Package ‘cbpManager’

April 7, 2026

Type Package

Title Generate, manage, and edit data and metadata files suitable for the import in cBioPortal for Cancer Genomics

Version 1.18.0

Date 2021-03-16

Description This R package provides an R Shiny application that enables the user to generate, manage, and edit data and metadata files suitable for the import in cBioPortal for Cancer Genomics.
Create cancer studies and edit its metadata. Upload mutation data of a patient that will be concatenated to the data_mutation_extended.txt file of the study.
Create and edit clinical patient data, sample data, and timeline data. Create custom timeline tracks for patients.

License AGPL-3 + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

VignetteBuilder knitr

URL https://arsenij-ust.github.io/cbpManager/index.html

BugReports https://github.com/arsenij-ust/cbpManager/issues

Depends shiny, shinydashboard

Imports utils, DT, htmltools, vroom, plyr, dplyr, magrittr, jsonlite, rapportools, basilisk, reticulate, shinyBS, shinycssloaders, rintrojs, rlang, markdown

Suggests knitr, BiocStyle, rmarkdown, testthat (>= 3.0.0)

StagedInstall no

Collate 'basilisk.R' 'dynamicTable.R' 'modulesResourceButtons.R' 'cbpManager-pkg.R' 'functions.R' 'global.R' 'cbpManager.R' 'shinyAppServer.R' 'shinyAppUI.R'

biocViews ImmunoOncology, DataImport, DataRepresentation, GUI, ThirdPartyClient, Preprocessing, Visualization

Config/testthat/edition 3

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

git_branch RELEASE_3_22

git_last_commit 5565444

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-04-07

Author Arsenij Ustjanzew [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1014-4521>), Federico Marini [aut] (ORCID: <https://orcid.org/0000-0003-3252-7758>)

Maintainer Arsenij Ustjanzew <arsenij.ustjanzew@gmail.com>

Contents

`addColumn_Server` *Server logic of module for adding a column*

Description

Server logic of module for adding a column

Usage

```
addColumn_Server(input, output, session, data)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	source data as data.frame

Value

reactive data.frame of modified source data

`addColumn_UI` *UI elements of module for adding a column*

Description

UI elements of module for adding a column

Usage

```
addColumn_UI(id, label = "Add column")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`addRowRc_Server` *Server logic of Resource tab module for adding a row*

Description

Server logic of Resource tab module for adding a row

Usage

```
addRowRc_Server(
  input,
  output,
  session,
  data,
  patient_ids = NULL,
  sample_ids = NULL,
  resource_ids = NULL,
  resource_type = c("definition", "sample", "patient", "study")
)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	Source data as data.frame
`patient_ids`	Reactive vector of existing patient IDs
`sample_ids`	Reactive data.frame of existing patient IDs and sample IDs
`resource_ids`	Reactive data.frame of data_resource_definition
`resource_type`	The type of the resource. Can be "definition", "sample", "patient", "study"

Value

reactive data.frame of modified source data

`addRowRc_UI` *UI elements of Resource tab module for adding a row*

Description

UI elements of Resource tab module for adding a row

Usage

```
addRowRc_UI(id, label = "Add")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`addRow_Server` *Server logic of module for adding a row*

Description

Server logic of module for adding a row

Usage

```
addRow_Server(
  input,
  output,
  session,
  data,
  patient_ids = NULL,
  dates_first_diagnosis = NULL,
  mode = c("timeline", "timepoint")
)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	source data as data.frame
`patient_ids`	reactive vector of existing patient IDs
`dates_first_diagnosis`	data.frame with dates of the first diagnosis and patient IDs
`mode`	Mode of the timeline data. Controls which columns are displayed.

Value

reactive data.frame of modified source data

`addRow_UI` *UI elements of module for adding a row*

Description

UI elements of module for adding a row

Usage

```
addRow_UI(id, label = "Add")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`cBioPortalToDataFrame` *Convert the cBioPortal sample- and patient-data file format into a data.frame*

