# Package ‘dccvalidator’

March 13, 2020

**Title**  Metadata Validation for Data Coordinating Centers

**Version** 0.2.0

**Description**  Performs checks for common metadata quality issues. Used by the data coordinating centers for the 'AMP-AD' consortium (<https://adknowledgeportal.synapse.org>), 'PsychENCODE' consortium (<http://www.psychencode.org>), and others to validate metadata prior to data releases.

**License**  MIT + file LICENSE

**URL**  https://sage-bionetworks.github.io/dccvalidator,
          https://github.com/Sage-Bionetworks/dccvalidator

**Depends**  R (>= 3.4), shinyBS

**Imports**  config, ggplot2, glue, golem, htmltools, knitr, markdown, purrr, reactive, readr, readxl, reticulate, rlang, shiny, shinydashboard, shinyjs, skimr, stats, tibble, tools, utils, visdat

**Suggests**  covr, jsonlite, jsonvalidate, rmarkdown (>= 1.16.2), stringr, testthat, withr

**VignetteBuilder**  knitr

**ByteCompile**  true

**Encoding**  UTF-8

**LazyData**  true

**RoxygenNote**  7.0.2

**SystemRequirements**  synapseclient (pypi.org/project/synapceclient)

**NeedsCompilation**  no

**Author**  Kara Woo [aut],
            Sage Bionetworks [cph],
            Nicole Kauer [aut, cre],
            Kelsey Montgomery [aut],
            Dean Attali [cph]

**Maintainer**  Nicole Kauer <nicole.kauer@sagebase.org>

**Repository**  CRAN

**Date/Publication**  2020-03-13 00:20:02 UTC
app_server

**Description**

Create the server-side component of the dccvalidator Shiny app.

**Usage**

```r
app_server(input, output, session)
```
app_ui

Arguments

input    Shiny input
output   Shiny output
session  Shiny session

Value

none

Examples

## Not run:
shinyApp(ui = app_ui, server = app_server)

## End(Not run)

app_ui

App UI

Description

Create the UI component of the dccvalidator Shiny app.

Usage

app_ui(request)

Arguments

request    Shiny request

Value

A shinydashboard page

Examples

## Not run:
shinyApp(ui = app_ui, server = app_server)

## End(Not run)
can_coerce  

Description

Checks if values are coercible to a given class. Because of inconsistencies in R’s built-in coercion functions (e.g. `as.numeric()` warns when it introduces NAs but `as.logical()` doesn’t; `as.integer()` will silently remove decimal places from numeric inputs) we check only for the specific coercions we want to allow, primarily allowing numeric, integer, or logical values to be considered valid even when the required type is character.

Usage

can_coerce(values, class)

Arguments

values  

Vector of values to check

class  

Class of interest

Details

This function is mainly in place so that we can automatically allow numeric read lengths, pH values, etc., which are defined as strings in our annotation vocabulary but can reasonably be numbers.

Additionally, this function will return TRUE if the values are integers and the desired class is numeric, and will return TRUE if the values are numeric but are whole numbers. 2.0 is considered coercible to integer, but 2.1 is not.

It will also allow the following capitalizations of boolean values: true, True, TRUE, false, False, FALSE. These are all treated as valid booleans by Synapse.

This function will not affect validation of enumerated values, regardless of their class. It is only used when validating annotations that have a required type but no enumerated values.

Value

Boolean value; TRUE if values are coercible to class, FALSE otherwise.

See Also

check_annotation_values()

Examples

# Not run because function is not exported
## Not run:
# Coercible:
can_coerce(1, "character")
can_coerce(TRUE, "character")
check_all

```r
can_coerce(1L, "character")
can_coerce(1L, "numeric")
can_coerce(1.0, "integer")

# Not coercible:
can_coerce("foo", "numeric")
can_coerce("foo", "logical")
can_coerce(2.5, "integer")
```

## End(Not run)

---

**Description**

Runs all validation checks. Requires an environment configuration (config) to be set. The config is expected to have templates for each metadataType, where individual and biospecimen depend on species and assay depends on the assay type. Additionally, there should be complete_columns for each metadataType.

**Usage**

```r
check_all(data, annotations, syn)
```

**Arguments**

- `data` A tibble or dataframe with the columns: name, metadataType, species, assay, file_data. The file_data column should be a list column containing a dataframe with the file data or NULL if the data does not exist. data is expected to have four rows, one for each metadataType: individual, biospecimen, assay, manifest. If file_data is NULL for a given metadataType, the metadataType should still be present.
- `annotations` A data frame of annotation definitions. Must contain at least three columns: key, value, and columnType.
- `syn` Synapse client object

**Value**

List of conditions

**Examples**

```r
## Not run:
syn <- synapse$Synapse()
synLOGIN()

annots <- get_synapse_annotations(syn = syn)
```
check_annotation_keys

Check annotation keys

Description

Checks that all annotation keys on a file, in a file view, or in a data frame are valid annotations.

check_annotation_keys() returns any invalid annotation keys; valid_annotation_keys() returns valid annotation keys.

