Package ‘bdpar’

June 24, 2021

Type Package

Title Big Data Preprocessing Architecture

Version 3.0.1

Description Provide a tool to easily build customized data flows to pre-process large volumes of information from different sources. To this end, ‘bdpar’ allows to (i) easily use and create new functionalities and (ii) develop new data source extractors according to the user needs. Additionally, the package provides by default a predefined data flow to extract and pre-process the most relevant information (tokens, dates, ... ) from some textual sources (SMS, Email, tweets, YouTube comments).

Date 2021-06-24

License GPL-3

URL https://github.com/miferreiro/bdpar

BugReports https://github.com/miferreiro/bdpar/issues

Depends R (>= 3.5.0)

Imports digest, parallel, R6, rlist, tools, utils

Suggests cld2, knitr, rex, rjson, rmarkdown, rtweet, stringi, stringr, testthat (>= 2.3.1), tuber

VignetteBuilder knitr

RoxygenNote 7.1.1

SystemRequirements Python (>= 2.7 or >= 3.6)

Encoding UTF-8

NeedsCompilation no

R topics documented:

AbbreviationPipe ........................................ 3
Bdpar ......................................................... 6
bdpar.log ................................................... 8
bdpar.Options ............................................. 9
bdparData .................................................. 12
Connections ............................................... 13
ContractionPipe ......................................... 15
DefaultPipeline ......................................... 17
DynamicPipeline ......................................... 20
ExtractorEml ............................................. 22
ExtractorFactory ......................................... 24
ExtractorSms .............................................. 26
ExtractorTwtid ............................................ 28
ExtractorYtbid ............................................. 30
File2Pipe ................................................... 32
FindEmojiPipe ............................................. 33
FindEmoticonPipe ........................................ 35
FindHashtagPipe ......................................... 37
FindUrlPipe ............................................... 39
FindUserNamePipe ....................................... 42
GenericPipe ............................................... 44
GenericPipeline ......................................... 46
GuessDatePipe ............................................ 47
GuessLanguagePipe ...................................... 49
Instance ................................................... 50
InterjectionPipe ......................................... 55
MeasureLengthPipe ...................................... 57
operator-pipe ............................................ 59
ResourceHandler ......................................... 60
runPipeline ............................................... 61
**Description**

`AbbreviationPipe` class is responsible for detecting the existing abbreviations in the data field of each `Instance`. Identified abbreviations are stored inside the `abbreviation` field of `Instance` class. Moreover if needed, is able to perform inline abbreviations replacement.

**Details**

`AbbreviationPipe` class requires the resource files (in json format) containing the correspondence between abbreviations and meaning. To this end, the language of the text indicated in the `property-LanguageName` should be contained in the resource file name (ie. `abbrev.xxx.json` where `xxx` is the value defined in the `propertyLanguageName`). The location of the resources should be defined in the "resources.abbreviations.path" field of `bdpar.Options` variable.

**Note**

`AbbreviationPipe` will automatically invalidate the `Instance` whenever the obtained data is empty.

**Inherit**

This class inherits from `GenericPipe` and implements the `pipe` abstract function.

**Super class**

`bdpar::GenericPipe` -> `AbbreviationPipe`

**Methods**

**Public methods:**
- `AbbreviationPipe$new()`
- `AbbreviationPipe$pipe()`  
- `AbbreviationPipe$findAbbreviation()`  
- `AbbreviationPipe$replaceAbbreviation()`  
- `AbbreviationPipe$getPropertyLanguageName()`  
- `AbbreviationPipe$getResourcesAbbreviationsPath()`
• `AbbreviationPipe$setResourcesAbbreviationsPath()`
• `AbbreviationPipe$clone()`

**Method** `new()`: Creates a `AbbreviationPipe` object.

*Usage:*

```r
AbbreviationPipe$new(
  propertyName = "abbreviation",
  propertyLanguageName = "language",
  alwaysBeforeDeps = list("GuessLanguagePipe"),
  notAfterDeps = list(),
  replaceAbbreviations = TRUE,
  resourcesAbbreviationsPath = NULL
)
```

*Arguments:*

- `propertyName` A `character` value. Name of the property associated with the `GenericPipe`.
- `propertyLanguageName` A `character` value. Name of the language property.
- `alwaysBeforeDeps` A `list` value. The dependencies alwaysBefore (`GenericPipes` that must be executed before this one).
- `notAfterDeps` A `list` value. The dependencies notAfter (`GenericPipes` that cannot be executed after this one).
- `replaceAbbreviations` A `logical` value. Indicates if the abbreviations are replaced or not.
- `resourcesAbbreviationsPath` A `character` value. Path of resource files (in json format) containing the correspondence between abbreviations and meaning.

**Method** `pipe()`: Preprocesses the `Instance` to obtain/replace the abbreviations. The abbreviations found in the data are added to the list of properties of the `Instance`.

*Usage:*

```r
AbbreviationPipe$pipe(instance)
```

*Arguments:*

- `instance` A `Instance` value. The `Instance` to preprocess.

*Returns:* The `Instance` with the modifications that have occurred in the pipe.

**Method** `findAbbreviation()`: Checks if the abbreviation is in the data.

*Usage:*

```r
AbbreviationPipe$findAbbreviation(data, abbreviation)
```

*Arguments:*

- `data` A `character` value. The text where abbreviation will be searched.
- `abbreviation` A `character` value. Indicates the abbreviation to find.

*Returns:* A `logical` value depending on whether the abbreviation is in the data.

**Method** `replaceAbbreviation()`: Replaces the `abbreviation` in the data for the `extendedAbbreviation`.

*Usage:*

```r
AbbreviationPipe$replaceAbbreviation(abbreviation, extendedAbbreviation, data)
```
**AbbreviationPipe**

*Arguments:*

abbreviation  A character value. Indicates the abbreviation to replace.
extendedAbbreviation  A character value. Indicates the string to replace for the abbreviations found.
data  A character value. The text where abbreviation will be replaced.

*Returns:* The data with the abbreviations replaced.

**Method** getPropertyLanguageName(): Gets the name of property language.

*Usage:*

AbbreviationPipe$getPropertyLanguageName()

*Returns:* Value of name of property language.

**Method** getResourcesAbbreviationsPath(): Gets the path of abbreviations resources.

*Usage:*

AbbreviationPipe$getResourcesAbbreviationsPath()

*Returns:* Value of path of abbreviations resources.

**Method** setResourcesAbbreviationsPath(): Sets the path of abbreviations resources.

*Usage:*

AbbreviationPipe$setResourcesAbbreviationsPath(path)

*Arguments:*

path  A character value. The new value of the path of abbreviations resources.

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

*Usage:*

AbbreviationPipe$clone(deep = FALSE)

*Arguments:*

deep  Whether to make a deep clone.

**See Also**

bdpar.Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe
Bdpar

Class to manage the preprocess of the files throughout the flow of pipes

Description

Bdpar class provides the static variables required to perform the whole data flow process. To this end Bdpar is in charge of (i) initialize the objects of handle the connections to APIs (Connections) and handles json resources (ResourceHandler) and (ii) executing the flow of pipes (inherited from GenericPipeline class) passed as argument.

Details

In the case that some pipe, defined on the workflow, needs some type of configuration, it can be defined through bdpar.Options variable which have different methods to support the functionality of different pipes.

Static variables

- **connections**: (Connections) object that handles the connections with YouTube and Twitter.
- **resourceHandler**: (ResourceHandler) object that handles the json resources files.

Methods

**Public methods**:

- Bdpar$new()
- Bdpar$execute()
- Bdpar$clone()

**Method** new(): Creates a Bdpar object. Initializes the static variables: connections and resourceHandler.

*Usage:*
Bdpar$new()

**Method** execute(): Preprocess files through the indicated flow of pipes.

*Usage:*
Bdpar$execute(path,
extractors = ExtractorFactory$new(),
pipeline = DefaultPipeline$new(),
cache = TRUE,
verbose = FALSE,
summary = FALSE)

*Arguments:*

path A character value. The path where the files to be processed are located.
extractors  A ExtractorFactory value. Class which implements the createInstance method to choose which type of Instance is created.

pipeline  A GenericPipeline value. Subclass of GenericPipeline, which implements the execute method. By default, it is the DefaultPipeline pipeline.

cache  (logical) flag indicating if the status of the instances will be stored after each pipe. This allows to avoid rejections of previously executed tasks, if the order and configuration of the pipe and pipeline is the same as what is stored in the cache.

verbose  (logical) flag indicating for printing messages, warnings and errors.

summary  (logical) flag indicating if a summary of the pipeline execution is provided or not.

Details:  In case of wanting to parallelize, it is necessary to indicate the number of cores to be used through bdpar.Options$set("numCores", numCores)

Returns:  The list of Instances that have been preprocessed.

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

Usage:
Bdpar$clone(deep = FALSE)

Arguments:
dee  Whether to make a deep clone.

See Also
bdpar.Options, Connections, DefaultPipeline, DynamicPipeline, GenericPipeline, Instance, ExtractorFactory, ResourceHandler, runPipeline

Examples

## Not run:

#If it is necessary to indicate any configuration, do it through:
#bdpar.Options$set(key, value)
#If the key is not initialized, do it through:
#bdpar.Options$add(key, value)

#If it is necessary parallelize, do it through:
#bdpar.Options$set("numCores", numCores)

#If it is necessary to change the behavior of the log, do it through:
#bdpar.Options$configureLog(console = TRUE, threshold = "INFO", file = NULL)

#Folder with the files to preprocess
path <- system.file("example",
                   package = "bdpar")

#Object which decides how creates the instances
extractors <- ExtractorFactory$new()

#Object which indicates the pipes’ flow
pipeline <- DefaultPipeline$new()
objectBdpar <- Bdpar$new()

# Starting file preprocessing...
objectBdpar$execute(path = path,
    extractors = extractors,
    pipeline = pipeline,
    cache = FALSE,
    verbose = FALSE,
    summary = TRUE)

## End(Not run)

bdpar.log

Write messages to the log at a given priority level using the custom bdpar log

Description

bdpar.log is responsible for managing the messages to show on the log.

Usage

bdpar.log(message, level = "INFO", className = NULL, methodName = NULL)

Arguments

message A string to be printed to the log with the corresponding priority level.
level The desired priority level (DEBUG,INFO,WARN,ERROR and FATAL). In the case of the FATAL level will be call to the stop function. Also, if the level is WARN, the message will be a warning.
className A string to indicated in which class is called to the log. If the value is NULL, this field is not shown in the log.
methodName A string to indicated in which method is called to the log. If the value is NULL, this field is not shown in the log.

Details

The format output is as following:
[currentTime][className][methodName][level] message
The type of message changes according to the level indicated:
- The DEBUG,INFO and ERROR levels return a text using the message function.
- The WARN level returns a text using the warning function.
- The FATAL level returns a text using the stop function.

Note

In the case of multithreading, the log will only be by file.
bdpar.Options

See Also
bdpar.Options

Examples

```r
## Not run:

# First step, configure the behavior of log
bdpar.options$configureLog(console = TRUE, threshold = "DEBUG", file = NULL)

message <- "Message example"

className <- "Class name example"

methodName <- "Method name example"

bdpar.log(message = message, level = "DEBUG", className = NULL, methodName = NULL)

bdpar.log(message = message, level = "INFO", className = className, methodName = methodName)

bdpar.log(message = message, level = "WARN", className = className, methodName = NULL)

bdpar.log(message = message, level = "ERROR", className = NULL, methodName = NULL)

bdpar.log(message = message, level = "FATAL", className = NULL, methodName = methodName)

## End(Not run)
```

---

bdpar.Options

Object to handle the keys/attributes/options common to all pipeline flow

Description

This class provides the necessary methods to manage a list of keys or options used along the pipe flow, both those provided by the default library and those implemented by the user.

Usage

bdpar.Options

Details

By default, the application initializes the object named bdpar.Options of type BdparOptions which is in charge of initializing the options used in the defined pipes.