Description

This function takes a file object (from read.table), removes the # symbol, sets the 5th row as the column names of the data.frame and removes the rows containing the priority, data type and column name. use read.table as follows: `read.table(file, sep='\t', colClasses = 'character', comment.char = '')`

Usage

```
cBioPortalToDataFrame(data)
```

Arguments

`data`	The data.frame of a cBioPortal sample/patient data file

Value

data.frame

Examples

```
df <- data.frame(
        V1=c("#attr_1", "#attribute 1", "#STRING", "#1", "ATTRIBUTE_1", "value_1"),
        V2=c("attr_2", "attribute 2", "STRING", "1", "ATTRIBUTE_2", "value_2")
    )
cbpManager:::cBioPortalToDataFrame(df)
```

`cbpManager` *Launch cbpManager*

Description

Launch the cbpManager Shiny application.

Usage

```
cbpManager(studyDir = NULL, logDir = NULL, returnAppObj = FALSE, ...)
```

Arguments

`studyDir`	Path to study folder containing studies of cBioPortal.
`logDir`	Path where a logfile should be saved. If NULL, logs are not stored
`returnAppObj`	Logical value, whether to return the app object if set to TRUE. Default behavior: directly runs the app (FALSE)
`...`	Further parameters that are used by `shiny::runApp`, e.g. `host` or `port`.

Value

shiny application object

Examples

```
if (interactive()) {
  cbpManager()
}
```

`cbpManager-pkg` *cbpManager*

Description

‘cbpManager‘ is an R package that provides an interactive Shiny-based graphical user interface for...

Author(s)

Arsenij Ustjanzew <`arsenij.ustjanzew@gmail.com`>

`check_input_dates` *Check the input of dates*

Description

Check the input of dates

Usage

```
check_input_dates(diagnosisDate, startDate = NULL, endDate = NULL)
```

Arguments

`diagnosisDate`	date of first diagnosis
`startDate`	start date of timeline event
`endDate`	end date of timeline event

Value

Returns a number indicating the warning

Examples

```
cbpManager:::check_input_dates(
    diagnosisDate = "2020-01-01",
    startDate = "2020-02-01",
    endDate = "2020-03-01"
) #returns 0
cbpManager:::check_input_dates(
    diagnosisDate = "2020-01-01",
    startDate = "2019-02-01"
) #returns 2
cbpManager:::check_input_dates(
    diagnosisDate = "2020-01-01",
    endDate = "2019-02-01"
) #returns 2
cbpManager:::check_input_dates(
    diagnosisDate = "2020-01-01",
    startDate = "2020-03-01",
    endDate = "2020-02-01"
) #returns 1
```

`convertDataFrame` *Convert the data.frame to the appropriate file format for cBioPortal*

Description

Convert the data.frame to the appropriate file format for cBioPortal

Usage

```
convertDataFrame(df)
```

Arguments

`df`	data.frame

Value

Data.frame formated for the cBioPortal file format

Examples

```
cbpManager:::convertDataFrame(
    data.frame(
        ATTRIBUTE1=c("attr_1", "attribute 1", "STRING", "value_a1"),
        ATTRIBUTE2=c("attr_2", "attribute 2", "STRING", "value_b1")
    )
)
```

`create_name` *Sanitize names*

Description

This function takes a character string, replaces spaces by underscores and runs make.names.

Usage

```
create_name(x, toupper = TRUE)
```

Arguments

`x`	A character string.
`toupper`	If TRUE, the name wil be upper-case; if FALSE, the name will be lower-case.

Value

A sanitized string.