Usage

check_annotation_keys(x, annotations, ...)

## S3 method for class 'NULL'
check_annotation_keys(x, annotations, ...)

## S3 method for class 'synapseclient.entity.File'
check_annotation_keys(x, annotations, syn, ...)

## S3 method for class 'data.frame'
check_annotation_keys(x, annotations, ...)

## S3 method for class 'synapseclient.table.CsvFileTable'
check_annotation_keys(x, annotations, ...)
check_annotation_keys

Arguments

- **x**: An object to check.
- **annotations**: A data frame of annotation definitions. Must contain at least three columns: `key`, `value`, and `columnType`.
- **...**: Additional parameters passed to `check_keys()`.
- **syn**: Synapse client object

Value

A condition object indicating whether keys match the given annotation dictionary. Erroneous keys are included as data within the object.

Methods (by class)

- **NULL**: Return NULL
- **synapseclient.entity.File**: Check annotation keys in a Synapse file
- **data.frame**: Check annotation keys in a data frame
- **synapseclient.table.CsvFileTable**: Check annotation keys for a Synapse table

See Also

- `valid_annotation_keys()`

Examples

```r
annots <- data.frame(
  key = c("assay", "fileFormat", "fileFormat", "fileFormat", "species"),
  value = c("rnaSeq", "fastq", "txt", "csv", "Human"),
  columnType = c("STRING", "STRING", "STRING", "STRING", "STRING")
)

dat1 <- data.frame(x = 1)
dat2 <- data.frame(assay = "rnaSeq")
check_annotation_keys(dat1, annots)
check_annotation_keys(dat2, annots)
```

## Not run:

```r
syn <- synapse$Synapse()
syn$login()
annots <- get_synapse_annotations(syn = syn)
my_file <- syn$get("syn11931757", downloadFile = FALSE)
check_annotation_keys(my_file, annots, syn)

dat <- data.frame(non_annotation = 5, assay = "rnaSeq")
check_annotation_keys(dat, annots)
```

```r
fv <- syn$tableQuery("SELECT * FROM syn17020234")
check_annotation_keys(fv, annots)
```

# If you don't specify an annotations data frame, these functions will
# download annotations automatically using `get_synapse_annotations()` (must
# be logged in to Synapse)
my_file <- syn$get("syn11931757", downloadFile = FALSE)
check_annotation_keys(my_file, syn = syn)

## End(Not run)

---

check_annotation_values

*Check annotation values*

**Description**

Checks that all annotation values are valid. It does not report on values for invalid keys; see `check_annotation_keys()`.

**Usage**

```r
check_annotation_values(x, annotations, ...)  
## S3 method for class 'NULL'
check_annotation_values(x, annotations, ...)  
## S3 method for class 'synapseclient.entity.File'
check_annotation_values(x, annotations, syn, ...)  
## S3 method for class 'data.frame'
check_annotation_values(x, annotations, ...)  
## S3 method for class 'synapseclient.table.CsvFileTable'
check_annotation_values(x, annotations, ...)  
```

**Arguments**

- `x` An object to check.
- `annotations` A data frame of annotation definitions. Must contain at least three columns: key, value, and columnType.
- `...` Additional options to `check_values()`
- `syn` Synapse client object

**Details**

If the allowable annotation values are an enumerated list, `check_annotation_values()` compares the values in the data to the values in this list. If there is no enumerated list of values and the annotation definition merely specifies a required type, then the values are checked against that type, with values that are coercible to the correct type treated as valid (see `can_coerce()`).
Value

A condition object indicating whether all annotation values are valid. Invalid annotation values are included as data within the object.

Methods (by class)

- **NULL**: Return NULL
- **synapseclient.entity.File**: Check annotation values on a Synapse file
- **data.frame**: Check annotation values in a data frame
- **synapseclient.table.CsvFileTable**: Check annotation values in a Synapse table

See Also

valid_annotation_values(), can_coerce()

