission, override_existing = var_override_existing, m
result <- api_instance$jobs_api$SaveJobConfig(var_name, var_job_submission, override_existing = var_override_exi
dput(result)


####################  StartJob  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to run jobs on
var_job_submission <- JobSubmission$new(c("fallbackAdducts_example"), c("enforcedAdducts_example"), c("detectabl
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Start computation for given compounds and with given parameters.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$StartJob(var_project_id, var_job_submission, opt_fields = var_opt_fieldsdata_file = "res
result <- api_instance$jobs_api$StartJob(var_project_id, var_job_submission, opt_fields = var_opt_fields)
dput(result)


####################  StartJobFromConfig  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to run jobs on
var_job_config_name <- "job_config_name_example" # character | name of the config to be used
var_request_body <- c("property_example") # array[character] | List of alignedFeatureIds to be computed
var_recompute <- "recompute_example" # character | enable or disable recompute. If null the stored value will be use
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Start computation for given compounds and with parameters from a stored job-config.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$StartJobFromConfig(var_project_id, var_job_config_name, var_request_body, recompute = va
result <- api_instance$jobs_api$StartJobFromConfig(var_project_id, var_job_config_name, var_request_body, recomp
dput(result)



## End(Not run)
```

JobSubmission *JobSubmission*

Description

Object to submit a job to be executed by SIRIUS

JobSubmission Class

Format

An `R6Class` generator object

Details

Create a new JobSubmission

Public fields

`fallbackAdducts` Fallback adducts are considered if the auto-detection did not find any indication for an ion mode. Pos Examples: [M+H]+,[M]+,[M+K]+,[M+Na]+,[M+H-H2O]+,[M+Na2-H]+,[M+2K-H]+,[M+NH4]+,[M+H3O]+,[M+MeOH+H]+,[M+ACN+H]+,[M+2ACN+H]+,[M+IPA+H]+,[M+ACN+N
Neg Examples: [M-H]-,[M]-,[M+K-2H]-,[M+Cl]-,[M-H2O-H]-,[M+Na-2H]-,[M+FA-H]-,[M+Br]-,[M+HAc-H]-,[M+TFA-H]-,[M+ACN-H]- list(character) [optional]

`enforcedAdducts` Enforced adducts that are always considered. Pos Examples: [M+H]+,[M]+,[M+K]+,[M+Na]+,[M+H-H2O]+,[M+Na2-H]+,[M+2K-H]+,[M+NH4]+,[M+H3O]+,[M+MeOH+H]+,[M+ACN+H]+,[M+2ACN+H]+,[M+IPA+
Neg Examples: [M-H]-,[M]-,[M+K-2H]-,[M+Cl]-,[M-H2O-H]-,[M+Na-2H]-,[M+FA-H]-,[M+Br]-,[M+HAc-H]-,[M+TFA-H]-,[M+ACN-H]- list(character) [optional]

`detectableAdducts` Detectable adducts which are only considered if there is an indication in the MS1 scan (e.g. correct mass delta). Pos Examples: [M+H]+,[M]+,[M+K]+,[M+Na]+,[M+H-H2O]+,[M+Na2-H]+,[M+2K-H]+,[M+NH4]+,[M+H3O]+,[M+MeOH+H]+,[M+ACN+H]+,[M+2ACN+H]+,[M+IPA+
Neg Examples: [M-H]-,[M]-,[M+K-2H]-,[M+Cl]-,[M-H2O-H]-,[M+Na-2H]-,[M+FA-H]-,[M+Br]-,[M+HAc-H]-,[M+TFA-H]-,[M+ACN-H]- list(character) [optional]

`recompute` Indicate if already existing result for a tool to be executed should be overwritten or not. character [optional]

`spectraSearchParams` SpectralLibrarySearch [optional]

`formulaIdParams` Sirius [optional]

`zodiacParams` Zodiac [optional]

`fingerprintPredictionParams` FingerprintPrediction [optional]

`canopusParams` Canopus [optional]

`structureDbSearchParams` StructureDbSearch [optional]

`msNovelistParams` MsNovelist [optional]

`configMap` As an alternative to the object based parameters, this map allows to store key value pairs of ALL SIRIUS parameters. All possible parameters can be retrieved from SIRIUS via the respective endpoint. named list(character) [optional]

`compoundIds` Compounds that should be the input for this Job Will be converted to the respective alignedFeatureIds for computation. At least one compoundId or alignedFeatureId needs to be specified. list(character) [optional]

`alignedFeatureIds` Features (aligned over runs) that should be the input for this Job At least one compoundId or alignedFeatureId needs to be specified. list(character) [optional]

Methods

Public methods:

- JobSubmission$new()
- JobSubmission$toJSON()
- JobSubmission$toList()
- JobSubmission$toSimpleType()
- JobSubmission$fromJSON()
- JobSubmission$toJSONString()
- JobSubmission$fromJSONString()
- JobSubmission$validateJSON()
- JobSubmission$toString()
- JobSubmission$isValid()
- JobSubmission$getInvalidFields()
- JobSubmission$print()
- JobSubmission$clone()

Method `new()`**:** Initialize a new JobSubmission class.

Usage:

```
JobSubmission$new(
  fallbackAdducts = NULL,
  enforcedAdducts = NULL,
  detectableAdducts = NULL,
  recompute = NULL,
  spectraSearchParams = NULL,
  formulaIdParams = NULL,
  zodiacParams = NULL,
  fingerprintPredictionParams = NULL,
  canopusParams = NULL,
  structureDbSearchParams = NULL,
  msNovelistParams = NULL,
  configMap = NULL,
  compoundIds = NULL,
  alignedFeatureIds = NULL,
  ...
)
```

Arguments:

`fallbackAdducts` Fallback adducts are considered if the auto-detection did not find any indication for an ion mode. Pos Examples: [M+H]+,[M]+,[M+K]+,[M+Na]+,[M+H-H2O]+,[M+Na2-H]+,[M+2K-H]+,[M+NH4]+,[M+H3O]+,[M+MeOH+H]+,[M+ACN+H]+,[M+2ACN+H]+,[M+IPA+H]+,[M+ACN
Neg Examples: [M-H]-,[M]-,[M+K-2H]-,[M+Cl]-,[M-H2O-H]-,[M+Na-2H]-,[M+FA-H]-,[M+Br]-,[M+HAc-H]-,[M+TFA-H]-,[M+ACN-H]-

`enforcedAdducts` Enforced adducts that are always considered. Pos Examples: [M+H]+,[M]+,[M+K]+,[M+Na]+,[M+H
H2O]+,[M+Na2-H]+,[M+2K-H]+,[M+NH4]+,[M+H3O]+,[M+MeOH+H]+,[M+ACN+H]+,[M+2ACN+H]+,[M+IP
Neg Examples: [M-H]-,[M]-,[M+K-2H]-,[M+Cl]-,[M-H2O-H]-,[M+Na-2H]-,[M+FA-H]-,[M+Br]-,[M+HAc-H]-,[M+TFA-H]-,[M+ACN-H]-

`detectableAdducts` Detectable adducts which are only considered if there is an indication in the MS1 scan (e.g. correct mass delta). Pos Examples: [M+H]+,[M]+,[M+K]+,[M+Na]+,[M+H-H2O]+,[M+Na2-H]+,[M+2K-H]+,[M+NH4]+,[M+H3O]+,[M+MeOH+H]+,[M+ACN+H]+,[M+2ACN+H]+,[M+IP
Neg Examples: [M-H]-,[M]-,[M+K-2H]-,[M+Cl]-,[M-H2O-H]-,[M+Na-2H]-,[M+FA-H]-,[M+Br]-,[M+HAc-H]-,[M+TFA-H]-,[M+ACN-H]-

`recompute` Indicate if already existing result for a tool to be executed should be overwritten or not.

`spectraSearchParams` spectraSearchParams

`formulaIdParams` formulaIdParams

`zodiacParams` zodiacParams

`fingerprintPredictionParams` fingerprintPredictionParams

`canopusParams` canopusParams

`structureDbSearchParams` structureDbSearchParams

`msNovelistParams` msNovelistParams

`configMap` As an alternative to the object based parameters, this map allows to store key value pairs of ALL SIRIUS parameters. All possible parameters can be retrieved from SIRIUS via the respective endpoint.

`compoundIds` Compounds that should be the input for this Job Will be converted to the respective alignedFeatureIds for computation. At least one compoundId or alignedFeatureId needs to be specified.

`alignedFeatureIds` Features (aligned over runs) that should be the input for this Job At least one compoundId or alignedFeatureId needs to be specified.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
JobSubmission$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
JobSubmission$toList()
```

Returns: JobSubmission as a base R list.

Examples:

```
# convert array of JobSubmission (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert JobSubmission to a base R type

Usage:

```
JobSubmission$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of JobSubmission

Usage:

```
JobSubmission$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of JobSubmission

Method `toJSONString()`**:** To JSON String

Usage:

```
JobSubmission$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: JobSubmission in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of JobSubmission

Usage:

```
JobSubmission$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of JobSubmission

Method `validateJSON()`**:** Validate JSON input with respect to JobSubmission and throw an exception if invalid

Usage:

```
JobSubmission$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
JobSubmission$toString()
```

Returns: String representation of JobSubmission

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
JobSubmission$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
JobSubmission$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
JobSubmission$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `JobSubmission$toList`
## ------------------------------------------------

# convert array of JobSubmission (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`LcmsSubmissionParameters` *LcmsSubmissionParameters*

Description

LcmsSubmissionParameters Class

LcmsSubmissionParameters Class

Format

An `R6Class` generator object

Details

Create a new LcmsSubmissionParameters

Public fields

`sampleNames` Sample names for each input file to link imported results, e.g. QuantTable back to the input data. If NULL or empty sample names will be derived from the input files. <p> The names are matched to the input files by index. Partial lists are allowed: a NULL entry and any input file without a corresponding entry get their name derived from the input file. Surplus entries that match no input file are ignored. <p> Names must neither be blank nor duplicated, otherwise the import is rejected. list(character) [optional]

`sampleTypes` Sample type for each input file to be used to compute fold changes between blank and sample runs If NULL or empty no fold changes will be computed during preprocessing. <p> The types are matched to the input files by index. In contrast to sampleNames either all or no sample types have to be given: if the number of types does not match the number of input files or if any type is NULL or blank, the import is rejected. list(character) [optional]

`alignLCMSRuns` Specifies whether LC/MS runs should be aligned character [optional]

`noiseIntensity` Noise level under which all peaks are considered to be likely noise. A peak has to be at least 3x noise level to be picked as feature. Peaks with MS/MS are still picked even though they might be below noise level. If not specified, the noise intensity is detected automatically from the data. We recommend NOT specifying this parameter, as the automated detection is usually sufficient. numeric [optional]

`traceMaxMassDeviation` Deviation [optional]

`alignMaxMassDeviation` Deviation [optional]

`alignMaxRetentionTimeDeviation` Maximum allowed retention time error in seconds for aligning features. If not specified, this parameter is estimated from the data. numeric [optional]

`minSNR` Minimum ratio between peak height and noise intensity for detecting features. By default, this value is 3. Features with good MS/MS are always picked independent of their intensity. For picking very low intensity features we recommend a min-snr of 2, but this will increase runtime and storage requirements numeric [optional]

Methods

Public methods:

- `LcmsSubmissionParameters$new()`
- `LcmsSubmissionParameters$toJSON()`
- `LcmsSubmissionParameters$toList()`
- `LcmsSubmissionParameters$toSimpleType()`
- `LcmsSubmissionParameters$fromJSON()`
- `LcmsSubmissionParameters$toJSONString()`
- `LcmsSubmissionParameters$fromJSONString()`
- `LcmsSubmissionParameters$validateJSON()`
- `LcmsSubmissionParameters$toString()`
- `LcmsSubmissionParameters$isValid()`
- `LcmsSubmissionParameters$getInvalidFields()`
- `LcmsSubmissionParameters$print()`
- `LcmsSubmissionParameters$clone()`

Method `new()`**:** Initialize a new LcmsSubmissionParameters class.

Usage:

```
LcmsSubmissionParameters$new(
  sampleNames = NULL,
  sampleTypes = NULL,
  alignLCMSRuns = TRUE,
  noiseIntensity = -1,
  traceMaxMassDeviation = NULL,
  alignMaxMassDeviation = NULL,
  alignMaxRetentionTimeDeviation = -1,
  minSNR = 3,
  ...
)
```

Arguments:

`sampleNames` Sample names for each input file to link imported results, e.g. QuantTable back to the input data. If NULL or empty sample names will be derived from the input files. <p> The names are matched to the input files by index. Partial lists are allowed: a NULL entry and any input file without a corresponding entry get their name derived from the input file. Surplus entries that match no input file are ignored. <p> Names must neither be blank nor duplicated, otherwise the import is rejected.

`sampleTypes` Sample type for each input file to be used to compute fold changes between blank and sample runs If NULL or empty no fold changes will be computed during preprocessing. <p> The types are matched to the input files by index. In contrast to sampleNames either all or no sample types have to be given: if the number of types does not match the number of input files or if any type is NULL or blank, the import is rejected.

`alignLCMSRuns` Specifies whether LC/MS runs should be aligned. Default to TRUE.

`noiseIntensity` Noise level under which all peaks are considered to be likely noise. A peak has to be at least 3x noise level to be picked as feature. Peaks with MS/MS are still picked even though they might be below noise level. If not specified, the noise intensity is detected automatically from the data. We recommend NOT specifying this parameter, as the automated detection is usually sufficient.. Default to -1.

`traceMaxMassDeviation` traceMaxMassDeviation

`alignMaxMassDeviation` alignMaxMassDeviation

`alignMaxRetentionTimeDeviation` Maximum allowed retention time error in seconds for aligning features. If not specified, this parameter is estimated from the data.. Default to -1.

`minSNR` Minimum ratio between peak height and noise intensity for detecting features. By default, this value is 3. Features with good MS/MS are always picked independent of their intensity. For picking very low intensity features we recommend a min-snr of 2, but this will increase runtime and storage requirements. Default to 3.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
LcmsSubmissionParameters$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
LcmsSubmissionParameters$toList()
```

Returns: LcmsSubmissionParameters as a base R list.

Examples:

```
# convert array of LcmsSubmissionParameters (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert LcmsSubmissionParameters to a base R type

Usage:

```
LcmsSubmissionParameters$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of LcmsSubmissionParameters

Usage:

```
LcmsSubmissionParameters$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LcmsSubmissionParameters

Method `toJSONString()`**:** To JSON String

Usage:

```
LcmsSubmissionParameters$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: LcmsSubmissionParameters in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of LcmsSubmissionParameters

Usage:

```
LcmsSubmissionParameters$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LcmsSubmissionParameters

Method `validateJSON()`**:** Validate JSON input with respect to LcmsSubmissionParameters and throw an exception if invalid

Usage:

```
LcmsSubmissionParameters$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
LcmsSubmissionParameters$toString()
```

Returns: String representation of LcmsSubmissionParameters

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
LcmsSubmissionParameters$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
LcmsSubmissionParameters$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
LcmsSubmissionParameters$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LcmsSubmissionParameters$toList`
## ------------------------------------------------

# convert array of LcmsSubmissionParameters (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`LicenseInfo` *LicenseInfo*

Description

LicenseInfo Class

LicenseInfo Class

Format

An `R6Class` generator object

Details

Create a new LicenseInfo

Public fields

`userEmail` Email address of the user account this license information belongs to. character [optional]

`userId` User ID (uid) of the user account this license information belongs to. character [optional]

`subscription` Subscription [optional]

`consumables` SubscriptionConsumables [optional]

`terms` list(Term) [optional]

Methods

Public methods:

- `LicenseInfo$new()`
- `LicenseInfo$toJSON()`
- `LicenseInfo$toList()`
- `LicenseInfo$toSimpleType()`
- `LicenseInfo$fromJSON()`
- `LicenseInfo$toJSONString()`
- `LicenseInfo$fromJSONString()`
- `LicenseInfo$validateJSON()`
- `LicenseInfo$toString()`
- `LicenseInfo$isValid()`
- `LicenseInfo$getInvalidFields()`
- `LicenseInfo$print()`
- `LicenseInfo$clone()`

Method `new()`**:** Initialize a new LicenseInfo class.

Usage:

```
LicenseInfo$new(
  userEmail = NULL,
  userId = NULL,
  subscription = NULL,
  consumables = NULL,
  terms = NULL,
  ...
)
```

Arguments:

`userEmail` Email address of the user account this license information belongs to.

`userId` User ID (uid) of the user account this license information belongs to.

`subscription` subscription

`consumables` consumables

`terms` terms

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
LicenseInfo$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
LicenseInfo$toList()
```

Returns: LicenseInfo as a base R list.

Examples:

```
# convert array of LicenseInfo (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert LicenseInfo to a base R type

Usage:

```
LicenseInfo$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of LicenseInfo

Usage:

```
LicenseInfo$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LicenseInfo

Method `toJSONString()`**:** To JSON String

Usage:

```
LicenseInfo$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: LicenseInfo in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of LicenseInfo

Usage:

```
LicenseInfo$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LicenseInfo

Method `validateJSON()`**:** Validate JSON input with respect to LicenseInfo and throw an exception if invalid

Usage:

```
LicenseInfo$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
LicenseInfo$toString()
```

Returns: String representation of LicenseInfo

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
LicenseInfo$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
LicenseInfo$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
LicenseInfo$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LicenseInfo$toList`
## ------------------------------------------------

# convert array of LicenseInfo (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`LipidAnnotation` *LipidAnnotation*

Description

LipidAnnotation Class

LipidAnnotation Class

Format

An `R6Class` generator object

Details

Create a new LipidAnnotation

Public fields

`lipidSpecies` Predicted lipid species in LIPID MAPS notation. NULL if not classified as lipid. character [optional]

`lipidMapsId` LIPID MAPS id of the predicted lipid class. character [optional]

`lipidClassName` Human-readable name of the predicted lipid class. character [optional]

`hypotheticalStructure` Hypothetical molecular structure of the predicted lipid species as SMILES. NULL if hypothetical structure not available. character [optional]

`chainsUnknown` True if the formula composition of the chains could not be determined from the MS/MS. character [optional]

Methods

Public methods:

- LipidAnnotation$new()
- LipidAnnotation$toJSON()
- LipidAnnotation$toList()
- LipidAnnotation$toSimpleType()
- LipidAnnotation$fromJSON()
- LipidAnnotation$toJSONString()
- LipidAnnotation$fromJSONString()
- LipidAnnotation$validateJSON()
- LipidAnnotation$toString()
- LipidAnnotation$isValid()
- LipidAnnotation$getInvalidFields()
- LipidAnnotation$print()
- LipidAnnotation$clone()

Method `new()`**:** Initialize a new LipidAnnotation class.

Usage:

```
LipidAnnotation$new(
  lipidSpecies = NULL,
  lipidMapsId = NULL,
  lipidClassName = NULL,
  hypotheticalStructure = NULL,
  chainsUnknown = NULL,
  ...
)
```

Arguments:

`lipidSpecies` Predicted lipid species in LIPID MAPS notation. NULL if not classified as lipid.

`lipidMapsId` LIPID MAPS id of the predicted lipid class.

`lipidClassName` Human-readable name of the predicted lipid class.

`hypotheticalStructure` Hypothetical molecular structure of the predicted lipid species as SMILES. NULL if hypothetical structure not available.

`chainsUnknown` True if the formula composition of the chains could not be determined from the MS/MS.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
LipidAnnotation$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
LipidAnnotation$toList()
```

Returns: LipidAnnotation as a base R list.

Examples:

```
# convert array of LipidAnnotation (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert LipidAnnotation to a base R type

Usage:

```
LipidAnnotation$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of LipidAnnotation

Usage:

```
LipidAnnotation$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LipidAnnotation

Method `toJSONString()`**:** To JSON String

Usage:

```
LipidAnnotation$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: LipidAnnotation in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of LipidAnnotation

Usage:

```
LipidAnnotation$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LipidAnnotation

Method `validateJSON()`**:** Validate JSON input with respect to LipidAnnotation and throw an exception if invalid

Usage:

```
LipidAnnotation$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
LipidAnnotation$toString()
```

Returns: String representation of LipidAnnotation

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
LipidAnnotation$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
LipidAnnotation$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
LipidAnnotation$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LipidAnnotation$toList`
## ------------------------------------------------

# convert array of LipidAnnotation (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`LoginAndAccountApi` *LoginAndAccount operations*

Description

LoginAndAccountApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `LoginAndAccountApi$new()`
- `LoginAndAccountApi$GetAccountInfo()`
- `LoginAndAccountApi$GetAccountInfoWithHttpInfo()`
- `LoginAndAccountApi$GetSignUpURL()`
- `LoginAndAccountApi$GetSignUpURLWithHttpInfo()`
- `LoginAndAccountApi$GetSubscriptions()`
- `LoginAndAccountApi$GetSubscriptionsWithHttpInfo()`
- `LoginAndAccountApi$IsLoggedIn()`
- `LoginAndAccountApi$IsLoggedInWithHttpInfo()`
- `LoginAndAccountApi$Login()`
- `LoginAndAccountApi$LoginWithHttpInfo()`
- `LoginAndAccountApi$Logout()`
- `LoginAndAccountApi$LogoutWithHttpInfo()`
- `LoginAndAccountApi$OpenPortal()`
- `LoginAndAccountApi$OpenPortalWithHttpInfo()`
- `LoginAndAccountApi$SelectSubscription()`
- `LoginAndAccountApi$SelectSubscriptionWithHttpInfo()`
- `LoginAndAccountApi$SignUp()`

- LoginAndAccountApi$SignUpWithHttpInfo()
- LoginAndAccountApi$clone()

Method `new()`**:** Initialize a new LoginAndAccountApi.

Usage:

```
LoginAndAccountApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `GetAccountInfo()`**:** Get information about the account currently logged in.

Usage:

```
LoginAndAccountApi$GetAccountInfo(include_subs = FALSE, data_file = NULL, ...)
```

Arguments:

`include_subs` (optional) include available and active subscriptions in <code>AccountInfo</code>. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: AccountInfo

Method `GetAccountInfoWithHttpInfo()`**:** Get information about the account currently logged in.

Usage:

```
LoginAndAccountApi$GetAccountInfoWithHttpInfo(
  include_subs = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_subs` (optional) include available and active subscriptions in <code>AccountInfo</code>. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (AccountInfo) with additional information such as HTTP status code, headers

Method `GetSignUpURL()`**:** Get SignUp URL (For signUp via web browser)

Usage:

```
LoginAndAccountApi$GetSignUpURL(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `GetSignUpURLWithHttpInfo()`**:** Get SignUp URL (For signUp via web browser)

Usage:

```
LoginAndAccountApi$GetSignUpURLWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `GetSubscriptions()`**:** Get available subscriptions of the account currently logged in.

Usage:

```
LoginAndAccountApi$GetSubscriptions(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[Subscription]

Method `GetSubscriptionsWithHttpInfo()`**:** Get available subscriptions of the account currently logged in.

Usage:

```
LoginAndAccountApi$GetSubscriptionsWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[Subscription]) with additional information such as HTTP status code, headers

Method `IsLoggedIn()`**:** Check if a user is logged in.

Usage:

```
LoginAndAccountApi$IsLoggedIn(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `IsLoggedInWithHttpInfo()`**:** Check if a user is logged in.

Usage:

```
LoginAndAccountApi$IsLoggedInWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `Login()`**:** Login into SIRIUS web services and activate default subscription if available.

Usage:

```
LoginAndAccountApi$Login(
  accept_terms,
  account_credentials,
  fail_when_logged_in = FALSE,
  include_subs = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`accept_terms`

`account_credentials` used to log in.

`fail_when_logged_in` (optional) if true request fails if an active login already exists. (default value: FALSE)

`include_subs` (optional) include available and active subscriptions in <code>AccountInfo</code>. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: AccountInfo

Method `LoginWithHttpInfo()`**:** Login into SIRIUS web services and activate default subscription if available.

Usage:

```
LoginAndAccountApi$LoginWithHttpInfo(
  accept_terms,
  account_credentials,
  fail_when_logged_in = FALSE,
  include_subs = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`accept_terms`

`account_credentials` used to log in.

`fail_when_logged_in` (optional) if true request fails if an active login already exists. (default value: FALSE)

`include_subs` (optional) include available and active subscriptions in <code>AccountInfo</code>. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (AccountInfo) with additional information such as HTTP status code, headers

Method `Logout()`**:** Logout from SIRIUS web services.

Usage:

```
LoginAndAccountApi$Logout(...)
```

Arguments:

`...` Other optional arguments

Returns: void

Method `LogoutWithHttpInfo()`**:** Logout from SIRIUS web services.

Usage:

```
LoginAndAccountApi$LogoutWithHttpInfo(...)
```

Arguments:

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `OpenPortal()`**:** Open User portal in browser.

Usage:

```
LoginAndAccountApi$OpenPortal(...)
```

Arguments:

`...` Other optional arguments

Returns: void

Method `OpenPortalWithHttpInfo()`**:** Open User portal in browser.

Usage:

```
LoginAndAccountApi$OpenPortalWithHttpInfo(...)
```

Arguments:

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `SelectSubscription()`**:** Select a subscription as active subscription to be used for computations.

Usage:

```
LoginAndAccountApi$SelectSubscription(sid, data_file = NULL, ...)
```

Arguments:

`sid`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: AccountInfo

Method `SelectSubscriptionWithHttpInfo()`**:** Select a subscription as active subscription to be used for computations.

Usage:

```
LoginAndAccountApi$SelectSubscriptionWithHttpInfo(sid, data_file = NULL, ...)
```

Arguments:

`sid`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (AccountInfo) with additional information such as HTTP status code, headers

Method `SignUp()`**:** Open SignUp window in system browser and return signUp link.

Usage:

```
LoginAndAccountApi$SignUp(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `SignUpWithHttpInfo()`**:** Open SignUp window in system browser and return signUp link.