The default fields on bdpar.Options are initialized, if needed, as shown bellow:
Cache functionality

If the bdpar cache is configured through the "cache" and "cache.folder" options, the status of the instances will be stored after each pipe. This allows to avoid rejections of previously executed tasks, if the order and configuration of the pipe and pipeline is the same as what is stored in the cache.

If you want to remove the cache, the cleanCache method does this task.

Parallel functionality

The parallelization of instances is configured through the "numCores" option, which indicates the number of cores that will be used in the processing.

In the case of parallelisation, only the log by file will work to allow collecting all the information produced by the cores.
Log configuration

The bdpar log is configured through the configureLog function. This system manages both the place to display the messages and the priority level of each message showing only the messages with a higher level than indicated in the threshold variable.

If you want to deactivate the bdpar log, the disableLog method in bdpar.Options does this task.

Methods

• **get**: obtains a specific option.
  - **Usage**: get(key)
  - **Value**: the value of the specific option.
  - **Arguments**:
    * **key**: (character) the name of the option to obtain.

• **add**: adds a option to the list of options
  - **Usage**: add(key, value)
  - **Arguments**:
    * **key**: (character) the name of the new option.
    * **propertyName**: (Object) the value of the new option.

• **set**: modifies the value of the one option.
  - **Usage**: set(key, value)
  - **Arguments**:
    * **key**: (character) the name of the new option.
    * **propertyName**: (Object) the value of the new option.

• **remove**: removes a specific option.
  - **Usage**: remove(key)
  - **Arguments**:
    * **key**: (character) the name of the option to remove.

• **getAll**: gets the list of options.
  - **Usage**: getAll()
  - **Value**: Value of options.

• **remove**: resets the option list to the initial state.
  - **Usage**: reset()

• **isSpecificOption**: checks for the existence of an specific option.
  - **Usage**: isSpecificProperty(key)
  - **Value**: A boolean results according to the existence of the specific option in the list of options
  - **Arguments**:
    * **key**: (character) the key of the option to check.

• **cleanCache**: Cleans the cache of executed pipelines. Deletes all files and directories that are in the path defined in "cache.folder" option.
  - **Usage**: cleanCache()
• **configureLog**: Configures the bdpar log. In the case of parallelisation, only the log by file will work.
  
  - *Usage*: `configureLog(console = TRUE, threshold = "INFO", file = NULL)`
  
  - *Arguments*:
    * **console**: (boolean) Shows the log on console or not.
    * **threshold**: (character) The logging threshold level. Messages with a lower priority level will be discarded.
    * **file**: (character) The file to write messages to. If it is NULL, the log in file will not be enabled.

• **disableLog**: Deactivates the bdpar log.
  
  - *Usage*: `disableLog()`

• **getLogConfiguration**: Print the bdpar log configuration.
  
  - *Usage*: `getLogConfiguration()`

### See Also

AbbreviationPipe, bdpar.log, Connections, ContractionPipe, ExtractorEml, ExtractorTwtid, ExtractorYtbid, GuessLanguagePipe, Instance, SlangPipe, StopWordPipe, TeeCSVPipe, %>%

---

**bdparData**

Example of the content of the files to be preprocessed.

---

**Description**

A manually collected data set containing e-mails and SMS messages from the nutritional and health domain classified as spam and non-spam (with a ratio of 50%). In addition the dataset contains two variables: (i) **path** which indicates the location of the target file and, (ii) **source** which contains the raw text comprising each file.

**Usage**

```
data(bdparData)
```

**Format**

A data frame with 20 rows and 2 variables:

- **path**  File path.
- **source**  File content.
Connections

Class to manage the connections with Twitter and YouTube

Description
The tasks of the functions that the Connections class has are to establish the connections and control the number of requests that have been made with the APIs of Twitter and YouTube.

Details
The way to indicate the keys of YouTube and Twitter has to be through fields of bdpar.Options variable:

[twitter]
- bdpar.Options$set("twitter.consumer.key",<<consumer_key>>)
- bdpar.Options$set("twitter.consumer.secret",<<consumer_secret>>)
- bdpar.Options$set("twitter.access.token",<<access_token>>)
- bdpar.Options$set("twitter.access.token.secret",<<access_token_secret>>)

[youtube]
- bdpar.Options$set("youtube.app.id",<<app_id>>)
- bdpar.Options$set("youtube.app.password",<<app_password>>)

Note
Fields of unused connections will be automatically ignored by the platform.

Methods
Public methods:
- Connections$new()
- Connections$getTwitterToken()
- Connections$startConnectionWithTwitter()
- Connections$checkRequestToTwitter()
- Connections$startConnectionWithYoutube()
- Connections$addNumRequestToYoutube()
- Connections$checkRequestToYoutube()
- Connections$getNumRequestMaxToYoutube()
- Connections$clone()

Method new(): Creates a Connections object.
Usage:
Connections$new()

Method getTwitterToken(): Gets the Twitter token ID.
Connections

Usage:
Connections$getTwitterToken()

Returns: Value of twitterToken.

Method startConnectionWithTwitter(): Responsible of establishing the connection to Twitter.

Usage:
Connections$startConnectionWithTwitter()

Method checkRequestToTwitter(): Function in charge of handling the connection with Twitter.

Usage:
Connections$checkRequestToTwitter()

Method startConnectionWithYoutube(): Function able to establish the connection with YouTube.

Usage:
Connections$startConnectionWithYoutube()

Method addNumRequestToYoutube(): Function that increases in one the number of request to YouTube.

Usage:
Connections$addNumRequestToYoutube()

Method checkRequestToYoutube(): Handles the connection with YouTube.

Usage:
Connections$checkRequestToYoutube()

Method getNumRequestMaxToYoutube(): Gets the number of maximum requests allowed by YouTube API.

Usage:
Connections$getNumRequestMaxToYoutube()

Returns: Value of number maximum of request to YouTube.

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

Usage:
Connections$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

See Also
bdpar.Options, ExtractorTwtid, ExtractorYtbid
Description

_ContractionPipe_ class is responsible for detecting the existing contractions in the **data** field of each **Instance**. Identified contractions are stored inside the **contraction** field of **Instance** class. Moreover if needed, is able to perform inline contractions replacement.

Details

_ContractionPipe_ class requires the resource files (in json format) containing the correspondence between contractions and meaning. To this end, the language of the text indicated in the **property-LanguageName** should be contained in the resource file name (ie. contr.xxx.json where xxx is the value defined in the **propertyLanguageName**). The location of the resources should be defined in the "**resources.contractions.path**" field of **bdpar.Options** variable.

Note

_ContractionPipe_ will automatically invalidate the **Instance** whenever the obtained data is empty.

Inherit

This class inherits from **GenericPipe** and implements the pipe abstract function.

Super class

`bdpar::GenericPipe -> ContractionPipe`

Methods

**Public methods:**

- `ContractionPipe$new()`
- `ContractionPipe$pipe()`
- `ContractionPipe$findContraction()`
- `ContractionPipe$replaceContraction()`
- `ContractionPipe$getPropertyLanguageName()`
- `ContractionPipe$getResourcesContractionsPath()`
- `ContractionPipe$setResourcesContractionsPath()`
- `ContractionPipe$clone()`

**Method** `new()`: Creates a **ContractionPipe** object.

*Usage:*
ContractionPipe$new(
  propertyName = "contractions",
  propertyLanguageName = "language",
  alwaysBeforeDeps = list("GuessLanguagePipe"),
  notAfterDeps = list(),
  replaceContractions = TRUE,
  resourcesContractionsPath = NULL
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
propertyLanguageName A character value. Name of the language property.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).
replaceContractions A logical value. Indicates if the contractions are replaced or not.
resourcesContractionsPath A character value. Path of resource files (in json format) containing the correspondence between contractions and meaning.

Method pipe(): Preprocesses the Instance to obtain/replace the contractions. The contractions found in the data are added to the list of properties of the Instance.

Usage:
ContractionPipe$pipe(instance)

Arguments:
instance A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method findContraction(): Checks if the contraction is in the data.

Usage:
ContractionPipe$findContraction(data, contraction)

Arguments:
data A character value. The text where contraction will be searched.
contraction A character value. Indicates the contraction to find.

Returns: A logical value depending on whether the contraction is in the data.

Method replaceContraction(): Replaces the contraction in the data for the extendedContraction.

Usage:
ContractionPipe$replaceContraction(contraction, extendedContraction, data)

Arguments:
contraction A character value. Indicates the contraction to replace.
extendedContraction A character value. Indicates the string to replace for the contractions found.
data A character value. The text where contraction will be replaced.

Returns: The data with the contractions replaced.

Method getPropertyLanguageName(): Gets the name of property language.

Usage:
ContractionPipe$getPropertyLanguageName()

Returns: Value of name of property language.

Method getResourcesContractionsPath(): Gets the path of contractions resources.

Usage:
ContractionPipe$getResourcesContractionsPath()

Returns: Value of path of contractions resources.

Method setResourcesContractionsPath(): Sets the path of contractions resources.

Usage:
ContractionPipe$setResourcesContractionsPath(path)

Arguments:
path A character value. The new value of the path of contractions resources.

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

Usage:
ContractionPipe$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also

AbbreviationPipe, bdpar.Options, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

DefaultPipeline

Class implementing a default pipelining process.

Description

This DefaultPipeline class inherits from the GenericPipeline class. Includes the execute method which provides a default pipelining implementation.
Details

The default flow is:

```
instance %>|%
  TargetAssigningPipe$new() %>|%
  StoreFileExtPipe$new() %>|%
  GuessDatePipe$new() %>|%
  File2Pipe$new() %>|%
  MeasureLengthPipe$new(propertyName = "length_before_cleaning_text") %>|%
  FindUserNamePipe$new() %>|%
  FindHashtagPipe$new() %>|%
  FindUrlPipe$new() %>|%
  FindEmoticonPipe$new() %>|%
  FindEmojiPipe$new() %>|%
  GuessLanguagePipe$new() %>|%
  ContractionPipe$new() %>|%
  AbbreviationPipe$new() %>|%
  SlangPipe$new() %>|%
  ToLowerCasePipe$new() %>|%
  InterjectionPipe$new() %>|%
  StopWordPipe$new() %>|%
  MeasureLengthPipe$new(propertyName = "length_after_cleaning_text") %>|%
  TeeCSVPipe$new()
```

Inherit

This class inherits from `GenericPipeline` and implements the `execute` abstract function.
Super class

`bdpar::GenericPipeline` -> `DefaultPipeline`

Methods

**Public methods:**

- `DefaultPipeline$new()`
- `DefaultPipeline$execute()`
- `DefaultPipeline$get()`
- `DefaultPipeline$print()`
- `DefaultPipeline$toString()`
- `DefaultPipeline$clone()`

**Method `new()`:** Creates a `DefaultPipeline` object.

*Usage:*  
`DefaultPipeline$new()`

**Method `execute()`:** Function where is implemented the flow of the `GenericPipes`.

*Usage:*  
`DefaultPipeline$execute(instance)`

*Arguments:*  
`instance` A `Instance` value. The `Instance` that is going to be processed.

*Returns:* The preprocessed `Instance`.

**Method `get()`:** Gets a list with containing the set of `GenericPipe`s of the pipeline.

*Usage:*  
`DefaultPipeline$get()`

*Returns:* The set of `GenericPipes` containing the pipeline.

**Method `print()`:** Prints pipeline representation. (Override print function)

*Usage:*  
`DefaultPipeline$print(...)`

*Arguments:*  
... Further arguments passed to or from other methods.

**Method `toString()`:** Returns a `character` representing the pipeline

*Usage:*  
`DefaultPipeline$toString()`

*Returns:* `DefaultPipeline character` representation

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

*Usage:*  
`DefaultPipeline$clone(deep = FALSE)`

*Arguments:*  
`deep` Whether to make a deep clone.
DynamicPipeline

Class implementing a dynamic pipelining process

Description

This DynamicPipeline class inherits from the GenericPipeline class. Includes the execute method which provides a dynamic pipelining implementation.