Examples

```r
annots <- data.frame(
  key = c("assay", "fileFormat", "fileFormat", "fileFormat", "species"),
  value = c("rnaSeq", "fastq", "txt", "csv", "Human"),
  columnType = c("STRING", "STRING", "STRING", "STRING", "STRING")
)

dat1 <- data.frame(assay = "not a valid assay")
dat2 <- data.frame(assay = "rnaSeq")
check_annotation_values(dat1, annots)
check_annotation_values(dat2, annots)

## Not run:
syn <- synapse$Synapse()
syn$login()

annots <- get_synapse_annotations(syn = syn)
my_file <- syn$get("syn11931757", downloadFile = FALSE)
check_annotation_values(my_file, annots)

dat <- data.frame(
  non_annotation = 5:7,
  assay = c("rnaSeq", "foo", "bar"),
  stringsAsFactors = FALSE
)
check_annotation_values(dat, annots)

fv <- synTableQuery("SELECT * FROM syn17020234")
check_annotation_values(fv, annots)

# If you don’t specify an annotations data frame, these functions will
# download annotations automatically using ‘get_synapse_annotations()’ (must
# be logged in to Synapse)
my_file <- syn$get("syn11931757", downloadFile = FALSE)
check_annotation_values(my_file, syn = syn)
```
# It is possible to whitelist certain certain values, or all values for
# certain keys:
check_annotation_values(dat, whitelist_keys = "assay", syn = syn)

ccheck_annotation_values(
  dat,
  whitelist_values = list(assay = c("foo")),
  syn = syn
)

## End(Not run)

---

### check_certified_user

**Check if user is certified**

---

**Description**

Check if user has completed and passed the Certified User Quiz.

**Usage**

```r
check_certified_user(id, syn)
```

**Arguments**

- `id` User ID
- `syn` Synapse client object

**Value**

A condition object indicating whether or not the given user is a certified Synapse user.

**Examples**

```r
## Not run:
syn <- synapse$Synapse()
syn$login()
check_certified_user("3384770")

## End(Not run)
```
check_cols_complete

Description

Check for complete columns in the data and fail (or warn) if incomplete. Missing columns that are required to be complete are ignored.

Usage

check_cols_complete(
  data,
  required_cols,
  empty_values = c(NA, ""),
  strict = TRUE,
  success_msg = "Required columns present are complete",
  fail_msg = "Some required columns are not complete"
)

Arguments

data     Data to check
required_cols A character vector of the required columns to check for completeness.
empty_values Values that are considered empty. Defaults to NA and "".
strict   If FALSE, return a "check_warn" object; if TRUE, return a "check_fail" object
success_msg Message indicating the check succeeded.
fail_msg   Message indicating the check failed.

Value

A condition object indicating whether the data contains columns that are not complete.

Examples

dat <- data.frame(specimenID = c("x", "y"), organ = c(NA, NA))
check_cols_complete(dat, c("specimenID", "organ"))
check_cols_empty

Check for empty columns

Description

Check for empty columns in the data and warn (or fail) if present. The function takes in a list of required column names that are not tested for emptiness. This is due to the existing function check_cols_complete(), which ensures that the required columns are complete. By ignoring the required columns in check_cols_empty(), there are no duplicated results for the same column in the event that a required column was also empty.

Usage

```r
check_cols_empty(
  data,
  empty_values = c(NA, ""),
  required_cols = NULL,
  strict = FALSE,
  success_msg = "No columns are empty",
  fail_msg = "Some columns are empty"
)
```

Arguments

- **data**: Data to check
- **empty_values**: Values that are considered empty. Defaults to NA and "".
- **required_cols**: A character vector of the required columns to check for completeness.
- **strict**: If FALSE, return a "check_warn" object; if TRUE, return a "check_fail" object
- **success_msg**: Message indicating the check succeeded.
- **fail_msg**: Message indicating the check failed.

Value

A condition object indicating whether the data contains columns that are empty.

Examples

```r
dat <- data.frame(specimenID = c("x", "y"), organ = c(NA, NA))
check_cols_empty(dat)
```
check_col_names

Check column names against their corresponding template

Description
Check column names against their corresponding template

Usage

check_col_names(
  data,
  template,
  success_msg = NULL,
  fail_msg = NULL,
  behavior = NULL
)

check_cols_manifest(
  data,
  id,
  success_msg = "All manifest columns present",
  fail_msg = "Missing columns in the manifest",
  ...
)

check_cols_individual(
  data,
  id,
  success_msg = "All individual metadata columns present",
  fail_msg = "Missing columns in the individual metadata file",
  ...
)

check_cols_assay(
  data,
  id,
  success_msg = "All assay metadata columns present",
  fail_msg = "Missing columns in the assay metadata file",
  ...
)

check_cols_biospecimen(
  data,
  id,
  success_msg = "All biospecimen columns present",
  fail_msg = "Missing columns in the biospecimen metadata file",
  ...
)
Arguments

- **data**: Data frame to check against template (manifest, individual metadata, or assay metadata)
- **template**: Character vector of column names from the template to check against
- **success_msg**: Message indicating the check succeeded.
- **fail_msg**: Message indicating the check failed.
- **behavior**: The intended behavior of the test
- **id**: Synapse ID of the template to check against
- **...**: Additional arguments passed to `syn$get()`

Value

A condition object indicating whether the required columns were present ("check_pass") or absent ("check_fail").