Examples

```
cbpManager:::create_name("Study name 1") #returns "STUDY_NAME_1"
cbpManager:::create_name("FANCY;name", toupper = FALSE) #returns "fancy.name"
```

`deleteColumn_Server` *Server logic of module for deleting a column*

Description

Server logic of module for deleting a column

Usage

```
deleteColumn_Server(input, output, session, data, exclude)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	source data as data.frame
`exclude`	column names that shoud be excluded from deletion

Value

reactive data.frame of modified source data

`deleteColumn_UI` *UI elements of module for deleting a column*

Description

UI elements of module for deleting a column

Usage

```
deleteColumn_UI(id, label = "Delete column(s)")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`deleteRowRc_Server` *Server logic of Resource tab module for deleting a row*

Description

Server logic of Resource tab module for deleting a row

Usage

```
deleteRowRc_Server(
  input,
  output,
  session,
  data,
  selected_row,
  mode = "default",
  sample_data = NULL,
  patient_data = NULL,
  study_data = NULL
)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	Source data as data.frame
`selected_row`	Index of the selected row from the table
`mode`	If 'recursive' the resources bind to the resource definition will be deleted.
`sample_data`	Data of the resource from type 'sample'
`patient_data`	Data of the resource from type 'patient'
`study_data`	Data of the resource from type 'study'

Value

reactive data.frame of modified source data

`deleteRowRc_UI` *UI elements of module for removing a row*

Description

UI elements of module for removing a row

Usage

```
deleteRowRc_UI(id, label = "Delete")
```

Arguments

`id`	Module id
`label`	Label of the button

Value

UI module

`deleteRow_Server` *Server logic of module for removing a row*

Description

Server logic of module for removing a row

Usage

```
deleteRow_Server(input, output, session, data, selected_row)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	source data as data.frame
`selected_row`	Index of the selected row from the table

Value

reactive data.frame of modified source data

`deleteRow_UI` *UI elements of module for removing a row*

Description

UI elements of module for removing a row

Usage

```
deleteRow_UI(id, label = "Delete")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`editRowRc_Server` *Server logic of Resource tab module for editing a row*

Description

Server logic of Resource tab module for editing a row

Usage

```
editRowRc_Server(
  input,
  output,
  session,
  data,
  patient_ids = NULL,
  sample_ids = NULL,
  resource_ids = NULL,
  selected_row = NULL,
  resource_type = c("definition", "sample", "patient", "study")
)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	Source data as data.frame
`patient_ids`	Reactive vector of existing patient IDs
`sample_ids`	Reactive data.frame of existing patient IDs and sample IDs
`resource_ids`	Reactive data.frame of data_resource_definition
`selected_row`	Index of the selected row
`resource_type`	The type of the resource. Can be "definition", "sample", "patient", "study"

Value

reactive data.frame of modified source data

`editRowRc_UI` *UI elements of module for editing a row*

Description

UI elements of module for editing a row

Usage

```
editRowRc_UI(id, label = "Edit")
```

Arguments

`id`	Module id
`label`	Label of the button

Value

UI module

`editRow_Server` *Server logic of module for editing a row*

Description

Server logic of module for editing a row

Usage

```
editRow_Server(
  input,
  output,
  session,
  data,
  patient_ids = NULL,
  dates_first_diagnosis = NULL,
  selected_row = NULL,
  mode = c("timeline", "timepoint")
)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	source data as data.frame
`patient_ids`	reactive vector of existing patient IDs
`dates_first_diagnosis`	data.frame with dates of the first diagnosis and patient IDs
`selected_row`	the index of the selected row
`mode`	Mode of the timeline data. Controls which columns are displayed.

Value

reactive data.frame of modified source data

`editRow_UI` *UI elements of module for editing a row*

Description

UI elements of module for editing a row

Usage

```
editRow_UI(id, label = "Edit")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`fncols` *Add empty column to a data.frame, if column does not exist in the data.frame*

Description

Add empty column to a data.frame, if column does not exist in the data.frame

Usage

```
fncols(data, cname)
```

Arguments

`data`	data.frame
`cname`	column name

Value

data.frame

Examples

```
cbpManager:::fncols(data.frame(a=c(1,2,3), b=c(4,5,6)), "new")
```

`generateOncotreeUIwidgets` *Create shiny UI-widget for specific columns of oncotree entries*