Inherit

This class inherits from GenericPipeline and implements the execute abstract function.

Super class

bdpar::GenericPipeline -> DynamicPipeline

Methods

Public methods:

- DynamicPipeline$new()
- DynamicPipeline$add()
- DynamicPipeline$removeByPos()
- DynamicPipeline$removeByPipe()
- DynamicPipeline$removeAll()
- DynamicPipeline$execute()
- DynamicPipeline$get()
- DynamicPipeline$print()
- DynamicPipeline$toString()
- DynamicPipeline$clone()

Method new(): Creates a DynamicPipeline object.

Usage:
DynamicPipeline$new(pipeline = NULL)

Arguments:
pipeline A list of GenericPipe objects. Initializes the flow of GenericPipe.

Method add(): Adds a GenericPipe or a GenericPipe list to the pipeline.

Usage:
DynamicPipeline$add(pipe, pos = NULL)

Arguments:
DynamicPipeline

pipe  A GenericPipe object or a list of GenericPipe objects.

pos  A (numeric) value. The value of the position to add. If it is NULL, GenericPipe is appended to the pipeline.

**Method** removeByPos(): Removes GenericPipes by the position on the pipeline.

**Usage:**
DynamicPipeline$removeByPos(pos)

**Arguments:**
pos  A (numeric) value. The value of the position to remove.

**Method** removeByPipe(): Removes GenericPipes by its name on the pipeline.

**Usage:**
DynamicPipeline$removeByPipe(pipe.name)

**Arguments:**
pipe.name  A (character) value. The GenericPipes name to remove.

**Method** removeAll(): Removes all GenericPipes included on pipeline.

**Usage:**
DynamicPipeline$removeAll()

**Method** execute(): Function where is implemented the flow of the GenericPipes.

**Usage:**
DynamicPipeline$execute(instance)

**Arguments:**
instance  A (Instance) value. The Instance that is going to be processed.

**Method** get(): Gets a list with containing the set of GenericPipes of the pipeline.

**Usage:**
DynamicPipeline$get()

**Returns:** The set of GenericPipes containing the pipeline.

**Method** print(): Prints pipeline representation. (Override print function)

**Usage:**
DynamicPipeline$print(...)

**Arguments:**
...  Further arguments passed to or from other methods.

**Method** toString(): Returns a character representing the pipeline

**Usage:**
DynamicPipeline$toString()

**Returns:** DynamicPipeline character representation

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

**Usage:**
DynamicPipeline$clone(deep = FALSE)

**Arguments:**
deep  Whether to make a deep clone.
ExtractorEml

See Also

bdpar.log, Instance, DefaultPipeline, GenericPipeline, GenericPipe.

ExtractorEml

Class to handle email files with eml extension

Description

This class inherits from the Instance class and implements the functions of extracting the text and the date from an eml type file.

Details

The way to indicate which part to choose in the email, when is a multipart email, is through the "ExtractorEML.mpaPartSelected" field of bdpar.Options variable.

Note

To be able to use this class it is necessary to have Python installed.

Inherit

This class inherits from Instance and implements the obtainSource and obtainDate abstracts functions.

Super class

bdpar::Instance -> ExtractorEml

Methods

Public methods:

• ExtractorEml$new()
• ExtractorEml$obtainDate()
• ExtractorEml$obtainSource()
• ExtractorEml$getPartSelectedOnMPAlternative()
• ExtractorEml$setPartSelectedOnMPAlternative()
• ExtractorEml$toString()
• ExtractorEml$clone()

Method new(): Creates a ExtractorEml object.

Usage:
ExtractorEml$new(path, PartSelectedOnMPAlternative = NULL)

Arguments:
path A character value. Path of the eml file.
PartSelectedOnMPAlternative is a character value. Configuration to read the eml files. If it is NULL, checks if is defined in the "extractorEML.mpaPartSelected" field of bdpar.Options variable.

Method obtainDate(): Obtains the date of the eml file. Calls the function read_emails and obtains the date of the file indicated in the path and then transforms it into the generic date format, that is "%a %b %d %H:%M:%S %Z %Y" (Example: "Thu May 02 06:52:36 UTC 2013").

Usage:
ExtractorEml$obtainDate()

Method obtainSource(): Obtains the source of the eml file. Calls the function read_emails and obtains the source of the file indicated in the path. In addition, it initializes the data with the initial source.

Usage:
ExtractorEml$obtainSource()

Method getPartSelectedOnMPAlternative(): Gets of PartSelectedOnMPAlternative variable.

Usage:
ExtractorEml$getPartSelectedOnMPAlternative()

Returns: Value of PartSelectedOnMPAlternative variable.

Method setPartSelectedOnMPAlternative(): Gets of PartSelectedOnMPAlternative variable.

Usage:
ExtractorEml$setPartSelectedOnMPAlternative(PartSelectedOnMPAlternative)

Arguments:
PartSelectedOnMPAlternative is a character value. The new value of PartSelectedOnMPAlternative variable.

Method toString(): Returns a character representing the instance

Usage:
ExtractorEml$toString()

Returns: Instance character representation

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

Usage:
ExtractorEml$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also
bdpar.Options, ExtractorSms, ExtractorTwtid, ExtractorYtbid, Instance
ExtractorFactory  
Class to handle the creation of Instance types

Description

ExtractorFactory class builds the appropriate Instance object according to the file extension. In the case of not finding the registered extension, the default extractor will be used if it has been previously configured.

Methods

Public methods:

- ExtractorFactory\$new()
- ExtractorFactory\$registerExtractor()
- ExtractorFactory\$setExtractor()
- ExtractorFactory\$setDefaultExtractor()
- ExtractorFactory\$removeExtractor()
- ExtractorFactory\$getAllExtractors()
- ExtractorFactory\$getDefaultExtractor()
- ExtractorFactory\$isSpecificExtractor()
- ExtractorFactory\$createInstance()
- ExtractorFactory\$reset()
- ExtractorFactory\$print()
- ExtractorFactory\$clone()

Method new(): Creates a ExtractorFactory object.

Usage:
ExtractorFactory\$new()

Method registerExtractor(): Adds an extractor to the list of extensions. If the extension is an empty string (""), the indicated extractor will be the default when there is no extractor associated with an extension.

Usage:
ExtractorFactory\$registerExtractor(extensions, extractor)

Arguments:

- extensions A character array. The names of the extension option.
- extractor A Object value. The extractor of the new extension.

Method setExtractor(): Modifies the extractor of the one extension.

Usage:
ExtractorFactory\$setExtractor(extension, extractor)

Arguments:
extension A character value. The name of the extension option.
extractor A Object value. The value of the new extractor.

**Method** `setDefaultExtractor()`: Modifies the extractor of the one extension. Assign NULL value to disable the default extractor.

*Usage:*
`ExtractorFactory$setDefaultExtractor(defaultExtractor)`

*Arguments:*
defaultExtractor A Object value. The value of the default extractor.

**Method** `removeExtractor()`: Removes a specific extractor thought the extension.

*Usage:*
`ExtractorFactory$removeExtractor(extension)`

*Arguments:*
extension A character value. The name of the extension to remove.

**Method** `getAllExtractors()`: Gets the list of extractors.

*Usage:*
`ExtractorFactory$getAllExtractors()`

*Returns*: Value of extractors.

**Method** `getDefaultExtractor()`: Gets the default extractor.

*Usage:*
`ExtractorFactory$getDefaultExtractor()`

*Returns*: Value of default extractor.

**Method** `isSpecificExtractor()`: Checks if exists an extractor for a specific extension.

*Usage:*
`ExtractorFactory$isSpecificExtractor(extension)`

*Arguments:*
extension A character value. The name of the extension to check

*Returns*: Value of extractors.

**Method** `createInstance()`: Builds the Instance object according to the file extension. In the case of not finding the registered extension, the default extractor will be used if it has been previously configured.

*Usage:*
`ExtractorFactory$createInstance(path)`

*Arguments:*
path A character value. Path of the file to create an Instance.

*Returns*: The Instance corresponding object according to the file extension.

**Method** `reset()`: Resets list of extractor to default state.
**Usage:**
ExtractorFactory$reset()

**Method** print(): Prints pipeline representation. (Override print function)

**Usage:**
ExtractorFactory$print(...)

**Arguments:**
... Further arguments passed to or from other methods.

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

**Usage:**
ExtractorFactory$clone(deep = FALSE)

**Arguments:**
deepl Whether to make a deep clone.

**See Also**
ExtractorEml, ExtractorSms, ExtractorTwtid, ExtractorYtbid, Instance

---

**ExtractorSms**

**Class to handle SMS files with tsms extension**

**Description**
This class that inherits from the **Instance** class and implements the functions of extracting the text and the date of an tsms type file.

**Details**
Due to the fact that the creation date of the message can not be extracted from the text of an SMS, the date will be initialized to empty.

**Inherit**
This class inherits from **Instance** and implements the **obtainSource** and **obtainDate** abstracts functions.

**Super class**
bdpar::Instance -> ExtractorSms
Methods

**Public methods:**

- `ExtractorSms$new()`
- `ExtractorSms$obtainDate()`
- `ExtractorSms$obtainSource()`
- `ExtractorSms$toString()`
- `ExtractorSms$clone()`

**Method new():** Creates a `ExtractorSms` object.

*Usage:*

```r
ExtractorSms$new(path)
```

*Arguments:*

- `path` A `character` value. Path of the tsms file.

**Method obtainDate():** Obtains the date of the SMS file.

*Usage:*

```r
ExtractorSms$obtainDate()
```

**Method obtainSource():** Obtains the source of the SMS file. Reads the file indicated in the path. In addition, it initializes the data field with the initial source.

*Usage:*

```r
ExtractorSms$obtainSource()
```

**Method toString():** Returns a `character` representing the instance

*Usage:*

```r
ExtractorSms$toString()
```

*Returns: Instance character representation*

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

*Usage:*

```r
ExtractorSms$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.

**See Also**

`ExtractorEml, ExtractorTwtid, ExtractorYtbid, Instance`
ExtractorTwtid

Class to handle tweets files with twtid extension

Description

This class inherits from the Instance class and implements the functions of extracting the text and the date of an twtid type file.

Details

Twitter connection is handled through the Connections class which loads the Twitter API credentials from the bdpar.Options object. Additionally, to increase the processing speed, each twitter query is stored in a cache to avoid the execution of duplicated queries. To enable this option, cache location should be in the "cache.twitter.path" field of bdpar.Options variable. This variable has to be the path to store the tweets and it is necessary that it has two folder named: "_spam_" and "_ham_"

Inherit

This class inherits from Instance and implements the obtainSource and obtainDate abstracts functions.

Super class

bdpar::Instance -> ExtractorTwtid

Methods

Public methods:

• ExtractorTwtid$new()
• ExtractorTwtid$obtainId()
• ExtractorTwtid$getId()
• ExtractorTwtid$obtainDate()
• ExtractorTwtid$obtainSource()
• ExtractorTwtid$toString()
• ExtractorTwtid$clone()

Method new(): Creates a ExtractorTwtid object.

Usage:
ExtractorTwtid$new(path, cachePath = NULL)

Arguments:
path A character value. Path of the twtid file.
cachePath A character value. Path of the cache location. If it is NULL, checks if is defined in the "cache.twitter.path" field of bdpar.Options variable.
Method `obtainId()`: Obtains the ID of a specific tweet. Reads the ID of the file indicated in the variable path.

*Usage:*

```
ExtractorTwtid$obtainId()
```

Method `getId()`: Gets the ID of a specific tweet.

*Usage:*

```
ExtractorTwtid$getId()
```

*Returns:* Value of tweet ID.

Method `obtainDate()`: Obtains the date from a specific tweet ID. If the tweet has been previously cached, the tweet date is loaded from cache path. Otherwise, the request is performed using Twitter API and the date is automatically formatted to "

*Usage:*

```
ExtractorTwtid$obtainDate()
```

Method `obtainSource()`: Obtains the source from a specific tweet ID. If the tweet has previously been cached, the source is loaded from cache path. Otherwise, the request is performed using Twitter API.

*Usage:*