See Also

dccvalidator::get_template()

Examples

template <- c("individualID", "specimenID", "assay")
dat <- data.frame(individualID = c("a", "b"), specimenID = c("a1", "b1"))
check_col_names(dat, template)
## Not run:
syn <- synapse$Synapse()
syn$login()
a <- data.frame(path = "/path/file.txt", parent = "syn123", assay = "rnaSeq")
check_cols_manifest(a, syn)
b <- data.frame(assay = "rnaSeq")
check_cols_manifest(b, syn)
## End(Not run)
**Usage**

```r
check_condition(msg, behavior, data, type)
```

**Arguments**

- `msg` Message to report
- `behavior` Statement of the correct behavior (i.e. what the higher level function was checking for)
- `data` Data to return (e.g. invalid values that need attention)
- `type` One of "check_pass", "check_warn", "check_fail"

**Value**

An S3 object of class "check_pass", "check_warn", or "check_fail"

**Examples**

```r
strict <- TRUE
check_condition(
  msg = "Some data is missing",
  behavior = "Files should be complete",
  data = c("specimenID", "assay"),
  type = ifelse(strict, "check_fail", "check_warn")
)
```

---

**check_files_manifest**  
**Check that files are present in manifest**

**Description**

Given a manifest and vector of file names, checks that the file names appear in the manifest. This is useful to ensure that metadata files (not just data files) are included in the manifest for upload.

**Usage**

```r
check_files_manifest(
  manifest,
  filenames,
  strict = FALSE,
  success_msg = "All required files are present in manifest",
  fail_msg = "Some files may be missing from manifest"
)
```
Arguments

manifest  The manifest as a data frame or tibble
filenames  File names to look for in the path column of the manifest
strict  If FALSE, return a "check_warn" object; if TRUE, return a "check_fail" object
success_msg  Message indicating the check succeeded.
fail_msg  Message indicating the check failed.

Value

A condition object indicating whether the files are present in the path column of the manifest

Examples

manifest <- data.frame(
  path = c("individual_metadata.csv", "biospecimen_metadata.csv"),
  parent = c("syn123", "syn123")
)
check_files_manifest(
  manifest,
  c(
    "individual_metadata.csv",
    "biospecimen_metadata.csv",
    "assay_metadata.csv"
  )
)

check_ids_match  Check ids

Description

Compare IDs (such as individual IDs or specimen IDs) between two data frames.
Ensure that all individual IDs in two data frames match.
Ensure that all specimen IDS in two data frames match

Usage

check_ids_match(
  x,
  y,
  idcol = c("individualID", "specimenID"),
  xname = NULL,
  yname = NULL,
  bidirectional = TRUE
)
check_indiv_ids_dup

check_indiv_ids_match(x, y, xname = NULL, yname = NULL, bidirectional = TRUE)

check_specimen_ids_match(
  x,
  y,
  xname = NULL,
  yname = NULL,
  bidirectional = TRUE
)

Arguments

x, y       Data frames to compare
idcol     Name of column containing ids to compare
xname, yname Names of x and y (to be used in resulting messages)
bidirectional Should mismatches from both x and y be reported? Defaults to TRUE; if FALSE, will return only IDs in y that are not present in x (IDs in x but not y will be ignored).

Value

A condition object indicating whether IDs match ("check_pass") or not ("check_fail"). Mismatched IDs are included as data within the object.

Examples

a <- data.frame(individualID = LETTERS[1:3])
b <- data.frame(individualID = LETTERS[1:4])
check_ids_match(a, b, idcol = "individualID", xname = "a", yname = "b")
a <- data.frame(individualID = LETTERS[1:3])
b <- data.frame(individualID = LETTERS[1:4])
check_specimen_ids_match(a, b, "individual", "biospecimen")
a <- data.frame(specimenID = LETTERS[1:3])
b <- data.frame(specimenID = LETTERS[1:4])
check_specimen_ids_match(a, b, "biospecimen", "assay")

check_indiv_ids_dup  Check uniqueness of individual and specimen IDs

Description

Check uniqueness of individual and specimen IDs
check_keys

Usage

check_indiv_ids_dup(
  data,
  empty_values = c(NA, ""),
  success_msg = "Individual IDs are unique",
  fail_msg = "Duplicate individual IDs found"
)

check_specimen_ids_dup(
  data,
  empty_values = c(NA, ""),
  success_msg = "Specimen IDs are unique",
  fail_msg = "Duplicate specimen IDs found"
)

Arguments

data Individual metadata file
empty_values Values that are considered empty. Defaults to NA and "".
success_msg Message indicating the check succeeded.
fail_msg Message indicating the check failed.