Usage:

```
LoginAndAccountApi$SignUpWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  GetAccountInfo  ####################

library(RSirius)
var_include_subs <- FALSE # character | include available and active subscriptions in <code>AccountInfo</code>. (Op

#Get information about the account currently logged in.
api_instance <- rsirius_api$new()
```

```
# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetAccountInfo(include_subs = var_include_subsdata_file = "result.txt")
result <- api_instance$login_and_account_api$GetAccountInfo(include_subs = var_include_subs)
dput(result)


####################  GetSignUpURL  ####################

library(RSirius)

#Get SignUp URL (For signUp via web browser)
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetSignUpURL(data_file = "result.txt")
result <- api_instance$login_and_account_api$GetSignUpURL()
dput(result)


####################  GetSubscriptions  ####################

library(RSirius)

#Get available subscriptions of the account currently logged in.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetSubscriptions(data_file = "result.txt")
result <- api_instance$login_and_account_api$GetSubscriptions()
dput(result)


####################  IsLoggedIn  ####################

library(RSirius)

#Check if a user is logged in.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$IsLoggedIn(data_file = "result.txt")
result <- api_instance$login_and_account_api$IsLoggedIn()
dput(result)


####################  Login  ####################

library(RSirius)
var_accept_terms <- "accept_terms_example" # character |
var_account_credentials <- AccountCredentials$new("username_example", "password_example", "refreshToken_example"
var_fail_when_logged_in <- FALSE # character | if true request fails if an active login already exists. (Optional)
var_include_subs <- FALSE # character | include available and active subscriptions in <code>AccountInfo</code>. (Op
```

```
#Login into SIRIUS web services and activate default subscription if available.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$Login(var_accept_terms, var_account_credentials, fail_when_logged_in = var_fail_when_log
result <- api_instance$login_and_account_api$Login(var_accept_terms, var_account_credentials, fail_when_logged_i
dput(result)


####################  Logout  ####################

library(RSirius)

#Logout from SIRIUS web services.
api_instance <- rsirius_api$new()

api_instance$login_and_account_api$Logout()


####################  OpenPortal  ####################

library(RSirius)

#Open User portal in browser.
api_instance <- rsirius_api$new()

api_instance$login_and_account_api$OpenPortal()


####################  SelectSubscription  ####################

library(RSirius)
var_sid <- "sid_example" # character |

#Select a subscription as active subscription to be used for computations.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$SelectSubscription(var_siddata_file = "result.txt")
result <- api_instance$login_and_account_api$SelectSubscription(var_sid)
dput(result)


####################  SignUp  ####################

library(RSirius)

#Open SignUp window in system browser and return signUp link.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$SignUp(data_file = "result.txt")
result <- api_instance$login_and_account_api$SignUp()
```

```
dput(result)


## End(Not run)
```

`LossEdge` *LossEdge*

Description

LossEdge Class

LossEdge Class

Format

An `R6Class` generator object

Details

Create a new LossEdge

Public fields

`sourceFragmentIdx` integer [optional]

`targetFragmentIdx` integer [optional]

`molecularFormula` character [optional]

`score` numeric [optional]

Methods

Public methods:

- `LossEdge$new()`
- `LossEdge$toJSON()`
- `LossEdge$toList()`
- `LossEdge$toSimpleType()`
- `LossEdge$fromJSON()`
- `LossEdge$toJSONString()`
- `LossEdge$fromJSONString()`
- `LossEdge$validateJSON()`
- `LossEdge$toString()`
- `LossEdge$isValid()`
- `LossEdge$getInvalidFields()`
- `LossEdge$print()`
- `LossEdge$clone()`

Method `new()`**:** Initialize a new LossEdge class.

Usage:

```
LossEdge$new(
  sourceFragmentIdx = NULL,
  targetFragmentIdx = NULL,
  molecularFormula = NULL,
  score = NULL,
  ...
)
```

Arguments:

`sourceFragmentIdx` sourceFragmentIdx

`targetFragmentIdx` targetFragmentIdx

`molecularFormula` molecularFormula

`score` score

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
LossEdge$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
LossEdge$toList()
```

Returns: LossEdge as a base R list.

Examples:

```
# convert array of LossEdge (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert LossEdge to a base R type

Usage:

```
LossEdge$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of LossEdge

Usage:

```
LossEdge$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LossEdge

Method `toJSONString()`**:** To JSON String

Usage:

```
LossEdge$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: LossEdge in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of LossEdge

Usage:

```
LossEdge$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of LossEdge

Method `validateJSON()`**:** Validate JSON input with respect to LossEdge and throw an exception if invalid

Usage:

```
LossEdge$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
LossEdge$toString()
```

Returns: String representation of LossEdge

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
LossEdge$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
LossEdge$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
LossEdge$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `LossEdge$toList`
## ------------------------------------------------

# convert array of LossEdge (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`MsData` *MsData*

Description

The MsData wraps all spectral input data belonging to a (aligned) feature. All spectra fields are optional. However, at least one Spectrum field needs to be set to create a valid MsData Object. The different types of spectra fields can be extended to adapt to other MassSpec measurement techniques not covered yet. <p> Each Feature can have: - One extracted isotope pattern (optional) - One merged MS/MS spectrum (optional) - One merged MS spectrum (optional) - many MS/MS spectra (optional) - many MS spectra (optional) <p> Each non-merged spectrum has an index which can be used to access the spectrum. <p> In the future we might add some additional information like chromatographic peak or something similar

MsData Class

Format

An `R6Class` generator object

Details

Create a new MsData

Public fields

`isotopePattern` BasicSpectrum [optional]

`mergedMs1` BasicSpectrum [optional]

`mergedMs2` BasicSpectrum [optional]

`ms1Spectra` list(BasicSpectrum) [optional]

`ms2Spectra` list(BasicSpectrum) [optional]

Methods

Public methods:

- MsData$new()
- MsData$toJSON()
- MsData$toList()
- MsData$toSimpleType()
- MsData$fromJSON()
- MsData$toJSONString()
- MsData$fromJSONString()
- MsData$validateJSON()
- MsData$toString()
- MsData$isValid()
- MsData$getInvalidFields()
- MsData$print()
- MsData$clone()

Method `new()`**:** Initialize a new MsData class.

Usage:

```
MsData$new(
  isotopePattern = NULL,
  mergedMs1 = NULL,
  mergedMs2 = NULL,
  ms1Spectra = NULL,
  ms2Spectra = NULL,
  ...
)
```

Arguments:

`isotopePattern` isotopePattern

`mergedMs1` mergedMs1

`mergedMs2` mergedMs2

`ms1Spectra` ms1Spectra

`ms2Spectra` ms2Spectra

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
MsData$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
MsData$toList()
```

Returns: MsData as a base R list.

Examples:

```
# convert array of MsData (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert MsData to a base R type

Usage:

```
MsData$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of MsData

Usage:

```
MsData$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of MsData

Method `toJSONString()`**:** To JSON String

Usage:

```
MsData$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: MsData in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of MsData

Usage:

```
MsData$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of MsData

Method `validateJSON()`**:** Validate JSON input with respect to MsData and throw an exception if invalid

Usage:

```
MsData$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
MsData$toString()
```

Returns: String representation of MsData

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
MsData$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
MsData$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
MsData$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `MsData$toList`
## ------------------------------------------------

# convert array of MsData (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

MsNovelist *MsNovelist*

Description

MsNovelist Class

MsNovelist Class

Format

An R6Class generator object

Details

Create a new MsNovelist

Public fields

`enabled` Indicates whether the tool is enabled. character [optional]

`numberOfCandidateToPredict` Number of structure candidates to be predicted by MsNovelist. Max Value 128. Values > 128 will be set to 128. The actual number of returned candidates might be lower due to duplicates being created by MsNovelist. integer [optional]

Methods

Public methods:

- `MsNovelist$new()`
- `MsNovelist$toJSON()`
- `MsNovelist$toList()`
- `MsNovelist$toSimpleType()`
- `MsNovelist$fromJSON()`
- `MsNovelist$toJSONString()`
- `MsNovelist$fromJSONString()`
- `MsNovelist$validateJSON()`
- `MsNovelist$toString()`
- `MsNovelist$isValid()`
- `MsNovelist$getInvalidFields()`
- `MsNovelist$print()`
- `MsNovelist$clone()`

Method `new()`**:** Initialize a new MsNovelist class.

Usage:

```
MsNovelist$new(enabled = NULL, numberOfCandidateToPredict = NULL, ...)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`numberOfCandidateToPredict` Number of structure candidates to be predicted by MsNovelist. Max Value 128. Values > 128 will be set to 128. The actual number of returned candidates might be lower due to duplicates being created by MsNovelist.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
MsNovelist$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
MsNovelist$toList()
```

Returns: MsNovelist as a base R list.

Examples:

```
# convert array of MsNovelist (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert MsNovelist to a base R type

Usage:

```
MsNovelist$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of MsNovelist

Usage:

```
MsNovelist$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of MsNovelist

Method `toJSONString()`**:** To JSON String

Usage:

```
MsNovelist$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: MsNovelist in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of MsNovelist

Usage:

```
MsNovelist$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of MsNovelist

Method `validateJSON()`**:** Validate JSON input with respect to MsNovelist and throw an exception if invalid

Usage:

```
MsNovelist$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
MsNovelist$toString()
```

Returns: String representation of MsNovelist

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
MsNovelist$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
MsNovelist$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
MsNovelist$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `MsNovelist$toList`
## ------------------------------------------------

# convert array of MsNovelist (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PagedModelAlignedFeature` *PagedModelAlignedFeature*

Description

PagedModelAlignedFeature Class

PagedModelAlignedFeature Class

Format

An `R6Class` generator object

Details

Create a new PagedModelAlignedFeature

Public fields

`content` list(AlignedFeature) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelAlignedFeature$new()`
- `PagedModelAlignedFeature$toJSON()`
- `PagedModelAlignedFeature$toList()`
- `PagedModelAlignedFeature$toSimpleType()`
- `PagedModelAlignedFeature$fromJSON()`
- `PagedModelAlignedFeature$toJSONString()`
- `PagedModelAlignedFeature$fromJSONString()`

- PagedModelAlignedFeature$validateJSON()
- PagedModelAlignedFeature$toString()
- PagedModelAlignedFeature$isValid()
- PagedModelAlignedFeature$getInvalidFields()
- PagedModelAlignedFeature$print()
- PagedModelAlignedFeature$clone()

Method `new()`**:** Initialize a new PagedModelAlignedFeature class.

Usage:

```
PagedModelAlignedFeature$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelAlignedFeature$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelAlignedFeature$toList()
```

Returns: PagedModelAlignedFeature as a base R list.

Examples:

```
# convert array of PagedModelAlignedFeature (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelAlignedFeature to a base R type

Usage:

```
PagedModelAlignedFeature$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelAlignedFeature

Usage:

```
PagedModelAlignedFeature$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelAlignedFeature

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelAlignedFeature$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelAlignedFeature in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelAlignedFeature

Usage:

```
PagedModelAlignedFeature$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelAlignedFeature

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelAlignedFeature and throw an exception if invalid

Usage:

```
PagedModelAlignedFeature$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelAlignedFeature$toString()
```

Returns: String representation of PagedModelAlignedFeature

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelAlignedFeature$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelAlignedFeature$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelAlignedFeature$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelAlignedFeature$toList`
## ------------------------------------------------

# convert array of PagedModelAlignedFeature (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PagedModelCompound` *PagedModelCompound*

Description

PagedModelCompound Class

PagedModelCompound Class

Format

An `R6Class` generator object

Details

Create a new PagedModelCompound

Public fields

`content` list(Compound) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- PagedModelCompound$new()
- PagedModelCompound$toJSON()
- PagedModelCompound$toList()
- PagedModelCompound$toSimpleType()
- PagedModelCompound$fromJSON()
- PagedModelCompound$toJSONString()
- PagedModelCompound$fromJSONString()
- PagedModelCompound$validateJSON()
- PagedModelCompound$toString()
- PagedModelCompound$isValid()
- PagedModelCompound$getInvalidFields()
- PagedModelCompound$print()
- PagedModelCompound$clone()

Method `new()`**:** Initialize a new PagedModelCompound class.

Usage:

```
PagedModelCompound$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelCompound$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelCompound$toList()
```

Returns: PagedModelCompound as a base R list.

Examples:

```
# convert array of PagedModelCompound (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelCompound to a base R type

Usage:

```
PagedModelCompound$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelCompound

Usage:

```
PagedModelCompound$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelCompound

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelCompound$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelCompound in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelCompound

Usage:

```
PagedModelCompound$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelCompound

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelCompound and throw an exception if invalid

Usage:

```
PagedModelCompound$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelCompound$toString()
```

Returns: String representation of PagedModelCompound

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelCompound$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelCompound$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelCompound$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelCompound$toList`
## ------------------------------------------------

# convert array of PagedModelCompound (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PagedModelFeature` *PagedModelFeature*

Description

PagedModelFeature Class

PagedModelFeature Class

Format

An `R6Class` generator object

Details

Create a new PagedModelFeature

Public fields

`content` list(Feature) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelFeature$new()`
- `PagedModelFeature$toJSON()`
- `PagedModelFeature$toList()`
- `PagedModelFeature$toSimpleType()`
- `PagedModelFeature$fromJSON()`
- `PagedModelFeature$toJSONString()`
- `PagedModelFeature$fromJSONString()`
- `PagedModelFeature$validateJSON()`
- `PagedModelFeature$toString()`
- `PagedModelFeature$isValid()`
- `PagedModelFeature$getInvalidFields()`
- `PagedModelFeature$print()`
- `PagedModelFeature$clone()`

Method `new()`**:** Initialize a new PagedModelFeature class.

Usage:

```
PagedModelFeature$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelFeature$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelFeature$toList()
```

Returns: PagedModelFeature as a base R list.

Examples:

```
# convert array of PagedModelFeature (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelFeature to a base R type

Usage:

```
PagedModelFeature$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelFeature

Usage:

```
PagedModelFeature$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelFeature

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelFeature$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelFeature in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelFeature

Usage:

```
PagedModelFeature$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelFeature

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelFeature and throw an exception if invalid

Usage:

```
PagedModelFeature$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelFeature$toString()
```

Returns: String representation of PagedModelFeature

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelFeature$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelFeature$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelFeature$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelFeature$toList`
## ------------------------------------------------

# convert array of PagedModelFeature (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

PagedModelFormulaCandidate
PagedModelFormulaCandidate

Description

PagedModelFormulaCandidate Class

PagedModelFormulaCandidate Class

Format

An R6Class generator object

Details

Create a new PagedModelFormulaCandidate

Public fields

`content` list(FormulaCandidate) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelFormulaCandidate$new()`
- `PagedModelFormulaCandidate$toJSON()`
- `PagedModelFormulaCandidate$toList()`
- `PagedModelFormulaCandidate$toSimpleType()`
- `PagedModelFormulaCandidate$fromJSON()`
- `PagedModelFormulaCandidate$toJSONString()`
- `PagedModelFormulaCandidate$fromJSONString()`
- `PagedModelFormulaCandidate$validateJSON()`
- `PagedModelFormulaCandidate$toString()`
- `PagedModelFormulaCandidate$isValid()`
- `PagedModelFormulaCandidate$getInvalidFields()`
- `PagedModelFormulaCandidate$print()`
- `PagedModelFormulaCandidate$clone()`

Method `new()`**:** Initialize a new PagedModelFormulaCandidate class.

Usage:

```
PagedModelFormulaCandidate$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelFormulaCandidate$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelFormulaCandidate$toList()
```

Returns: PagedModelFormulaCandidate as a base R list.

Examples:

```
# convert array of PagedModelFormulaCandidate (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelFormulaCandidate to a base R type

Usage:

```
PagedModelFormulaCandidate$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelFormulaCandidate

Usage:

```
PagedModelFormulaCandidate$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelFormulaCandidate

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelFormulaCandidate$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelFormulaCandidate in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelFormulaCandidate

Usage:

```
PagedModelFormulaCandidate$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelFormulaCandidate

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelFormulaCandidate and throw an exception if invalid

Usage:

```
PagedModelFormulaCandidate$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelFormulaCandidate$toString()
```

Returns: String representation of PagedModelFormulaCandidate

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelFormulaCandidate$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelFormulaCandidate$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelFormulaCandidate$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelFormulaCandidate$toList`
## ------------------------------------------------

# convert array of PagedModelFormulaCandidate (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

PagedModelJob *PagedModelJob*

Description

PagedModelJob Class

PagedModelJob Class

Format

An `R6Class` generator object

Details

Create a new PagedModelJob

Public fields

`content` list(Job) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelJob$new()`
- `PagedModelJob$toJSON()`
- `PagedModelJob$toList()`
- `PagedModelJob$toSimpleType()`
- `PagedModelJob$fromJSON()`
- `PagedModelJob$toJSONString()`
- `PagedModelJob$fromJSONString()`
- `PagedModelJob$validateJSON()`

- PagedModelJob$toString()
- PagedModelJob$isValid()
- PagedModelJob$getInvalidFields()
- PagedModelJob$print()
- PagedModelJob$clone()

Method `new()`**:** Initialize a new PagedModelJob class.

Usage:

```
PagedModelJob$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelJob$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelJob$toList()
```

Returns: PagedModelJob as a base R list.

Examples:

```
# convert array of PagedModelJob (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelJob to a base R type

Usage:

```
PagedModelJob$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelJob

Usage:

```
PagedModelJob$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelJob

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelJob$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelJob in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelJob

Usage:

```
PagedModelJob$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelJob

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelJob and throw an exception if invalid

Usage:

```
PagedModelJob$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelJob$toString()
```

Returns: String representation of PagedModelJob

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelJob$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelJob$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelJob$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelJob$toList`
## ------------------------------------------------

# convert array of PagedModelJob (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

PagedModelRun *PagedModelRun*

Description

PagedModelRun Class

PagedModelRun Class

Format

An `R6Class` generator object

Details

Create a new PagedModelRun

Public fields

`content` list(Run) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelRun$new()`
- `PagedModelRun$toJSON()`
- `PagedModelRun$toList()`
- `PagedModelRun$toSimpleType()`
- `PagedModelRun$fromJSON()`
- `PagedModelRun$toJSONString()`
- `PagedModelRun$fromJSONString()`
- `PagedModelRun$validateJSON()`

- PagedModelRun$toString()
- PagedModelRun$isValid()
- PagedModelRun$getInvalidFields()
- PagedModelRun$print()
- PagedModelRun$clone()

Method `new()`**:** Initialize a new PagedModelRun class.

Usage:

```
PagedModelRun$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelRun$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelRun$toList()
```

Returns: PagedModelRun as a base R list.

Examples:

```
# convert array of PagedModelRun (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelRun to a base R type

Usage:

```
PagedModelRun$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelRun

Usage:

```
PagedModelRun$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelRun

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelRun$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelRun in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelRun

Usage:

```
PagedModelRun$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelRun

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelRun and throw an exception if invalid

Usage:

```
PagedModelRun$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelRun$toString()
```

Returns: String representation of PagedModelRun

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelRun$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelRun$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelRun$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelRun$toList`
## ------------------------------------------------

# convert array of PagedModelRun (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PagedModelSpectralLibraryMatch`
PagedModelSpectralLibraryMatch

Description

PagedModelSpectralLibraryMatch Class

PagedModelSpectralLibraryMatch Class

Format

An `R6Class` generator object

Details

Create a new PagedModelSpectralLibraryMatch

Public fields

`content` list(SpectralLibraryMatch) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelSpectralLibraryMatch$new()`
- `PagedModelSpectralLibraryMatch$toJSON()`
- `PagedModelSpectralLibraryMatch$toList()`
- `PagedModelSpectralLibraryMatch$toSimpleType()`
- `PagedModelSpectralLibraryMatch$fromJSON()`
- `PagedModelSpectralLibraryMatch$toJSONString()`
- `PagedModelSpectralLibraryMatch$fromJSONString()`

- PagedModelSpectralLibraryMatch$validateJSON()
- PagedModelSpectralLibraryMatch$toString()
- PagedModelSpectralLibraryMatch$isValid()
- PagedModelSpectralLibraryMatch$getInvalidFields()
- PagedModelSpectralLibraryMatch$print()
- PagedModelSpectralLibraryMatch$clone()

Method `new()`**:** Initialize a new PagedModelSpectralLibraryMatch class.

Usage:

```
PagedModelSpectralLibraryMatch$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelSpectralLibraryMatch$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelSpectralLibraryMatch$toList()
```

Returns: PagedModelSpectralLibraryMatch as a base R list.

Examples:

```
# convert array of PagedModelSpectralLibraryMatch (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelSpectralLibraryMatch to a base R type

Usage:

```
PagedModelSpectralLibraryMatch$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelSpectralLibrary-Match

Usage:

```
PagedModelSpectralLibraryMatch$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelSpectralLibraryMatch

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelSpectralLibraryMatch$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelSpectralLibraryMatch in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelSpectralLibraryMatch

Usage:

```
PagedModelSpectralLibraryMatch$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelSpectralLibraryMatch

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelSpectralLibraryMatch and throw an exception if invalid

Usage:

```
PagedModelSpectralLibraryMatch$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelSpectralLibraryMatch$toString()
```

Returns: String representation of PagedModelSpectralLibraryMatch

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelSpectralLibraryMatch$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelSpectralLibraryMatch$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelSpectralLibraryMatch$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelSpectralLibraryMatch$toList`
## ------------------------------------------------

# convert array of PagedModelSpectralLibraryMatch (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PagedModelStructureCandidateFormula` *PagedModelStructureCandidateFormula*

Description

PagedModelStructureCandidateFormula Class

PagedModelStructureCandidateFormula Class

Format

An `R6Class` generator object

Details

Create a new PagedModelStructureCandidateFormula

Public fields

`content` list(StructureCandidateFormula) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- PagedModelStructureCandidateFormula$new()
- PagedModelStructureCandidateFormula$toJSON()
- PagedModelStructureCandidateFormula$toList()
- PagedModelStructureCandidateFormula$toSimpleType()
- PagedModelStructureCandidateFormula$fromJSON()
- PagedModelStructureCandidateFormula$toJSONString()
- PagedModelStructureCandidateFormula$fromJSONString()
- PagedModelStructureCandidateFormula$validateJSON()
- PagedModelStructureCandidateFormula$toString()
- PagedModelStructureCandidateFormula$isValid()
- PagedModelStructureCandidateFormula$getInvalidFields()
- PagedModelStructureCandidateFormula$print()
- PagedModelStructureCandidateFormula$clone()

Method `new()`**:** Initialize a new PagedModelStructureCandidateFormula class.

Usage:

```
PagedModelStructureCandidateFormula$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelStructureCandidateFormula$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelStructureCandidateFormula$toList()
```

Returns: PagedModelStructureCandidateFormula as a base R list.

Examples:

```
# convert array of PagedModelStructureCandidateFormula (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelStructureCandidateFormula to a base R type

Usage:

```
PagedModelStructureCandidateFormula$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelStructureCandidateFormula

Usage:

```
PagedModelStructureCandidateFormula$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelStructureCandidateFormula

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelStructureCandidateFormula$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelStructureCandidateFormula in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelStructureCandidateFormula

Usage:

```
PagedModelStructureCandidateFormula$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelStructureCandidateFormula

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelStructureCandidateFormula and throw an exception if invalid

Usage:

```
PagedModelStructureCandidateFormula$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelStructureCandidateFormula$toString()
```

Returns: String representation of PagedModelStructureCandidateFormula

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

`PagedModelStructureCandidateFormula$isValid()`

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

`PagedModelStructureCandidateFormula$getInvalidFields()`

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

`PagedModelStructureCandidateFormula$print()`

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

Usage:

`PagedModelStructureCandidateFormula$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelStructureCandidateFormula$toList`
## ------------------------------------------------

# convert array of PagedModelStructureCandidateFormula (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PagedModelStructureCandidateScored`

PagedModelStructureCandidateScored

Description

PagedModelStructureCandidateScored Class

PagedModelStructureCandidateScored Class

Format

An `R6Class` generator object

Details

Create a new PagedModelStructureCandidateScored

Public fields

`content` list(StructureCandidateScored) [optional]

`page` PageMetadata [optional]

Methods

Public methods:

- `PagedModelStructureCandidateScored$new()`
- `PagedModelStructureCandidateScored$toJSON()`
- `PagedModelStructureCandidateScored$toList()`
- `PagedModelStructureCandidateScored$toSimpleType()`
- `PagedModelStructureCandidateScored$fromJSON()`
- `PagedModelStructureCandidateScored$toJSONString()`
- `PagedModelStructureCandidateScored$fromJSONString()`
- `PagedModelStructureCandidateScored$validateJSON()`
- `PagedModelStructureCandidateScored$toString()`
- `PagedModelStructureCandidateScored$isValid()`
- `PagedModelStructureCandidateScored$getInvalidFields()`
- `PagedModelStructureCandidateScored$print()`
- `PagedModelStructureCandidateScored$clone()`

Method `new()`**:** Initialize a new PagedModelStructureCandidateScored class.

Usage:

```
PagedModelStructureCandidateScored$new(content = NULL, page = NULL, ...)
```

Arguments:

`content` content

`page` page

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PagedModelStructureCandidateScored$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PagedModelStructureCandidateScored$toList()
```

Returns: PagedModelStructureCandidateScored as a base R list.

Examples:

```
# convert array of PagedModelStructureCandidateScored (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PagedModelStructureCandidateScored to a base R type

Usage:

```
PagedModelStructureCandidateScored$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PagedModelStructureCandidateScored

Usage:

```
PagedModelStructureCandidateScored$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelStructureCandidateScored

Method `toJSONString()`**:** To JSON String

Usage:

```
PagedModelStructureCandidateScored$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PagedModelStructureCandidateScored in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PagedModelStructureCandidateScored

Usage:

```
PagedModelStructureCandidateScored$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PagedModelStructureCandidateScored

Method `validateJSON()`**:** Validate JSON input with respect to PagedModelStructureCandidateScored and throw an exception if invalid

Usage:

```
PagedModelStructureCandidateScored$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PagedModelStructureCandidateScored$toString()
```

Returns: String representation of PagedModelStructureCandidateScored

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PagedModelStructureCandidateScored$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PagedModelStructureCandidateScored$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PagedModelStructureCandidateScored$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PagedModelStructureCandidateScored$toList`
## ------------------------------------------------

# convert array of PagedModelStructureCandidateScored (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

PageMetadata *PageMetadata*

Description

PageMetadata Class

PageMetadata Class

Format

An `R6Class` generator object

Details

Create a new PageMetadata

Public fields

`size` integer [optional]

`number` integer [optional]

`totalElements` integer [optional]

`totalPages` integer [optional]

Methods

Public methods:

- `PageMetadata$new()`
- `PageMetadata$toJSON()`
- `PageMetadata$toList()`
- `PageMetadata$toSimpleType()`
- `PageMetadata$fromJSON()`
- `PageMetadata$toJSONString()`
- `PageMetadata$fromJSONString()`
- `PageMetadata$validateJSON()`
- `PageMetadata$toString()`
- `PageMetadata$isValid()`
- `PageMetadata$getInvalidFields()`
- `PageMetadata$print()`
- `PageMetadata$clone()`

Method `new()`**:** Initialize a new PageMetadata class.

Usage:

```
PageMetadata$new(
  size = NULL,
  number = NULL,
  totalElements = NULL,
  totalPages = NULL,
  ...
)
```

Arguments:

`size` size

`number` number

`totalElements` totalElements

`totalPages` totalPages

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PageMetadata$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PageMetadata$toList()
```

Returns: PageMetadata as a base R list.

Examples:

```
# convert array of PageMetadata (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PageMetadata to a base R type

Usage:

```
PageMetadata$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PageMetadata

Usage:

```
PageMetadata$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PageMetadata

Method `toJSONString()`**:** To JSON String

Usage:

```
PageMetadata$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PageMetadata in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PageMetadata

Usage:

```
PageMetadata$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PageMetadata

Method `validateJSON()`**:** Validate JSON input with respect to PageMetadata and throw an exception if invalid

Usage:

```
PageMetadata$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PageMetadata$toString()
```

Returns: String representation of PageMetadata

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PageMetadata$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PageMetadata$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PageMetadata$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PageMetadata$toList`
## ------------------------------------------------

# convert array of PageMetadata (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ParentPeak` *ParentPeak*

Description

Link from annotated fragment peak to its parent fragment peak connected by their neutral loss.

ParentPeak Class

Format

An `R6Class` generator object

Details

Create a new ParentPeak

Public fields

`parentIdx` Index to the parent peak connected by this loss in this particular spectrum integer

`parentFragmentId` Identifier of the parent fragment connected via this loss. Can be used to map fragments and peaks among fragmentation trees and spectra. integer

`lossFormula` Molecular formula of the neutral loss that connects these two peaks. character

Methods

Public methods:

- `ParentPeak$new()`
- `ParentPeak$toJSON()`
- `ParentPeak$toList()`
- `ParentPeak$toSimpleType()`
- `ParentPeak$fromJSON()`
- `ParentPeak$toJSONString()`

- ParentPeak$fromJSONString()
- ParentPeak$validateJSON()
- ParentPeak$toString()
- ParentPeak$isValid()
- ParentPeak$getInvalidFields()
- ParentPeak$print()
- ParentPeak$clone()

Method `new()`**:** Initialize a new ParentPeak class.

Usage:

```
ParentPeak$new(parentIdx, parentFragmentId, lossFormula, ...)
```

Arguments:

`parentIdx` Index to the parent peak connected by this loss in this particular spectrum

`parentFragmentId` Identifier of the parent fragment connected via this loss. Can be used to map fragments and peaks among fragmentation trees and spectra.

`lossFormula` Molecular formula of the neutral loss that connects these two peaks.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ParentPeak$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ParentPeak$toList()
```

Returns: ParentPeak as a base R list.