```
ExtractorTwtid$obtainSource()
```

Method `toString()`: Returns a `character` representing the instance

*Usage:*

```
ExtractorTwtid$toString()
```

*Returns:* `Instance character` representation

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

*Usage:*

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

*Arguments:*

depth Whether to make a deep clone.

See Also

`bdpar.Options`, `Connections`, `ExtractorEml`, `ExtractorSms`, `ExtractorYtbid`, `Instance`, `
ExtractorYtbid  

Class to handle comments of YouTube files with ytbid extension

Description

This class inherits from the Instance class and implements the functions of extracting the text and the date of an ytbid type file.

Details

YouTube connection is handled through the Connections class which loads the YouTube API credentials from the bdpar.Options object. Additionally, to increase the processing speed, each Youtube query is stored in a cache to avoid the execution of duplicated queries. To enable this option, cache location should be in the "cache.youtube.path" field of bdpar.Options variable. This variable has to be the path to store the comments and it is necessary that it has two folder named: "_spam_" and "_ham_"

Inherit

This class inherits from Instance and implements the obtainSource and obtainDate abstracts functions.

Super class

bdpar::Instance -> ExtractorYtbid

Methods

Public methods:

• ExtractorYtbid$new()
• ExtractorYtbid$obtainId()
• ExtractorYtbid$getId()
• ExtractorYtbid$obtainDate()
• ExtractorYtbid$obtainSource()
• ExtractorYtbid$toString()
• ExtractorYtbid$clone()

Method new(): Creates a ExtractorYtbid object.

Usage:

ExtractorYtbid$new(path, cachePath = NULL)

Arguments:

path A character value. Path of the ytbid file.

cachePath A character value. Path of the cache location. If it is NULL, checks if is defined in the "cache.youtube.path" field of bdpar.Options variable.
Method `obtainId()`: Obtains the ID of the specific Youtube's comment. Reads the ID of the file indicated in the variable path.

Usage:
ExtractorYtbid$obtainId()

Method `getId()`: Gets the ID of an specific Youtube's comment.

Usage:
ExtractorYtbid$getId()

Returns: Value of Youtube’s comment ID.

Method `obtainDate()`: Obtains the date from a specific comment ID. If the comment has been previously cached the comment date is loaded from cache path. Otherwise, the request is performed using YouTube API and the date is then formatted to the established standard.

Usage:
ExtractorYtbid$obtainDate()

Method `obtainSource()`: Obtains the source from a specific comment ID. If the comment has previously been cached the source is loaded from cache path. Otherwise, the request is performed using on YouTube API.

Usage:
ExtractorYtbid$obtainSource()

Method `toString()`: Returns a character representing the instance

Usage:
ExtractorYtbid$toString()

Returns: Instance character representation

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

Usage:
ExtractorYtbid$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

See Also
bdpar.Options, Connections, ExtractorEml, ExtractorSms, ExtractorTwtid, Instance
File2Pipe

Class to obtain the source field of an Instance

Description

Obtains the source using the method which implements the subclass of Instance.

Note

File2Pipe will automatically invalidate the Instance whenever the obtained source is empty or not in UTF-8 format.

Inherit

This class inherits from GenericPipe and implements the pipe abstract function.

Super class

bdpar::GenericPipe -> File2Pipe

Methods

Public methods:

• File2Pipe$new()
• File2Pipe$pipe()
• File2Pipe$clone()

Method new(): Creates a File2Pipe object.

Usage:

File2Pipe$new(
  propertyName = "source",
  alwaysBeforeDeps = list("TargetAssigningPipe"),
  notAfterDeps = list()
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).

Method pipe(): Preprocesses the Instance to obtain the source.

Usage:

File2Pipe$pipe(instance)

Arguments:
instance  A Instance value. The Instance to preprocess.

Returns:  The Instance with the modifications that have occurred in the pipe.

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

Usage:
File2Pipe$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

See Also
AbbreviationPipe, ContractionPipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

FindEmojiPipe  Class to find and/or replace the emoji on the data field of an Instance

Description
This class is responsible of detecting the existing emojis in the data field of each Instance. Identified emojis are stored inside the emoji field of Instance class. Moreover if required, is able to perform inline emoji replacement.

Details
FindEmojiPipe use the emoji list provided by rtweet package.

Note
FindEmojiPipe will automatically invalidate the Instance whenever the obtained data is empty.

Inherit
This class inherits from GenericPipe and implements the pipe abstract function.

Super class
bdpar::GenericPipe -> FindEmojiPipe
Methods

Public methods:

• `FindEmojiPipe$new()`
• `FindEmojiPipe$pipe()`
• `FindEmojiPipe$findEmoji()`
• `FindEmojiPipe$replaceEmoji()`
• `FindEmojiPipe$clone()`

Method `new()`: Creates a `FindEmojiPipe` object.

Usage:
`FindEmojiPipe$new(`
  `propertyName = "Emojis",
  alwaysBeforeDeps = list(),
  notAfterDeps = list(),
  replaceEmojis = TRUE`
)`

Arguments:

propertyName A `character` value. Name of the property associated with the `GenericPipe`.
alwaysBeforeDeps A `list` value. The dependencies alwaysBefore (`GenericPipes` that must be executed before this one).
notAfterDeps A `list` value. The dependencies notAfter (`GenericPipes` that cannot be executed after this one).
replaceEmojis A `logical` value. Indicates if the emojis are replaced.
propertyLanguageName A `character` value. Name of the language property.

Method `pipe()`: Preprocessors the `Instance` to obtain/replace the emojis. The emojis found in the data are added to the list of properties of the `Instance`.

Usage:
`FindEmojiPipe$pipe(instance)`

Arguments:

instance A `Instance` value. The `Instance` to preprocess.

Returns: The `Instance` with the modifications that have occurred in the pipe.

Method `findEmoji()`: Checks if the emoji is in the data.

Usage:
`FindEmojiPipe$findEmoji(data, emoji)`

Arguments:

data A `character` value. The text where emoji will be searched.
emoji A `character` value. Indicates the emoji to find.

Returns: A `logical` value depending on whether the emoji is in the data.

Method `replaceEmoji()`: Replaces the `emoji` in the data for the `extendedEmoji`.

Usage:
FindEmoticonPipe

FindEmojiPipe$replaceEmoji(emoji, extendedEmoji, data)

Arguments:
emoji A character value. Indicates the emoji to replace.
extendedEmoji A character value. Indicates the string to replace for the emojis found.
data A character value. The text where emoji will be replaced.

Returns: The data with the emojis replaced.

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

Usage:
FindEmojiPipe$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also
AbbreviationPipe, ContractionPipe, File2Pipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

FindEmoticonPipe Class to find and/or remove the emoticons on the data field of an Instance

Description
This class is responsible of detecting the existing emoticons in the data field of each Instance. Identified emoticons are stored inside the emoticon field of Instance class. Moreover if required, is able to perform inline emoticon removal.

Details
The regular expression indicated in the emoticonPattern variable is used to identify emoticons.

Note
FindEmoticonPipe will automatically invalidate the Instance whenever the obtained data is empty.

Inherit
This class inherits from GenericPipe and implements the pipe abstract function.

Super class
bdpar::GenericPipe -> FindEmoticonPipe
Public fields

emoticonPattern: A character value. The regular expression to detect emoticons.

Methods

Public methods:

• FindEmoticonPipe$new()
• FindEmoticonPipe$pipe()
• FindEmoticonPipe$findEmoticon()
• FindEmoticonPipe$removeEmoticon()
• FindEmoticonPipe$clone()

Method new(): Creates a FindEmoticonPipe object.

Usage:
FindEmoticonPipe$new(
  propertyName = "emoticon",
  alwaysBeforeDeps = list(),
  notAfterDeps = list("FindHashtagPipe"),
  removeEmoticons = TRUE
)

Arguments:

propertyName: A character value. Name of the property associated with the GenericPipe.
alwaysBeforeDeps: A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps: A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).
removeEmoticons: A logical value. Indicates if the emoticons are removed.
propertyLanguageName: A character value. Name of the language property.

Method pipe(): Preprocesses the Instance to obtain/remove the emoticons. The emoticons found in the data are added to the list of properties of the Instance.

Usage:
FindEmoticonPipe$pipe(instance)

Arguments:

instance: A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method findEmoticon(): Finds the emoticons in the data.

Usage:
FindEmoticonPipe$findEmoticon(data)

Arguments:

data: A character value. The text to search the emoticons.

Returns: The list with emoticons found.
### Method removeEmoticon():
Removes the *emoticons* in the data.

**Usage:**
```r
FindEmoticonPipe$removeEmoticon(data)
```

**Arguments:**
- **data** A *character* value. The text where emoticons will be removed.

**Returns:** The data with the emoticons removed.

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

**Usage:**
```r
FindEmoticonPipe$clone(deep = FALSE)
```

**Arguments:**
- **deep** Whether to make a deep clone.

---

### See Also
- `AbbreviationPipe`
- `ContractionPipe`
- `File2Pipe`
- `FindEmojiPipe`
- `FindHashtagPipe`
- `FindUrlPipe`
- `FindUserNamePipe`
- `GuessDatePipe`
- `GuessLanguagePipe`
- `Instance`
- `InterjectionPipe`
- `MeasureLengthPipe`
- `GenericPipe`
- `SlangPipe`
- `StopWordPipe`
- `StoreFileExtPipe`
- `TargetAssigningPipe`
- `TeeCSVPipe`
- `ToLowerCasePipe`

---

### Description

This class is responsible of detecting the existing hashtags in the *data* field of each *Instance*. Identified hashtags are stored inside the *hashtag* field of *Instance* class. Moreover if required, is able to perform inline hashtag removement.

### Details

The regular expression indicated in the `hashtagPattern` variable is used to identify hashtags.

### Note

`FindHashtagPipe` will automatically invalidate the *Instance* whenever the obtained data is empty.

### Inherit

This class inherits from `GenericPipe` and implements the pipe abstract function.

### Super class

`bdpar::GenericPipe` -> `FindHashtagPipe`
Public fields

hashtagPattern A character value. The regular expression to detect hashtags.

Methods

Public methods:

- FindHashtagPipe$new()
- FindHashtagPipe$pipe()
- FindHashtagPipe$findHashtag()
- FindHashtagPipe$removeHashtag()
- FindHashtagPipe$clone()

Method new(): Creates a FindHashtagPipe object.

Usage:
FindHashtagPipe$new(
  propertyName = "hashtag",
  alwaysBeforeDeps = list(),
  notAfterDeps = list(),
  removeHashtags = TRUE
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).
removeHashtags A logical value. Indicates if the hashtags are removed.
propertyLanguageName A character value. Name of the language property.

Method pipe(): Preprocesses the Instance to obtain/remove the hashtags. The hashtags found in the data are added to the list of properties of the Instance.

Usage:
FindHashtagPipe$pipe(instance)

Arguments:

instance A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method findHashtag(): Finds the hashtags in the data.

Usage:
FindHashtagPipe$findHashtag(data)

Arguments:

data A character value. The text to search the hashtags.

Returns: The list with hashtags found.
**Method** removeHashtag(): Removes the hashtags in the data.

*Usage:*
FindHashtagPipe$removeHashtag(data)

*Arguments:*
data A **character** value. The text where hashtags will be removed.

*Returns:*
The data with the hashtags removed.

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

*Usage:*
FindHashtagPipe$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.

---

**See Also**
AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

FindUrlPipe

---

**Class to find and/or remove the URLs on the data field of an Instance**

---

**Description**
This class is responsible of detecting the existing URLs in the **data** field of each **Instance**. Identified URLs are stored inside the **URLs** field of **Instance** class. Moreover if required, is able to perform inline URLs removement.

**Details**
The regular expressions indicated in the **URLPatterns** variable are used to identify URLs.

**Note**
*FindUrlPipe* will automatically invalidate the **Instance** whenever the obtained data is empty.

**Inherit**
This class inherits from **GenericPipe** and implements the pipe abstract function.

**Super class**
bdpar::GenericPipe -> FindUrlPipe
Public fields

- **URLPattern** A character value. The regular expression to detect URLs.
- **EmailPattern** A character value. The regular expression to detect emails.

Methods

Public methods:

- `FindUrlPipe$new()`
- `FindUrlPipe$pipe()`
- `FindUrlPipe$findUrl()`
- `FindUrlPipe$removeUrl()`
- `FindUrlPipe$putNamesURLPattern()`
- `FindUrlPipe$getURLPatterns()`
- `FindUrlPipe$setURLPatterns()`
- `FindUrlPipe$getNamesURLPatterns()`
- `FindUrlPipe$setNamesURLPatterns()`
- `FindUrlPipe$clone()`

Method `new()`: Creates a **FindUrlPipe** object.

**Usage:**