Value

A condition object indicating whether the individual IDs in the individual metadata file are unique.

Examples

dat1 <- data.frame(individualID = c("x", "y", "z", "z"))
check_indiv_ids_dup(dat1)

dat2 <- data.frame(
  individualID = c("x", "y", "z"),
  specimenID = c("a", "a", "b")
)
check_specimen_ids_dup(dat2)

check_keys Check that a given set of keys are all present in an annotations dictionary

Description

Check that a given set of keys are all present in an annotations dictionary.
check_keys

Usage

check_keys(
  x,
  annotations,
  whitelist_keys = NULL,
  success_msg = "All annotation keys are valid",
  fail_msg = "Some annotation keys are invalid",
  annots_link = "https://shinypro.synapse.org/users/nsanati/annotationUI/",
  return_valid = FALSE,
  syn
)

Arguments

x                  A data frame of annotation data
annotations        A data frame of annotations to check against
whitelist_keys     A character vector of annotation keys to whitelist. If provided, all values for the
                    given keys will be treated as valid.
success_msg        Message indicating the check succeeded.
fail_msg           Message indicating the check failed.
annots_link        Link to a definition of the annotations being used in the project
return_valid       Should the function return valid values? Defaults to FALSE (i.e. the function will return invalid values).
syn                Synapse client object

Value

A condition object indicating whether keys match the given annotation dictionary. Erroneous keys are included as data within the object.

Examples

annots <- data.frame(
  key = c("fileFormat", "fileFormat"),
  value = c("txt", "csv"),
  columnType = c("STRING", "STRING"),
  stringsAsFactors = FALSE
)
check_keys("fileFormat", annots)
check_keys("x", annots)
check_pass

Create custom conditions for reporting

Description

These functions create custom condition objects with subclasses "check_pass", "check_warn", and "check_fail" (inheriting from "message", "warning", or "error", respectively). Validation functions such as `dccvalidator::check_col_names()` use these to report results and provide additional data on the source of errors or invalid data if needed.

Usage

```r
check_pass(msg, behavior, data = NULL)
check_warn(msg, behavior, data = NULL)
check_fail(msg, behavior, data = NULL)
```

Arguments

- **msg** Message to report
- **behavior** Statement of the correct behavior (i.e. what the higher level function was checking for)
- **data** Data to return (e.g. invalid values that need attention)

Value

An S3 object of class "check_pass", "check_warn", or "check_fail"

Examples

```r
check_pass(msg = "Success!", behavior = "Files should be complete")
check_warn(
  msg = "Warning, some data is missing",
  behavior = "Files should be complete",
  data = c("specimenID", "assay") # columns with missing data
)
check_fail(
  msg = "Error, some required data is missing",
  behavior = "Files should be complete",
  data = c("specimenID", "assay") # columns with missing data
)
```
check_schema_df  

Check a data frame of data against a JSON Schema

Description

Each row of the data frame will be converted to JSON and validated against the given schema.

Usage

check_schema_df(
  df,  
  schema,  
  success_msg = "Data is valid against the schema",  
  fail_msg = "Data is invalid against the schema"
)

Arguments

df  
A data frame whose rows will be converted into JSON and validated

schema  
Contents of the json schema, or a filename containing a schema.

success_msg  
Message indicating the check succeeded.

fail_msg  
Message indicating the check failed.

Value

A condition object indicating whether the data is valid against the schema.

Examples

if (requireNamespace("jsonvalidate", quietly = TRUE) &  
  requireNamespace("jsonlite", quietly = TRUE)) {
  dat <- data.frame(  
    x = c(NA, 1, NA),  
    y = c(NA, NA, "foo")  
  )  
  schema <- '{  
    "$schema": "http://json-schema.org/draft-04/schema#",  
    "properties": {  
      "x": {  
        "type": "integer"  
      },  
      "y": {  
        "type": "integer"  
      }  
    },  
    "required": ["x", "y"]  
  }  
  check_schema_df(  
    df,  
    schema,  
    success_msg = "Data is valid against the schema",  
    fail_msg = "Data is invalid against the schema"
  )
}
check_schema_json

check_schema_json

Check data against a JSON Schema

Description

Check a JSON blob against a JSON Schema.

Usage

check_schema_json(
  json,
  schema,
  success_msg = "Data is valid against the schema",
  fail_msg = "Data is invalid against the schema"
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>json</td>
<td>Contents of a json object, or a filename containing one.</td>
</tr>
<tr>
<td>schema</td>
<td>Contents of the json schema, or a filename containing a schema.</td>
</tr>
<tr>
<td>success_msg</td>
<td>Message indicating the check succeeded.</td>
</tr>
<tr>
<td>fail_msg</td>
<td>Message indicating the check failed.</td>
</tr>
</tbody>
</table>

Value

A condition object indicating whether the data is valid against the schema.