Examples:

```
# convert array of ParentPeak (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ParentPeak to a base R type

Usage:

```
ParentPeak$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ParentPeak

Usage:

```
ParentPeak$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ParentPeak

Method `toJSONString()`**:** To JSON String

Usage:

```
ParentPeak$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ParentPeak in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ParentPeak

Usage:

```
ParentPeak$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ParentPeak

Method `validateJSON()`**:** Validate JSON input with respect to ParentPeak and throw an exception if invalid

Usage:

```
ParentPeak$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ParentPeak$toString()
```

Returns: String representation of ParentPeak

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ParentPeak$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ParentPeak$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ParentPeak$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ParentPeak$toList`
## ------------------------------------------------

# convert array of ParentPeak (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PeakAnnotation` *PeakAnnotation*

Description

PeakAnnotation Class

PeakAnnotation Class

Format

An `R6Class` generator object

Details

Create a new PeakAnnotation

Public fields

`fragmentId` Identifier of the peak/fragment. Can be used to map fragments and peaks among fragmentation trees and spectra. integer

`molecularFormula` Molecular formula that has been annotated to this peak character [optional]

`adduct` Adduct that has been annotated to this peak character [optional]

`exactMass` Exact mass of the annotated molecular formula and adduct numeric [optional]

`massDeviationMz` Absolute mass deviation of the exact mass to the measured peak mass in mDa numeric [optional]

`massDeviationPpm` Relative mass deviation of the exact mass to the measured peak mass in ppm numeric [optional]

`recalibratedMassDeviationMz` Absolute mass deviation of the exact mass to the recalibrated peak mass in mDa numeric [optional]

`recalibratedMassDeviationPpm` Relative mass deviation of the exact mass to the recalibrated peak mass in ppm numeric [optional]

`parentPeak` ParentPeak [optional]

`substructureAtoms` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Array/List of indices of the atoms of the structure candidate that are part of this fragments substructure (highlighted atoms) list(integer) [optional]

`substructureBonds` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Array/List of indices of the bonds of the structure candidate that are part of this fragments substructure (highlighted bonds) Null if substructure annotation not available or not requested. list(integer) [optional]

`substructureBondsCut` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Array/List of indices of the bonds of the structure candidate that need to be cut to produce this fragments substructure (highlighted cut bonds). Null if substructure annotation not available or not requested. list(integer) [optional]

`substructureScore` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. This score roughly reflects the probability of this fragment forming. This is the score of the path from root to this node which has the maximal score or \"profit\". The score of a path is equal to the sum of scores of its contained fragments and edges. Note: Refers to 'totalScore' in CombinatorialNode Null if substructure annotation not available or not requested. numeric [optional]

`hydrogenRearrangements` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Number of hydrogens rearrangements needed to match the substructure to the fragment formula. Null if substructure annotation not available or not requested. integer [optional]

Methods

Public methods:

- `PeakAnnotation$new()`
- `PeakAnnotation$toJSON()`
- `PeakAnnotation$toList()`
- `PeakAnnotation$toSimpleType()`
- `PeakAnnotation$fromJSON()`
- `PeakAnnotation$toJSONString()`
- `PeakAnnotation$fromJSONString()`

- PeakAnnotation$validateJSON()
- PeakAnnotation$toString()
- PeakAnnotation$isValid()
- PeakAnnotation$getInvalidFields()
- PeakAnnotation$print()
- PeakAnnotation$clone()

Method `new()`**:** Initialize a new PeakAnnotation class.

Usage:

```
PeakAnnotation$new(
  fragmentId,
  molecularFormula = NULL,
  adduct = NULL,
  exactMass = NULL,
  massDeviationMz = NULL,
  massDeviationPpm = NULL,
  recalibratedMassDeviationMz = NULL,
  recalibratedMassDeviationPpm = NULL,
  parentPeak = NULL,
  substructureAtoms = NULL,
  substructureBonds = NULL,
  substructureBondsCut = NULL,
  substructureScore = NULL,
  hydrogenRearrangements = NULL,
  ...
)
```

Arguments:

`fragmentId` Identifier of the peak/fragment. Can be used to map fragments and peaks among fragmentation trees and spectra.

`molecularFormula` Molecular formula that has been annotated to this peak

`adduct` Adduct that has been annotated to this peak

`exactMass` Exact mass of the annotated molecular formula and adduct

`massDeviationMz` Absolute mass deviation of the exact mass to the measured peak mass in mDa

`massDeviationPpm` Relative mass deviation of the exact mass to the measured peak mass in ppm

`recalibratedMassDeviationMz` Absolute mass deviation of the exact mass to the recalibrated peak mass in mDa

`recalibratedMassDeviationPpm` Relative mass deviation of the exact mass to the recalibrated peak mass in ppm

`parentPeak` parentPeak

`substructureAtoms` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Array/List of indices of the atoms of the structure candidate that are part of this fragments substructure (highlighted atoms)

`substructureBonds` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Array/List of indices of the bonds of the structure candidate that are part of this fragments substructure (highlighted bonds) Null if substructure annotation not available or not requested.

`substructureBondsCut` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Array/List of indices of the bonds of the structure candidate that need to be cut to produce this fragments substructure (highlighted cut bonds). Null if substructure annotation not available or not requested.

`substructureScore` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. This score roughly reflects the probability of this fragment forming. This is the score of the path from root to this node which has the maximal score or \"profit\". The score of a path is equal to the sum of scores of its contained fragments and edges. Note: Refers to 'totalScore' in CombinatorialNode Null if substructure annotation not available or not requested.

`hydrogenRearrangements` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. Number of hydrogens rearrangements needed to match the substructure to the fragment formula. Null if substructure annotation not available or not requested.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PeakAnnotation$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PeakAnnotation$toList()
```

Returns: PeakAnnotation as a base R list.

Examples:

```
# convert array of PeakAnnotation (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PeakAnnotation to a base R type

Usage:

```
PeakAnnotation$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PeakAnnotation

Usage:

```
PeakAnnotation$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PeakAnnotation

Method `toJSONString()`**:** To JSON String

Usage:

```
PeakAnnotation$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PeakAnnotation in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PeakAnnotation

Usage:

```
PeakAnnotation$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PeakAnnotation

Method `validateJSON()`**:** Validate JSON input with respect to PeakAnnotation and throw an exception if invalid

Usage:

```
PeakAnnotation$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PeakAnnotation$toString()
```

Returns: String representation of PeakAnnotation

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PeakAnnotation$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PeakAnnotation$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PeakAnnotation$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PeakAnnotation$toList`
## ------------------------------------------------

# convert array of PeakAnnotation (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`PeakPair` *PeakPair*

Description

PeakPair Class

PeakPair Class

Format

An `R6Class` generator object

Details

Create a new PeakPair

Public fields

`queryPeak` integer

`referencePeak` integer

Methods

Public methods:

- PeakPair$new()
- PeakPair$toJSON()
- PeakPair$toList()
- PeakPair$toSimpleType()
- PeakPair$fromJSON()
- PeakPair$toJSONString()
- PeakPair$fromJSONString()
- PeakPair$validateJSON()
- PeakPair$toString()
- PeakPair$isValid()
- PeakPair$getInvalidFields()
- PeakPair$print()
- PeakPair$clone()

Method `new()`**:** Initialize a new PeakPair class.

Usage:

```
PeakPair$new(queryPeak, referencePeak, ...)
```

Arguments:

`queryPeak` queryPeak

`referencePeak` referencePeak

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
PeakPair$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
PeakPair$toList()
```

Returns: PeakPair as a base R list.

Examples:

```
# convert array of PeakPair (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert PeakPair to a base R type

Usage:

```
PeakPair$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of PeakPair

Usage:

```
PeakPair$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PeakPair

Method `toJSONString()`**:** To JSON String

Usage:

```
PeakPair$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: PeakPair in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of PeakPair

Usage:

```
PeakPair$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of PeakPair

Method `validateJSON()`**:** Validate JSON input with respect to PeakPair and throw an exception if invalid

Usage:

```
PeakPair$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
PeakPair$toString()
```

Returns: String representation of PeakPair

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
PeakPair$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
PeakPair$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
PeakPair$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `PeakPair$toList`
## ------------------------------------------------

# convert array of PeakPair (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ProblemDetail` *ProblemDetail*

Description

ProblemDetail Class

ProblemDetail Class

Format

An `R6Class` generator object

Details

Create a new ProblemDetail

Public fields

`type` character [optional]

`title` character [optional]

`status` integer [optional]

`detail` character [optional]

`instance` character [optional]

`properties` named list(object) [optional]

Methods

Public methods:

- `ProblemDetail$new()`
- `ProblemDetail$toJSON()`
- `ProblemDetail$toList()`
- `ProblemDetail$toSimpleType()`
- `ProblemDetail$fromJSON()`
- `ProblemDetail$toJSONString()`
- `ProblemDetail$fromJSONString()`
- `ProblemDetail$validateJSON()`
- `ProblemDetail$toString()`
- `ProblemDetail$isValid()`
- `ProblemDetail$getInvalidFields()`
- `ProblemDetail$print()`
- `ProblemDetail$clone()`

Method `new()`**:** Initialize a new ProblemDetail class.

Usage:

```
ProblemDetail$new(
  type = NULL,
  title = NULL,
  status = NULL,
  detail = NULL,
  instance = NULL,
  properties = NULL,
  ...
)
```

Arguments:

`type` type

`title` title

`status` status

`detail` detail

`instance` instance

`properties` properties

... Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ProblemDetail$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ProblemDetail$toList()
```

Returns: ProblemDetail as a base R list.

Examples:

```
# convert array of ProblemDetail (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ProblemDetail to a base R type

Usage:

```
ProblemDetail$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ProblemDetail

Usage:

```
ProblemDetail$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ProblemDetail

Method `toJSONString()`**:** To JSON String

Usage:

```
ProblemDetail$toJSONString(...)
```

Arguments:

... Parameters passed to 'jsonlite::toJSON'

Returns: ProblemDetail in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ProblemDetail

Usage:

```
ProblemDetail$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ProblemDetail

Method `validateJSON()`**:** Validate JSON input with respect to ProblemDetail and throw an exception if invalid

Usage:

```
ProblemDetail$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ProblemDetail$toString()
```

Returns: String representation of ProblemDetail

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ProblemDetail$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ProblemDetail$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ProblemDetail$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ProblemDetail$toList`
## ------------------------------------------------

# convert array of ProblemDetail (x) to a data frame
## Not run:
library(purrr)
```

```
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ProjectChangeEvent` *ProjectChangeEvent*

Description

ProjectChangeEvent Class

ProjectChangeEvent Class

Format

An `R6Class` generator object

Details

Create a new ProjectChangeEvent

Public fields

`eventType` character [optional]

`projectId` character [optional]

`compoundId` character [optional]

`featuredId` character [optional]

`formulaId` character [optional]

`structureInChIKey` character [optional]

Methods

Public methods:

- `ProjectChangeEvent$new()`
- `ProjectChangeEvent$toJSON()`
- `ProjectChangeEvent$toList()`
- `ProjectChangeEvent$toSimpleType()`
- `ProjectChangeEvent$fromJSON()`
- `ProjectChangeEvent$toJSONString()`
- `ProjectChangeEvent$fromJSONString()`
- `ProjectChangeEvent$validateJSON()`
- `ProjectChangeEvent$toString()`
- `ProjectChangeEvent$isValid()`

- ProjectChangeEvent$getInvalidFields()
- ProjectChangeEvent$print()
- ProjectChangeEvent$clone()

Method `new()`**:** Initialize a new ProjectChangeEvent class.

Usage:

```
ProjectChangeEvent$new(
  eventType = NULL,
  projectId = NULL,
  compoundId = NULL,
  featuredId = NULL,
  formulaId = NULL,
  structureInChIKey = NULL,
  ...
)
```

Arguments:

`eventType` eventType

`projectId` projectId

`compoundId` compoundId

`featuredId` featuredId

`formulaId` formulaId

`structureInChIKey` structureInChIKey

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ProjectChangeEvent$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ProjectChangeEvent$toList()
```

Returns: ProjectChangeEvent as a base R list.

Examples:

```
# convert array of ProjectChangeEvent (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ProjectChangeEvent to a base R type

Usage:

```
ProjectChangeEvent$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ProjectChangeEvent

Usage:

```
ProjectChangeEvent$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ProjectChangeEvent

Method `toJSONString()`**:** To JSON String

Usage:

```
ProjectChangeEvent$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ProjectChangeEvent in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ProjectChangeEvent

Usage:

```
ProjectChangeEvent$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ProjectChangeEvent

Method `validateJSON()`**:** Validate JSON input with respect to ProjectChangeEvent and throw an exception if invalid

Usage:

```
ProjectChangeEvent$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ProjectChangeEvent$toString()
```

Returns: String representation of ProjectChangeEvent

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ProjectChangeEvent$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ProjectChangeEvent$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ProjectChangeEvent$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ProjectChangeEvent$toList`
## ------------------------------------------------

# convert array of ProjectChangeEvent (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ProjectInfo` *ProjectInfo*

Description

ProjectInfo Class

ProjectInfo Class

Format

An `R6Class` generator object

Details

Create a new ProjectInfo

Public fields

`projectId` A user-selected unique name of the project for easy access. character [optional]

`location` Storage location of the project. character [optional]

`description` Description of this project. character [optional]

`type` Type of this project. NULL if project type has not yet been specified by importing data. character [optional]

`compatible` Indicates whether computed results (e.g. fingerprints, compound classes) are compatible with the backend. If true, the project is up-to-date and there are no restrictions regarding usage. If false, the project is incompatible and therefore \"read only\" until the incompatible results have been removed. See the updateProject endpoint for further information. If NULL, the information has not been requested. character [optional]

`numOfFeatures` Number of features (aligned over runs) in this project. If NULL, information has not been requested (See OptField 'sizeInformation'). integer [optional]

`numOfCompounds` Number of compounds (group of ion identities) in this project. If NULL, information has not been requested (See OptField 'sizeInformation') or might be unavailable for this project type. integer [optional]

`numOfBytes` Size in bytes this project consumes on disk. If NULL, information has not been requested (See OptField 'sizeInformation'). integer [optional]

`detectedAdducts` Set of all detected adducts available in this project. list(character) [optional]

Methods

Public methods:

- `ProjectInfo$new()`
- `ProjectInfo$toJSON()`
- `ProjectInfo$toList()`
- `ProjectInfo$toSimpleType()`
- `ProjectInfo$fromJSON()`
- `ProjectInfo$toJSONString()`
- `ProjectInfo$fromJSONString()`
- `ProjectInfo$validateJSON()`
- `ProjectInfo$toString()`
- `ProjectInfo$isValid()`
- `ProjectInfo$getInvalidFields()`
- `ProjectInfo$print()`
- `ProjectInfo$clone()`

Method `new()`**:** Initialize a new ProjectInfo class.

Usage:

```
ProjectInfo$new(
  projectId = NULL,
  location = NULL,
  description = NULL,
```

```
  type = NULL,
  compatible = NULL,
  numOfFeatures = NULL,
  numOfCompounds = NULL,
  numOfBytes = NULL,
  detectedAdducts = NULL,
  ...
)
```

Arguments:

`projectId` A user-selected unique name of the project for easy access.

`location` Storage location of the project.

`description` Description of this project.

`type` Type of this project. NULL if project type has not yet been specified by importing data.

`compatible` Indicates whether computed results (e.g. fingerprints, compound classes) are compatible with the backend. If true, the project is up-to-date and there are no restrictions regarding usage. If false, the project is incompatible and therefore \"read only\" until the incompatible results have been removed. See the updateProject endpoint for further information. If NULL, the information has not been requested.

`numOfFeatures` Number of features (aligned over runs) in this project. If NULL, information has not been requested (See OptField 'sizeInformation').

`numOfCompounds` Number of compounds (group of ion identities) in this project. If NULL, information has not been requested (See OptField 'sizeInformation') or might be unavailable for this project type.

`numOfBytes` Size in bytes this project consumes on disk. If NULL, information has not been requested (See OptField 'sizeInformation').

`detectedAdducts` Set of all detected adducts available in this project.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ProjectInfo$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ProjectInfo$toList()
```

Returns: ProjectInfo as a base R list.

Examples:

```
# convert array of ProjectInfo (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ProjectInfo to a base R type

Usage:

```
ProjectInfo$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ProjectInfo

Usage:

```
ProjectInfo$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ProjectInfo

Method `toJSONString()`**:** To JSON String

Usage:

```
ProjectInfo$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ProjectInfo in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ProjectInfo

Usage:

```
ProjectInfo$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ProjectInfo

Method `validateJSON()`**:** Validate JSON input with respect to ProjectInfo and throw an exception if invalid

Usage:

```
ProjectInfo$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ProjectInfo$toString()
```

Returns: String representation of ProjectInfo

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ProjectInfo$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ProjectInfo$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ProjectInfo$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ProjectInfo$toList`
## ------------------------------------------------

# convert array of ProjectInfo (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ProjectsApi` *Projects operations*

Description

ProjectsApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `ProjectsApi$new()`
- `ProjectsApi$BuildSearchIndex()`
- `ProjectsApi$BuildSearchIndexWithHttpInfo()`
- `ProjectsApi$CloseProject()`
- `ProjectsApi$CloseProjectWithHttpInfo()`
- `ProjectsApi$Create()`
- `ProjectsApi$CreateWithHttpInfo()`
- `ProjectsApi$CreateProject()`
- `ProjectsApi$CreateProjectWithHttpInfo()`
- `ProjectsApi$GetCanopusClassyFireData()`
- `ProjectsApi$GetCanopusClassyFireDataWithHttpInfo()`
- `ProjectsApi$GetCanopusNpcData()`
- `ProjectsApi$GetCanopusNpcDataWithHttpInfo()`
- `ProjectsApi$GetFingerIdData()`
- `ProjectsApi$GetFingerIdDataWithHttpInfo()`
- `ProjectsApi$GetProject()`
- `ProjectsApi$GetProjectWithHttpInfo()`
- `ProjectsApi$GetProjects()`
- `ProjectsApi$GetProjectsWithHttpInfo()`
- `ProjectsApi$ImportMsRunData()`
- `ProjectsApi$ImportMsRunDataWithHttpInfo()`
- `ProjectsApi$ImportMsRunDataAsJob()`
- `ProjectsApi$ImportMsRunDataAsJobWithHttpInfo()`
- `ProjectsApi$ImportPreprocessedData()`
- `ProjectsApi$ImportPreprocessedDataWithHttpInfo()`
- `ProjectsApi$ImportPreprocessedDataAsJob()`
- `ProjectsApi$ImportPreprocessedDataAsJobWithHttpInfo()`
- `ProjectsApi$OpenProject()`
- `ProjectsApi$OpenProjectWithHttpInfo()`
- `ProjectsApi$clone()`

Method `new()`**:** Initialize a new ProjectsApi.

Usage:

```
ProjectsApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `BuildSearchIndex()`**:** Create a search index for the given project.

Usage:

```
ProjectsApi$BuildSearchIndex(project_id, force = FALSE, ...)
```

Arguments:

`project_id` unique name/identifier of the project to create the index for.

`force` (optional) if true an existing index will be deleted and recreated. (default value: FALSE)

`...` Other optional arguments

Returns: void

Method `BuildSearchIndexWithHttpInfo()`**:** Create a search index for the given project.

Usage:

```
ProjectsApi$BuildSearchIndexWithHttpInfo(project_id, force = FALSE, ...)
```

Arguments:

`project_id` unique name/identifier of the project to create the index for.

`force` (optional) if true an existing index will be deleted and recreated. (default value: FALSE)

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `CloseProject()`**:** Close project-space and remove it from the application

Usage:

```
ProjectsApi$CloseProject(project_id, compact = FALSE, ...)
```

Arguments:

`project_id` unique name/identifier of the project-space to be closed.

`compact` (optional) if true, compact project storage after closing. DEPRECATED: Compacting acts on the local filesystem and will likely be removed in a later version. (default value: FALSE)

`...` Other optional arguments

Returns: void

Method `CloseProjectWithHttpInfo()`**:** Close project-space and remove it from the application

Usage:

```
ProjectsApi$CloseProjectWithHttpInfo(project_id, compact = FALSE, ...)
```

Arguments:

`project_id` unique name/identifier of the project-space to be closed.

`compact` (optional) if true, compact project storage after closing. DEPRECATED: Compacting acts on the local filesystem and will likely be removed in a later version. (default value: FALSE)

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `Create()`**:** Create and open a new project with unique autogenerated projectId.

Usage:

```
ProjectsApi$Create(opt_fields = list("none"), data_file = NULL, ...)
```

Arguments:

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: ProjectInfo

Method `CreateWithHttpInfo()`**:** Create and open a new project with unique autogenerated projectId.

Usage:

```
ProjectsApi$CreateWithHttpInfo(
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (ProjectInfo) with additional information such as HTTP status code, headers

Method `CreateProject()`**:** Create and open a new project-space at given location and make it accessible via the given projectId.

Usage:

```
ProjectsApi$CreateProject(
  project_id,
  path_to_project = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` unique name/identifier that shall be used to access the newly created project-space. Must consist only of [a-zA-Z0-9_-].

`path_to_project` (optional) local file path where the project will be created. If NULL, project will be stored by its projectId in default project location. DEPRECATED: This parameter relies on the local filesystem and will likely be removed in later versions of this API to allow for more flexible use cases.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: ProjectInfo

Method `CreateProjectWithHttpInfo()`**:** Create and open a new project-space at given location and make it accessible via the given projectId.

Usage:

```
ProjectsApi$CreateProjectWithHttpInfo(
  project_id,
  path_to_project = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` unique name/identifier that shall be used to access the newly created project-space. Must consist only of [a-zA-Z0-9_-].

`path_to_project` (optional) local file path where the project will be created. If NULL, project will be stored by its projectId in default project location. DEPRECATED: This parameter relies on the local filesystem and will likely be removed in later versions of this API to allow for more flexible use cases.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (ProjectInfo) with additional information such as HTTP status code, headers

Method `GetCanopusClassyFireData()`**:** Get CANOPUS prediction vector definition for ClassyFire classes

Usage:

```
ProjectsApi$GetCanopusClassyFireData(project_id, charge, data_file = NULL, ...)
```

Arguments:

`project_id`

`charge`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `GetCanopusClassyFireDataWithHttpInfo()`**:** Get CANOPUS prediction vector definition for ClassyFire classes

Usage:

```
ProjectsApi$GetCanopusClassyFireDataWithHttpInfo(
  project_id,
  charge,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id`

`charge`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `GetCanopusNpcData()`**:** Get CANOPUS prediction vector definition for NPC classes

Usage:

```
ProjectsApi$GetCanopusNpcData(project_id, charge, data_file = NULL, ...)
```

Arguments:

`project_id`

`charge`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `GetCanopusNpcDataWithHttpInfo()`**:** Get CANOPUS prediction vector definition for NPC classes

Usage:

```
ProjectsApi$GetCanopusNpcDataWithHttpInfo(
  project_id,
  charge,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id`

`charge`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `GetFingerIdData()`**:** Get CSI:FingerID fingerprint (prediction vector) definition

Usage:

```
ProjectsApi$GetFingerIdData(project_id, charge, data_file = NULL, ...)
```

Arguments:

`project_id`

`charge`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: character

Method `GetFingerIdDataWithHttpInfo()`**:** Get CSI:FingerID fingerprint (prediction vector) definition

Usage:

```
ProjectsApi$GetFingerIdDataWithHttpInfo(
  project_id,
  charge,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id`

`charge`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (character) with additional information such as HTTP status code, headers

Method `GetProject()`**:** Get project space info by its projectId.

Usage:

```
ProjectsApi$GetProject(
  project_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` unique name/identifier of the project-space to be accessed.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: ProjectInfo

Method `GetProjectWithHttpInfo()`**:** Get project space info by its projectId.

Usage:

```
ProjectsApi$GetProjectWithHttpInfo(
  project_id,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` unique name/identifier of the project-space to be accessed.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (ProjectInfo) with additional information such as HTTP status code, headers

Method `GetProjects()`**:** List opened project spaces.

Usage:

```
ProjectsApi$GetProjects(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[ProjectInfo]

Method `GetProjectsWithHttpInfo()`**:** List opened project spaces.

Usage:

```
ProjectsApi$GetProjectsWithHttpInfo(data_file = NULL, ...)
```

Arguments:

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[ProjectInfo]) with additional information such as HTTP status code, headers

Method `ImportMsRunData()`**:** Import and align full MS runs from various formats into the specified project Possible formats: mzML, mzXML.

Usage:

```
ProjectsApi$ImportMsRunData(
  project_id,
  input_files,
  parameters,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` Project-space to import into.

`input_files` files to import into project

`parameters`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: ImportResult

Method `ImportMsRunDataWithHttpInfo()`**:** Import and align full MS runs from various formats into the specified project Possible formats: mzML, mzXML.

Usage:

```
ProjectsApi$ImportMsRunDataWithHttpInfo(
  project_id,
  input_files,
  parameters,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` Project-space to import into.

`input_files` files to import into project

`parameters`

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (ImportResult) with additional information such as HTTP status code, headers

Method `ImportMsRunDataAsJob()`**:** Import and align full MS runs from various formats into the specified project as background job.

Usage:

```
ProjectsApi$ImportMsRunDataAsJob(
  project_id,
  input_files,
  parameters,
  opt_fields = list("progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` Project-space to import into.

`input_files` files to import into project

`parameters`

`opt_fields` (optional) Set of optional fields to be included. Use 'none' only to override defaults. (default value: ["progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Job

Method `ImportMsRunDataAsJobWithHttpInfo()`**:** Import and align full MS runs from various formats into the specified project as background job.

Usage:

```
ProjectsApi$ImportMsRunDataAsJobWithHttpInfo(
  project_id,
  input_files,
```

```
  parameters,
  opt_fields = list("progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` Project-space to import into.

`input_files` files to import into project

`parameters`

`opt_fields` (optional) Set of optional fields to be included. Use 'none' only to override defaults. (default value: ["progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Job) with additional information such as HTTP status code, headers

Method `ImportPreprocessedData()`**:** Import already preprocessed ms/ms data from various formats into the specified project Possible formats: ms, mgf, cef, msp.

Usage:

```
ProjectsApi$ImportPreprocessedData(
  project_id,
  input_files,
  ignore_formulas = FALSE,
  allow_ms1_only = TRUE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`input_files` files to import into project

`ignore_formulas` (optional) No description (default value: FALSE)

`allow_ms1_only` (optional) No description (default value: TRUE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: ImportResult

Method `ImportPreprocessedDataWithHttpInfo()`**:** Import already preprocessed ms/ms data from various formats into the specified project Possible formats: ms, mgf, cef, msp.

Usage:

```
ProjectsApi$ImportPreprocessedDataWithHttpInfo(
  project_id,
  input_files,
  ignore_formulas = FALSE,
  allow_ms1_only = TRUE,
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`input_files` files to import into project

`ignore_formulas` (optional) No description (default value: FALSE)

`allow_ms1_only` (optional) No description (default value: TRUE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (ImportResult) with additional information such as HTTP status code, headers

Method `ImportPreprocessedDataAsJob()`**:** Import ms/ms data from the given format into the specified project-space as background job.