```r
FindUrlPipe$new(
  propertyName = "URLs",
  alwaysBeforeDeps = list(),
  notAfterDeps = list("FindUrlPipe"),
  removeUrls = TRUE,
  URLPatterns = list(self$URLPattern, self$EmailPattern),
  namesURLPatterns = list("UrlPattern", "EmailPattern")
)
```

**Arguments:**

- `propertyName` A character value. Name of the property associated with the **GenericPipe**.
- `alwaysBeforeDeps` A list value. The dependencies alwaysBefore (**GenericPipes** that must be executed before this one).
- `notAfterDeps` A list value. The dependencies notAfter (**GenericPipes** that cannot be executed after this one).
- `removeUrls` A logical value. Indicates if the URLs are removed.
- `URLPatterns` A list value. The regex to find URLs.
- `namesURLPatterns` A list value. The names of regex.
- `propertyLanguageName` A character value. Name of the language property.

Method `pipe()`: Preprocesses the **Instance** to obtain/remove the URLs. The URLs found in the data are added to the list of properties of the **Instance**.

**Usage:**

```r
FindUrlPipe$pipe(instance)
```

**Arguments:**
instance A **Instance** value. The **Instance** to preprocess.

*Returns:* The **Instance** with the modifications that have occurred in the pipe.

**Method** `findUrl()`: Finds the **URLs** in the data.

*Usage:*
`FindUrlPipe$findUrl(pattern, data)`

*Arguments:*
- **pattern** A **character** value. The regex to find URLs.
- **data** A **character** value. The text to find the URLs.

*Returns:* The list with URLs found.

**Method** `removeUrl()`: Removes the URL in the data.

*Usage:*
`FindUrlPipe$removeUrl(pattern, data)`

*Arguments:*
- **pattern** A **character** value. The regex to find URLs.
- **data** A **character** value. The text to remove the URLs.

*Returns:* The data with URLs removed.

**Method** `putNamesURLPattern()`: Sets the names to **URL patterns** result.

*Usage:*
`FindUrlPipe$putNamesURLPattern(resultOfURLPatterns)`

*Arguments:*
- **resultOfURLPatterns** A **list** value. The list with URLs found.

*Returns:* The URLs found with the names of URL pattern.

**Method** `getURLPatterns()`: Gets the **URL patterns**.

*Usage:*
`FindUrlPipe$getURLPatterns()`

*Returns:* Value of **URL patterns**.

**Method** `setURLPatterns()`: Sets the **URL patterns**.

*Usage:*
`FindUrlPipe$setURLPatterns(URLPatterns)`

*Arguments:*
- **URLPatterns** A **list** value. The new value of the **URL patterns**.

**Method** `getNamesURLPatterns()`: Gets the names of **URLs**.

*Usage:*
`FindUrlPipe$getNamesURLPatterns()`

*Returns:* Value of names of URLs.
**Method** setNamesURLPatterns(): Sets the names of URLs.

*Usage:*

```
FindUrlPipe$setNamesURLPatterns(namesURLPatterns)
```

*Arguments:*

namesURLPatterns  A list value. The new value of the names of URLs.

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

*Usage:*

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

*Arguments:*

deep  Whether to make a deep clone.

---

**FindUserNamePipe**  
*Class to find and/or remove the users on the data field of an Instance*

**Description**

This class is responsible of detecting the existing user names in the data field of each Instance. Identified user names are stored inside the **userName** field of Instance class. Moreover if required, is able to perform inline user name removement.

**Details**

The regular expressions indicated in the userPattern variable are used to identify user names.

**Note**

**FindUserNamePipe** will automatically invalidate the Instance whenever the obtained data is empty.

**Inherit**

This class inherits from **GenericPipe** and implements the pipe abstract function.

**Super class**

```
bdpar::GenericPipe -> FindUserNamePipe
```

**Public fields**

userPattern  A character value. The regular expression to detect name users.
Methods

Public methods:

- `FindUserNamePipe$new()`
- `FindUserNamePipe$pipe()`
- `FindUserNamePipe$findUserName()`
- `FindUserNamePipe$removeUserName()`
- `FindUserNamePipe$clone()`

**Method new()**: Creates a `FindEmoticonPipe` object.

**Usage**:

```r
FindUserNamePipe$new(
  propertyName = "userName",
  alwaysBeforeDeps = list(),
  notAfterDeps = list(),
  removeUser = TRUE
)
```

**Arguments**:

- `propertyName` A `character` value. Name of the property associated with the `GenericPipe`.
- `alwaysBeforeDeps` A `list` value. The dependencies alwaysBefore (`GenericPipes` that must be executed before this one).
- `notAfterDeps` A `list` value. The dependencies notAfter (`GenericPipes` that cannot be executed after this one).
- `removeUser` A `logical` value. Indicates if the name users are removed.
- `propertyLanguageName` A `character` value. Name of the language property.

**Method pipe()**: Preprocesses the `Instance` to obtain/remove the name users. The emoticons found in the data are added to the list of properties of the `Instance`.

**Usage**:

```r
FindUserNamePipe$pipe(instance)
```

**Arguments**:

- `instance` A `Instance` value. The `Instance` to preprocess.

**Returns**: The `Instance` with the modifications that have occurred in the pipe.

**Method findUserName()**: Finds the `name users` in the data.

**Usage**:

```r
FindUserNamePipe$findUserName(data)
```

**Arguments**:

- `data` A `character` value. The text to search the name users.

**Returns**: The `list` with name users found.

**Method removeUserName()**: Removes the `name users` in the data.

**Usage**:

```r
FindUserNamePipe$removeUserName(data)
```
Arguments:
data  A character value. The text where name users will be removed.

Returns: The data with the name users removed.

Method clone(): The objects of this class are cloneable with this method.
Usage:
FindUserNamePipe$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

See Also
AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe,
FindUrlPipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe,
GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe,
ToLowerCasePipe

GenericPipe

Abstract super class that handles the management of the Pipes

Description
Provides the required methods to successfully handle each GenericPipe class.

Methods
Public methods:
• GenericPipe$new()
• GenericPipe$pipe()
• GenericPipe$getPropertyName()
• GenericPipe$getAlwaysBeforeDeps()
• GenericPipe$getNotAfterDeps()
• GenericPipe$setPropertyName()
• GenericPipe$setAlwaysBeforeDeps()
• GenericPipe$setNotAfterDeps()
• GenericPipe$hash()
• GenericPipe$clone()

Method new(): Creates a GenericPipe object.
Usage:
GenericPipe$new(propertyName, alwaysBeforeDeps, notAfterDeps)

Arguments:
propertyName  A character value. Name of the property associated with the Pipe.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (Pipes that must be executed before this one).

notAfterDeps A list value. The dependencies notAfter (Pipes that cannot be executed after this one).

**Method pipe():** Abstract method to preprocess the Instance.

*Usage:*
GenericPipe$pipe(instance)

*Arguments:*
instance A Instance value. The Instance to preprocess.

*Returns:* The preprocessed Instance.

**Method getPropertyName():** Gets of name of property.

*Usage:*
GenericPipe$getPropertyName()

*Returns:* Value of name of property.

**Method getAlwaysBeforeDeps():** Gets of the dependencies always before.

*Usage:*
GenericPipe$getAlwaysBeforeDeps()

*Returns:* Value of dependencies always before.

**Method getNotAfterDeps():** Gets of the dependencies not after.

*Usage:*
GenericPipe$getNotAfterDeps()

*Returns:* Value of dependencies not after.

**Method setPropertyName():** Changes the value of property’s name.

*Usage:*
GenericPipe$setPropertyName(propertyName)

*Arguments:*
propertyName A character value. The new value of the property’s name.

**Method setAlwaysBeforeDeps():** Changes the value of dependencies always before.

*Usage:*
GenericPipe$setAlwaysBeforeDeps(alwaysBeforeDeps)

*Arguments:*
alwaysBeforeDeps A list value. The new value of the dependencies always before.

**Method setNotAfterDeps():** Changes the value of dependencies not after.

*Usage:*
GenericPipe$setNotAfterDeps(notAfterDeps)

*Arguments:*
notAfterDeps A list value. The new value of the dependencies not after.

**Method** hash(): Generates an identification of pipe based on its fields.

*Usage:*
GenericPipe$hash(algo = "md5")

*Arguments:*
algo Algorithm to be applied. Options: "md5", "sha1", "crc32", "sha256", "sha512", "xxhash32", "xxhash64", "murmur32", "spookyhash"

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

*Usage:*
GenericPipe$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.

**See Also**
AbbreviationPipe, bdpar.log, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

**GenericPipeline**

Abstract super class implementing the pipelining process

**Description**

Abstract super class to establish the flow of Pipes.

**Methods**

**Public methods:**

- GenericPipeline$new()
- GenericPipeline$execute()
- GenericPipeline$get()
- GenericPipeline$toString()
- GenericPipeline$clone()

**Method** new(): Creates a GenericPipeline object.

*Usage:*
GenericPipeline$new()

**Method** execute(): Function where is implemented the flow of the GenericPipes.

*Usage:*

GuessDatePipe

GenericPipeline$execute(instance)

*Arguments:*
instance A *Instance* value. The *Instance* that is going to be processed.

*Returns:* The preprocessed *Instance*.

**Method get():** Gets a list with containing the set of *GenericPipes* of the pipeline.

*Usage:*
GenericPipeline$get()

*Returns:* The set of *GenericPipes* containing the pipeline.

**Method toString():** Returns a *character* representing the pipeline.

*Usage:*
GenericPipeline$toString()

*Details:* This function allows to set a place to define a *character* representation of the structure of a pipeline.

*Returns:* *GenericPipeline character* representation

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

*Usage:*
GenericPipeline$clone(deep = FALSE)

*Arguments:*
depth Whether to make a deep clone.

See Also

bdpar.log, DefaultPipeline, DynamicPipeline, Instance, GenericPipe, %>%

---

**GuessDatePipe**

*Class to obtain the date field of an Instance*

**Description**

Obtains the *date* using the method which implements the subclass of *Instance*.

**Inherit**

This class inherit from *GenericPipe* and implements the pipe abstract function.

**Super class**

`bdpar::GenericPipe` -> `GuessDatePipe`
Methods

Public methods:

• `GuessDatePipe$new()`
• `GuessDatePipe$pipe()`
• `GuessDatePipe$clone()`

Method `new()`: Creates a `GuessDatePipe` object.

*Usage:*

`GuessDatePipe$new(
  propertyName = "date",
  alwaysBeforeDeps = list("TargetAssigningPipe"),
  notAfterDeps = list()
)`

*Arguments:*

`propertyName` A `character` value. Name of the property associated with the `GenericPipe`.

`alwaysBeforeDeps` A `list` value. The dependencies alwaysBefore (`GenericPipes` that must be executed before this one).

`notAfterDeps` A `list` value. The dependencies notAfter (`GenericPipes` that cannot be executed after this one).

Method `pipe()`: Preprocesses the `Instance` to obtain the date.

*Usage:*

`GuessDatePipe$pipe(instance)`

*Arguments:*

`instance` A `Instance` value. The `Instance` to preprocess.

*Returns:* The `Instance` with the modifications that have occurred in the pipe.

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

*Usage:*

`GuessDatePipe$clone(deep = FALSE)`

*Arguments:*

`deep` Whether to make a deep clone.

See Also

**GuessLanguagePipe**

*Class to guess the language of an Instance*

**Description**

This class allows guess the language by using language detector of library cld2. Creates the **language** property which indicates the idiom text. Optionally, it is possible to choose the language provided by Twitter.

**Details**

To obtain the language of the tweets, it will be verified that there is a json file with the information stored in memory. On the other hand, it is necessary define the "cache.twitter.path" field of *bdpar.Options* variable to know where the information of tweets are saved.

**Note**

The Pipe will invalidate the Instance if the language of the data can not be detect.

**Inherit**

This class inherits from *GenericPipe* and implements the pipe abstract function.

**Super class**

*bdpar::GenericPipe* -> GuessLanguagePipe

**Methods**

**Public methods:**

- *GuessLanguagePipe$new()*
- *GuessLanguagePipe$pipe()*
- *GuessLanguagePipe$getLanguage()*
- *GuessLanguagePipe$clone()*

**Method** `new()`: Creates a *GuessLanguagePipe* object.

*Usage:*