Examples

```r
if (requireNamespace("jsonvalidator", quietly = TRUE)) {
  schema <- '{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "properties": {
      "x": {
        "type": "integer"
      }
    },
    "required": ["x"]
  }
  json_valid <- '{
    "x": 3
  }
  json_invalid <- '{
    "x": 1.5
  }
```
check_team_membership

Description

Check if a user is a member of any of the given teams.

Usage

check_team_membership(teams, user, syn)

Arguments

- **teams**: Team IDs to check membership in
- **user**: User to check (e.g. output from syn$getUserProfile())
- **syn**: Synapse client object

Value

A condition object indicating whether the Synapse user is a member of the given team(s).

Examples

```r
## Not run:
syn <- synapse$Synapse()
syn$login()
user <- syn$getUserProfile("dcctravistest")
check_team_membership(teams = "3396691", user = user, syn = syn)
check_team_membership(
  teams = c("3397398", "3377637"),
  user = user,
  syn = syn
)
## End(Not run)
```
check_values  

Check a set of keys and their values

Description

Check a set of keys and their values

Usage

check_values(
  x,
  annotations,
  whitelist_keys = NULL,
  whitelist_values = NULL,
  success_msg = "All annotation values are valid",
  fail_msg = "Some annotation values are invalid",
  return_valid = FALSE,
  annots_link = "https://shinypro.synapse.org/users/nsanati/annotationUI/",
  syn
)

Arguments

x  A data frame of annotation data
annotations  A data frame of annotations to check against
whitelist_keys  A character vector of annotation keys to whitelist. If provided, all values for the
given keys will be treated as valid.
whitelist_values  A named list of keys (as the names) and values (as vectors) to whitelist
success_msg  Message indicating the check succeeded.
fail_msg  Message indicating the check failed.
return_valid  Should the function return valid values? Defaults to FALSE (i.e. the function will return invalid values).
annots_link  Link to a definition of the annotations being used in the project
syn  Synapse client object

Value

If return_valid = FALSE: a condition object indicating whether all annotation values are valid. Invalid annotation values are included as data within the object: a named list where each element corresponds to a key that contains invalid values, and the contents of each element is a vector of invalid values. If return_valid = TRUE: a named list of the valid annotation keys and values.
Examples

```r
annots <- data.frame(
  key = c("fileFormat", "fileFormat"),
  value = c("txt", "csv"),
  columnType = c("STRING", "STRING"),
  stringsAsFactors = FALSE
)
dat <- data.frame(
  fileFormat = c("wrong", "txt", "csv", "wrong again"),
  stringsAsFactors = FALSE
)
check_values(dat, annots)
```

---

df_to_json_list  
*Convert data frame to JSON*

**Description**

Given a data frame, converts each row to a JSON blob and returns the results in a list, to make it easier to iteratively validate data with `check_schema_df()`.

**Usage**

```r
df_to_json_list(df)
```

**Arguments**

- `df`  
  A data frame

**Value**

A list of JSON blobs

**See Also**

`check_schema`

**Examples**

```r
if (requireNamespace("jsonlite", quietly = TRUE)) {
  dat <- data.frame(
    x = c(NA, 1L)
  )
df_to_json_list(dat)
}
```
file_summary_ui  UI for the file summary module

Description

Creates the UI for the file summary module, complete with a drop-down selection box, and two tabs, one for a file overview and one for file details.

Gives functionality to the file summary UI, populating the drop-down menu with available files to choose from, and showing both an overview and detailed summary of a chosen file.

Usage

```r
file_summary_ui(id)
```

```r
file_summary_server(input, output, session, file_data)
```

Arguments

- **id** the id
- **input** the input variables from `shiny::callModule()`
- **output** the output variables from `shiny::callModule()`
- **session** the session from `shiny::callModule()`
- **file_data** a reactive, named list of file data in data frames or tibbles

Value

html UI for file summary

Examples

```r
library("shiny")
library("shinydashboard")

server <- function(input, output) {
  # Create some simple file data
  data <- reactive({
    list(
      "individual" = data.frame(
        individualID = c("a", "b", "c"),
        age = c(23, 24, 24),
        stringsAsFactors = FALSE
      ),
      "biospecimen" = data.frame(
        individualID = c("a", "b", "c"),
        specimenID = c("a1", "b1", "c1"),
        isReal = c(FALSE, FALSE, FALSE),
        stringsAsFactors = FALSE
      )
    )
  })
```

get_synapse_annotations

Get Synapse annotations

Description

Download current annotation values from Synapse and provide them as a data frame.