Usage:

```
ProjectsApi$ImportPreprocessedDataAsJob(
  project_id,
  input_files,
  ignore_formulas = FALSE,
  allow_ms1_only = TRUE,
  opt_fields = list("progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`input_files` files to import into project

`ignore_formulas` (optional) No description (default value: FALSE)

`allow_ms1_only` (optional) No description (default value: TRUE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Job

Method `ImportPreprocessedDataAsJobWithHttpInfo()`**:** Import ms/ms data from the given format into the specified project-space as background job.

Usage:

```
ProjectsApi$ImportPreprocessedDataAsJobWithHttpInfo(
  project_id,
  input_files,
  ignore_formulas = FALSE,
  allow_ms1_only = TRUE,
  opt_fields = list("progress"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to import into.

`input_files` files to import into project

`ignore_formulas` (optional) No description (default value: FALSE)

`allow_ms1_only` (optional) No description (default value: TRUE)

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: ["progress"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Job) with additional information such as HTTP status code, headers

Method `OpenProject()`**:** Open an existing project-space and make it accessible via the given projectId.

Usage:

```
ProjectsApi$OpenProject(
  project_id,
  path_to_project = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` unique name/identifier that shall be used to access the opened project-space. Must consist only of [a-zA-Z0-9_-].

`path_to_project` (optional) local file path to open the project from. If NULL, project will be loaded by its projectId from default project location. DEPRECATED: This parameter relies on the local filesystem and will likely be removed in later versions of this API to allow for more flexible use cases.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: ProjectInfo

Method `OpenProjectWithHttpInfo()`**:** Open an existing project-space and make it accessible via the given projectId.

Usage:

```
ProjectsApi$OpenProjectWithHttpInfo(
  project_id,
  path_to_project = NULL,
  opt_fields = list("none"),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` unique name/identifier that shall be used to access the opened project-space. Must consist only of [a-zA-Z0-9_-].

`path_to_project` (optional) local file path to open the project from. If NULL, project will be loaded by its projectId from default project location. DEPRECATED: This parameter relies on the local filesystem and will likely be removed in later versions of this API to allow for more flexible use cases.

`opt_fields` (optional) No description (default value: ["none"])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (ProjectInfo) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  BuildSearchIndex  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | unique name/identifier of the project to create the index for.
var_force <- FALSE # character | if true an existing index will be deleted and recreated. (Optional)

#Create a search index for the given project.
api_instance <- rsirius_api$new()

api_instance$projects_api$BuildSearchIndex(var_project_id, force = var_force)


####################  CloseProject  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | unique name/identifier of the project-space to be closed.
var_compact <- FALSE # character | if true, compact project storage after closing. DEPRECATED: Compacting acts on th

#Close project-space and remove it from the application
api_instance <- rsirius_api$new()

api_instance$projects_api$CloseProject(var_project_id, compact = var_compact)


####################  Create  ####################

library(RSirius)
var_opt_fields <- c("none") # array[character] |  (Optional)
```

```
#Create and open a new project with unique autogenerated projectId.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$Create(opt_fields = var_opt_fieldsdata_file = "result.txt")
result <- api_instance$projects_api$Create(opt_fields = var_opt_fields)
dput(result)


####################  CreateProject  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | unique name/identifier that shall be used to access the newly c
var_path_to_project <- "path_to_project_example" # character | local file path where the project will be created. I
var_opt_fields <- c("none") # array[character] |  (Optional)

#Create and open a new project-space at given location and make it accessible via the given projectId.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$CreateProject(var_project_id, path_to_project = var_path_to_project, opt_fields = var_op
result <- api_instance$projects_api$CreateProject(var_project_id, path_to_project = var_path_to_project, opt_fie
dput(result)


####################  GetCanopusClassyFireData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character |
var_charge <- 56 # integer |

#Get CANOPUS prediction vector definition for ClassyFire classes
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCanopusClassyFireData(var_project_id, var_chargedata_file = "result.txt")
result <- api_instance$projects_api$GetCanopusClassyFireData(var_project_id, var_charge)
dput(result)


####################  GetCanopusNpcData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character |
var_charge <- 56 # integer |

#Get CANOPUS prediction vector definition for NPC classes
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetCanopusNpcData(var_project_id, var_chargedata_file = "result.txt")
result <- api_instance$projects_api$GetCanopusNpcData(var_project_id, var_charge)
```

```
dput(result)


####################  GetFingerIdData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character |
var_charge <- 56 # integer |

#Get CSI:FingerID fingerprint (prediction vector) definition
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetFingerIdData(var_project_id, var_chargedata_file = "result.txt")
result <- api_instance$projects_api$GetFingerIdData(var_project_id, var_charge)
dput(result)


####################  GetProject  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | unique name/identifier of the project-space to be accessed.
var_opt_fields <- c("none") # array[character] |  (Optional)

#Get project space info by its projectId.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetProject(var_project_id, opt_fields = var_opt_fieldsdata_file = "result.txt")
result <- api_instance$projects_api$GetProject(var_project_id, opt_fields = var_opt_fields)
dput(result)


####################  GetProjects  ####################

library(RSirius)

#List opened project spaces.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetProjects(data_file = "result.txt")
result <- api_instance$projects_api$GetProjects()
dput(result)


####################  ImportMsRunData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | Project-space to import into.
var_input_files <- c(123) # array[data.frame] | files to import into project
var_parameters <- LcmsSubmissionParameters$new(c("sampleNames_example"), c("sampleTypes_example"), "alignLCMSRun
```

```
#Import and align full MS runs from various formats into the specified project  Possible formats: mzML, mzXML.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$ImportMsRunData(var_project_id, var_input_files, var_parametersdata_file = "result.txt")
result <- api_instance$projects_api$ImportMsRunData(var_project_id, var_input_files, var_parameters)
dput(result)


####################  ImportMsRunDataAsJob  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | Project-space to import into.
var_input_files <- c(123) # array[data.frame] | files to import into project
var_parameters <- LcmsSubmissionParameters$new(c("sampleNames_example"), c("sampleTypes_example"), "alignLCMSRun
var_opt_fields <- c("none") # array[character] | Set of optional fields to be included. Use 'none' only to override

#Import and align full MS runs from various formats into the specified project as background job.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$ImportMsRunDataAsJob(var_project_id, var_input_files, var_parameters, opt_fields = var_o
result <- api_instance$projects_api$ImportMsRunDataAsJob(var_project_id, var_input_files, var_parameters, opt_fi
dput(result)


####################  ImportPreprocessedData  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to import into.
var_input_files <- c(123) # array[data.frame] | files to import into project
var_ignore_formulas <- FALSE # character |  (Optional)
var_allow_ms1_only <- TRUE # character |  (Optional)

#Import already preprocessed ms/ms data from various formats into the specified project  Possible formats: ms, mgf,
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$ImportPreprocessedData(var_project_id, var_input_files, ignore_formulas = var_ignore_for
result <- api_instance$projects_api$ImportPreprocessedData(var_project_id, var_input_files, ignore_formulas = va
dput(result)


####################  ImportPreprocessedDataAsJob  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to import into.
var_input_files <- c(123) # array[data.frame] | files to import into project
var_ignore_formulas <- FALSE # character |  (Optional)
var_allow_ms1_only <- TRUE # character |  (Optional)
var_opt_fields <- c("none") # array[character] | set of optional fields to be included. Use 'none' only to override

#Import ms/ms data from the given format into the specified project-space as background job.
```

```
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$ImportPreprocessedDataAsJob(var_project_id, var_input_files, ignore_formulas = var_ignor
result <- api_instance$projects_api$ImportPreprocessedDataAsJob(var_project_id, var_input_files, ignore_formulas
dput(result)


####################  OpenProject  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | unique name/identifier that shall be used to access the opened
var_path_to_project <- "path_to_project_example" # character | local file path to open the project from. If NULL, pr
var_opt_fields <- c("none") # array[character] |  (Optional)

#Open an existing project-space and make it accessible via the given projectId.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$OpenProject(var_project_id, path_to_project = var_path_to_project, opt_fields = var_opt_
result <- api_instance$projects_api$OpenProject(var_project_id, path_to_project = var_path_to_project, opt_field
dput(result)



## End(Not run)
```

QuantTable *QuantTable*

Description

Quantification of features or compounds within the runs they have been detected in. Rows refer to the quantified objects, columns to the runs. Values that could not be quantified are NaN.

QuantTable Class

Format

An `R6Class` generator object

Details

Create a new QuantTable

Public fields

`quantificationMeasure` character [optional]

`rowType` character [optional]

`rowIds` Ids of the quantified objects, features or compounds depending on the row type. list(character) [optional]

`columnIds` Ids of the runs the objects are quantified in. list(character) [optional]

`columnNames` Names of the runs the objects are quantified in, in the order of columnIds. <p> Optional field, only present if requested, since the table can be read by run id alone. <p> The name is the one the run carries in the project. It is either the sample name given when the data was imported, or, if none was given, derived from the measurement itself: the run id inside the file, and only failing that the file name. A name is therefore not necessarily the name of the file the run came from, and it is not guaranteed to be unique. Use columnSources to identify the input file, and columnIds to identify the run. list(character) [optional]

`columnSources` Files the runs were imported from, in the order of columnIds, to relate a column back to the input data. Same value as the source of the corresponding run. <p> Optional field, only present if requested, since it is not needed to read the table. list(character) [optional]

`values` list(list(numeric)) [optional]

Methods

Public methods:

- `QuantTable$new()`
- `QuantTable$toJSON()`
- `QuantTable$toList()`
- `QuantTable$toSimpleType()`
- `QuantTable$fromJSON()`
- `QuantTable$toJSONString()`
- `QuantTable$fromJSONString()`
- `QuantTable$validateJSON()`
- `QuantTable$toString()`
- `QuantTable$isValid()`
- `QuantTable$getInvalidFields()`
- `QuantTable$print()`
- `QuantTable$clone()`

Method `new()`**:** Initialize a new QuantTable class.

Usage:

```
QuantTable$new(
  quantificationMeasure = NULL,
  rowType = NULL,
  rowIds = NULL,
  columnIds = NULL,
  columnNames = NULL,
  columnSources = NULL,
  values = NULL,
  ...
)
```

Arguments:

`quantificationMeasure` quantificationMeasure

`rowType` rowType

`rowIds` Ids of the quantified objects, features or compounds depending on the row type.

`columnIds` Ids of the runs the objects are quantified in.

`columnNames` Names of the runs the objects are quantified in, in the order of columnIds. <p> Optional field, only present if requested, since the table can be read by run id alone. <p> The name is the one the run carries in the project. It is either the sample name given when the data was imported, or, if none was given, derived from the measurement itself: the run id inside the file, and only failing that the file name. A name is therefore not necessarily the name of the file the run came from, and it is not guaranteed to be unique. Use columnSources to identify the input file, and columnIds to identify the run.

`columnSources` Files the runs were imported from, in the order of columnIds, to relate a column back to the input data. Same value as the source of the corresponding run. <p> Optional field, only present if requested, since it is not needed to read the table.

`values` values

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
QuantTable$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
QuantTable$toList()
```

Returns: QuantTable as a base R list.

Examples:

```
# convert array of QuantTable (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert QuantTable to a base R type

Usage:

```
QuantTable$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of QuantTable

Usage:

```
QuantTable$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of QuantTable

Method `toJSONString()`**:** To JSON String

Usage:

```
QuantTable$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: QuantTable in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of QuantTable

Usage:

```
QuantTable$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of QuantTable

Method `validateJSON()`**:** Validate JSON input with respect to QuantTable and throw an exception if invalid

Usage:

```
QuantTable$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
QuantTable$toString()
```

Returns: String representation of QuantTable

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
QuantTable$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
QuantTable$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
QuantTable$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `QuantTable$toList`
## ------------------------------------------------

# convert array of QuantTable (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`QuantTableOptField` *QuantTableOptField*

Description

QuantTableOptField Class

Format

An `R6Class` generator object

Methods

Public methods:

- `QuantTableOptField$new()`
- `QuantTableOptField$toJSON()`
- `QuantTableOptField$fromJSON()`
- `QuantTableOptField$toJSONString()`
- `QuantTableOptField$fromJSONString()`
- `QuantTableOptField$clone()`

Method `new()`**:** Initialize a new QuantTableOptField class.

Usage:

```
QuantTableOptField$new(...)
```

Arguments:

`...` Optional arguments.

Method `toJSON()`**:** To JSON String

Usage:

```
QuantTableOptField$toJSON()
```

Returns: QuantTableOptField in JSON format

Method `fromJSON()`**:** Deserialize JSON string into an instance of QuantTableOptField

Usage:

```
QuantTableOptField$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of QuantTableOptField

Method `toJSONString()`**:** To JSON String

Usage:

```
QuantTableOptField$toJSONString()
```

Returns: QuantTableOptField in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of QuantTableOptField

Usage:

```
QuantTableOptField$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of QuantTableOptField

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`rsirius_api` *ApiClient*

Description

ApiClient Class

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

RSirius API Class

A single point of access to the RSirius API.

NOTE: This class is auto generated by OpenAPI Generator (https://openapi-generator.tech). Ref: https://openapi-generator.tech Do not edit the class manually.

Public fields

`api_client` API client

`actuator_api` an instance of ActuatorApi

`compounds_api` an instance of CompoundsApi

`features_api` an instance of FeaturesApi

`gui_api` an instance of GuiApi

`info_api` an instance of InfoApi

`jobs_api` an instance of JobsApi

`login_and_account_api` an instance of LoginAndAccountApi

`projects_api` an instance of ProjectsApi

`runs_api` an instance of RunsApi

`searchable_databases_api` an instance of SearchableDatabasesApi

Methods

Public methods:

- rsirius_api$new()
- rsirius_api$wait_for_job_completion()
- rsirius_api$clone()

Method `new()`**:** Initialize a new RSirius API Class.

Usage:

```
rsirius_api$new(api_client)
```

Arguments:

`api_client` An instance of API client (optional).

Method `wait_for_job_completion()`**:**

Usage:

```
rsirius_api$wait_for_job_completion(project_id, job_id, timeout_in_sec = NULL)
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Run *Run*

Description

Run Class

Run Class

Format

An `R6Class` generator object

Details

Create a new Run

Public fields

`runId` Identifier character [optional]

`name` Informative, human-readable name of the run character [optional]

`source` Source location character [optional]

`chromatography` Chromatography the run was measured with, e.g. 'Liquid Chromatography'. character [optional]

`ionization` Ionization the run was measured with, named as in the HUPO PSI-MS controlled vocabulary, e.g. 'electrospray ionization'. character [optional]

`fragmentation` Fragmentation the run was measured with, named as in the HUPO PSI-MS controlled vocabulary, e.g. 'beam-type collision-induced dissociation'. character [optional]

`massAnalyzers` Mass analyzers of the instrument the run was measured on, named as in the HUPO PSI-MS controlled vocabulary, e.g. 'orbitrap'. list(character) [optional]

`tags` Key: tagName, value: tag named list(Tag) [optional]

Methods

Public methods:

- `Run$new()`
- `Run$toJSON()`
- `Run$toList()`
- `Run$toSimpleType()`
- `Run$fromJSON()`
- `Run$toJSONString()`
- `Run$fromJSONString()`
- `Run$validateJSON()`
- `Run$toString()`
- `Run$isValid()`
- `Run$getInvalidFields()`
- `Run$print()`
- `Run$clone()`

Method `new()`**:** Initialize a new Run class.

Usage:

```
Run$new(
  runId = NULL,
  name = NULL,
  source = NULL,
  chromatography = NULL,
  ionization = NULL,
  fragmentation = NULL,
  massAnalyzers = NULL,
  tags = NULL,
  ...
)
```

Arguments:

`runId` Identifier

`name` Informative, human-readable name of the run

`source` Source location

`chromatography` Chromatography the run was measured with, e.g. 'Liquid Chromatography'.

`ionization` Ionization the run was measured with, named as in the HUPO PSI-MS controlled vocabulary, e.g. 'electrospray ionization'.

`fragmentation` Fragmentation the run was measured with, named as in the HUPO PSI-MS controlled vocabulary, e.g. 'beam-type collision-induced dissociation'.

`massAnalyzers` Mass analyzers of the instrument the run was measured on, named as in the HUPO PSI-MS controlled vocabulary, e.g. 'orbitrap'.

`tags` Key: tagName, value: tag

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Run$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Run$toList()
```

Returns: Run as a base R list.

Examples:

```
# convert array of Run (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Run to a base R type

Usage:

```
Run$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Run

Usage:

```
Run$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Run

Method `toJSONString()`**:** To JSON String

Usage:

```
Run$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Run in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Run

Usage:

```
Run$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Run

Method `validateJSON()`**:** Validate JSON input with respect to Run and throw an exception if invalid

Usage:

```
Run$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Run$toString()
```

Returns: String representation of Run

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Run$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Run$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Run$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Run$toList`
## ------------------------------------------------

# convert array of Run (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

RunOptField *RunOptField*

Description

RunOptField Class

Format

An `R6Class` generator object

Methods

Public methods:

- `RunOptField$new()`
- `RunOptField$toJSON()`
- `RunOptField$fromJSON()`
- `RunOptField$toJSONString()`
- `RunOptField$fromJSONString()`
- `RunOptField$clone()`

Method `new()`**:** Initialize a new RunOptField class.

Usage:

```
RunOptField$new(...)
```

Arguments:

`...` Optional arguments.

Method `toJSON()`**:** To JSON String

Usage:

```
RunOptField$toJSON()
```

Returns: RunOptField in JSON format

Method `fromJSON()`**:** Deserialize JSON string into an instance of RunOptField

Usage:

```
RunOptField$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of RunOptField

Method `toJSONString()`**:** To JSON String

Usage:

```
RunOptField$toJSONString()
```

Returns: RunOptField in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of RunOptField

Usage:

```
RunOptField$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of RunOptField

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`RunsApi` *Runs operations*

Description

RunsApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `RunsApi$new()`
- `RunsApi$GetRun()`
- `RunsApi$GetRunWithHttpInfo()`
- `RunsApi$GetRunsPage()`
- `RunsApi$GetRunsPageWithHttpInfo()`
- `RunsApi$GetRunsSearchableFields()`
- `RunsApi$GetRunsSearchableFieldsWithHttpInfo()`
- `RunsApi$clone()`

Method `new()`**:** Initialize a new RunsApi.

Usage:

```
RunsApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `GetRun()`**:** Get run with the given identifier from the specified project-space.

Usage:

```
RunsApi$GetRun(project_id, run_id, opt_fields = list(), data_file = NULL, ...)
```

Arguments:

`project_id` project-space to read from.

`run_id` identifier of run to access.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: [])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: Run

Method `GetRunWithHttpInfo()`**:** Get run with the given identifier from the specified project-space.

Usage:

```
RunsApi$GetRunWithHttpInfo(
  project_id,
  run_id,
  opt_fields = list(),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project-space to read from.

`run_id` identifier of run to access.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: [])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (Run) with additional information such as HTTP status code, headers

Method `GetRunsPage()`**:** Get runs in the given project-space

Usage:

```
RunsApi$GetRunsPage(
  project_id,
  search_query = NULL,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list(),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to get runs from.

`search_query` (optional) Optional search query in lucene syntax. Omit this parameter to page over all runs.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: [])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: PagedModelRun

Method `GetRunsPageWithHttpInfo()`**:** Get runs in the given project-space

Usage:

```
RunsApi$GetRunsPageWithHttpInfo(
  project_id,
  search_query = NULL,
  page = 0,
  size = 20,
  sort = NULL,
  opt_fields = list(),
  data_file = NULL,
  ...
)
```

Arguments:

`project_id` project space to get runs from.

`search_query` (optional) Optional search query in lucene syntax. Omit this parameter to page over all runs.

`page` (optional) Zero-based page index (0..N) (default value: 0)

`size` (optional) The size of the page to be returned (default value: 20)

`sort` (optional) Sorting criteria in the format: property,(asc|desc). Default sort order is ascending. Multiple sort criteria are supported.

`opt_fields` (optional) set of optional fields to be included. Use 'none' only to override defaults. (default value: [])

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (PagedModelRun) with additional information such as HTTP status code, headers

Method `GetRunsSearchableFields()`**:** Get all fields that can be used in the searchQuery parameter of run endpoints

Usage:

```
RunsApi$GetRunsSearchableFields(project_id, data_file = NULL, ...)
```

Arguments:

`project_id` project space to read the searchable run fields from.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableField]

Method `GetRunsSearchableFieldsWithHttpInfo()`**:** Get all fields that can be used in the searchQuery parameter of run endpoints

Usage:

```
RunsApi$GetRunsSearchableFieldsWithHttpInfo(project_id, data_file = NULL, ...)
```

Arguments:

`project_id` project space to read the searchable run fields from.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableField]) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
####################  GetRun  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project-space to read from.
var_run_id <- "run_id_example" # character | identifier of run to access.
var_opt_fields <- c(RunOptField$new()) # array[RunOptField] | set of optional fields to be included. Use 'none' onl

#Get run with the given identifier from the specified project-space.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetRun(var_project_id, var_run_id, opt_fields = var_opt_fieldsdata_file = "result.txt")
result <- api_instance$runs_api$GetRun(var_project_id, var_run_id, opt_fields = var_opt_fields)
dput(result)


####################  GetRunsPage  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project space to get runs from.
var_search_query <- "search_query_example" # character | Optional search query in lucene syntax. Omit this paramete
var_page <- 0 # integer | Zero-based page index (0..N) (Optional)
var_size <- 20 # integer | The size of the page to be returned (Optional)
var_sort <- c("inner_example") # array[character] | Sorting criteria in the format: property,(asc|desc). Default so
var_opt_fields <- c(RunOptField$new()) # array[RunOptField] | set of optional fields to be included. Use 'none' onl

#Get runs in the given project-space
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetRunsPage(var_project_id, search_query = var_search_query, page = var_page, size = var_
result <- api_instance$runs_api$GetRunsPage(var_project_id, search_query = var_search_query, page = var_page, siz
dput(result)


####################  GetRunsSearchableFields  ####################

library(RSirius)
var_project_id <- "project_id_example" # character | project space to read the searchable run fields from.

#Get all fields that can be used in the searchQuery parameter of run endpoints
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetRunsSearchableFields(var_project_iddata_file = "result.txt")
result <- api_instance$runs_api$GetRunsSearchableFields(var_project_id)
dput(result)
```

```
## End(Not run)
```

`SearchableDatabase` *SearchableDatabase*

Description

SearchableDatabase Class

SearchableDatabase Class

Format

An `R6Class` generator object

Details

Create a new SearchableDatabase

Public fields

`databaseId` A unique identifier or name of the database. Should only contain file path and url save characters For user databases this is usually the file name. character

`customDb` Indicates whether the database is a user managed custom database or if it is a database that is included in SIRIUS which cannot be modified. character

`searchable` True when this database can be used as a search parameter. False if the database is just an additional filter that can be applied after search. character

`dbDate` Date on which the data was imported / database was created. character [optional]

`dbVersion` Database schema version. integer [optional]

`updateNeeded` If true the database version is outdated and the database needs to be updated or re-imported before it can be used. character

`numberOfStructures` Number of unique compounds available in this database. integer [optional]

`numberOfFormulas` Number of different molecular formulas available in this database. integer [optional]

`numberOfReferenceSpectra` Number of reference spectra available in this database integer [optional]

`errorMessage` Error message if the database could not be loaded character [optional]

`displayName` display name of the database Should be short character [optional]

`location` Storage location of user database Might be NULL for non-user databases or if default location is used. character [optional]

`matchRtOfReferenceSpectra` Indicates whether this database shall be used to use retention time information for library matching. Typically used for in-house spectral libraries that were measured on the same chromatographic setup as the samples being analysed, so that retention times are comparable. character [optional]

Methods

Public methods:

- SearchableDatabase$new()
- SearchableDatabase$toJSON()
- SearchableDatabase$toList()
- SearchableDatabase$toSimpleType()
- SearchableDatabase$fromJSON()
- SearchableDatabase$toJSONString()
- SearchableDatabase$fromJSONString()
- SearchableDatabase$validateJSON()
- SearchableDatabase$toString()
- SearchableDatabase$isValid()
- SearchableDatabase$getInvalidFields()
- SearchableDatabase$print()
- SearchableDatabase$clone()

Method `new()`**:** Initialize a new SearchableDatabase class.

Usage:

```
SearchableDatabase$new(
  databaseId,
  customDb,
  searchable,
  updateNeeded,
  dbDate = NULL,
  dbVersion = NULL,
  numberOfStructures = NULL,
  numberOfFormulas = NULL,
  numberOfReferenceSpectra = NULL,
  errorMessage = NULL,
  displayName = NULL,
  location = NULL,
  matchRtOfReferenceSpectra = FALSE,
  ...
)
```

Arguments:

`databaseId` A unique identifier or name of the database. Should only contain file path and url save characters For user databases this is usually the file name.

`customDb` Indicates whether the database is a user managed custom database or if it is a database that is included in SIRIUS which cannot be modified.

`searchable` True when this database can be used as a search parameter. False if the database is just an additional filter that can be applied after search.

`updateNeeded` If true the database version is outdated and the database needs to be updated or re-imported before it can be used.

`dbDate` Date on which the data was imported / database was created.

`dbVersion` Database schema version.

`numberOfStructures` Number of unique compounds available in this database.

`numberOfFormulas` Number of different molecular formulas available in this database.

`numberOfReferenceSpectra` Number of reference spectra available in this database

`errorMessage` Error message if the database could not be loaded

`displayName` display name of the database Should be short

`location` Storage location of user database Might be NULL for non-user databases or if default location is used.

`matchRtOfReferenceSpectra` Indicates whether this database shall be used to use retention time information for library matching. Typically used for in-house spectral libraries that were measured on the same chromatographic setup as the samples being analysed, so that retention times are comparable.. Default to FALSE.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SearchableDatabase$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SearchableDatabase$toList()
```

Returns: SearchableDatabase as a base R list.

Examples:

```
# convert array of SearchableDatabase (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SearchableDatabase to a base R type

Usage:

```
SearchableDatabase$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SearchableDatabase

Usage:

```
SearchableDatabase$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableDatabase

Method `toJSONString()`**:** To JSON String

Usage:

```
SearchableDatabase$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SearchableDatabase in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SearchableDatabase

Usage:

```
SearchableDatabase$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableDatabase

Method `validateJSON()`**:** Validate JSON input with respect to SearchableDatabase and throw an exception if invalid

Usage:

```
SearchableDatabase$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SearchableDatabase$toString()
```

Returns: String representation of SearchableDatabase

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SearchableDatabase$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SearchableDatabase$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SearchableDatabase$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SearchableDatabase$toList`
## ------------------------------------------------

# convert array of SearchableDatabase (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SearchableDatabaseParameters` *SearchableDatabaseParameters*

Description

SearchableDatabaseParameters Class

SearchableDatabaseParameters Class

Format

An `R6Class` generator object

Details

Create a new SearchableDatabaseParameters

Public fields

`displayName` display name of the database Should be short character [optional]

`location` Storage location of user database Might be NULL for non-user databases or if default location is used. character [optional]

`matchRtOfReferenceSpectra` Indicates whether this database shall be used to use retention time information for library matching. Typically used for in-house spectral libraries that were measured on the same chromatographic setup as the samples being analyzed, so that retention times are comparable. character [optional]

Methods

Public methods:

- `SearchableDatabaseParameters$new()`
- `SearchableDatabaseParameters$toJSON()`
- `SearchableDatabaseParameters$toList()`

- SearchableDatabaseParameters$toSimpleType()
- SearchableDatabaseParameters$fromJSON()
- SearchableDatabaseParameters$toJSONString()
- SearchableDatabaseParameters$fromJSONString()
- SearchableDatabaseParameters$validateJSON()
- SearchableDatabaseParameters$toString()
- SearchableDatabaseParameters$isValid()
- SearchableDatabaseParameters$getInvalidFields()
- SearchableDatabaseParameters$print()
- SearchableDatabaseParameters$clone()

Method `new()`**:** Initialize a new SearchableDatabaseParameters class.