Description

Create shiny UI-widget for specific columns of oncotree entries

Usage

```
generateOncotreeUIwidgets(
  colname,
  mode = c("add", "edit"),
  tab = c("Patient", "Sample")
)
```

Arguments

`colname`	column name
`mode`	determines the inputId prefix of the UI-widget
`tab`	"Patient", "Sample" - The used tab; sets the html id prefix of the input

Value

A oncotree specific shiny UI-widget

Examples

```
oncotree <- jsonlite::fromJSON(system.file("extdata", "oncotree.json", package = "cbpManager"))
cancer_type <- unique(oncotree$mainType[which(!is.na(oncotree$mainType))])
cbpManager:::generateOncotreeUIwidgets("CANCER_TYPE", "add")
```

`generateUIwidgets` *Generate UI input widget*

Description

Generate UI input widget

Usage

```
generateUIwidgets(
  colname,
  mode = c("add", "edit"),
  tab = c("Patient", "Sample", "Mutation"),
  data = NULL,
  selected_row = NULL,
  patientIDs = NULL,
  sampleIDs = NULL
)
```

Arguments

`colname`	A character string - the name of the column, that will be the label of the input
`mode`	"add" or "edit" - whether to use existing values or not
`tab`	"Patient", "Sample" - The used tab; sets the html id prefix of the input
`data`	A data.frame.
`selected_row`	A number indicating the row number of the selected row in the data.frame.
`patientIDs`	Vector of patient IDs used for drop down menu of the PATIENT_ID column

Value

A sanitized string.

Examples

```
cbpManager:::generateUIwidgets(colname = "attribute", mode = "add", tab = "Patient")
```

`getSampleIDs` *Get Sample IDs associated with Patient IDs from the data_clinical_sample.txt file*

Description

Get Sample IDs associated with Patient IDs from the data_clinical_sample.txt file

Usage

```
getSampleIDs(file_path, patIDs)
```

Arguments

`file_path`	A character string.
`patIDs`	A character string.

Value

vector with Sample IDs

Examples

```
cbpManager:::getSampleIDs(
    system.file("study/testpatient/data_clinical_sample.txt", package = "cbpManager"),
    patIDs = "Testpatient")
```

`importPatientData` *Import patient data into current study data.frames*

Description

Import patient data into current study data.frames

Usage

```
importPatientData(
  mode = c("patient", "sample", "mutations", "timelines"),
  file_name,
  file_path,
  patIDs,
  data,
  associatedSampleIDs = NULL
)
```

Arguments

`mode`	Defines the type of imported data
`file_name`	Filename of source data
`file_path`	Filepath with filename of source data
`patIDs`	PATIENT_IDs of patients that should be imported
`data`	Source data, to be subsetted according to patIDs
`associatedSampleIDs`	The sample IDs associated to the patIDs

Value

data.frame

`IsDate` *Check if input is in the appropriate date format*

Description

Check if input is in the appropriate date format

Usage

```
IsDate(mydate, date.format = "%Y-%m-%d")
```

Arguments

`mydate`	date
`date.format`	string describig the date format

Value

boolean

Examples

```
cbpManager:::IsDate("2020-02-20")
cbpManager:::IsDate("20.01.2020", date.format = "%d.%m.%Y")
```

`saveResource_Server` *Server logic of module for saving the resource data*

Description

Server logic of module for saving the resource data

Usage

```
saveResource_Server(
  input,
  output,
  session,
  data,
  study_id,
  data_filename,
  meta_filename,
  resource_type = c("SAMPLE", "DEFINITION", "PATIENT", "STUDY")
)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	Source data as data.frame
`study_id`	The current study ID
`data_filename`	File name of the data file
`meta_filename`	file name of the meta file
`resource_type`	The type of the resource. Can be "definition", "sample", "patient", "study"

Value

boolean value; TRUE if function was used.

`saveResource_UI` *UI elements of module for saving the resource data*

Description

UI elements of module for saving the resource data

Usage

```
saveResource_UI(id, label = "Save")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`saveTimeline_Server` *Server logic of module for saving the source data*

Description

Server logic of module for saving the source data

Usage

```
saveTimeline_Server(input, output, session, data, study_id)
```

Arguments

`input`	Shiny input
`output`	Shiny output
`session`	Shiny session
`data`	source data as data.frame
`study_id`	the current study ID

Value

boolean value; TRUE if function was used.