```r
GuessLanguagePipe$new(
  propertyName = "language",
  alwaysBeforeDeps = list("StoreFileExtPipe", "TargetAssigningPipe"),
  notAfterDeps = list(),
  languageTwitter = TRUE
)
```

**Arguments:**

- `propertyName` A character value. Name of the property associated with the *GenericPipe*. 
Instance

alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).

notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).

languageTwitter A logical value. Indicates whether for the Instances of type twtid the language that returns the API is obtained or the detector is applied.

Method pipe(): Preprocesses the Instance to obtain the language of the data.

Usage:
GuessLanguagePipe$pipe(instance)

Arguments:
instance A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method getLanguage(): Guesses the language of data.

Usage:
GuessLanguagePipe$getLanguage(data)

Arguments:
data A character value. The text to guess the language.


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

Usage:
GuessLanguagePipe$clone(deep = FALSE)

Arguments:
dep Whether to make a deep clone.

See Also

AbbreviationPipe, bdpar.Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

| Instance | Abstract super class that handles the management of the Instances |

Description

Provides the required methods to successfully handle each Instance class.
Methods

**Public methods:**

- `Instance$new()`  
- `Instance$obtainDate()`  
- `Instance$obtainSource()`  
- `Instance$getDate()`  
- `Instance$getSource()`  
- `Instance$getPath()`  
- `Instance$getData()`  
- `Instance$getProperties()`  
- `Instance$setSource()`  
- `Instance$setData()`  
- `Instance$setDate()`  
- `Instance$setProperties()`  
- `Instance$addProperties()`  
- `Instance$getSpecificProperty()`  
- `Instance$isSpecificProperty()`  
- `Instance$setSpecificProperty()`  
- `Instance$getNamesOfProperties()`  
- `Instance$isInstanceValid()`  
- `Instance$invalidate()`  
- `Instance$getFlowPipes()`  
- `Instance$addFlowPipes()`  
- `Instance$getBanPipes()`  
- `Instance$addBanPipes()`  
- `Instance$checkCompatibility()`  
- `Instance$toString()`  
- `Instance$clone()`  

**Method** `new()`: Creates a `Instance` object.

Usage:

`Instance$new(path)`

Arguments:

- path A character value. Path of the file.

**Method** `obtainDate()`: Abstract function responsible for obtaining the date of the `Instance`.

Usage:

`Instance$obtainDate()`

**Method** `obtainSource()`: Abstract function responsible for determining the source of the `Instance`.

Usage:
Method getDate(): Gets the date.
Usage:
Instance.getDate()
Returns: Value of date.

Method getSource(): Gets the source.
Usage:
Instance.getSource()
Returns: Value of source.

Method getPath(): Gets the path.
Usage:
Instance.getPath()
Returns: Value of path.

Method getData(): Gets the data.
Usage:
Instance.getData()
Returns: Value of data.

Method getProperties(): Gets the properties
Usage:
Instance.getProperties()
Returns: Value of properties.

Method setSource(): Modifies the source value.
Usage:
Instance.setSource(source)
Arguments:
source A character value. The new value of source.

Method setData(): Modifies the data value.
Usage:
Instance.setData(data)
Arguments:
data A character value. The new value of data.

Method setDate(): Modifies the date value.
Usage:
Instance.setDate(date)
Arguments:
date  A character value. The new value of date.

Method setProperties(): Modifies the properties value.

Usage:
Instance$setProperties(properties)

Arguments:
properties A list value. The new list of properties.

Method addProperties(): Adds a property to the list of the properties.

Usage:
Instance$addProperties(propertyValue, propertyName)

Arguments:
propertyValue A Object value. The value of the new property.
propertyName A character value. The name of the new property.

Method getSpecificProperty(): Obtains a specific property.

Usage:
Instance$getSpecificProperty(propertyName)

Arguments:
propertyName A character value. The name of the property to obtain.

Returns: The value of the specific property.

Method isSpecificProperty(): Checks for the existence of an specific property.

Usage:
Instance$isSpecificProperty(propertyName)

Arguments:
propertyName A character value. The name of the property to check.

Returns: A logical results according to the existence of the specific property in the list of properties.

Method setSpecificProperty(): Modifies the value of the one property.

Usage:
Instance$setSpecificProperty(propertyName, propertyValue)

Arguments:
propertyName A character value. The name of the property.
propertyValue A Object value. The new value of the property.

Method getNamesOfProperties(): Gets of the names of all properties.

Usage:
Instance$getNamesOfProperties()

Returns: The names of properties.

Method isInstanceValid(): Checks if the Instance is valid.
Usage:
Instance$isInstanceValid()

Returns: Value of isValid flag.

**Method invalidate():** Forces the invalidation of an specific Instance.

Usage:
Instance$invalidate()

**Method getFlowPipes():** Gets the list of the flow of GenericPipe.

Usage:
Instance$getFlowPipes()

Returns: Names of the GenericPipe used.

**Method addFlowPipes():** Gets the list of the flow of GenericPipe.

Usage:
Instance$addFlowPipes(namePipe)

Arguments:
namePipe A **character** value. Name of the new **GenericPipe** to be added in the **GenericPipeline**.

**Method getBanPipes():** Gets an array with containing all the ban **GenericPipe**.

Usage:
Instance$getBanPipes()

Returns: Value of ban **GenericPipe** array.

**Method addBanPipes():** Added the name of the Pipe to the array that keeps the track of **GenericPipes** having running after restrictions.

Usage:
Instance$addBanPipes(namePipe)

Arguments:
namePipe A **character** value. **GenericPipe** name to be introduced into the ban array.

**Method checkCompatibility():** Check compatibility between **GenericPipes**.

Usage:
Instance$checkCompatibility(namePipe, alwaysBefore)

Arguments:
namePipe A **character** value. The name of the **GenericPipe** name to check the compatibility.
alwaysBefore A **list** value. **GenericPipes** that the **Instance** had to go through.

**Method toString():** Returns a **character** representing the instance

Usage:
Instance$toString()

Returns: **Instance character** representation

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

Usage:
Instance$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
InterjectionPipe

See Also

ExtractorEml, ExtractorSms, ExtractorTwtid, ExtractorYtbid

InterjectionPipe

Class to find and/or remove the interjections on the data field of an Instance

Description

InterjectionPipe class is responsible for detecting the existing interjections in the data field of each Instance. Identified interjections are stored inside the interjection field of Instance class. Moreover if needed, is able to perform inline interjections removal.

Details

InterjectionPipe class requires the resource files (in json format) containing the list of interjections. To this end, the language of the text indicated in the propertyLanguageName should be contained in the resource file name (ie. interj.xxx.json where xxx is the value defined in the propertyLanguageName). The location of the resources should be defined in the "resources.interjections.path" field of bdpar.Options variable.

Note

InterjectionPipe will automatically invalidate the Instance whenever the obtained data is empty.

Inherit

This class inherits from GenericPipe and implements the pipe abstract function.

Super class

bdpar::GenericPipe -> InterjectionPipe

Methods

Public methods:

- InterjectionPipe$new()
- InterjectionPipe$pipe()
- InterjectionPipe$findInterjection()
- InterjectionPipe$removeInterjection()
- InterjectionPipe$getPropertyLanguageName()
- InterjectionPipe$getResourcesInterjectionsPath()
- InterjectionPipe$setResourcesInterjectionsPath()
- InterjectionPipe$clone()

Method new(): Creates a InterjectionPipe object.
Usage:
InterjectionPipe$new(
  propertyName = "interjection",
  propertyLanguageName = "language",
  alwaysBeforeDeps = list("GuessLanguagePipe"),
  notAfterDeps = list(),
  removeInterjections = TRUE,
  resourcesInterjectionsPath = NULL
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
propertyLanguageName A character value. Name of the language property.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).
removeInterjections A logical value. Indicates if the interjections are removed or not.
resourcesInterjectionsPath A character value. Path of resource files (in json format) containing the interjections.

Method pipe(): Preprocesses the Instance to obtain/remove the interjections. The interjections found in the data are added to the list of properties of the Instance.

Usage:
InterjectionPipe$pipe(instance)

Arguments:

instance A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method findInterjection(): Checks if the interjection is in the data.

Usage:
InterjectionPipe$findInterjection(data, interjection)

Arguments:

data A character value. The text where interjection will be searched.
interjection A character value. Indicates the interjection to find.

Returns: A logical value depending on whether the interjection is in the data.

Method removeInterjection(): Removes the interjection in the data.

Usage:
InterjectionPipe$removeInterjection(interjection, data)

Arguments:

interjection A character value. Indicates the interjection to remove.
data A character value. The text where interjection will be removed.

Returns: The data with the interjections removed.
**MeasureLengthPipe**

**Method** `getPropertyLanguageName()`: Gets the name of property language.

*Usage:*
InterjectionPipe$getPropertyLanguageName()

*Returns:* Value of name of property language.

**Method** `getResourcesInterjectionsPath()`: Gets the path of interjections resources.

*Usage:*
InterjectionPipe$getResourceInterjectionsPath()

*Returns:* Value of path of interjections resources.

**Method** `setResourcesInterjectionsPath()`: Sets the path of interjections resources.

*Usage:*
InterjectionPipe$setResourcesInterjectionsPath(path)

*Arguments:*
path A character value. The new value of the path of interjections resources.

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

*Usage:*
InterjectionPipe$clone(deep = FALSE)

*Arguments:*
deep Whether to make a deep clone.

**See Also**

AbbreviationPipe, bdpar::Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

**MeasureLengthPipe**

Class to obtain the length of the data field of an Instance

**Description**

This class is responsible of obtaining the length of the *data* field of each *Instance*. Creates the *length* property which indicates the length of the text. The property's name is customize thought the class constructor.

**Inherit**

This class inherits from *GenericPipe* and implements the pipe abstract function.

**Super class**

`bdpar::GenericPipe` -> MeasureLengthPipe
Methods

Public methods:

- `MeasureLengthPipe$new()`
- `MeasureLengthPipe$pipe()`
- `MeasureLengthPipe$getLength()`
- `MeasureLengthPipe$clone()`

**Method new():** Creates a `File2Pipe` object.

*Usage:*

```r
MeasureLengthPipe$new(
  propertyName = "length",
  alwaysBeforeDeps = list(),
  notAfterDeps = list(),
  nchar_conf = TRUE
)
```

*Arguments:*

- `propertyName` A character value. Name of the property associated with the `GenericPipe`.
- `alwaysBeforeDeps` A list value. The dependencies alwaysBefore (`GenericPipes` that must be executed before this one).
- `notAfterDeps` A list value. The dependencies notAfter (`GenericPipes` that cannot be executed after this one).
- `nchar_conf` A logical value. Indicates if the pipe uses nchar or object.size.

**Method pipe():** Preprocesses the `Instance` to obtain the length of data.

*Usage:*

```r
MeasureLengthPipe$pipe(instance)
```

*Arguments:*

- `instance` A `Instance` value. The `Instance` to preprocess.

*Returns:* The `Instance` with the modifications that have occurred in the pipe.

**Method getLength():** Preprocesses the `Instance` to obtain the length of data.

*Usage:*

```r
MeasureLengthPipe$getLength(data, nchar_conf = TRUE)
```

*Arguments:*

- `data` A character value. The text to preprocess.
- `nchar_conf` A logical value. Indicates if the pipe uses nchar or object.size.

*Returns:* The `Instance` with the modifications that have occurred in the pipe.

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

*Usage:*

```r
MeasureLengthPipe$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.
See Also

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

**Description**

Defines a customized forward pipe operator extending the features of classical %>%%. Concretely %>|% is able to stop the pipelining process whenever an Instance has been invalidated. This issue, avoids executing the whole pipelining process for the invalidated Instance and therefore reduce the time and resources used to complete the whole process.