Usage:

```
SearchableDatabaseParameters$new(
  displayName = NULL,
  location = NULL,
  matchRtOfReferenceSpectra = FALSE,
  ...
)
```

Arguments:

`displayName` display name of the database Should be short

`location` Storage location of user database Might be NULL for non-user databases or if default location is used.

`matchRtOfReferenceSpectra` Indicates whether this database shall be used to use retention time information for library matching. Typically used for in-house spectral libraries that were measured on the same chromatographic setup as the samples being analyzed, so that retention times are comparable.. Default to FALSE.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SearchableDatabaseParameters$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SearchableDatabaseParameters$toList()
```

Returns: SearchableDatabaseParameters as a base R list.

Examples:

```
# convert array of SearchableDatabaseParameters (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
```

```
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SearchableDatabaseParameters to a base R type

Usage:

```
SearchableDatabaseParameters$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SearchableDatabaseParameters

Usage:

```
SearchableDatabaseParameters$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableDatabaseParameters

Method `toJSONString()`**:** To JSON String

Usage:

```
SearchableDatabaseParameters$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SearchableDatabaseParameters in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SearchableDatabaseParameters

Usage:

```
SearchableDatabaseParameters$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableDatabaseParameters

Method `validateJSON()`**:** Validate JSON input with respect to SearchableDatabaseParameters and throw an exception if invalid

Usage:

```
SearchableDatabaseParameters$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SearchableDatabaseParameters$toString()
```

Returns: String representation of SearchableDatabaseParameters

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SearchableDatabaseParameters$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SearchableDatabaseParameters$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SearchableDatabaseParameters$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SearchableDatabaseParameters$toList`
## ------------------------------------------------

# convert array of SearchableDatabaseParameters (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SearchableDatabasesApi` *SearchableDatabases operations*

Description

SearchableDatabasesApi

Format

An `R6Class` generator object

Details

SIRIUS Nightsky API

REST API that provides the full functionality of SIRIUS and its web services as background service. It is intended as entry-point for scripting languages and software integration SDKs.This API is exposed by SIRIUS 6

The version of the OpenAPI document: 3.2 Generated by: https://openapi-generator.tech

Public fields

`api_client` Handles the client-server communication.

Methods

Public methods:

- `SearchableDatabasesApi$new()`
- `SearchableDatabasesApi$AddDatabases()`
- `SearchableDatabasesApi$AddDatabasesWithHttpInfo()`
- `SearchableDatabasesApi$CreateDatabase()`
- `SearchableDatabasesApi$CreateDatabaseWithHttpInfo()`
- `SearchableDatabasesApi$GetCustomDatabases()`
- `SearchableDatabasesApi$GetCustomDatabasesWithHttpInfo()`
- `SearchableDatabasesApi$GetDatabase()`
- `SearchableDatabasesApi$GetDatabaseWithHttpInfo()`
- `SearchableDatabasesApi$GetDatabases()`
- `SearchableDatabasesApi$GetDatabasesWithHttpInfo()`
- `SearchableDatabasesApi$GetIncludedDatabases()`
- `SearchableDatabasesApi$GetIncludedDatabasesWithHttpInfo()`
- `SearchableDatabasesApi$ImportIntoDatabase()`
- `SearchableDatabasesApi$ImportIntoDatabaseWithHttpInfo()`
- `SearchableDatabasesApi$RemoveDatabase()`
- `SearchableDatabasesApi$RemoveDatabaseWithHttpInfo()`
- `SearchableDatabasesApi$UpdateDatabase()`
- `SearchableDatabasesApi$UpdateDatabaseWithHttpInfo()`
- `SearchableDatabasesApi$clone()`

Method `new()`**:** Initialize a new SearchableDatabasesApi.

Usage:

```
SearchableDatabasesApi$new(api_client)
```

Arguments:

`api_client` An instance of API client.

Method `AddDatabases()`**:** [DEPRECATED] This endpoint is based on local file paths and will likely be replaced in future versions of this API.

Usage:

```
SearchableDatabasesApi$AddDatabases(request_body, data_file = NULL, ...)
```

Arguments:

`request_body` local file paths of the database files (.siriusdb) to register. Each must exist, must not already be registered, and its name must not collide with an existing database.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableDatabase]

Method `AddDatabasesWithHttpInfo()`**:** [DEPRECATED] This endpoint is based on local file paths and will likely be replaced in future versions of this API.

Usage:

```
SearchableDatabasesApi$AddDatabasesWithHttpInfo(
  request_body,
  data_file = NULL,
  ...
)
```

Arguments:

`request_body` local file paths of the database files (.siriusdb) to register. Each must exist, must not already be registered, and its name must not collide with an existing database.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableDatabase]) with additional information such as HTTP status code, headers

Method `CreateDatabase()`**:** Create a new, empty custom database

Usage:

```
SearchableDatabasesApi$CreateDatabase(
  database_id,
  searchable_database_parameters = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the new database. Must be URL-safe, that is letters, digits, '-' and '_' only, and must not be in use by another database.

`searchable_database_parameters` (optional) optional settings for the new database. If omitted, the database is created in the default custom database directory with default settings. Supply a location to place the database file elsewhere, a displayName for the user interface, and matchRtOfReferenceSpectra for in-house libraries whose retention times are comparable to the measured samples.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: SearchableDatabase

Method `CreateDatabaseWithHttpInfo()`**:** Create a new, empty custom database

Usage:

```
SearchableDatabasesApi$CreateDatabaseWithHttpInfo(
  database_id,
  searchable_database_parameters = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the new database. Must be URL-safe, that is letters, digits, '-' and '_' only, and must not be in use by another database.

`searchable_database_parameters` (optional) optional settings for the new database. If omitted, the database is created in the default custom database directory with default settings. Supply a location to place the database file elsewhere, a displayName for the user interface, and matchRtOfReferenceSpectra for in-house libraries whose retention times are comparable to the measured samples.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (SearchableDatabase) with additional information such as HTTP status code, headers

Method `GetCustomDatabases()`**:** List only the custom databases, that is the structure databases and spectral libraries the user has created or added.

Usage:

```
SearchableDatabasesApi$GetCustomDatabases(
  include_stats = FALSE,
  include_with_errors = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_stats` (optional) if true, the number of structures, formulas and reference spectra is included per database. Slower, since the database files have to be read. (default value: FALSE)

`include_with_errors` (optional) if true, databases that could not be loaded are listed as well, carrying the reason in their errorMessage field. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableDatabase]

Method `GetCustomDatabasesWithHttpInfo()`**:** List only the custom databases, that is the structure databases and spectral libraries the user has created or added.

Usage:

```
SearchableDatabasesApi$GetCustomDatabasesWithHttpInfo(
  include_stats = FALSE,
  include_with_errors = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_stats` (optional) if true, the number of structures, formulas and reference spectra is included per database. Slower, since the database files have to be read. (default value: FALSE)

`include_with_errors` (optional) if true, databases that could not be loaded are listed as well, carrying the reason in their errorMessage field. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableDatabase]) with additional information such as HTTP status code, headers

Method `GetDatabase()`**:** Get a single searchable database by its id.

Usage:

```
SearchableDatabasesApi$GetDatabase(
  database_id,
  include_stats = TRUE,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the database to retrieve, as reported by the listing endpoints.

`include_stats` (optional) if true (the default here), the number of structures, formulas and reference spectra is included. (default value: TRUE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: SearchableDatabase

Method `GetDatabaseWithHttpInfo()`**:** Get a single searchable database by its id.

Usage:

```
SearchableDatabasesApi$GetDatabaseWithHttpInfo(
  database_id,
  include_stats = TRUE,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the database to retrieve, as reported by the listing endpoints.

`include_stats` (optional) if true (the default here), the number of structures, formulas and reference spectra is included. (default value: TRUE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (SearchableDatabase) with additional information such as HTTP status code, headers

Method `GetDatabases()`**:** List all searchable databases, both the ones included in SIRIUS and the custom ones added by the user

Usage:

```
SearchableDatabasesApi$GetDatabases(
  include_stats = FALSE,
  include_with_errors = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_stats` (optional) if true, the number of structures, formulas and reference spectra is included per database. Computing these counts touches the database files, so requesting them is noticeably slower than a plain listing. (default value: FALSE)

`include_with_errors` (optional) if true, databases that could not be loaded are listed as well, carrying the reason in their errorMessage field. Use this to show a broken database to the user instead of silently hiding it. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableDatabase]

Method `GetDatabasesWithHttpInfo()`**:** List all searchable databases, both the ones included in SIRIUS and the custom ones added by the user

Usage:

```
SearchableDatabasesApi$GetDatabasesWithHttpInfo(
  include_stats = FALSE,
  include_with_errors = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_stats` (optional) if true, the number of structures, formulas and reference spectra is included per database. Computing these counts touches the database files, so requesting them is noticeably slower than a plain listing. (default value: FALSE)

`include_with_errors` (optional) if true, databases that could not be loaded are listed as well, carrying the reason in their errorMessage field. Use this to show a broken database to the user instead of silently hiding it. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableDatabase]) with additional information such as HTTP status code, headers

Method `GetIncludedDatabases()`**:** List only the databases that ship with SIRIUS, such as PubChem and the bio databases.

Usage:

```
SearchableDatabasesApi$GetIncludedDatabases(
  include_stats = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_stats` (optional) if true, the number of structures, formulas and reference spectra is included per database. Slower, since the database files have to be read. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: array[SearchableDatabase]

Method `GetIncludedDatabasesWithHttpInfo()`**:** List only the databases that ship with SIRIUS, such as PubChem and the bio databases.

Usage:

```
SearchableDatabasesApi$GetIncludedDatabasesWithHttpInfo(
  include_stats = FALSE,
  data_file = NULL,
  ...
)
```

Arguments:

`include_stats` (optional) if true, the number of structures, formulas and reference spectra is included per database. Slower, since the database files have to be read. (default value: FALSE)

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (array[SearchableDatabase]) with additional information such as HTTP status code, headers

Method `ImportIntoDatabase()`**:** Start import of structure and spectra files into the specified database.

Usage:

```
SearchableDatabasesApi$ImportIntoDatabase(
  database_id,
  input_files,
  buffer_size = 1000,
  bio_transformer_parameters = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the custom database to import into. Must exist.

`input_files` files to import into project

`buffer_size` (optional) number of compounds to keep in memory before writing them to the database. Raise it to speed up large imports on machines with enough RAM. (default value: 1000)

`bio_transformer_parameters` (optional) No description

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: SearchableDatabase

Method `ImportIntoDatabaseWithHttpInfo()`**:** Start import of structure and spectra files into the specified database.

Usage:

```
SearchableDatabasesApi$ImportIntoDatabaseWithHttpInfo(
  database_id,
  input_files,
  buffer_size = 1000,
  bio_transformer_parameters = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the custom database to import into. Must exist.

`input_files` files to import into project

`buffer_size` (optional) number of compounds to keep in memory before writing them to the database. Raise it to speed up large imports on machines with enough RAM. (default value: 1000)

`bio_transformer_parameters` (optional) No description

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (SearchableDatabase) with additional information such as HTTP status code, headers

Method `RemoveDatabase()`**:** Remove a custom database from this SIRIUS instance, and optionally delete it from disk

Usage:

```
SearchableDatabasesApi$RemoveDatabase(database_id, delete = FALSE, ...)
```

Arguments:

`database_id` id of the database to remove.

`delete` (optional) if true, the database file is deleted from disk and the data is lost. If false (the default), only the registration is removed and the file is kept, so the database can be registered again later. (default value: FALSE)

`...` Other optional arguments

Returns: void

Method `RemoveDatabaseWithHttpInfo()`**:** Remove a custom database from this SIRIUS instance, and optionally delete it from disk

Usage:

```
SearchableDatabasesApi$RemoveDatabaseWithHttpInfo(
  database_id,
  delete = FALSE,
  ...
)
```

Arguments:

`database_id` id of the database to remove.

`delete` (optional) if true, the database file is deleted from disk and the data is lost. If false (the default), only the registration is removed and the file is kept, so the database can be registered again later. (default value: FALSE)

`...` Other optional arguments

Returns: API response (void) with additional information such as HTTP status code, headers

Method `UpdateDatabase()`**:** Change the settings of an existing custom database

Usage:

```
SearchableDatabasesApi$UpdateDatabase(
  database_id,
  searchable_database_parameters = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the database to update.

`searchable_database_parameters` (optional) the settings to apply.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: SearchableDatabase

Method `UpdateDatabaseWithHttpInfo()`**:** Change the settings of an existing custom database

Usage:

```
SearchableDatabasesApi$UpdateDatabaseWithHttpInfo(
  database_id,
  searchable_database_parameters = NULL,
  data_file = NULL,
  ...
)
```

Arguments:

`database_id` id of the database to update.

`searchable_database_parameters` (optional) the settings to apply.

`data_file` (optional) name of the data file to save the result

`...` Other optional arguments

Returns: API response (SearchableDatabase) with additional information such as HTTP status code, headers

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
####################  AddDatabases  ####################

library(RSirius)
var_request_body <- c("property_example") # array[character] | local file paths of the database files (.siriusdb) t

#[DEPRECATED] This endpoint is based on local file paths and will likely be replaced in future versions of this API.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$AddDatabases(var_request_bodydata_file = "result.txt")
result <- api_instance$searchable_databases_api$AddDatabases(var_request_body)
dput(result)


####################  CreateDatabase  ####################

library(RSirius)
var_database_id <- "database_id_example" # character | id of the new database. Must be URL-safe, that is letters, di
var_searchable_database_parameters <- SearchableDatabaseParameters$new("displayName_example", "location_example

#Create a new, empty custom database
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$CreateDatabase(var_database_id, searchable_database_parameters = var_searchable_databas
result <- api_instance$searchable_databases_api$CreateDatabase(var_database_id, searchable_database_parameters =
dput(result)


####################  GetCustomDatabases  ####################

library(RSirius)
var_include_stats <- FALSE # character | if true, the number of structures, formulas and reference spectra is includ
var_include_with_errors <- FALSE # character | if true, databases that could not be loaded are listed as well, carry

#List only the custom databases, that is the structure databases and spectral libraries the user has  created or adde
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
```

```
# result <- api_instance$GetCustomDatabases(include_stats = var_include_stats, include_with_errors = var_include_
result <- api_instance$searchable_databases_api$GetCustomDatabases(include_stats = var_include_stats, include_wi
dput(result)


####################  GetDatabase  ####################

library(RSirius)
var_database_id <- "database_id_example" # character | id of the database to retrieve, as reported by the listing en
var_include_stats <- TRUE # character | if true (the default here), the number of structures, formulas and reference

#Get a single searchable database by its id.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDatabase(var_database_id, include_stats = var_include_statsdata_file = "result.txt")
result <- api_instance$searchable_databases_api$GetDatabase(var_database_id, include_stats = var_include_stats)
dput(result)


####################  GetDatabases  ####################

library(RSirius)
var_include_stats <- FALSE # character | if true, the number of structures, formulas and reference spectra is includ
var_include_with_errors <- FALSE # character | if true, databases that could not be loaded are listed as well, carry

#List all searchable databases, both the ones included in SIRIUS and the custom ones added by the user
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetDatabases(include_stats = var_include_stats, include_with_errors = var_include_with_e
result <- api_instance$searchable_databases_api$GetDatabases(include_stats = var_include_stats, include_with_err
dput(result)


####################  GetIncludedDatabases  ####################

library(RSirius)
var_include_stats <- FALSE # character | if true, the number of structures, formulas and reference spectra is includ

#List only the databases that ship with SIRIUS, such as PubChem and the bio databases.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$GetIncludedDatabases(include_stats = var_include_statsdata_file = "result.txt")
result <- api_instance$searchable_databases_api$GetIncludedDatabases(include_stats = var_include_stats)
dput(result)


####################  ImportIntoDatabase  ####################

library(RSirius)
var_database_id <- "database_id_example" # character | id of the custom database to import into. Must exist.
```

```
var_input_files <- c(123) # array[data.frame] | files to import into project
var_buffer_size <- 1000 # integer | number of compounds to keep in memory before writing them to the
var_bio_transformer_parameters <- BioTransformerParameters$new("RULE_BASED", "BT_RULE_BASED", c(BioTransformerSe

#Start import of structure and spectra files into the specified database.
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$ImportIntoDatabase(var_database_id, var_input_files, buffer_size = var_buffer_size, bio_
result <- api_instance$searchable_databases_api$ImportIntoDatabase(var_database_id, var_input_files, buffer_size
dput(result)


####################  RemoveDatabase  ####################

library(RSirius)
var_database_id <- "database_id_example" # character | id of the database to remove.
var_delete <- FALSE # character | if true, the database file is deleted from disk and the data is lost. If false (the

#Remove a custom database from this SIRIUS instance, and optionally delete it from disk
api_instance <- rsirius_api$new()

api_instance$searchable_databases_api$RemoveDatabase(var_database_id, delete = var_delete)


####################  UpdateDatabase  ####################

library(RSirius)
var_database_id <- "database_id_example" # character | id of the database to update.
var_searchable_database_parameters <- SearchableDatabaseParameters$new("displayName_example", "location_example

#Change the settings of an existing custom database
api_instance <- rsirius_api$new()

# to save the result into a file, simply add the optional `data_file` parameter, e.g.
# result <- api_instance$UpdateDatabase(var_database_id, searchable_database_parameters = var_searchable_databas
result <- api_instance$searchable_databases_api$UpdateDatabase(var_database_id, searchable_database_parameters =
dput(result)



## End(Not run)
```

SearchableField *SearchableField*

Description

Describes one field of the search index that can be used in lucene search queries (searchQuery parameter). Use this information to build valid queries, e.g. which fields support range queries ([300 TO 400]) and which support word based (full text) search.

SearchableField Class

Format

An `R6Class` generator object

Details

Create a new SearchableField

Public fields

`name` Name of the field to be used in search queries, e.g. <code>ionMass:[300 TO 400]</code>. Nested fields are addressed with dot notation. Dynamic, map-like fields are reported as concrete, directly usable field names - one entry per key currently present in the project index - rather than a wildcard template, e.g. <code>tags.<tagName></code>, an element symbol in <code>...molecularFormula.<element></code>, or a database id in <code>topAnnotations.matchedDatabases.<dbId></code>. character

`fieldType` SearchableFieldType

`fullTextSearch` If true, the field content is split into words and can be searched word by word (full text search). If false, TEXT fields only match as exact terms (though wildcards and regex are still possible). character [optional]

`sortable` If true, search results can be sorted by this field. character [optional]

`defaultSearchField` If true, this field is searched when a query term does not specify a field name. character [optional]

`possibleValues` The values this field can take, exactly as they are indexed, or null if the field accepts free text. <p> Reported for ENUM and BOOLEAN fields, for TEXT fields that hold a closed vocabulary (the compound class ontologies), and for fields whose values are project state: a tag restricted by its definition, or the adducts detected in this project. Searchable fields are described per project, so the latter are the values actually present in it rather than everything that could ever occur. <p> Note that the indexed value is not always the notation you would expect: an adduct is indexed as '[M + H]+', with spaces. Use the values as offered rather than constructing them. list(character) [optional]

`description` Optional human-readable description of the field content, as shown in the API documentation. Plain text suitable for direct display (e.g. tooltips); deliberate paragraph breaks are newlines. character [optional]

`significantSuffixLength` How many trailing dot-separated segments of the field name carry its meaning, for compact display. <code>1</code> for a normal field (show the terminal segment, e.g. <code>lipid</code>); <code>2</code> for a dynamic map key (show field + key, e.g. <code>matchedDatabases.GNPS</code>, <code>qualities.PEAK_QUALITY</code>, <code>tags.pfas</code>); larger for multi-segment keys. Clients may show the last <code>significantSuffixLength</code> name segments as a short label; the full field name always stays authoritative for queries. integer

Methods

Public methods:

- SearchableField$new()
- SearchableField$toJSON()
- SearchableField$toList()
- SearchableField$toSimpleType()
- SearchableField$fromJSON()
- SearchableField$toJSONString()
- SearchableField$fromJSONString()
- SearchableField$validateJSON()
- SearchableField$toString()
- SearchableField$isValid()
- SearchableField$getInvalidFields()
- SearchableField$print()
- SearchableField$clone()

Method `new()`**:** Initialize a new SearchableField class.

Usage:

```
SearchableField$new(
  name,
  fieldType,
  significantSuffixLength,
  fullTextSearch = NULL,
  sortable = NULL,
  defaultSearchField = NULL,
  possibleValues = NULL,
  description = NULL,
  ...
)
```

Arguments:

`name` Name of the field to be used in search queries, e.g. <code>ionMass:[300 TO 400]</code>. Nested fields are addressed with dot notation. Dynamic, map-like fields are reported as concrete, directly usable field names - one entry per key currently present in the project index - rather than a wildcard template, e.g. <code>tags.<tagName></code>, an element symbol in <code>...molecularFormula.<element></code>, or a database id in <code>topAnnotations.matchedDatabases.<dbId></code>.

`fieldType` fieldType

`significantSuffixLength` How many trailing dot-separated segments of the field name carry its meaning, for compact display. <code>1</code> for a normal field (show the terminal segment, e.g. <code>lipid</code>); <code>2</code> for a dynamic map key (show field + key, e.g. <code>matchedDatabases.GNPS</code>, <code>qualities.PEAK_QUALITY</code>, <code>tags.pfas</code>); larger for multi-segment keys. Clients may show the last <code>significantSuffixLength</c
name segments as a short label; the full field name always stays authoritative for queries.

`fullTextSearch` If true, the field content is split into words and can be searched word by word (full text search). If false, TEXT fields only match as exact terms (though wildcards and regex are still possible).

`sortable` If true, search results can be sorted by this field.

`defaultSearchField` If true, this field is searched when a query term does not specify a field name.

`possibleValues` The values this field can take, exactly as they are indexed, or null if the field accepts free text. <p> Reported for ENUM and BOOLEAN fields, for TEXT fields that hold a closed vocabulary (the compound class ontologies), and for fields whose values are project state: a tag restricted by its definition, or the adducts detected in this project. Searchable fields are described per project, so the latter are the values actually present in it rather than everything that could ever occur. <p> Note that the indexed value is not always the notation you would expect: an adduct is indexed as '[M + H]+', with spaces. Use the values as offered rather than constructing them.

`description` Optional human-readable description of the field content, as shown in the API documentation. Plain text suitable for direct display (e.g. tooltips); deliberate paragraph breaks are newlines.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SearchableField$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SearchableField$toList()
```

Returns: SearchableField as a base R list.

Examples:

```
# convert array of SearchableField (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SearchableField to a base R type

Usage:

```
SearchableField$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SearchableField

Usage:

```
SearchableField$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableField

Method `toJSONString()`**:** To JSON String

Usage:

```
SearchableField$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SearchableField in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SearchableField

Usage:

```
SearchableField$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableField

Method `validateJSON()`**:** Validate JSON input with respect to SearchableField and throw an exception if invalid

Usage:

```
SearchableField$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SearchableField$toString()
```

Returns: String representation of SearchableField

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SearchableField$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SearchableField$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SearchableField$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SearchableField$toList`
## ------------------------------------------------

# convert array of SearchableField (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SearchableFieldType` *SearchableFieldType*

Description

SearchableFieldType Class

Format

An `R6Class` generator object

Methods

Public methods:

- `SearchableFieldType$new()`
- `SearchableFieldType$toJSON()`
- `SearchableFieldType$fromJSON()`
- `SearchableFieldType$toJSONString()`
- `SearchableFieldType$fromJSONString()`
- `SearchableFieldType$clone()`

Method `new()`**:** Initialize a new SearchableFieldType class.

Usage:

```
SearchableFieldType$new(...)
```

Arguments:

`...` Optional arguments.

Method `toJSON()`**:** To JSON String

Usage:

```
SearchableFieldType$toJSON()
```

Returns: SearchableFieldType in JSON format

Method `fromJSON()`**:** Deserialize JSON string into an instance of SearchableFieldType

Usage:

```
SearchableFieldType$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableFieldType

Method `toJSONString()`**:** To JSON String

Usage:

```
SearchableFieldType$toJSONString()
```

Returns: SearchableFieldType in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SearchableFieldType

Usage:

```
SearchableFieldType$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SearchableFieldType

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`SecurityContext` *SecurityContext*

Description

SecurityContext Class

SecurityContext Class

Format

An `R6Class` generator object

Details

Create a new SecurityContext

Public fields

`principal` SecurityContextPrincipal [optional]

Methods

Public methods:

- SecurityContext$new()
- SecurityContext$toJSON()
- SecurityContext$toList()
- SecurityContext$toSimpleType()
- SecurityContext$fromJSON()
- SecurityContext$toJSONString()
- SecurityContext$fromJSONString()
- SecurityContext$validateJSON()
- SecurityContext$toString()
- SecurityContext$isValid()
- SecurityContext$getInvalidFields()
- SecurityContext$print()
- SecurityContext$clone()

Method `new()`**:** Initialize a new SecurityContext class.

Usage:

```
SecurityContext$new(principal = NULL, ...)
```

Arguments:

`principal` principal

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SecurityContext$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SecurityContext$toList()
```

Returns: SecurityContext as a base R list.

Examples:

```
# convert array of SecurityContext (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SecurityContext to a base R type

Usage:

```
SecurityContext$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SecurityContext

Usage:

```
SecurityContext$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SecurityContext

Method `toJSONString()`**:** To JSON String

Usage:

```
SecurityContext$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SecurityContext in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SecurityContext

Usage:

```
SecurityContext$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SecurityContext

Method `validateJSON()`**:** Validate JSON input with respect to SecurityContext and throw an exception if invalid

Usage:

```
SecurityContext$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SecurityContext$toString()
```

Returns: String representation of SecurityContext

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SecurityContext$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SecurityContext$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SecurityContext$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SecurityContext$toList`
## ------------------------------------------------

# convert array of SecurityContext (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SecurityContextPrincipal`

SecurityContextPrincipal

Description

SecurityContextPrincipal Class

SecurityContextPrincipal Class

Format

An `R6Class` generator object

Details

Create a new SecurityContextPrincipal

Public fields

`name` character [optional]

Methods

Public methods:

- `SecurityContextPrincipal$new()`
- `SecurityContextPrincipal$toJSON()`
- `SecurityContextPrincipal$toList()`
- `SecurityContextPrincipal$toSimpleType()`
- `SecurityContextPrincipal$fromJSON()`
- `SecurityContextPrincipal$toJSONString()`
- `SecurityContextPrincipal$fromJSONString()`
- `SecurityContextPrincipal$validateJSON()`
- `SecurityContextPrincipal$toString()`
- `SecurityContextPrincipal$isValid()`
- `SecurityContextPrincipal$getInvalidFields()`
- `SecurityContextPrincipal$print()`
- `SecurityContextPrincipal$clone()`

Method `new()`**:** Initialize a new SecurityContextPrincipal class.

Usage:

```
SecurityContextPrincipal$new(name = NULL, ...)
```

Arguments:

`name` name

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SecurityContextPrincipal$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SecurityContextPrincipal$toList()
```

Returns: SecurityContextPrincipal as a base R list.