`saveTimeline_UI` *UI elements of module for saving the data*

Description

UI elements of module for saving the data

Usage

```
saveTimeline_UI(id, label = "Save")
```

Arguments

`id`	module id
`label`	label of the button

Value

UI module

`setupConda_cbpManager` *Install conda environment with basilisk before launching the app*

Description

Install conda environment with basilisk before launching the app

Usage

```
setupConda_cbpManager()
```

Value

Nothing to return

Examples

```
## Not run:
setupConda_cbpManager()

## End(Not run)
```

`shinyAppServer` *Shiny app server function*

Description

Shiny app server function

Usage

```
shinyAppServer(input, output, session)
```

Arguments

`input`	provided by shiny
`output`	provided by shiny
`session`	provided by shiny

Value

nothing to return

`shinyAppUI` *Shiny app server object create the shiny application user interface*

Description

Shiny app server object create the shiny application user interface

Usage

```
shinyAppUI
```

Format

An object of class `shiny.tag` of length 3.

`updateOncotreeUIwidgets` *Updates UI-widgets for specific columns of oncotree entries*

Description

Updates UI-widgets for specific columns of oncotree entries

Usage

```
updateOncotreeUIwidgets(
  session,
  row_last_clicked,
  mode = c("add", "edit"),
  tab = c("Patient", "Sample")
)
```

Arguments

`session`	Shiny session
`row_last_clicked`	the index of the row last clicked in the oncotree_table
`mode`	determines the inputId prefix of the UI-widget
`tab`	"Patient", "Sample" - The used tab; sets the html id pr

Value

nothing to return

`validateResourceDefinition` *Validate resource_definition input*

Description

Validate resource_definition input

Usage

```
validateResourceDefinition(values, resourceDf, mode = "add")
```

Arguments

`values`	List of input values
`resourceDf`	data.frame of data_resource_definition
`mode`	The mode of the function ('add' or 'edit')

Value

boolean

`validateResourcePatient` *Validate resource_patient input*

Description

Validate resource_patient input

Usage

```
validateResourcePatient(values)
```

Arguments

`values`	List of input values

Value

boolean

`validateResourceSample` *Validate resource_sample input*

Description

Validate resource_sample input

Usage

```
validateResourceSample(values)
```

Arguments

`values`	List of input values

Value

boolean

`validateResourceStudy` *Validate resource_study input*

Description

Validate resource_study input

Usage

```
validateResourceStudy(values)
```

Arguments

`values`	List of input values

Value

boolean

`ValidationDependencies` *Validation Dependencies*

Description

Vector defining a set of Python dependencies and versions required to operate with the validation scripts for cBioPortal

Usage

```
.validation_dependencies
```

Format

A character vector containing the pinned versions of all Python packages on which the import validation depends.

`writeLogfile` *Write a line to a logfile containing the date, time, username (from Shinyproxy), and the name of the modified file.*

Description

Write a line to a logfile containing the date, time, username (from Shinyproxy), and the name of the modified file.

Usage

```
writeLogfile(outdir, modified_file, log_filename = "cbpManager_logfile.txt")
```

Arguments

`outdir`	directory, where the logfile should be saved
`modified_file`	Name of the modified file
`log_filename`	Name of the logfile

Value

Nothing to return

Examples

```
cbpManager:::writeLogfile(tempdir(), "data_clinical_patient.txt")
```

Index