**Usage**

\[\text{lhs} \ %|\% \ \text{rhs}\]

**Arguments**

- \text{lhs} \quad \text{an Instance object.}
- \text{rhs} \quad \text{a function call using the bdpar semantics.}

**Value**

The Instance modified by the methods it has traversed.

**Details**

This is the %>|% operator of the modified magrittr library to both (i) to stop the flow when the Instance is invalid and (ii) automatically call the pipe function of the R6 objects passing through it (iii) to check the dependencies of the Instance and (iv) to manage the pipeline cache.

The usage structure would be as shown below:

\[
\text{instance} \ %>|% \\
\text{pipeObject}$\text{new}() \ %>|% \\
\text{pipeObject}$\text{new}(<\text{argument1}>, <\text{argument2}, ...>) \ %>|% \\
\text{pipeObject}$\text{new}()
\]

**Note**

Pipelining process is automatically stopped if the Instance is invalid.
See Also

`bdpar::Options::Instance::GenericPipe`

---

**ResourceHandler**

Class that handles different types of resources.

**Description**

Class that handles different types of resources.

**Details**

It is a class that allows store the resources that are needed in the `GenericPipes` to avoid having to repeatedly read from the file. File resources of type json are read and stored in memory.

**Methods**

**Public methods:**

- `ResourceHandler::new()`
- `ResourceHandler::isLoadResource()`
- `ResourceHandler::getResources()`
- `ResourceHandler::setResources()`
- `ResourceHandler::getNamesResources()`
- `ResourceHandler::clone()`

**Method new():** Creates a `ResourceHandler` object.

*Usage:

`ResourceHandler::new()`*

**Method isLoadResource():** From the resource path, it is checked if they have already been loaded. In this case, the list of the requested resource is returned. Otherwise, the resource variable is added to the list of resources, and the resource list is returned. In the event that the resource file does not exist, NULL is returned.

*Usage:

`ResourceHandler::isLoadResource(pathResource)`*

*Arguments:

`pathResource` A (character) value. The resource file path.

*Returns: The resources list is returned, if they exist.

**Method getResources():** Gets of resources variable.

*Usage:

`ResourceHandler::getResources()`*

*Returns: The value of resources variable.
Method setResources(): Sets of resources variable.

Usage:
ResourceHandler$setResources(resources)

Arguments:
resources  The new value of resources.

Method getNamesResources(): Gets of names of resources

Usage:
ResourceHandler$getNamesResources()

Returns: Value of names of resources.

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

Usage:
ResourceHandler$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

---

runPipeline

*Initiates the pipelining process*

---

**Description**

**runPipeline** is responsible for easily initialize the pipelining preprocessing process.

**Usage**

```
runPipeline(path, extractors = ExtractorFactory$new(),
  pipeline = DefaultPipeline$new(), cache = TRUE, verbose = FALSE, summary = FALSE)
```

**Arguments**

- **path** *(character)* path where the files to be preprocessed are located.
- **extractors** *(ExtractorFactory)* object implementing the method `createInstance` to choose which type of `Instance` is created.
- **pipeline** *(GenericPipeline)* subclass of `GenericPipeline`, which implements the whole pipeling process.
- **cache** *(logical)* flag indicating if the status of the instances will be stored after each pipe. This allows to avoid rejections of previously executed tasks, if the order and configuration of the pipe and pipeline is the same as what is stored in the cache.
- **verbose** *(logical)* flag indicating for printing messages, warnings and errors.
- **summary** *(logical)* flag indicating if a summary of the pipeline execution is provided or not.
Value
List of Instance that have been preprocessed.

Details
In the case that some pipe, defined on the workflow, needs some type of configuration, it can be defined through bdpar.Options variable which have different methods to support the functionality of different pipes.

See Also
Bdpar, bdpar.Options, Connections, DefaultPipeline, DynamicPipeline, GenericPipeline, Instance, ExtractorFactory, ResourceHandler

Examples
```r
## Not run:

# If it is necessary to indicate any existing configuration key, do it through:
# bdpar.Options$set(key, value)
# If the key is not initialized, do it through:
# bdpar.Options$add(key, value)

# If it is necessary parallelize, do it through:
# bdpar.Options$set("numCores", numCores)

# If it is necessary to change the behavior of the log, do it through:
# bdpar.Options$configureLog(console = TRUE, threshold = "INFO", file = NULL)

# Folder with the files to preprocess
path <- system.file("example",
                   package = "bdpar")

# Object which decides how creates the instances
extractors <- ExtractorFactory$new()

# Object which indicates the pipes' flow
pipeline <- DefaultPipeline$new()

# Starting file preprocessing...
runPipeline(path = path,
            extractors = extractors,
            pipeline = pipeline,
            cache = FALSE,
            verbose = FALSE,
            summary = TRUE)

## End(Not run)
```
**Description**

*SlangPipe* class is responsible for detecting the existing slangs in the *data* field of each *Instance*. Identified slangs are stored inside the *slang* field of *Instance* class. Moreover if needed, is able to perform inline slangs replacement.

**Details**

*SlangPipe* class requires the resource files (in json format) containing the correspondence between slangs and meaning. To this end, the language of the text indicated in the *propertyLanguageName* should be contained in the resource file name (ie. slang.xxx.json where xxx is the value defined in the *propertyLanguageName*). The location of the resources should be defined in the "resources.slangs.path" field of *bdpar.Options* variable.

**Note**

*SlangPipe* will automatically invalidate the *Instance* whenever the obtained data is empty.

**Inherit**

This class inherits from *GenericPipe* and implements the *pipe* abstract function.

**Super class**

*bdpar::GenericPipe* \rightarrow SlangPipe

**Methods**

Public methods:

- *SlangPipe$new()*
- *SlangPipe$pipe()*
- *SlangPipe$findSlang()*
- *SlangPipe$replaceSlang()*
- *SlangPipe$getPropertyLanguageName()*
- *SlangPipe$getResourcesSlangsPath()*
- *SlangPipe$setResourcesSlangsPath()*
- *SlangPipe$clone()*

**Method** *new()*: Creates a *SlangPipe* object.

*Usage:*
SlangPipe$new(
  propertyName = "langpropname",
  propertyLanguageName = "language",
  alwaysBeforeDeps = list("GuessLanguagePipe"),
  notAfterDeps = list(),
  replaceSlangs = TRUE,
  resourcesSlangsPath = NULL
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
propertyLanguageName A character value. Name of the language property.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).
replaceSlangs A logical value. Indicates if the slangs are replaced or not.
resourcesSlangsPath A character value. Path of resource files (in json format) containing the correspondence between slangs and meaning.

Method pipe(): Preprocesses the Instance to obtain/replace the slangs. The slangs found in the data are added to the list of properties of the Instance.

Usage:
SlangPipe$pipe(instance)

Arguments:
instance A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method findSlang(): Checks if the slang is in the data.

Usage:
SlangPipe$findSlang(data, slang)

Arguments:
data A character value. The text where slang will be searched.
slang A character value. Indicates the slang to find.

Returns: A logical value depending on whether the slang is in the data.

Method replaceSlang(): Replaces the slang in the data for the extendedSlang.

Usage:
SlangPipe$replaceSlang(slang, extendedSlang, data)

Arguments:
slang A character value. Indicates the slang to replace.
extendedSlang A character value. Indicates the string to replace for the slangs found.
data A character value. The text where slang will be replaced.

Returns: The data with the slangs replaced.
StopWordPipe

Method getPropertyLanguageName(): Gets the name of property language.
Usage:
SlangPipe$getPropertyLanguageName()
Returns: Value of name of property language.

Method getResourcesSlangsPath(): Gets the path of slangs resources.
Usage:
SlangPipe$getResourcesSlangsPath()
Returns: Value of path of slangs resources.

Method setResourcesSlangsPath(): Sets the path of slangs resources.
Usage:
SlangPipe$setResourcesSlangsPath(path)
Arguments:
path A character value. The new value of the path of slangs resources.

Method clone(): The objects of this class are cloneable with this method.
Usage:
SlangPipe$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

See Also
AbbreviationPipe, bdpar.Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe,
FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance,
InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, StopWordPipe, StoreFileExtPipe,
TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

StopWordPipe

Class to find and/or remove the stop words on the data field of an
Instance

Description
StopWordPipe class is responsible for detecting the existing stop words in the data field of each
Instance. Identified stop words are stored inside the contraction field of Instance class. Moreover if needed, is able to perform inline stop words removal.

Details
StopWordPipe class requires the resource files (in json format) containing the list of stop words. To this end, the language of the text indicated in the propertyLanguageName should be contained in the resource file name (ie. xxx.json where xxx is the value defined in the propertyLanguageName ). The location of the resources should be defined in the "resources.stopwords.path" field of bdpar.Options variable.
**Note**

*StopWordPipe* will automatically invalidate the *Instance* whenever the obtained data is empty.

**Inherit**

This class inherits from *GenericPipe* and implements the pipe abstract function.

**Super class**

`bdpar::GenericPipe` -> *StopWordPipe*

**Methods**

**Public methods:**

- `StopWordPipe$new()`
- `StopWordPipe$pipe()`
- `StopWordPipe$findStopWord()`
- `StopWordPipe$removeStopWord()`
- `StopWordPipe$getPropertyLanguageName()`
- `StopWordPipe$getResourcesStopWordsPath()`
- `StopWordPipe$setResourcesStopWordsPath()`
- `StopWordPipe$clone()`

**Method** `new()`: Creates a *StopWordPipe* object.

*Usage:*

```r
StopWordPipe$new(
  propertyName = "stopWord",
  propertyLanguageName = "language",
  alwaysBeforeDeps = list("GuessLanguagePipe"),
  notAfterDeps = list("AbbreviationPipe"),
  removeStopWords = TRUE,
  resourcesStopWordsPath = NULL
)
```

*Arguments:*

- `propertyName` A *character* value. Name of the property associated with the *GenericPipe*.
- `propertyLanguageName` A *character* value. Name of the language property.
- `alwaysBeforeDeps` A *list* value. The dependencies alwaysBefore (*GenericPipes* that must be executed before this one).
- `notAfterDeps` A *list* value. The dependencies notAfter (*GenericPipes* that cannot be executed after this one).
- `removeStopWords` A *logical* value. Indicates if the stop words are removed or not.
- `resourcesStopWordsPath` A *character* value. Path of resource files (in json format) containing the stop words.

**Method** `pipe()`: Preprocesses the *Instance* to obtain/remove the stop words. The stop words found in the data are added to the list of properties of the *Instance*. 
StopWordPipe

Usage:
StopWordPipe$pipe(instance)

Arguments:
instant A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method findStopWord(): Checks if the stop word is in the data.

Usage:
StopWordPipe$findStopWord(data, stopWord)

Arguments:
data A character value. The text where stop word will be searched.
stopWord A character value. Indicates the stop word to find.

Returns: A logical value depending on whether the stop word is in the data.

Method removeStopWord(): Removes the stop word in the data.

Usage:
StopWordPipe$removeStopWord(stopWord, data)

Arguments:
stopWord A character value. Indicates the stop word to remove.
data A character value. The text where stop word will be removed.