Examples:

```
# convert array of SecurityContextPrincipal (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SecurityContextPrincipal to a base R type

Usage:

```
SecurityContextPrincipal$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SecurityContextPrincipal

Usage:

```
SecurityContextPrincipal$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SecurityContextPrincipal

Method `toJSONString()`**:** To JSON String

Usage:

```
SecurityContextPrincipal$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SecurityContextPrincipal in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SecurityContextPrincipal

Usage:

```
SecurityContextPrincipal$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SecurityContextPrincipal

Method `validateJSON()`**:** Validate JSON input with respect to SecurityContextPrincipal and throw an exception if invalid

Usage:

```
SecurityContextPrincipal$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SecurityContextPrincipal$toString()
```

Returns: String representation of SecurityContextPrincipal

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SecurityContextPrincipal$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SecurityContextPrincipal$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SecurityContextPrincipal$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SecurityContextPrincipal$toList`
## ------------------------------------------------

# convert array of SecurityContextPrincipal (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SimplePeak` *SimplePeak*

Description

SimplePeak Class

SimplePeak Class

Format

An `R6Class` generator object

Details

Create a new SimplePeak

Public fields

`mz` numeric [optional]

`intensity` numeric [optional]

Methods

Public methods:

- `SimplePeak$new()`
- `SimplePeak$toJSON()`
- `SimplePeak$toList()`
- `SimplePeak$toSimpleType()`
- `SimplePeak$fromJSON()`
- `SimplePeak$toJSONString()`
- `SimplePeak$fromJSONString()`
- `SimplePeak$validateJSON()`
- `SimplePeak$toString()`
- `SimplePeak$isValid()`
- `SimplePeak$getInvalidFields()`
- `SimplePeak$print()`
- `SimplePeak$clone()`

Method `new()`**:** Initialize a new SimplePeak class.

Usage:

```
SimplePeak$new(mz = NULL, intensity = NULL, ...)
```

Arguments:

`mz` mz

`intensity` intensity

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SimplePeak$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SimplePeak$toList()
```

Returns: SimplePeak as a base R list.

Examples:

```
# convert array of SimplePeak (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SimplePeak to a base R type

Usage:

```
SimplePeak$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SimplePeak

Usage:

```
SimplePeak$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SimplePeak

Method `toJSONString()`**:** To JSON String

Usage:

```
SimplePeak$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SimplePeak in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SimplePeak

Usage:

```
SimplePeak$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SimplePeak

Method `validateJSON()`**:** Validate JSON input with respect to SimplePeak and throw an exception if invalid

Usage:

```
SimplePeak$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SimplePeak$toString()
```

Returns: String representation of SimplePeak

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SimplePeak$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SimplePeak$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SimplePeak$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SimplePeak$toList`
## ------------------------------------------------

# convert array of SimplePeak (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

Sirius *Sirius*

Description

User/developer friendly parameter subset for the Formula/SIRIUS tool Can use results from Spectral library search tool.

Sirius Class

Format

An `R6Class` generator object

Details

Create a new Sirius

Public fields

`enabled` Indicates whether the tool is enabled. character [optional]

`profile` Instrument specific profile for internal algorithms Just select what comes closest to the instrument that was used for measuring the data. character [optional]

`numberOfCandidates` Number of formula candidates to keep as result list (Formula Candidates). integer [optional]

`numberOfCandidatesPerIonization` Use this parameter if you want to force SIRIUS to report at least NumberOfCandidatesPerIonization results per ionization. if <= 0, this parameter will have no effect and just the top NumberOfCandidates results will be reported. integer [optional]

`massAccuracyMS2ppm` Maximum allowed mass deviation. Only molecular formulas within this mass window are considered. numeric [optional]

`isotopeMs2Settings` Specify how isotope patterns in MS/MS should be handled. <p> FILTER: When filtering is enabled, molecular formulas are excluded if their theoretical isotope pattern does not match the theoretical one, even if their MS/MS pattern has high score. <p> SCORE: Use them for SCORING. To use this the instrument should produce clear MS/MS isotope patterns <p> IGNORE: Ignore that there might be isotope patterns in MS/MS character [optional]

`filterByIsotopePattern` When filtering is enabled, molecular formulas are excluded if their theoretical isotope pattern does not match the measured one, even if their MS/MS pattern has high score. character [optional]

`enforceElGordoFormula` El Gordo may predict that an MS/MS spectrum is a lipid spectrum. If enabled, the corresponding molecular formula will be enforced as a molecular formula candidate. character [optional]

`performBottomUpSearch` If true, molecular formula generation via bottom up search is enabled. character [optional]

`performDenovoBelowMz` Specifies the m/z below which de novo molecular formula generation is enabled. Set to 0 to disable de novo molecular formula generation. numeric [optional]

`formulaSearchDBs` List of structure databases to extract molecular formulas from, to reduce the formula search space. SIRIUS is quite good at de novo formula annotation, so only enable if you have a good reason. list(character) [optional]

`applyFormulaConstraintsToDBAndBottomUpSearch` By default, the formula (element) constraints are only applied to de novo molecular formula generation. If true, the constraints are as well applied to database search and bottom up search. character [optional]

`enforcedFormulaConstraints` These configurations hold the information how to autodetect elements based on the given formula constraints. Note: If the compound is already assigned to a specific molecular formula, this annotation is ignored. <p> Enforced: Enforced elements are always considered character [optional]

`fallbackFormulaConstraints` These configurations hold the information how to autodetect elements based on the given formula constraints. Note: If the compound is already assigned to a specific molecular formula, this annotation is ignored. <p> Fallback: Fallback elements are used, if the auto-detection fails (e.g. no isotope pattern available) character [optional]

`detectableElements` These configurations hold the information how to autodetect elements based on the given formula constraints. Note: If the compound is already assigned to a specific molecular formula, this annotation is ignored. <p> Detectable: Detectable elements are added to the chemical alphabet, if there are indications for them (e.g. in isotope pattern) list(character) [optional]

`ilpTimeout` Timeout [optional]

`useHeuristic` UseHeuristic [optional]

`injectSpecLibMatchFormulas` If true formula candidates that belong to spectral library matches above a certain threshold will be injected and preserved for further analyses, no matter which score they have or which filter is applied character [optional]

`minScoreToInjectSpecLibMatch` Similarity Threshold to inject formula candidates no matter which score/rank they have or which filter settings are applied. If the threshold is >= 0, formula candidates with a reference spectrum similarity above the threshold will be injected. numeric [optional]

`minPeaksToInjectSpecLibMatch` Matching peaks threshold to inject formula candidates no matter which score they have or which filter is applied. integer [optional]

Methods

Public methods:

- `Sirius$new()`
- `Sirius$toJSON()`
- `Sirius$toList()`
- `Sirius$toSimpleType()`
- `Sirius$fromJSON()`
- `Sirius$toJSONString()`
- `Sirius$fromJSONString()`
- `Sirius$validateJSON()`
- `Sirius$toString()`
- `Sirius$isValid()`

- Sirius$getInvalidFields()
- Sirius$print()
- Sirius$clone()

Method `new()`**:** Initialize a new Sirius class.

Usage:

```
Sirius$new(
  enabled = NULL,
  profile = NULL,
  numberOfCandidates = NULL,
  numberOfCandidatesPerIonization = NULL,
  massAccuracyMS2ppm = NULL,
  isotopeMs2Settings = NULL,
  filterByIsotopePattern = NULL,
  enforceElGordoFormula = NULL,
  performBottomUpSearch = NULL,
  performDenovoBelowMz = NULL,
  formulaSearchDBs = NULL,
  applyFormulaConstraintsToDBAndBottomUpSearch = NULL,
  enforcedFormulaConstraints = NULL,
  fallbackFormulaConstraints = NULL,
  detectableElements = NULL,
  ilpTimeout = NULL,
  useHeuristic = NULL,
  injectSpecLibMatchFormulas = NULL,
  minScoreToInjectSpecLibMatch = NULL,
  minPeaksToInjectSpecLibMatch = NULL,
  ...
)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`profile` Instrument specific profile for internal algorithms Just select what comes closest to the instrument that was used for measuring the data.

`numberOfCandidates` Number of formula candidates to keep as result list (Formula Candidates).

`numberOfCandidatesPerIonization` Use this parameter if you want to force SIRIUS to report at least NumberOfCandidatesPerIonization results per ionization. if <= 0, this parameter will have no effect and just the top NumberOfCandidates results will be reported.

`massAccuracyMS2ppm` Maximum allowed mass deviation. Only molecular formulas within this mass window are considered.

`isotopeMs2Settings` Specify how isotope patterns in MS/MS should be handled. <p> FILTER: When filtering is enabled, molecular formulas are excluded if their theoretical isotope pattern does not match the theoretical one, even if their MS/MS pattern has high score. <p> SCORE: Use them for SCORING. To use this the instrument should produce clear MS/MS isotope patterns <p> IGNORE: Ignore that there might be isotope patterns in MS/MS

`filterByIsotopePattern` When filtering is enabled, molecular formulas are excluded if their theoretical isotope pattern does not match the measured one, even if their MS/MS pattern has high score.

`enforceElGordoFormula` El Gordo may predict that an MS/MS spectrum is a lipid spectrum. If enabled, the corresponding molecular formula will be enforced as a molecular formula candidate.

`performBottomUpSearch` If true, molecular formula generation via bottom up search is enabled.

`performDenovoBelowMz` Specifies the m/z below which de novo molecular formula generation is enabled. Set to 0 to disable de novo molecular formula generation.

`formulaSearchDBs` List of structure databases to extract molecular formulas from, to reduce the formula search space. SIRIUS is quite good at de novo formula annotation, so only enable if you have a good reason.

`applyFormulaConstraintsToDBAndBottomUpSearch` By default, the formula (element) constraints are only applied to de novo molecular formula generation. If true, the constraints are as well applied to database search and bottom up search.

`enforcedFormulaConstraints` These configurations hold the information how to autodetect elements based on the given formula constraints. Note: If the compound is already assigned to a specific molecular formula, this annotation is ignored. <p> Enforced: Enforced elements are always considered

`fallbackFormulaConstraints` These configurations hold the information how to autodetect elements based on the given formula constraints. Note: If the compound is already assigned to a specific molecular formula, this annotation is ignored. <p> Fallback: Fallback elements are used, if the auto-detection fails (e.g. no isotope pattern available)

`detectableElements` These configurations hold the information how to autodetect elements based on the given formula constraints. Note: If the compound is already assigned to a specific molecular formula, this annotation is ignored. <p> Detectable: Detectable elements are added to the chemical alphabet, if there are indications for them (e.g. in isotope pattern)

`ilpTimeout` ilpTimeout

`useHeuristic` useHeuristic

`injectSpecLibMatchFormulas` If true formula candidates that belong to spectral library matches above a certain threshold will be injected and preserved for further analyses, no matter which score they have or which filter is applied

`minScoreToInjectSpecLibMatch` Similarity Threshold to inject formula candidates no matter which score/rank they have or which filter settings are applied. If the threshold is >= 0, formula candidates with a reference spectrum similarity above the threshold will be injected.

`minPeaksToInjectSpecLibMatch` Matching peaks threshold to inject formula candidates no matter which score they have or which filter is applied.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Sirius$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Sirius$toList()
```

Returns: Sirius as a base R list.

Examples:

```
# convert array of Sirius (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Sirius to a base R type

Usage:

```
Sirius$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Sirius

Usage:

```
Sirius$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Sirius

Method `toJSONString()`**:** To JSON String

Usage:

```
Sirius$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Sirius in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Sirius

Usage:

```
Sirius$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Sirius

Method `validateJSON()`**:** Validate JSON input with respect to Sirius and throw an exception if invalid

Usage:

```
Sirius$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Sirius$toString()
```

Returns: String representation of Sirius

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Sirius$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Sirius$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Sirius$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Sirius$toList`
## ------------------------------------------------

# convert array of Sirius (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

SpectralLibraryMatch *SpectralLibraryMatch*

Description

SpectralLibraryMatch Class

SpectralLibraryMatch Class

Format

An R6Class generator object

Details

Create a new SpectralLibraryMatch

Public fields

specMatchId character [optional]

rank integer [optional]

similarity Similarity between query and reference spectrum numeric

sharedPeaks Number of shared/matched peaks integer [optional]

sharedPeakMapping List of paired/matched peak indices. Maps indices of peaks from the query spectrum (mass sorted) to indices of matched peaks in the reference spectrum (mass sorted) list(PeakPair) [optional]

querySpectrumIndex integer

dbName character [optional]

dbId character [optional]

uuid integer

splash character [optional]

molecularFormula character [optional]

adduct character [optional]

exactMass numeric [optional]

smiles character [optional]

type character [optional]

inchiKey character

referenceSpectrumType character [optional]

referenceSpectrum BasicSpectrum [optional]

querySpectrumType character [optional]

Methods

Public methods:

- SpectralLibraryMatch$new()
- SpectralLibraryMatch$toJSON()
- SpectralLibraryMatch$toList()
- SpectralLibraryMatch$toSimpleType()
- SpectralLibraryMatch$fromJSON()
- SpectralLibraryMatch$toJSONString()
- SpectralLibraryMatch$fromJSONString()
- SpectralLibraryMatch$validateJSON()
- SpectralLibraryMatch$toString()
- SpectralLibraryMatch$isValid()
- SpectralLibraryMatch$getInvalidFields()
- SpectralLibraryMatch$print()
- SpectralLibraryMatch$clone()

Method `new()`**:** Initialize a new SpectralLibraryMatch class.

Usage:

```
SpectralLibraryMatch$new(
  similarity,
  querySpectrumIndex,
  uuid,
  inchiKey,
  specMatchId = NULL,
  rank = NULL,
  sharedPeaks = NULL,
  sharedPeakMapping = NULL,
  dbName = NULL,
  dbId = NULL,
  splash = NULL,
  molecularFormula = NULL,
  adduct = NULL,
  exactMass = NULL,
  smiles = NULL,
  type = "IDENTITY",
  referenceSpectrumType = "SPECTRUM",
  referenceSpectrum = NULL,
  querySpectrumType = NULL,
  ...
)
```

Arguments:

`similarity` Similarity between query and reference spectrum

`querySpectrumIndex` querySpectrumIndex

`uuid` uuid

`inchiKey` inchiKey

`specMatchId` specMatchId

`rank` rank

`sharedPeaks` Number of shared/matched peaks

`sharedPeakMapping` List of paired/matched peak indices. Maps indices of peaks from the query spectrum (mass sorted) to indices of matched peaks in the reference spectrum (mass sorted)

`dbName` dbName

`dbId` dbId

`splash` splash

`molecularFormula` molecularFormula

`adduct` adduct

`exactMass` exactMass

`smiles` smiles

`type` type. Default to "IDENTITY".

`referenceSpectrumType` referenceSpectrumType. Default to "SPECTRUM".

`referenceSpectrum` referenceSpectrum

`querySpectrumType` querySpectrumType

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SpectralLibraryMatch$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SpectralLibraryMatch$toList()
```

Returns: SpectralLibraryMatch as a base R list.

Examples:

```
# convert array of SpectralLibraryMatch (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SpectralLibraryMatch to a base R type

Usage:

```
SpectralLibraryMatch$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SpectralLibraryMatch

Usage:

```
SpectralLibraryMatch$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectralLibraryMatch

Method `toJSONString()`**:** To JSON String

Usage:

```
SpectralLibraryMatch$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SpectralLibraryMatch in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SpectralLibraryMatch

Usage:

```
SpectralLibraryMatch$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectralLibraryMatch

Method `validateJSON()`**:** Validate JSON input with respect to SpectralLibraryMatch and throw an exception if invalid

Usage:

```
SpectralLibraryMatch$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SpectralLibraryMatch$toString()
```

Returns: String representation of SpectralLibraryMatch

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SpectralLibraryMatch$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SpectralLibraryMatch$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SpectralLibraryMatch$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SpectralLibraryMatch$toList`
## ------------------------------------------------

# convert array of SpectralLibraryMatch (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SpectralLibraryMatchSummary` *SpectralLibraryMatchSummary*

Description

SpectralLibraryMatchSummary Class

SpectralLibraryMatchSummary Class

Format

An `R6Class` generator object

Details

Create a new SpectralLibraryMatchSummary

Public fields

`bestMatch` SpectralLibraryMatch [optional]

`spectralMatchCount` integer

`referenceSpectraCount` integer

`databaseCompoundCount` integer

Methods

Public methods:

- SpectralLibraryMatchSummary$new()
- SpectralLibraryMatchSummary$toJSON()
- SpectralLibraryMatchSummary$toList()
- SpectralLibraryMatchSummary$toSimpleType()
- SpectralLibraryMatchSummary$fromJSON()
- SpectralLibraryMatchSummary$toJSONString()
- SpectralLibraryMatchSummary$fromJSONString()
- SpectralLibraryMatchSummary$validateJSON()
- SpectralLibraryMatchSummary$toString()
- SpectralLibraryMatchSummary$isValid()
- SpectralLibraryMatchSummary$getInvalidFields()
- SpectralLibraryMatchSummary$print()
- SpectralLibraryMatchSummary$clone()

Method `new()`**:** Initialize a new SpectralLibraryMatchSummary class.

Usage:

```
SpectralLibraryMatchSummary$new(
  spectralMatchCount,
  referenceSpectraCount,
  databaseCompoundCount,
  bestMatch = NULL,
  ...
)
```

Arguments:

`spectralMatchCount` spectralMatchCount

`referenceSpectraCount` referenceSpectraCount

`databaseCompoundCount` databaseCompoundCount

`bestMatch` bestMatch

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SpectralLibraryMatchSummary$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SpectralLibraryMatchSummary$toList()
```

Returns: SpectralLibraryMatchSummary as a base R list.

Examples:

```
# convert array of SpectralLibraryMatchSummary (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SpectralLibraryMatchSummary to a base R type

Usage:

```
SpectralLibraryMatchSummary$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SpectralLibraryMatchSummary

Usage:

```
SpectralLibraryMatchSummary$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectralLibraryMatchSummary

Method `toJSONString()`**:** To JSON String

Usage:

```
SpectralLibraryMatchSummary$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SpectralLibraryMatchSummary in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SpectralLibraryMatchSummary

Usage:

```
SpectralLibraryMatchSummary$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectralLibraryMatchSummary

Method `validateJSON()`**:** Validate JSON input with respect to SpectralLibraryMatchSummary and throw an exception if invalid

Usage:

```
SpectralLibraryMatchSummary$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SpectralLibraryMatchSummary$toString()
```

Returns: String representation of SpectralLibraryMatchSummary

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SpectralLibraryMatchSummary$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SpectralLibraryMatchSummary$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SpectralLibraryMatchSummary$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SpectralLibraryMatchSummary$toList`
## ------------------------------------------------

# convert array of SpectralLibraryMatchSummary (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

SpectralLibrarySearch *SpectralLibrarySearch*

Description

User/developer friendly parameter subset for the Spectral library search tool.

SpectralLibrarySearch Class

Format

An R6Class generator object

Details

Create a new SpectralLibrarySearch

Public fields

enabled Indicates whether the tool is enabled. character [optional]

spectraSearchDBs Structure Databases with Reference spectra to search in. <p> Defaults to BIO + Custom Databases. Possible values are available via the Database API. list(character) [optional]

precursorDeviationPpm Maximum allowed mass deviation in ppm for matching the precursor. If not specified, the same value as for the peaks is used. numeric [optional]

minSimilarity Minimal spectral similarity of a spectral match to be considered a hit. numeric [optional]

minNumOfPeaks Minimal number of matching peaks of a spectral match to be considered a hit. integer [optional]

enableAnalogueSearch Enable analogue search in addition to the identity spectral library search character [optional]

minSimilarityAnalogue Minimal spectral similarity of a spectral match to be considered an analogue hit. numeric [optional]

minNumOfPeaksAnalogue Minimal number of matching peaks of a spectral match to be considered an analogue hit. integer [optional]

scoring NO LONGER SUPPORTED (IGNORED) Specify scoring method to match spectra INTENSITY: Intensity weighted. Each peak matches at most one peak in the other spectrum. GAUSSIAN: Treat peaks as (un-normalized) Gaussians and score overlapping areas of PDFs. Each peak might score against multiple peaks in the other spectrum. MODIFIED_COSINE: This algorithm requires that there is at most one pair of peaks (u,v) where the m/z of u and v are within the allowed mass tolerance. To be used for analog search with different precursor masses. character [optional]

peakDeviationPpm NO LONGER SUPPORTED (IGNORED) Maximum allowed mass deviation in ppm for matching peaks. numeric [optional]

Methods

Public methods:

- SpectralLibrarySearch$new()
- SpectralLibrarySearch$toJSON()
- SpectralLibrarySearch$toList()
- SpectralLibrarySearch$toSimpleType()
- SpectralLibrarySearch$fromJSON()
- SpectralLibrarySearch$toJSONString()
- SpectralLibrarySearch$fromJSONString()
- SpectralLibrarySearch$validateJSON()
- SpectralLibrarySearch$toString()
- SpectralLibrarySearch$isValid()
- SpectralLibrarySearch$getInvalidFields()
- SpectralLibrarySearch$print()
- SpectralLibrarySearch$clone()

Method `new()`**:** Initialize a new SpectralLibrarySearch class.

Usage:

```
SpectralLibrarySearch$new(
  enabled = NULL,
  spectraSearchDBs = NULL,
  precursorDeviationPpm = NULL,
  minSimilarity = NULL,
  minNumOfPeaks = NULL,
  enableAnalogueSearch = NULL,
  minSimilarityAnalogue = NULL,
  minNumOfPeaksAnalogue = NULL,
  scoring = NULL,
  peakDeviationPpm = NULL,
  ...
)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`spectraSearchDBs` Structure Databases with Reference spectra to search in. <p> Defaults to BIO + Custom Databases. Possible values are available via the Database API.

`precursorDeviationPpm` Maximum allowed mass deviation in ppm for matching the precursor. If not specified, the same value as for the peaks is used.

`minSimilarity` Minimal spectral similarity of a spectral match to be considered a hit.

`minNumOfPeaks` Minimal number of matching peaks of a spectral match to be considered a hit.

`enableAnalogueSearch` Enable analogue search in addition to the identity spectral library search

`minSimilarityAnalogue` Minimal spectral similarity of a spectral match to be considered an analogue hit.

`minNumOfPeaksAnalogue` Minimal number of matching peaks of a spectral match to be considered an analogue hit.

`scoring` NO LONGER SUPPORTED (IGNORED) Specify scoring method to match spectra INTENSITY: Intensity weighted. Each peak matches at most one peak in the other spectrum. GAUSSIAN: Treat peaks as (un-normalized) Gaussians and score overlapping areas of PDFs. Each peak might score against multiple peaks in the other spectrum. MODIFIED_COSINE: This algorithm requires that there is at most one pair of peaks (u,v) where the m/z of u and v are within the allowed mass tolerance. To be used for analog search with different precursor masses.

`peakDeviationPpm` NO LONGER SUPPORTED (IGNORED) Maximum allowed mass deviation in ppm for matching peaks.

... Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SpectralLibrarySearch$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SpectralLibrarySearch$toList()
```

Returns: SpectralLibrarySearch as a base R list.

Examples:

```
# convert array of SpectralLibrarySearch (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SpectralLibrarySearch to a base R type

Usage:

```
SpectralLibrarySearch$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SpectralLibrarySearch

Usage:

```
SpectralLibrarySearch$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectralLibrarySearch

Method `toJSONString()`**:** To JSON String

Usage:

```
SpectralLibrarySearch$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SpectralLibrarySearch in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SpectralLibrarySearch

Usage:

```
SpectralLibrarySearch$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectralLibrarySearch

Method `validateJSON()`**:** Validate JSON input with respect to SpectralLibrarySearch and throw an exception if invalid

Usage:

```
SpectralLibrarySearch$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SpectralLibrarySearch$toString()
```

Returns: String representation of SpectralLibrarySearch

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SpectralLibrarySearch$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SpectralLibrarySearch$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SpectralLibrarySearch$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SpectralLibrarySearch$toList`
## ------------------------------------------------

# convert array of SpectralLibrarySearch (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`SpectrumAnnotation` *SpectrumAnnotation*

Description

SpectrumAnnotation Class

SpectrumAnnotation Class

Format

An `R6Class` generator object

Details

Create a new SpectrumAnnotation

Public fields

`molecularFormula` Molecular formula that has been annotated to this spectrum character [optional]

`adduct` Adduct that has been annotated to this spectrum character [optional]

`exactMass` Exact mass based on the annotated molecular formula and ionization numeric [optional]

`massDeviationMz` Absolute mass deviation of the exact mass to the precursor mass (precursorMz) of this spectrum in mDa numeric [optional]

`massDeviationPpm` Relative mass deviation of the exact mass to the precursor mass (precursorMz) of this spectrum in ppm numeric [optional]

`structureAnnotationSmiles` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> Smiles of the structure candidate used to derive substructure peak annotations via epimetheus insilico fragmentation Substructure highlighting (bond and atom indices) refer to this specific SMILES. If you standardize or canonicalize this SMILES in any way the indices of substructure highlighting might not match correctly anymore. <p> Null if substructure annotation not available or not requested. character [optional]

`structureAnnotationName` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> Name of the structure candidate used to derive substructure peak annotations via epimetheus insilico fragmentation. <p> Null if substructure annotation not available or not requested. character [optional]

`structureAnnotationSvg` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> SVG graphics of the structure candidate used to derive substructure peak annotations via epimetheus insilico fragmentation Substructure highlighting (bond and atom indices) refers to this SVG. <p> Null if substructure annotation not available or not requested. character [optional]

`structureAnnotationScore` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> Overall score of all substructure annotations computed for this structure candidate (structureAnnotationSmiles) <p> Null if substructure annotation not available or not requested. numeric [optional]

Methods

Public methods:

- `SpectrumAnnotation$new()`
- `SpectrumAnnotation$toJSON()`
- `SpectrumAnnotation$toList()`
- `SpectrumAnnotation$toSimpleType()`
- `SpectrumAnnotation$fromJSON()`
- `SpectrumAnnotation$toJSONString()`
- `SpectrumAnnotation$fromJSONString()`
- `SpectrumAnnotation$validateJSON()`
- `SpectrumAnnotation$toString()`
- `SpectrumAnnotation$isValid()`
- `SpectrumAnnotation$getInvalidFields()`
- `SpectrumAnnotation$print()`
- `SpectrumAnnotation$clone()`

Method `new()`**:** Initialize a new SpectrumAnnotation class.

Usage:

```
SpectrumAnnotation$new(
  molecularFormula = NULL,
  adduct = NULL,
  exactMass = NULL,
  massDeviationMz = NULL,
  massDeviationPpm = NULL,
  structureAnnotationSmiles = NULL,
  structureAnnotationName = NULL,
  structureAnnotationSvg = NULL,
  structureAnnotationScore = NULL,
  ...
)
```

Arguments:

`molecularFormula` Molecular formula that has been annotated to this spectrum

`adduct` Adduct that has been annotated to this spectrum

`exactMass` Exact mass based on the annotated molecular formula and ionization

`massDeviationMz` Absolute mass deviation of the exact mass to the precursor mass (precursorMz) of this spectrum in mDa

`massDeviationPpm` Relative mass deviation of the exact mass to the precursor mass (precursorMz) of this spectrum in ppm

`structureAnnotationSmiles` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> Smiles of the structure candidate used to derive substructure peak annotations via epimetheus insilico fragmentation Substructure highlighting (bond and atom indices) refer to this specific SMILES. If you standardize or canonicalize this SMILES in any way the indices of substructure highlighting might not match correctly anymore. <p> Null if substructure annotation not available or not requested.

`structureAnnotationName` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> Name of the structure candidate used to derive substructure peak annotations via epimetheus insilico fragmentation. <p> Null if substructure annotation not available or not requested.

`structureAnnotationSvg` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> SVG graphics of the structure candidate used to derive substructure peak annotations via epimetheus insilico fragmentation Substructure highlighting (bond and atom indices) refers to this SVG. <p> Null if substructure annotation not available or not requested.

`structureAnnotationScore` EXPERIMENTAL: This field is experimental and may be changed (or even removed) without notice until it is declared stable. <p> Overall score of all substructure annotations computed for this structure candidate (structureAnnotationSmiles) <p> Null if substructure annotation not available or not requested.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SpectrumAnnotation$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SpectrumAnnotation$toList()
```

Returns: SpectrumAnnotation as a base R list.