Usage

get_synapse_annotations(synID = "syn10242922", syn)

Arguments

- synID: The Synapse ID of a table to query from. Defaults to "syn10242922"
- syn: Synapse client object

Value

A data frame containing all annotation keys, descriptions, column types, maximum sizes, values, value descriptions, sources, and the name of the annotation’s parent module.
get_synapse_table

**get_synapse_table**

*Get Synapse table*

### Description

Get the contents of a Synapse table as a data frame

### Usage

```r
get_synapse_table(synID, syn)
```

### Arguments

- **synID**
  - The Synapse ID of a table to query from. Defaults to "syn10242922"
- **syn**
  - Synapse client object

### Value

Data frame of table contents

### Examples

```r
## Not run:
syn <- synapse$Synapse()
syn$login()
get_synapse_annotations(synID = "syn10242922", syn = syn)

## End(Not run)
```
get_template

Get a template

Description
Get a template

Usage
get_template(synID, syn, ...)

Arguments
- synID: Synapse ID of an excel or csv file containing a metadata template
- syn: Synapse client object
- ...: Additional arguments passed to syn$get()

Value
Character vector of template column names

Examples
## Not run:
syn <- synapse$Synapse()
syn$login()
get_template("syn12973252", syn = syn)
## End(Not run)

report_unsatisfied_requirements
Create a modal dialog if user is not in required team(s) or certified

Description
Takes the output from check_team_membership() and check_certified_user(). If the user is not in the required teams or certified, creates a modal dialog indicating which teams they need to belong to and how to request access.

Usage
report_unsatisfied_requirements(membership, certified, syn)
**Arguments**

- membership: Output from `check_team_membership()`
- certified: Output from `check_certified_user()`
- syn: Synapse client object

**Value**

If user is not certified or in the required teams, a modal dialog describing which requirements are not met.

**Examples**

```r
## Not run:
syn <- synapse$Synapse()
syn$login()
user <- syn$getUserProfile("dcctravistest")
membership <- check_team_membership(
  teams = "3396691",
  user = user,
  syn = syn
)
certified <- check_certified_user(user$ownerId, syn = syn)
report_unsatisfied_requirements(membership, certified, syn = syn)

## End(Not run)
```

---

**results_boxes_ui**

UI function for results boxes module

**Description**

This function outputs the html tags needed to create UI for the successes, warnings, and failures results boxes.

This gives functionality to the results boxes module UI, attaching titles and populating the validation results.

**Usage**

```r
results_boxes_ui(id)
results_boxes_server(input, output, session, results)
```
Arguments

- **id**
  The module id.

- **input**
  The input from `shiny::callModule()`.

- **output**
  The output from `shiny::callModule()`.

- **session**
  The session from `shiny::callModule()`.

- **results**
  List of the validation results. If `NULL`, box titles will be default strings (i.e. “Successess (0)”; otherwise, the boxes will be populated with the results.

Value

The html UI for the module.

Examples

```r
library("shiny")
library("shinydashboard")

server <- function(input, output) {
  # Create some sample results
  res <- list(
    check_pass(msg = "All good!", behavior = "Values should be >10"),
    check_fail(
      msg = "Some values are too small",
      behavior = "Values should be > 10",
      data = c(5.5, 1.3)
    )
  )
  # Show results in boxes
  callModule(results_boxes_server, "Validation Results", res)
}

ui <- function(request) {
  dashboardPage(
    header = dashboardHeader(),
    sidebar = dashboardSidebar(),
    body = dashboardBody(
      includeCSS(
        system.file("app/www/custom.css", package = "dccvalidator")
      ),
      results_boxes_ui("Validation Results")
    )
  )
}

## Not run:
shinyApp(ui, server)

## End(Not run)
**run_app**  
*Run the Shiny application*

**Description**  
Run the Shiny application

**Usage**  
run_app(...) 

**Arguments**  
...  
Additional golem options passed to `golem::with_golem_options()`

**Value**  
Shiny app with additional golem options passed via ...

**Examples**  
```r  
## Not run:  
library("dccvalidator")  
run_app()  
## End(Not run)  
```  

---

**valid_annotation_keys**  
*Valid annotation keys*

**Description**  
Checks for and returns the valid annotation keys in a data frame, Synapse file, or Synapse file view.

**Usage**  
valid_annotation_keys(x, annotations, ...) 

---

**valid_annotation_keys**  
*Valid annotation keys*

**Description**  
Checks for and returns the valid annotation keys in a data frame, Synapse file, or Synapse file view.

**Usage**  
valid_annotation_keys(x, annotations, ...)  

## S3 method for class 'NULL'  
valid_annotation_keys(x, annotations, ...)  

## S3 method for class 'synapseclient.entity.File'  
valid_annotation_keys(x, annotations, syn, ...)  

## S3 method for class 'data.frame'  
valid_annotation_keys(x, annotations, ...)
valid_annotation_values

## S3 method for class 'synapseclient.table.CsvFileTable'
valid_annotation_keys(x, annotations, ...)

**Arguments**

- `x` An object to check.
- `annotations` A data frame of annotation definitions. Must contain at least three columns: `key`, `value`, and `columnType`.
- `...` Additional parameters passed to `check_keys()`
- `syn` Synapse client object

**Value**

A vector of valid annotation keys present in `x`.

**Methods (by class)**

- **NULL**: Return NULL
- **synapseclient.entity.File**: Valid annotation keys on a Synapse file
- **data.frame**: Valid annotation keys in a data frame
- **synapseclient.table.CsvFileTable**: Valid annotation keys in a Synapse table

**Examples**