Returns: The data with the stop words removed.

Method getPropertyLanguageName(): Gets the name of property language.

Usage:
StopWordPipe$getPropertyLanguageName()

Returns: Value of name of property language.

Method getResourcesStopWordsPath(): Gets the path of stop words resources.

Usage:
StopWordPipe(getResourcesStopWordsPath())

Returns: Value of path of stop words resources.

Method setResourcesStopWordsPath(): Sets the path of stop words resources.

Usage:
StopWordPipe$setResourcesStopWordsPath(path)

Arguments:
path A character value. The new value of the path of stop words resources.

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

Usage:
StopWordPipe$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
StoreFileExtPipe

See Also

AbbreviationPipe, bdpar::Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

StoreFileExtPipe  
Class to get the file's extension field of an Instance

Description

Gets the extension of a file. Creates the extension property which indicates extension of the file.

Note

StoreFileExtPipe will automatically invalidate the Instance if it is not able to find the extension from the path field.

Inherit

This class inherits from GenericPipe and implements the pipe abstract function.

Super class

bdpar::GenericPipe -> StoreFileExtPipe

Methods

Public methods:

- StoreFileExtPipe$new()
- StoreFileExtPipe$pipe()
- StoreFileExtPipe$obtainExtension()
- StoreFileExtPipe$clone()

Method new(): Creates a StoreFileExtPipe object.

Usage:

StoreFileExtPipe$new(
  propertyName = "extension",
  alwaysBeforeDeps = list(),
  notAfterDeps = list()
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps  A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).

**Method** pipe(): Preprocesses the Instance to obtain the extension of Instance.

*Usage:*
StoreFileExtPipe$pipe(instance)

*Arguments:*
instance  A Instance value. The Instance to preprocess.

*Returns:* The Instance with the modifications that have occurred in the pipe.

**Method** obtainExtension(): Gets of extension of the path.

*Usage:*
StoreFileExtPipe$obtainExtension(path)

*Arguments:*
path  A character value. The path of the file to get the extension.

*Returns:* Extension of the path.

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

*Usage:*
StoreFileExtPipe$clone(deep = FALSE)

*Arguments:*
deep  Whether to make a deep clone.

**See Also**

AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, TargetAssigningPipe, TeeCSVPipe, ToLowerCasePipe

---

**TargetAssigningPipe**  
Class to get the target field of the Instance

**Description**

This class allows searching in the path the target of the Instance.

**Details**

The targets that are searched can be controlled through the constructor of the class where target- 
sName will be the string that is searched within the path and targets has the values that the property can take.
Inherit
This class inherits from GenericPipe and implements the pipe abstract function.

Super class
bdpar::GenericPipe -> TargetAssigningPipe

Methods

Public methods:
• TargetAssigningPipe$new()
• TargetAssigningPipe$pipe()
• TargetAssigningPipe$getTarget()
• TargetAssigningPipe$checkTarget()
• TargetAssigningPipe$getTargets()
• TargetAssigningPipe$clone()

Method new(): Creates a TargetAssigningPipe object.

Usage:
TargetAssigningPipe$new(
    targets = list("ham", "spam"),
    targetsName = list("_ham_", "_spam_"),
    propertyName = "target",
    alwaysBeforeDeps = list(),
    notAfterDeps = list()
)

Arguments:
 targets A list value. Name of the targets property.
 targetsName A list value. The name of folders.
 propertyName A character value. Name of the property associated with the GenericPipe.
 alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
 notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).

Method pipe(): Preprocesses the Instance to obtain the target.

Usage:
TargetAssigningPipe$pipe(instance)

Arguments:
 instance A Instance value. The Instance to preprocess.

Returns: The Instance with the modifications that have occurred in the pipe.

Method getTarget(): Gets the target from a path.

Usage:
TeeCSVPipe

TargetAssigningPipe$getTarget(path)

Arguments:
path A character value. The path to analyze.

Returns: The target of the path.

Method checkTarget(): Checks if the target is in the path.

Usage:
TargetAssigningPipe$checkTarget(target, path)

Arguments:
target A character value. The target to find in the path.
path A character value. The path to analyze.

Returns: if the target is found, returns target, else returns "".

Method getTargets(): Gets of targets.

Usage:
TargetAssigningPipe$getTargets()

Returns: Value of targets.

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

Usage:
TargetAssigningPipe$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

See Also
AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TeeCSVPipe, ToLowerCasePipe

TeeCSVPipe

Class to handle a CSV with the properties field of the preprocessed
Instance

Description
Complete a CSV with the properties of the preprocessed Instance.

Details
The path to save the properties should be defined in the "teeCSVPipe.output.path" field of bd-par.Options variable.
Inherit

This class inherits from `GenericPipe` and implements the `pipe` abstract function.

Super class

`bdpar::GenericPipe` -> `TeeCSVPipe`

Methods

**Public methods:**

- `TeeCSVPipe$new()`
- `TeeCSVPipe$pipe()`
- `TeeCSVPipe$clone()`

**Method `new()`:** Creates a `TeeCSVPipe` object.

*Usage:*

```r
TeeCSVPipe$new(
  propertyName = "",
  alwaysBeforeDeps = list(),
  notAfterDeps = list(),
  withData = TRUE,
  withSource = TRUE,
  outputPath = NULL
)
```

*Arguments:*

- `propertyName` A `character` value. Name of the property associated with the `GenericPipe`.
- `alwaysBeforeDeps` A `list` value. The dependencies alwaysBefore (`GenericPipes` that must be executed before this one).
- `notAfterDeps` A `list` value. The dependencies notAfter (`GenericPipes` that cannot be executed after this one).
- `withData` A `logical` value. Indicates if the data is added to CSV.
- `withSource` A `logical` value. Indicates if the source is added to CSV.
- `outputPath` A `character` value. The path of CSV.

**Method `pipe()`:** Completes the CSV with the preprocessed `Instance`.

*Usage:*

```r
TeeCSVPipe$pipe(instance)
```

*Arguments:*

- `instance` A `Instance` value. The `Instance` to preprocess.

*Returns:* The `Instance` with the modifications that have occurred in the pipe.

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

*Usage:*

```r
TeeCSVPipe$clone(deep = FALSE)
```

*Arguments:*

- `deep` Whether to make a deep clone.
ToLowerCasePipe

See Also

AbbreviationPipe, bdpar::Options, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashTagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, ToLowerCasePipe

ToLowerCasePipe

Description

Class to convert the data field of an Instance to lower case.

Inherit

This class inherits from GenericPipe and implements the pipe abstract function.

Super class

bdpar::GenericPipe -> ToLowerCasePipe

Methods

Public methods:

• ToLowerCasePipe$new()
• ToLowerCasePipe$pipe()
• ToLowerCaseCasePipe$toLowerCase()
• ToLowerCasePipe$clone()

Method new(): Creates a ToLowerCasePipe object.

Usage:
ToLowerCasePipe$new(
  propertyName = "", 
  alwaysBeforeDeps = list(),
  notAfterDeps = list()
)

Arguments:

propertyName A character value. Name of the property associated with the GenericPipe.
alwaysBeforeDeps A list value. The dependencies alwaysBefore (GenericPipes that must be executed before this one).
notAfterDeps A list value. The dependencies notAfter (GenericPipes that cannot be executed after this one).

Method pipe(): Preprocesses the Instance to convert the data to lower case.

Usage:
ToLowerCasePipe

ToLowerCasePipe$pipe(instance)

Arguments:
instance  A Instance value. The Instance to preprocess.

Returns:  The Instance with the modifications that have occurred in the pipe.

Method toLowerCase(): Converts the data to lower case

Usage:
ToLowerCasePipe$toLowerCase(data)

Arguments:
data  A character value. Text to preprocess.

Returns:  The data in lower case.

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

Usage:
ToLowerCasePipe$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

See Also
AbbreviationPipe, ContractionPipe, File2Pipe, FindEmojiPipe, FindEmoticonPipe, FindHashtagPipe, FindUrlPipe, FindUserNamePipe, GuessDatePipe, GuessLanguagePipe, Instance, InterjectionPipe, MeasureLengthPipe, GenericPipe, ResourceHandler, SlangPipe, StopWordPipe, StoreFileExtPipe, TargetAssigningPipe, TeeCSVPipe
Index

* datasets
  bdparData, 12

   AbbreviationPipe, 3, 3, 4, 12, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   Bdpar, 6, 6, 62
   bdpar.log, 8, 12, 20, 22, 46, 47
   bdpar.Options, 3, 5–7, 9, 9, 11, 13–15, 17, 22, 23, 28–31, 49, 50, 55, 57, 60, 62, 63, 65, 68, 71, 73
   bdpar:::GenericPipe, 3, 15, 32, 33, 35, 37, 39, 42, 47, 49, 55, 57, 63, 66, 68, 70, 72, 73
   bdpar:::GenericPipeline, 19, 20
   bdpar:::Instance, 22, 26, 28, 30
   bdparData, 12

   character, 4–6, 16, 17, 19, 21–25, 27–32, 34–45, 47–54, 56–58, 64–74
   Connections, 6, 7, 12, 13, 13, 28–31, 62
   ContractionPipe, 5, 12, 15, 15, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   DefaultPipeline, 7, 17, 17, 19, 22, 47, 62
   DynamicPipeline, 7, 20, 20, 21, 47, 62

   ExtractorEml, 12, 22, 22, 26, 27, 29, 31, 55
   ExtractorFactory, 7, 24, 24, 62
   ExtractorSms, 23, 26, 26, 27, 29, 31, 55
   ExtractorTwtid, 12, 14, 23, 26–28, 28, 31, 55
   ExtractorYtbid, 12, 14, 23, 26, 27, 29, 30, 30, 55

   File2Pipe, 5, 17, 32, 32, 35, 37, 39, 42, 44, 46, 48, 50, 57–59, 65, 68, 69, 71, 73, 74

   FindEmojiPipe, 5, 17, 33, 33, 34, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   FindEmoticonPipe, 5, 17, 33, 35, 36, 39, 42–44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   FindHashtagPipe, 5, 17, 33, 35, 37, 37, 38, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   FindUrlPipe, 5, 17, 33, 35, 37, 39, 39, 40, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   FindUserNamePipe, 5, 17, 33, 37, 37, 39, 42, 42, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74


   GenericPipeline, 6, 7, 17, 18, 20, 22, 46, 46, 47, 54, 61, 62

   GuessDatePipe, 5, 17, 33, 35, 37, 39, 42, 44, 46, 47, 48, 50, 57, 59, 65, 68, 69, 71, 73, 74

   GuessLanguagePipe, 5, 12, 17, 33, 35, 37, 39, 42, 44, 46, 48, 49, 49, 57, 59, 65, 68, 69, 71, 73, 74

   Instance, 3–5, 7, 12, 15–17, 19–50, 50, 51, 53–74

   InterjectionPipe, 5, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 55, 55, 59, 65, 68, 69, 71, 73, 74

   list, 4, 16, 20, 21, 32, 34, 36, 38, 40–43, 45, 46, 48, 50, 53, 54, 56, 58, 64, 66, 68–70, 72, 73

   logical, 4, 16, 34, 36, 38, 40, 43, 50, 56, 58, 64, 66, 67, 72
MeasureLengthPipe, 5, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 65, 68, 69, 71, 73, 74
message, 8
operator-pipe, 59
ResourceHandler, 5–7, 17, 46, 57, 59, 60, 62, 63, 68, 69, 71, 73, 74
runPipeline, 7, 61
SlangPipe, 5, 12, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 63, 66, 68, 69, 71, 73, 74
stop, 8
StopWordPipe, 5, 12, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 66, 68, 69, 71, 73, 74
StoreFileExtPipe, 5, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 68, 71, 73, 74
TargetAssigningPipe, 5, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 70, 73, 74
TeeCSVPipe, 5, 12, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 72, 74
ToLowerCasePipe, 5, 17, 33, 35, 37, 39, 42, 44, 46, 48, 50, 57, 59, 65, 68, 69, 71, 73, 73

warning, 8