Examples:

```
# convert array of SpectrumAnnotation (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
```

```
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SpectrumAnnotation to a base R type

Usage:

```
SpectrumAnnotation$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SpectrumAnnotation

Usage:

```
SpectrumAnnotation$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectrumAnnotation

Method `toJSONString()`**:** To JSON String

Usage:

```
SpectrumAnnotation$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SpectrumAnnotation in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SpectrumAnnotation

Usage:

```
SpectrumAnnotation$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SpectrumAnnotation

Method `validateJSON()`**:** Validate JSON input with respect to SpectrumAnnotation and throw an exception if invalid

Usage:

```
SpectrumAnnotation$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SpectrumAnnotation$toString()
```

Returns: String representation of SpectrumAnnotation

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SpectrumAnnotation$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SpectrumAnnotation$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SpectrumAnnotation$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SpectrumAnnotation$toList`
## ------------------------------------------------

# convert array of SpectrumAnnotation (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`StoredJobSubmission` *StoredJobSubmission*

Description

StoredJobSubmission Class

StoredJobSubmission Class

Format

An `R6Class` generator object

Details

Create a new StoredJobSubmission

Public fields

`name` Unique name to identify this JobSubmission (job config). character

`editable` False for predefined configs which are not editable and not removable. character

`jobSubmission` JobSubmission

Methods

Public methods:

- `StoredJobSubmission$new()`
- `StoredJobSubmission$toJSON()`
- `StoredJobSubmission$toList()`
- `StoredJobSubmission$toSimpleType()`
- `StoredJobSubmission$fromJSON()`
- `StoredJobSubmission$toJSONString()`
- `StoredJobSubmission$fromJSONString()`
- `StoredJobSubmission$validateJSON()`
- `StoredJobSubmission$toString()`
- `StoredJobSubmission$isValid()`
- `StoredJobSubmission$getInvalidFields()`
- `StoredJobSubmission$print()`
- `StoredJobSubmission$clone()`

Method `new()`**:** Initialize a new StoredJobSubmission class.

Usage:

```
StoredJobSubmission$new(name, editable, jobSubmission, ...)
```

Arguments:

`name` Unique name to identify this JobSubmission (job config).

`editable` False for predefined configs which are not editable and not removable.

`jobSubmission` jobSubmission

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
StoredJobSubmission$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
StoredJobSubmission$toList()
```

Returns: StoredJobSubmission as a base R list.

Examples:

```
# convert array of StoredJobSubmission (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert StoredJobSubmission to a base R type

Usage:

```
StoredJobSubmission$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of StoredJobSubmission

Usage:

```
StoredJobSubmission$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StoredJobSubmission

Method `toJSONString()`**:** To JSON String

Usage:

```
StoredJobSubmission$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: StoredJobSubmission in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of StoredJobSubmission

Usage:

```
StoredJobSubmission$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StoredJobSubmission

Method `validateJSON()`**:** Validate JSON input with respect to StoredJobSubmission and throw an exception if invalid

Usage:

```
StoredJobSubmission$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
StoredJobSubmission$toString()
```

Returns: String representation of StoredJobSubmission

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
StoredJobSubmission$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
StoredJobSubmission$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
StoredJobSubmission$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StoredJobSubmission$toList`
## ------------------------------------------------

# convert array of StoredJobSubmission (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`StructureCandidate` *StructureCandidate*

Description

StructureCandidate Class

StructureCandidate Class

Format

An `R6Class` generator object

Details

Create a new StructureCandidate

Public fields

`inchiKey` InChIKey of the structure candidate. Searching with a full 27 character key matches its 2D (skeleton) part, so stereoisomers of the searched structure are found alike. character [optional]

`smiles` character [optional]

`structureName` Name of the structure candidate. Searching also resolves the searched term as a PubChem synonym, so a structure is found by any of its common names. character [optional]

`structureSvg` SVG graphics of the structure candidate OPTIONAL: needs to be added by parameter character [optional]

`dbLinks` List of structure database links belonging to the structure candidate OPTIONAL: needs to be added by parameter list(DBLink) [optional]

`spectralLibraryMatches` List of spectral library matches belonging to the structure candidate OPTIONAL: needs to be added by parameter list(SpectralLibraryMatch) [optional]

`xlogP` numeric [optional]

Methods

Public methods:

- `StructureCandidate$new()`
- `StructureCandidate$toJSON()`
- `StructureCandidate$toList()`
- `StructureCandidate$toSimpleType()`
- `StructureCandidate$fromJSON()`
- `StructureCandidate$toJSONString()`
- `StructureCandidate$fromJSONString()`
- `StructureCandidate$validateJSON()`
- `StructureCandidate$toString()`

- StructureCandidate$isValid()
- StructureCandidate$getInvalidFields()
- StructureCandidate$print()
- StructureCandidate$clone()

Method `new()`**:** Initialize a new StructureCandidate class.

Usage:

```
StructureCandidate$new(
  inchiKey = NULL,
  smiles = NULL,
  structureName = NULL,
  structureSvg = NULL,
  dbLinks = NULL,
  spectralLibraryMatches = NULL,
  xlogP = NULL,
  ...
)
```

Arguments:

`inchiKey` InChIKey of the structure candidate. Searching with a full 27 character key matches its 2D (skeleton) part, so stereoisomers of the searched structure are found alike.

`smiles` smiles

`structureName` Name of the structure candidate. Searching also resolves the searched term as a PubChem synonym, so a structure is found by any of its common names.

`structureSvg` SVG graphics of the structure candidate OPTIONAL: needs to be added by parameter

`dbLinks` List of structure database links belonging to the structure candidate OPTIONAL: needs to be added by parameter

`spectralLibraryMatches` List of spectral library matches belonging to the structure candidate OPTIONAL: needs to be added by parameter

`xlogP` xlogP

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
StructureCandidate$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
StructureCandidate$toList()
```

Returns: StructureCandidate as a base R list.

Examples:

```
# convert array of StructureCandidate (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert StructureCandidate to a base R type

Usage:

```
StructureCandidate$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of StructureCandidate

Usage:

```
StructureCandidate$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureCandidate

Method `toJSONString()`**:** To JSON String

Usage:

```
StructureCandidate$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: StructureCandidate in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of StructureCandidate

Usage:

```
StructureCandidate$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureCandidate

Method `validateJSON()`**:** Validate JSON input with respect to StructureCandidate and throw an exception if invalid

Usage:

```
StructureCandidate$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
StructureCandidate$toString()
```

Returns: String representation of StructureCandidate

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
StructureCandidate$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
StructureCandidate$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
StructureCandidate$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StructureCandidate$toList`
## ------------------------------------------------

# convert array of StructureCandidate (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`StructureCandidateFormula` *StructureCandidateFormula*

Description

StructureCandidateFormula Class

StructureCandidateFormula Class

Format

An `R6Class` generator object

Details

Create a new StructureCandidateFormula

Public fields

`inchiKey` InChIKey of the structure candidate. Searching with a full 27 character key matches its 2D (skeleton) part, so stereoisomers of the searched structure are found alike. character [optional]

`smiles` character [optional]

`structureName` Name of the structure candidate. Searching also resolves the searched term as a PubChem synonym, so a structure is found by any of its common names. character [optional]

`structureSvg` SVG graphics of the structure candidate OPTIONAL: needs to be added by parameter character [optional]

`dbLinks` List of structure database links belonging to the structure candidate OPTIONAL: needs to be added by parameter list(DBLink) [optional]

`spectralLibraryMatches` List of spectral library matches belonging to the structure candidate OPTIONAL: needs to be added by parameter list(SpectralLibraryMatch) [optional]

`xlogP` numeric [optional]

`rank` The overall rank of this candidate among all candidates of this feature. integer [optional]

`csiScore` CSI:FingerID score of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID This is the score used for ranking structure candidates numeric [optional]

`tanimotoSimilarity` Tanimoto similarity of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID numeric [optional]

`mcesDistToTopHit` Maximum Common Edge Subgraph (MCES) distance to the top scoring hit (CSI:FingerID) in a candidate list. numeric [optional]

`fingerprint` BinaryFingerprint [optional]

`molecularFormula` Molecular formula of this candidate character [optional]

`adduct` Adduct of this candidate character [optional]

`formulaId` Id of the corresponding Formula candidate character [optional]

Methods

Public methods:

- StructureCandidateFormula$new()
- StructureCandidateFormula$toJSON()
- StructureCandidateFormula$toList()
- StructureCandidateFormula$toSimpleType()
- StructureCandidateFormula$fromJSON()
- StructureCandidateFormula$toJSONString()
- StructureCandidateFormula$fromJSONString()
- StructureCandidateFormula$validateJSON()
- StructureCandidateFormula$toString()
- StructureCandidateFormula$isValid()
- StructureCandidateFormula$getInvalidFields()
- StructureCandidateFormula$print()
- StructureCandidateFormula$clone()

Method `new()`**:** Initialize a new StructureCandidateFormula class.

Usage:

```
StructureCandidateFormula$new(
  inchiKey = NULL,
  smiles = NULL,
  structureName = NULL,
  structureSvg = NULL,
  dbLinks = NULL,
  spectralLibraryMatches = NULL,
  xlogP = NULL,
  rank = NULL,
  csiScore = NULL,
  tanimotoSimilarity = NULL,
  mcesDistToTopHit = NULL,
  fingerprint = NULL,
  molecularFormula = NULL,
  adduct = NULL,
  formulaId = NULL,
  ...
)
```

Arguments:

`inchiKey` InChIKey of the structure candidate. Searching with a full 27 character key matches its 2D (skeleton) part, so stereoisomers of the searched structure are found alike.

`smiles` smiles

`structureName` Name of the structure candidate. Searching also resolves the searched term as a PubChem synonym, so a structure is found by any of its common names.

`structureSvg` SVG graphics of the structure candidate OPTIONAL: needs to be added by parameter

`dbLinks` List of structure database links belonging to the structure candidate OPTIONAL: needs to be added by parameter

`spectralLibraryMatches` List of spectral library matches belonging to the structure candidate OPTIONAL: needs to be added by parameter

`xlogP` xlogP

`rank` The overall rank of this candidate among all candidates of this feature.

`csiScore` CSI:FingerID score of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID This is the score used for ranking structure candidates

`tanimotoSimilarity` Tanimoto similarity of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID

`mcesDistToTopHit` Maximum Common Edge Subgraph (MCES) distance to the top scoring hit (CSI:FingerID) in a candidate list.

`fingerprint` fingerprint

`molecularFormula` Molecular formula of this candidate

`adduct` Adduct of this candidate

`formulaId` Id of the corresponding Formula candidate

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
StructureCandidateFormula$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
StructureCandidateFormula$toList()
```

Returns: StructureCandidateFormula as a base R list.

Examples:

```
# convert array of StructureCandidateFormula (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert StructureCandidateFormula to a base R type

Usage:

```
StructureCandidateFormula$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of StructureCandidateFormula

Usage:

```
StructureCandidateFormula$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureCandidateFormula

Method `toJSONString()`**:** To JSON String

Usage:

```
StructureCandidateFormula$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: StructureCandidateFormula in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of StructureCandidateFormula

Usage:

```
StructureCandidateFormula$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureCandidateFormula

Method `validateJSON()`**:** Validate JSON input with respect to StructureCandidateFormula and throw an exception if invalid

Usage:

```
StructureCandidateFormula$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
StructureCandidateFormula$toString()
```

Returns: String representation of StructureCandidateFormula

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
StructureCandidateFormula$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
StructureCandidateFormula$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
StructureCandidateFormula$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StructureCandidateFormula$toList`
## ------------------------------------------------

# convert array of StructureCandidateFormula (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`StructureCandidateScored`

StructureCandidateScored

Description

StructureCandidateScored Class

StructureCandidateScored Class

Format

An `R6Class` generator object

Details

Create a new StructureCandidateScored

Public fields

`inchiKey` InChIKey of the structure candidate. Searching with a full 27 character key matches its 2D (skeleton) part, so stereoisomers of the searched structure are found alike. character [optional]

`smiles` character [optional]

`structureName` Name of the structure candidate. Searching also resolves the searched term as a PubChem synonym, so a structure is found by any of its common names. character [optional]

`structureSvg` SVG graphics of the structure candidate OPTIONAL: needs to be added by parameter character [optional]

`dbLinks` List of structure database links belonging to the structure candidate OPTIONAL: needs to be added by parameter list(DBLink) [optional]

`spectralLibraryMatches` List of spectral library matches belonging to the structure candidate OPTIONAL: needs to be added by parameter list(SpectralLibraryMatch) [optional]

`xlogP` numeric [optional]

`rank` The overall rank of this candidate among all candidates of this feature. integer [optional]

`csiScore` CSI:FingerID score of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID This is the score used for ranking structure candidates numeric [optional]

`tanimotoSimilarity` Tanimoto similarity of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID numeric [optional]

`mcesDistToTopHit` Maximum Common Edge Subgraph (MCES) distance to the top scoring hit (CSI:FingerID) in a candidate list. numeric [optional]

`fingerprint` BinaryFingerprint [optional]

Methods

Public methods:

- `StructureCandidateScored$new()`
- `StructureCandidateScored$toJSON()`
- `StructureCandidateScored$toList()`
- `StructureCandidateScored$toSimpleType()`
- `StructureCandidateScored$fromJSON()`
- `StructureCandidateScored$toJSONString()`
- `StructureCandidateScored$fromJSONString()`
- `StructureCandidateScored$validateJSON()`
- `StructureCandidateScored$toString()`
- `StructureCandidateScored$isValid()`
- `StructureCandidateScored$getInvalidFields()`
- `StructureCandidateScored$print()`
- `StructureCandidateScored$clone()`

Method `new()`**:** Initialize a new StructureCandidateScored class.

Usage:

```
StructureCandidateScored$new(
  inchiKey = NULL,
  smiles = NULL,
  structureName = NULL,
  structureSvg = NULL,
  dbLinks = NULL,
  spectralLibraryMatches = NULL,
  xlogP = NULL,
  rank = NULL,
  csiScore = NULL,
  tanimotoSimilarity = NULL,
  mcesDistToTopHit = NULL,
  fingerprint = NULL,
  ...
)
```

Arguments:

`inchiKey` InChIKey of the structure candidate. Searching with a full 27 character key matches its 2D (skeleton) part, so stereoisomers of the searched structure are found alike.

`smiles` smiles

`structureName` Name of the structure candidate. Searching also resolves the searched term as a PubChem synonym, so a structure is found by any of its common names.

`structureSvg` SVG graphics of the structure candidate OPTIONAL: needs to be added by parameter

`dbLinks` List of structure database links belonging to the structure candidate OPTIONAL: needs to be added by parameter

`spectralLibraryMatches` List of spectral library matches belonging to the structure candidate OPTIONAL: needs to be added by parameter

`xlogP` xlogP

`rank` The overall rank of this candidate among all candidates of this feature.

`csiScore` CSI:FingerID score of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID This is the score used for ranking structure candidates

`tanimotoSimilarity` Tanimoto similarity of the fingerprint of this compound to the predicted fingerprint of CSI:FingerID

`mcesDistToTopHit` Maximum Common Edge Subgraph (MCES) distance to the top scoring hit (CSI:FingerID) in a candidate list.

`fingerprint` fingerprint

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
StructureCandidateScored$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
StructureCandidateScored$toList()
```

Returns: StructureCandidateScored as a base R list.

Examples:

```
# convert array of StructureCandidateScored (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert StructureCandidateScored to a base R type

Usage:

```
StructureCandidateScored$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of StructureCandidateScored

Usage:

```
StructureCandidateScored$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureCandidateScored

Method `toJSONString()`**:** To JSON String

Usage:

```
StructureCandidateScored$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: StructureCandidateScored in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of StructureCandidateScored

Usage:

```
StructureCandidateScored$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureCandidateScored

Method `validateJSON()`**:** Validate JSON input with respect to StructureCandidateScored and throw an exception if invalid

Usage:

```
StructureCandidateScored$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
StructureCandidateScored$toString()
```

Returns: String representation of StructureCandidateScored

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
StructureCandidateScored$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
StructureCandidateScored$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
StructureCandidateScored$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StructureCandidateScored$toList`
## ------------------------------------------------

# convert array of StructureCandidateScored (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`StructureDbSearch` *StructureDbSearch*

Description

User/developer friendly parameter subset for the CSI:FingerID structure db search tool. Needs results from the FingerprintPrediction and CANOPUS tools. Non-null parameters in this object override their equivalent value in the config map.

StructureDbSearch Class

Format

An `R6Class` generator object

Details

Create a new StructureDbSearch

Public fields

`enabled` Indicates whether the tool is enabled. character [optional]

`structureSearchDBs` Structure databases to search in, If expansive search is enabled this DB selection will be expanded to PubChem if no high confidence hit was found in the selected databases. <p> Defaults to BIO + Custom Databases. Possible values are available via the Database API. list(character) [optional]

`tagStructuresWithLipidClass` Candidates matching the lipid class estimated by El Gordo will be tagged. The lipid class will only be available if El Gordo predicts that the MS/MS is a lipid spectrum. If this parameter is set to 'false' El Gordo will still be executed and e.g. improve the fragmentation tree, but the matching structure candidates will not be tagged if they match lipid class. character [optional]

`expansiveSearchConfidenceMode` Expansive search mode. Expansive search will expand the search space to whole PubChem in case no hit with reasonable confidence was found in one of the specified databases (structureSearchDBs). <p> Possible Values OFF - No expansive search is performed EXACT - Use confidence score in exact mode: Only molecular structures identical to the true structure should count as correct identification. APPROXIMATE - Use confidence score in approximate mode: Molecular structures hits that are close to the true structure should count as correct identification. character [optional]

Methods

Public methods:

- `StructureDbSearch$new()`
- `StructureDbSearch$toJSON()`
- `StructureDbSearch$toList()`
- `StructureDbSearch$toSimpleType()`

- StructureDbSearch$fromJSON()
- StructureDbSearch$toJSONString()
- StructureDbSearch$fromJSONString()
- StructureDbSearch$validateJSON()
- StructureDbSearch$toString()
- StructureDbSearch$isValid()
- StructureDbSearch$getInvalidFields()
- StructureDbSearch$print()
- StructureDbSearch$clone()

Method `new()`**:** Initialize a new StructureDbSearch class.

Usage:

```
StructureDbSearch$new(
  enabled = NULL,
  structureSearchDBs = NULL,
  tagStructuresWithLipidClass = NULL,
  expansiveSearchConfidenceMode = NULL,
  ...
)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`structureSearchDBs` Structure databases to search in, If expansive search is enabled this DB selection will be expanded to PubChem if no high confidence hit was found in the selected databases. <p> Defaults to BIO + Custom Databases. Possible values are available via the Database API.

`tagStructuresWithLipidClass` Candidates matching the lipid class estimated by El Gordo will be tagged. The lipid class will only be available if El Gordo predicts that the MS/MS is a lipid spectrum. If this parameter is set to 'false' El Gordo will still be executed and e.g. improve the fragmentation tree, but the matching structure candidates will not be tagged if they match lipid class.

`expansiveSearchConfidenceMode` Expansive search mode. Expansive search will expand the search space to whole PubChem in case no hit with reasonable confidence was found in one of the specified databases (structureSearchDBs). <p> Possible Values OFF - No expansive search is performed EXACT - Use confidence score in exact mode: Only molecular structures identical to the true structure should count as correct identification. APPROXIMATE - Use confidence score in approximate mode: Molecular structures hits that are close to the true structure should count as correct identification.

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
StructureDbSearch$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
StructureDbSearch$toList()
```

Returns: StructureDbSearch as a base R list.

Examples:

```
# convert array of StructureDbSearch (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert StructureDbSearch to a base R type

Usage:

```
StructureDbSearch$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of StructureDbSearch

Usage:

```
StructureDbSearch$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureDbSearch

Method `toJSONString()`**:** To JSON String

Usage:

```
StructureDbSearch$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: StructureDbSearch in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of StructureDbSearch

Usage:

```
StructureDbSearch$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of StructureDbSearch

Method `validateJSON()`**:** Validate JSON input with respect to StructureDbSearch and throw an exception if invalid

Usage:

```
StructureDbSearch$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
StructureDbSearch$toString()
```

Returns: String representation of StructureDbSearch

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
StructureDbSearch$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
StructureDbSearch$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
StructureDbSearch$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `StructureDbSearch$toList`
## ------------------------------------------------

# convert array of StructureDbSearch (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Subscription` *Subscription*

Description

Subscription Class

Subscription Class

Format

An `R6Class` generator object

Details

Create a new Subscription

Public fields

`sid` Unique identifier of this subscription character [optional]

`subscriberId` ID of the owner of the subscription. This can be the ID of any SubscriptionOwner (e.g. TagGroup or User) depending on the level at which a subscription is valid. character [optional]

`subscriberName` Optional name of the owner of this subscription character [optional]

`expirationDate` character [optional]

`startDate` character [optional]

`countQueries` character [optional]

`instanceLimit` Limit of instances (features) that can be computed with this subscription integer [optional]

`instanceHashRecordingTime` Hash is used to allow recomputing identical data without increasing counted instances (features). The recording time is the amount of time an instance is memorized. integer [optional]

`maxQueriesPerInstance` Maximum number of queries (e.g. prediction) that can be performed for one instance before it is counted another time. integer [optional]

`maxUserAccounts` integer [optional]

`serviceUrl` character [optional]

`description` character [optional]

`name` character [optional]

`tos` character [optional]

`pp` character [optional]

`allowedFeatures` AllowedFeatures [optional]

Methods

Public methods:

- Subscription$new()
- Subscription$toJSON()
- Subscription$toList()
- Subscription$toSimpleType()
- Subscription$fromJSON()
- Subscription$toJSONString()
- Subscription$fromJSONString()
- Subscription$validateJSON()
- Subscription$toString()
- Subscription$isValid()
- Subscription$getInvalidFields()
- Subscription$print()
- Subscription$clone()

Method `new()`**:** Initialize a new Subscription class.

Usage:

```
Subscription$new(
  sid = NULL,
  subscriberId = NULL,
  subscriberName = NULL,
  expirationDate = NULL,
  startDate = NULL,
  countQueries = NULL,
  instanceLimit = NULL,
  instanceHashRecordingTime = NULL,
  maxQueriesPerInstance = NULL,
  maxUserAccounts = NULL,
  serviceUrl = NULL,
  description = NULL,
  name = NULL,
  tos = NULL,
  pp = NULL,
  allowedFeatures = NULL,
  ...
)
```

Arguments:

`sid` Unique identifier of this subscription

`subscriberId` ID of the owner of the subscription. This can be the ID of any SubscriptionOwner (e.g. TagGroup or User) depending on the level at which a subscription is valid.

`subscriberName` Optional name of the owner of this subscription

`expirationDate` expirationDate

`startDate` startDate

`countQueries` countQueries

`instanceLimit` Limit of instances (features) that can be computed with this subscription

`instanceHashRecordingTime` Hash is used to allow recomputing identical data without increasing counted instances (features). The recording time is the amount of time an instance is memorized.

`maxQueriesPerInstance` Maximum number of queries (e.g. prediction) that can be performed for one instance before it is counted another time.

`maxUserAccounts` maxUserAccounts

`serviceUrl` serviceUrl

`description` description

`name` name

`tos` tos

`pp` pp

`allowedFeatures` allowedFeatures

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Subscription$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Subscription$toList()
```

Returns: Subscription as a base R list.

Examples:

```
# convert array of Subscription (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Subscription to a base R type

Usage:

```
Subscription$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Subscription

Usage:

```
Subscription$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Subscription

Method `toJSONString()`**:** To JSON String

Usage:

```
Subscription$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Subscription in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Subscription

Usage:

```
Subscription$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Subscription

Method `validateJSON()`**:** Validate JSON input with respect to Subscription and throw an exception if invalid

Usage:

```
Subscription$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Subscription$toString()
```

Returns: String representation of Subscription

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Subscription$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Subscription$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Subscription$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Subscription$toList`
## ------------------------------------------------

# convert array of Subscription (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

SubscriptionConsumables *SubscriptionConsumables*

Description

SubscriptionConsumables Class

SubscriptionConsumables Class

Format

An `R6Class` generator object

Details

Create a new SubscriptionConsumables

Public fields

`pendingJobs` integer [optional]

`countedCompounds` integer [optional]

Methods

Public methods:

- `SubscriptionConsumables$new()`
- `SubscriptionConsumables$toJSON()`
- `SubscriptionConsumables$toList()`
- `SubscriptionConsumables$toSimpleType()`
- `SubscriptionConsumables$fromJSON()`
- `SubscriptionConsumables$toJSONString()`
- `SubscriptionConsumables$fromJSONString()`

- SubscriptionConsumables$validateJSON()
- SubscriptionConsumables$toString()
- SubscriptionConsumables$isValid()
- SubscriptionConsumables$getInvalidFields()
- SubscriptionConsumables$print()
- SubscriptionConsumables$clone()

Method `new()`**:** Initialize a new SubscriptionConsumables class.

Usage:

```
SubscriptionConsumables$new(pendingJobs = NULL, countedCompounds = NULL, ...)
```

Arguments:

`pendingJobs` pendingJobs

`countedCompounds` countedCompounds

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
SubscriptionConsumables$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
SubscriptionConsumables$toList()
```

Returns: SubscriptionConsumables as a base R list.

Examples:

```
# convert array of SubscriptionConsumables (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert SubscriptionConsumables to a base R type

Usage:

```
SubscriptionConsumables$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of SubscriptionConsumables

Usage:

```
SubscriptionConsumables$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SubscriptionConsumables

Method `toJSONString()`**:** To JSON String

Usage:

```
SubscriptionConsumables$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: SubscriptionConsumables in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of SubscriptionConsumables

Usage:

```
SubscriptionConsumables$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of SubscriptionConsumables

Method `validateJSON()`**:** Validate JSON input with respect to SubscriptionConsumables and throw an exception if invalid

Usage:

```
SubscriptionConsumables$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
SubscriptionConsumables$toString()
```

Returns: String representation of SubscriptionConsumables

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
SubscriptionConsumables$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
SubscriptionConsumables$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
SubscriptionConsumables$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `SubscriptionConsumables$toList`
## ------------------------------------------------

# convert array of SubscriptionConsumables (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Tag` *Tag*

Description

Tag Class

Tag Class

Format

An `R6Class` generator object

Details

Create a new Tag

Public fields

`tagName` Name of the tag as defined by the corresponding TagDefinition Links the tag object to its definition. character

`value` AnyValue [optional]

Methods

Public methods:

- Tag$new()
- Tag$toJSON()
- Tag$toList()
- Tag$toSimpleType()
- Tag$fromJSON()
- Tag$toJSONString()
- Tag$fromJSONString()
- Tag$validateJSON()
- Tag$toString()
- Tag$isValid()
- Tag$getInvalidFields()
- Tag$print()
- Tag$clone()

Method `new()`**:** Initialize a new Tag class.