```r
annots <- data.frame(
  key = c("assay", "fileFormat", "fileFormat", "fileFormat", "species"),
  value = c("rnaSeq", "fastq", "txt", "csv", "Human"),
  columnType = c("STRING", "STRING", "STRING", "STRING", "STRING")
)
dat1 <- data.frame(x = 1)
dat2 <- data.frame(assay = "rnaSeq")
valid_annotation_keys(dat1, annots)
valid_annotation_keys(dat2, annots)
```

---

**valid_annotation_values**

*Valid annotation values*

**Description**

Checks for and returns the valid annotation values in a data frame, Synapse file, or Synapse file view.
valid_annotation_values

Usage

valid_annotation_values(x, annotations, ...)

## S3 method for class `NULL`
valid_annotation_values(x, annotations, ...)

## S3 method for class `synapseclient.entity.File`
valid_annotation_values(x, annotations, syn, ...)

## S3 method for class `data.frame`
valid_annotation_values(x, annotations, ...)

## S3 method for class `synapseclient.table.CsvFileTable`
valid_annotation_values(x, annotations, ...)

Arguments

- **x** An object to check.
- **annotations** A data frame of annotation definitions. Must contain at least three columns: key, value, and columnType.
- **...** Additional options to check_values()
- **syn** Synapse client object

Value

A named list of valid annotation values.

Methods (by class)

- **NULL**: Return NULL
- **synapseclient.entity.File**: Valid annotation values on a Synapse file
- **data.frame**: Valid annotation values in a data frame
- **synapseclient.table.CsvFileTable**: Valid annotation values in a Synapse table

Examples

```r
annots <- data.frame(
  key = c("assay", "fileFormat", "fileFormat", "fileFormat", "species"),
  value = c("rnaSeq", "fastq", "txt", "csv", "Human"),
  columnType = c("STRING", "STRING", "STRING", "STRING", "STRING")
)
dat1 <- data.frame(assay = "not a valid assay")
dat2 <- data.frame(assay = "rnaSeq")
valid_annotation_values(dat1, annots)
valid_annotation_values(dat2, annots)
```
with_busy_indicator_ui

Show busy indicator

Description

These functions add button feedback features including: disabling the button while processing requested function, showing a spinning wheel while processing requested function, displaying a green checkmark showing success upon completion, or displaying an error message if the function requested failed. They require the development version of shinyjs (>= 1.0.1.9006). With earlier versions, the buttons will succeed but visual indicator feedback will not appear.

Usage

```r
with_busy_indicator_ui(button)

with_busy_indicator_server(button_id, expr)
```

Arguments

- `button` A Shiny actionButton
- `button_id` id of shiny actionButton
- `expr` the code to run when the button is clicked

Details

Wrap the button in this function to attach visual features.

Redistributed with minor modifications under MIT License from: https://github.com/daattali/advanced-shiny/blob/de590d593a0871848a31afd82584637decc972/busy-indicator/helpers.R

Hint for making this work with modules by mmoisse in PR#11: https://github.com/daattali/advanced-shiny/pull/11

Copyright 2019 Dean Attali

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Value

Shiny UI and server logic

Author(s)

Dean Attali dean@attalitech.com

Examples

library("shiny")

server <- function(input, output) {
    observeEvent(input$action, {
    with_busy_indicator_server("action", {
        Sys.sleep(1)
      output$value <- renderPrint("Success!")
    })
    })
}

ui <- fluidPage(
    includeCSS(
      system.file("app/www/custom.css", package = "dccvalidator")
    ),
    with_busy_indicator_ui(actionButton("action", label = "Action"),
      fluidRow(column(2, textOutput("value")))
  )

## Not run:
shinyApp(ui, server)

## End(Not run)