Usage:

```
Tag$new(tagName, value = NULL, ...)
```

Arguments:

`tagName` Name of the tag as defined by the corresponding TagDefinition Links the tag object to its definition.

`value` value

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Tag$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Tag$toList()
```

Returns: Tag as a base R list.

Examples:

```
# convert array of Tag (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Tag to a base R type

Usage:

```
Tag$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Tag

Usage:

```
Tag$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Tag

Method `toJSONString()`**:** To JSON String

Usage:

```
Tag$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Tag in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Tag

Usage:

```
Tag$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Tag

Method `validateJSON()`**:** Validate JSON input with respect to Tag and throw an exception if invalid

Usage:

```
Tag$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Tag$toString()
```

Returns: String representation of Tag

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Tag$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Tag$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Tag$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Tag$toList`
## ------------------------------------------------

# convert array of Tag (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Term` *Term*

Description

Term Class

Term Class

Format

An `R6Class` generator object

Details

Create a new Term

Public fields

`name` character [optional]

`link` character [optional]

Methods

Public methods:

- `Term$new()`
- `Term$toJSON()`
- `Term$toList()`
- `Term$toSimpleType()`
- `Term$fromJSON()`
- `Term$toJSONString()`
- `Term$fromJSONString()`
- `Term$validateJSON()`
- `Term$toString()`
- `Term$isValid()`
- `Term$getInvalidFields()`
- `Term$print()`
- `Term$clone()`

Method `new()`**:** Initialize a new Term class.

Usage:

```
Term$new(name = NULL, link = NULL, ...)
```

Arguments:

`name` name

`link` link

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Term$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Term$toList()
```

Returns: Term as a base R list.

Examples:

```
# convert array of Term (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Term to a base R type

Usage:

```
Term$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Term

Usage:

```
Term$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Term

Method `toJSONString()`**:** To JSON String

Usage:

```
Term$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Term in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Term

Usage:

```
Term$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Term

Method `validateJSON()`**:** Validate JSON input with respect to Term and throw an exception if invalid

Usage:

```
Term$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Term$toString()
```

Returns: String representation of Term

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Term$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Term$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Term$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Term$toList`
## ------------------------------------------------

# convert array of Term (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

Timeout *Timeout*

Description

Timeout Class

Timeout Class

Format

An R6Class generator object

Details

Create a new Timeout

Public fields

numberOfSecondsPerDecomposition integer [optional]

numberOfSecondsPerInstance integer [optional]

Methods

Public methods:

- Timeout$new()
- Timeout$toJSON()
- Timeout$toList()
- Timeout$toSimpleType()
- Timeout$fromJSON()
- Timeout$toJSONString()
- Timeout$fromJSONString()
- Timeout$validateJSON()
- Timeout$toString()
- Timeout$isValid()
- Timeout$getInvalidFields()
- Timeout$print()
- Timeout$clone()

Method new(): Initialize a new Timeout class.

Usage:

```
Timeout$new(
  numberOfSecondsPerDecomposition = NULL,
  numberOfSecondsPerInstance = NULL,
  ...
)
```

Arguments:

`numberOfSecondsPerDecomposition` numberOfSecondsPerDecomposition

`numberOfSecondsPerInstance` numberOfSecondsPerInstance

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Timeout$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Timeout$toList()
```

Returns: Timeout as a base R list.

Examples:

```
# convert array of Timeout (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Timeout to a base R type

Usage:

```
Timeout$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Timeout

Usage:

```
Timeout$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Timeout

Method `toJSONString()`**:** To JSON String

Usage:

```
Timeout$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Timeout in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Timeout

Usage:

```
Timeout$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Timeout

Method `validateJSON()`**:** Validate JSON input with respect to Timeout and throw an exception if invalid

Usage:

```
Timeout$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Timeout$toString()
```

Returns: String representation of Timeout

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Timeout$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Timeout$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Timeout$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Timeout$toList`
## ------------------------------------------------

# convert array of Timeout (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`UseHeuristic` *UseHeuristic*

Description

UseHeuristic Class

UseHeuristic Class

Format

An `R6Class` generator object

Details

Create a new UseHeuristic

Public fields

`useHeuristicAboveMz` integer [optional]

`useOnlyHeuristicAboveMz` integer [optional]

Methods

Public methods:

- `UseHeuristic$new()`
- `UseHeuristic$toJSON()`
- `UseHeuristic$toList()`
- `UseHeuristic$toSimpleType()`
- `UseHeuristic$fromJSON()`
- `UseHeuristic$toJSONString()`
- `UseHeuristic$fromJSONString()`
- `UseHeuristic$validateJSON()`

- UseHeuristic$toString()
- UseHeuristic$isValid()
- UseHeuristic$getInvalidFields()
- UseHeuristic$print()
- UseHeuristic$clone()

Method `new()`**:** Initialize a new UseHeuristic class.

Usage:

```
UseHeuristic$new(
  useHeuristicAboveMz = NULL,
  useOnlyHeuristicAboveMz = NULL,
  ...
)
```

Arguments:

`useHeuristicAboveMz` useHeuristicAboveMz

`useOnlyHeuristicAboveMz` useOnlyHeuristicAboveMz

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
UseHeuristic$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
UseHeuristic$toList()
```

Returns: UseHeuristic as a base R list.

Examples:

```
# convert array of UseHeuristic (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert UseHeuristic to a base R type

Usage:

```
UseHeuristic$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of UseHeuristic

Usage:

```
UseHeuristic$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of UseHeuristic

Method `toJSONString()`**:** To JSON String

Usage:

```
UseHeuristic$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: UseHeuristic in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of UseHeuristic

Usage:

```
UseHeuristic$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of UseHeuristic

Method `validateJSON()`**:** Validate JSON input with respect to UseHeuristic and throw an exception if invalid

Usage:

```
UseHeuristic$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
UseHeuristic$toString()
```

Returns: String representation of UseHeuristic

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
UseHeuristic$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
UseHeuristic$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
UseHeuristic$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `UseHeuristic$toList`
## ------------------------------------------------

# convert array of UseHeuristic (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`WebServerNamespace` *WebServerNamespace*

Description

WebServerNamespace Class

WebServerNamespace Class

Format

An `R6Class` generator object

Details

Create a new WebServerNamespace

Public fields

`value` character [optional]

Methods

Public methods:

- WebServerNamespace$new()
- WebServerNamespace$toJSON()
- WebServerNamespace$toList()
- WebServerNamespace$toSimpleType()
- WebServerNamespace$fromJSON()
- WebServerNamespace$toJSONString()
- WebServerNamespace$fromJSONString()
- WebServerNamespace$validateJSON()
- WebServerNamespace$toString()
- WebServerNamespace$isValid()
- WebServerNamespace$getInvalidFields()
- WebServerNamespace$print()
- WebServerNamespace$clone()

Method `new()`**:** Initialize a new WebServerNamespace class.

Usage:

```
WebServerNamespace$new(value = NULL, ...)
```

Arguments:

`value` value

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
WebServerNamespace$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
WebServerNamespace$toList()
```

Returns: WebServerNamespace as a base R list.

Examples:

```
# convert array of WebServerNamespace (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert WebServerNamespace to a base R type

Usage:

```
WebServerNamespace$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of WebServerNamespace

Usage:

```
WebServerNamespace$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of WebServerNamespace

Method `toJSONString()`**:** To JSON String

Usage:

```
WebServerNamespace$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: WebServerNamespace in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of WebServerNamespace

Usage:

```
WebServerNamespace$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of WebServerNamespace

Method `validateJSON()`**:** Validate JSON input with respect to WebServerNamespace and throw an exception if invalid

Usage:

```
WebServerNamespace$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
WebServerNamespace$toString()
```

Returns: String representation of WebServerNamespace

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
WebServerNamespace$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
WebServerNamespace$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
WebServerNamespace$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `WebServerNamespace$toList`
## ------------------------------------------------

# convert array of WebServerNamespace (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`Zodiac` *Zodiac*

Description

User/developer friendly parameter subset for the ZODIAC tool (network-based molecular formula re-ranking). Needs results from the Formula/SIRIUS tool.

Zodiac Class

Format

An `R6Class` generator object

Details

Create a new Zodiac

Public fields

`enabled` Indicates whether the tool is enabled. character [optional]

`consideredCandidatesAt300Mz` Maximum number of candidate molecular formulas (fragmentation trees computed by SIRIUS) per compound which are considered by ZODIAC for compounds below 300 m/z. integer [optional]

`consideredCandidatesAt800Mz` Maximum number of candidate molecular formulas (fragmentation trees computed by SIRIUS) per compound which are considered by ZODIAC for compounds above 800 m/z. integer [optional]

`runInTwoSteps` By default, ZODIAC runs a 2-step approach: first running only 'good quality compounds', and afterwards including the remaining ones. character [optional]

`edgeFilterThresholds` ZodiacEdgeFilterThresholds [optional]

`gibbsSamplerParameters` ZodiacEpochs [optional]

`librarySearchAnchors` ZodiacLibraryScoring [optional]

`analogueSearchAnchors` ZodiacAnalogueNodes [optional]

Methods

Public methods:

- `Zodiac$new()`
- `Zodiac$toJSON()`
- `Zodiac$toList()`
- `Zodiac$toSimpleType()`
- `Zodiac$fromJSON()`
- `Zodiac$toJSONString()`
- `Zodiac$fromJSONString()`
- `Zodiac$validateJSON()`
- `Zodiac$toString()`
- `Zodiac$isValid()`
- `Zodiac$getInvalidFields()`
- `Zodiac$print()`
- `Zodiac$clone()`

Method `new()`**:** Initialize a new Zodiac class.

Usage:

```
Zodiac$new(
  enabled = NULL,
  consideredCandidatesAt300Mz = NULL,
  consideredCandidatesAt800Mz = NULL,
  runInTwoSteps = NULL,
  edgeFilterThresholds = NULL,
  gibbsSamplerParameters = NULL,
  librarySearchAnchors = NULL,
  analogueSearchAnchors = NULL,
  ...
)
```

Arguments:

`enabled` Indicates whether the tool is enabled.

`consideredCandidatesAt300Mz` Maximum number of candidate molecular formulas (fragmentation trees computed by SIRIUS) per compound which are considered by ZODIAC for compounds below 300 m/z.

`consideredCandidatesAt800Mz` Maximum number of candidate molecular formulas (fragmentation trees computed by SIRIUS) per compound which are considered by ZODIAC for compounds above 800 m/z.

`runInTwoSteps` By default, ZODIAC runs a 2-step approach: first running only 'good quality compounds', and afterwards including the remaining ones.

`edgeFilterThresholds` edgeFilterThresholds

`gibbsSamplerParameters` gibbsSamplerParameters

`librarySearchAnchors` librarySearchAnchors

`analogueSearchAnchors` analogueSearchAnchors

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
Zodiac$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
Zodiac$toList()
```

Returns: Zodiac as a base R list.

Examples:

```
# convert array of Zodiac (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert Zodiac to a base R type

Usage:

```
Zodiac$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of Zodiac

Usage:

```
Zodiac$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Zodiac

Method `toJSONString()`**:** To JSON String

Usage:

```
Zodiac$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: Zodiac in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of Zodiac

Usage:

```
Zodiac$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of Zodiac

Method `validateJSON()`**:** Validate JSON input with respect to Zodiac and throw an exception if invalid

Usage:

```
Zodiac$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
Zodiac$toString()
```

Returns: String representation of Zodiac

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
Zodiac$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
Zodiac$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
Zodiac$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `Zodiac$toList`
## ------------------------------------------------

# convert array of Zodiac (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ZodiacAnalogueNodes` *ZodiacAnalogueNodes*

Description

ZodiacAnalogueNodes Class

ZodiacAnalogueNodes Class

Format

An `R6Class` generator object

Details

Create a new ZodiacAnalogueNodes

Public fields

`enabled` character [optional]

`minSimilarity` numeric [optional]

`minSharedPeaks` integer [optional]

Methods

Public methods:

- `ZodiacAnalogueNodes$new()`
- `ZodiacAnalogueNodes$toJSON()`
- `ZodiacAnalogueNodes$toList()`
- `ZodiacAnalogueNodes$toSimpleType()`
- `ZodiacAnalogueNodes$fromJSON()`
- `ZodiacAnalogueNodes$toJSONString()`
- `ZodiacAnalogueNodes$fromJSONString()`

- ZodiacAnalogueNodes$validateJSON()
- ZodiacAnalogueNodes$toString()
- ZodiacAnalogueNodes$isValid()
- ZodiacAnalogueNodes$getInvalidFields()
- ZodiacAnalogueNodes$print()
- ZodiacAnalogueNodes$clone()

Method `new()`**:** Initialize a new ZodiacAnalogueNodes class.

Usage:

```
ZodiacAnalogueNodes$new(
  enabled = NULL,
  minSimilarity = NULL,
  minSharedPeaks = NULL,
  ...
)
```

Arguments:

`enabled` enabled

`minSimilarity` minSimilarity

`minSharedPeaks` minSharedPeaks

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ZodiacAnalogueNodes$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ZodiacAnalogueNodes$toList()
```

Returns: ZodiacAnalogueNodes as a base R list.

Examples:

```
# convert array of ZodiacAnalogueNodes (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ZodiacAnalogueNodes to a base R type

Usage:

```
ZodiacAnalogueNodes$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ZodiacAnalogueNodes

Usage:

```
ZodiacAnalogueNodes$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacAnalogueNodes

Method `toJSONString()`**:** To JSON String

Usage:

```
ZodiacAnalogueNodes$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ZodiacAnalogueNodes in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ZodiacAnalogueNodes

Usage:

```
ZodiacAnalogueNodes$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacAnalogueNodes

Method `validateJSON()`**:** Validate JSON input with respect to ZodiacAnalogueNodes and throw an exception if invalid

Usage:

```
ZodiacAnalogueNodes$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ZodiacAnalogueNodes$toString()
```

Returns: String representation of ZodiacAnalogueNodes

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ZodiacAnalogueNodes$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ZodiacAnalogueNodes$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ZodiacAnalogueNodes$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ZodiacAnalogueNodes$toList`
## ------------------------------------------------

# convert array of ZodiacAnalogueNodes (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ZodiacEdgeFilterThresholds`

ZodiacEdgeFilterThresholds

Description

ZodiacEdgeFilterThresholds Class

ZodiacEdgeFilterThresholds Class

Format

An `R6Class` generator object

Details

Create a new ZodiacEdgeFilterThresholds

Public fields

`thresholdFilter` numeric [optional]

`minLocalCandidates` integer [optional]

`minLocalConnections` integer [optional]

Methods

Public methods:

- ZodiacEdgeFilterThresholds$new()
- ZodiacEdgeFilterThresholds$toJSON()
- ZodiacEdgeFilterThresholds$toList()
- ZodiacEdgeFilterThresholds$toSimpleType()
- ZodiacEdgeFilterThresholds$fromJSON()
- ZodiacEdgeFilterThresholds$toJSONString()
- ZodiacEdgeFilterThresholds$fromJSONString()
- ZodiacEdgeFilterThresholds$validateJSON()
- ZodiacEdgeFilterThresholds$toString()
- ZodiacEdgeFilterThresholds$isValid()
- ZodiacEdgeFilterThresholds$getInvalidFields()
- ZodiacEdgeFilterThresholds$print()
- ZodiacEdgeFilterThresholds$clone()

Method `new()`**:** Initialize a new ZodiacEdgeFilterThresholds class.

Usage:

```
ZodiacEdgeFilterThresholds$new(
  thresholdFilter = NULL,
  minLocalCandidates = NULL,
  minLocalConnections = NULL,
  ...
)
```

Arguments:

`thresholdFilter` thresholdFilter

`minLocalCandidates` minLocalCandidates

`minLocalConnections` minLocalConnections

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ZodiacEdgeFilterThresholds$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ZodiacEdgeFilterThresholds$toList()
```

Returns: ZodiacEdgeFilterThresholds as a base R list.

Examples:

```
# convert array of ZodiacEdgeFilterThresholds (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ZodiacEdgeFilterThresholds to a base R type

Usage:

```
ZodiacEdgeFilterThresholds$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ZodiacEdgeFilterThresholds

Usage:

```
ZodiacEdgeFilterThresholds$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacEdgeFilterThresholds

Method `toJSONString()`**:** To JSON String

Usage:

```
ZodiacEdgeFilterThresholds$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ZodiacEdgeFilterThresholds in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ZodiacEdgeFilterThresholds

Usage:

```
ZodiacEdgeFilterThresholds$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacEdgeFilterThresholds

Method `validateJSON()`**:** Validate JSON input with respect to ZodiacEdgeFilterThresholds and throw an exception if invalid

Usage:

```
ZodiacEdgeFilterThresholds$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ZodiacEdgeFilterThresholds$toString()
```

Returns: String representation of ZodiacEdgeFilterThresholds

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ZodiacEdgeFilterThresholds$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ZodiacEdgeFilterThresholds$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ZodiacEdgeFilterThresholds$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ZodiacEdgeFilterThresholds$toList`
## ------------------------------------------------

# convert array of ZodiacEdgeFilterThresholds (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

ZodiacEpochs *ZodiacEpochs*

Description

ZodiacEpochs Class

ZodiacEpochs Class

Format

An `R6Class` generator object

Details

Create a new ZodiacEpochs

Public fields

`iterations` integer [optional]

`burnInPeriod` integer [optional]

`numberOfMarkovChains` integer [optional]

Methods

Public methods:

- `ZodiacEpochs$new()`
- `ZodiacEpochs$toJSON()`
- `ZodiacEpochs$toList()`
- `ZodiacEpochs$toSimpleType()`
- `ZodiacEpochs$fromJSON()`
- `ZodiacEpochs$toJSONString()`
- `ZodiacEpochs$fromJSONString()`
- `ZodiacEpochs$validateJSON()`
- `ZodiacEpochs$toString()`
- `ZodiacEpochs$isValid()`
- `ZodiacEpochs$getInvalidFields()`
- `ZodiacEpochs$print()`
- `ZodiacEpochs$clone()`

Method `new()`**:** Initialize a new ZodiacEpochs class.

Usage:

```
ZodiacEpochs$new(
  iterations = NULL,
  burnInPeriod = NULL,
  numberOfMarkovChains = NULL,
  ...
)
```

Arguments:

`iterations` iterations

`burnInPeriod` burnInPeriod

`numberOfMarkovChains` numberOfMarkovChains

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ZodiacEpochs$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ZodiacEpochs$toList()
```

Returns: ZodiacEpochs as a base R list.

Examples:

```
# convert array of ZodiacEpochs (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ZodiacEpochs to a base R type

Usage:

```
ZodiacEpochs$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ZodiacEpochs

Usage:

```
ZodiacEpochs$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacEpochs

Method `toJSONString()`**:** To JSON String

Usage:

```
ZodiacEpochs$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ZodiacEpochs in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ZodiacEpochs

Usage:

```
ZodiacEpochs$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacEpochs

Method `validateJSON()`**:** Validate JSON input with respect to ZodiacEpochs and throw an exception if invalid

Usage:

```
ZodiacEpochs$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ZodiacEpochs$toString()
```

Returns: String representation of ZodiacEpochs

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ZodiacEpochs$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ZodiacEpochs$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ZodiacEpochs$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ZodiacEpochs$toList`
## ------------------------------------------------

# convert array of ZodiacEpochs (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

`ZodiacLibraryScoring` *ZodiacLibraryScoring*

Description

ZodiacLibraryScoring Class

ZodiacLibraryScoring Class

Format

An `R6Class` generator object

Details

Create a new ZodiacLibraryScoring

Public fields

`enabled` character [optional]

`minSimilarity` numeric [optional]

Methods

Public methods:

- `ZodiacLibraryScoring$new()`
- `ZodiacLibraryScoring$toJSON()`
- `ZodiacLibraryScoring$toList()`
- `ZodiacLibraryScoring$toSimpleType()`
- `ZodiacLibraryScoring$fromJSON()`
- `ZodiacLibraryScoring$toJSONString()`
- `ZodiacLibraryScoring$fromJSONString()`
- `ZodiacLibraryScoring$validateJSON()`

- ZodiacLibraryScoring$toString()
- ZodiacLibraryScoring$isValid()
- ZodiacLibraryScoring$getInvalidFields()
- ZodiacLibraryScoring$print()
- ZodiacLibraryScoring$clone()

Method `new()`**:** Initialize a new ZodiacLibraryScoring class.

Usage:

```
ZodiacLibraryScoring$new(enabled = NULL, minSimilarity = NULL, ...)
```

Arguments:

`enabled` enabled

`minSimilarity` minSimilarity

`...` Other optional arguments.

Method `toJSON()`**:** Convert to an R object. This method is deprecated. Use 'toSimpleType()' instead.

Usage:

```
ZodiacLibraryScoring$toJSON()
```

Method `toList()`**:** Convert to a List

Convert the R6 object to a list to work more easily with other tooling.

Usage:

```
ZodiacLibraryScoring$toList()
```

Returns: ZodiacLibraryScoring as a base R list.

Examples:

```
# convert array of ZodiacLibraryScoring (x) to a data frame
\dontrun{
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df
}
```

Method `toSimpleType()`**:** Convert ZodiacLibraryScoring to a base R type

Usage:

```
ZodiacLibraryScoring$toSimpleType()
```

Returns: A base R type, e.g. a list or numeric/character array.

Method `fromJSON()`**:** Deserialize JSON string into an instance of ZodiacLibraryScoring

Usage:

```
ZodiacLibraryScoring$fromJSON(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacLibraryScoring

Method `toJSONString()`**:** To JSON String

Usage:

```
ZodiacLibraryScoring$toJSONString(...)
```

Arguments:

`...` Parameters passed to 'jsonlite::toJSON'

Returns: ZodiacLibraryScoring in JSON format

Method `fromJSONString()`**:** Deserialize JSON string into an instance of ZodiacLibraryScoring

Usage:

```
ZodiacLibraryScoring$fromJSONString(input_json)
```

Arguments:

`input_json` the JSON input

Returns: the instance of ZodiacLibraryScoring

Method `validateJSON()`**:** Validate JSON input with respect to ZodiacLibraryScoring and throw an exception if invalid

Usage:

```
ZodiacLibraryScoring$validateJSON(input)
```

Arguments:

`input` the JSON input

Method `toString()`**:** To string (JSON format)

Usage:

```
ZodiacLibraryScoring$toString()
```

Returns: String representation of ZodiacLibraryScoring

Method `isValid()`**:** Return true if the values in all fields are valid.

Usage:

```
ZodiacLibraryScoring$isValid()
```

Returns: true if the values in all fields are valid.

Method `getInvalidFields()`**:** Return a list of invalid fields (if any).

Usage:

```
ZodiacLibraryScoring$getInvalidFields()
```

Returns: A list of invalid fields (if any).

Method `print()`**:** Print the object

Usage:

```
ZodiacLibraryScoring$print()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## ------------------------------------------------
## Method `ZodiacLibraryScoring$toList`
## ------------------------------------------------

# convert array of ZodiacLibraryScoring (x) to a data frame
## Not run:
library(purrr)
library(tibble)
df <- x |> map(\(y)y$toList()) |> map(as_tibble) |> list_rbind()
df

## End(Not run)
```

Index

AccountCredentials, 4
AccountInfo, 8
ActuatorApi, 11
AlignedFeature, 13, *65*, *271*
AllowedFeatures, 18, *435*
AnnotatedMsMsData, 22
AnnotatedPeak, 25, *29*
AnnotatedSpectrum, *22*, 28, *181*
AnyValue, 33, *442*
ApiClient, 35
ApiResponse, 39

BackgroundComputationsStateEvent, 40
BasicSpectrum, 44, *126*, *209*, *264*, *398*
BinaryFingerprint, 48, *422*, *427*
BioTransformerParameters, 51
BioTransformerSequenceStep, *52*, 55

Canopus, 58, *237*
CanopusPrediction, 61, *181*
Compound, 65, *274*
CompoundClass, *62*, 69, *73*
CompoundClasses, 73, *101*, *104*, *121*, *181*
CompoundImport, 76
CompoundsApi, 79
ComputedSubtools, *14*, 90
ConnectionCheck, 93
ConnectionError, *94*, 97
ConsensusAnnotationsCSI, *65*, 100
ConsensusAnnotationsDeNovo, *65*, 104

DataImportEvent, 108
DBLink, 111, *418*, *422*, *427*
Deviation, 114, *181*, *242*

Feature, 117, *278*
FeatureAnnotations, *14*, 121
FeatureImport, *76*, 125
FeaturesApi, 129
FingerprintPrediction, 177, *237*
FormulaCandidate, *121*, 180, *281*
FragmentationTree, *181*, 185
FragmentNode, *185*, 188

GuiApi, 192
GuiInfo, 195

ImportResult, 198
Info, 202
InfoApi, 206
IsotopePatternAnnotation, *181*, 208

Job, *41*, 212, *284*
JobProgress, *212*, 216
JobsApi, 219
JobSubmission, 236, *415*

LcmsSubmissionParameters, 241
LicenseInfo, *94*, 246
LipidAnnotation, *181*, 249
LoginAndAccountApi, 253
LossEdge, *185*, 261

MsData, *14*, 264
MsNovelist, *237*, 268

PagedModelAlignedFeature, 271
PagedModelCompound, 274
PagedModelFeature, 277
PagedModelFormulaCandidate, 281
PagedModelJob, 284
PagedModelRun, 287
PagedModelSpectralLibraryMatch, 290
PagedModelStructureCandidateFormula, 293
PagedModelStructureCandidateScored, 296
PageMetadata, *271*, *274*, *278*, *281*, *284*, *287*, *290*, *293*, *297*, 300
ParentPeak, 303, *307*
PeakAnnotation, *25*, 306

PeakPair, 311, *398*
ProblemDetail, 314
ProjectChangeEvent, 318
ProjectInfo, 321
ProjectsApi, 325

QuantTable, 341
QuantTableOptField, 345

rsirius_api, 347
Run, *287*, 348
RunOptField, 352
RunsApi, 353

SearchableDatabase, 358
SearchableDatabaseParameters, 362
SearchableDatabasesApi, 365
SearchableField, 376
SearchableFieldType, *377*, 381
SecurityContext, 382
SecurityContextPrincipal, *382*, 385
SimplePeak, *29*, *44*, 388
Sirius, *237*, 392
SpectralLibraryMatch, *290*, 398, *402*, *418*, *422*, *427*
SpectralLibraryMatchSummary, 402
SpectralLibrarySearch, *237*, 406
SpectrumAnnotation, *29*, 410
StoredJobSubmission, 414
StructureCandidate, *101*, 418
StructureCandidateFormula, *293*, 422
StructureCandidateScored, *121*, *297*, 426
StructureDbSearch, *237*, 431
Subscription, *8*, *246*, 435
SubscriptionConsumables, *246*, 439

Tag, *14*, *65*, *349*, 442
Term, *246*, 445
Timeout, *393*, 449

UseHeuristic, *393*, 452

WebServerNamespace, 455

Zodiac, *237*, 458
ZodiacAnalogueNodes, *459*, 462
ZodiacEdgeFilterThresholds, *459*, 465
ZodiacEpochs, *459*, 469
ZodiacLibraryScoring, *459*, 472