Package ‘redcapAPI’

May 16, 2024

Type Package
Title Interface to 'REDCap'
Version 2.9.1
Maintainer Shawn Garbett <shawn.garbett@vumc.org>
Description Access data stored in 'REDCap' databases using the Application Programming Interface (API). 'REDCap' (Research Electronic Data CAPture; <https://projectredcap.org>, Harris, et al. (2009) <doi:10.1016/j.jbi.2008.08.010>, Harris, et al. (2019) <doi:10.1016/j.jbi.2019.103208>) is a web application for building and managing online surveys and databases developed at Vanderbilt University. The API allows users to access data and project meta data (such as the data dictionary) from the web programmatically. The redcapAPI package facilitates the process of accessing data with options to prepare an analysis-ready data set consistent with the definitions in a database's data dictionary.

License GPL-2
Depends R (>= 3.5.0)
Imports checkmate, chron, httr, labelVector, lubridate, keyring, getPass, yaml
LazyLoad yes
Suggests testthat (>= 3.0.0), Hmisc, rstudioapi, mockery
URL https://github.com/vubiostat/redcapAPI
BugReports https://github.com/vubiostat/redcapAPI/issues
Encoding UTF-8
RoxygenNote 7.3.1
NeedsCompilation no
Author Benjamin Nutter [ctb, aut], Shawn Garbett [cre, ctb] (<https://orcid.org/0000-0003-4079-5621>), Savannah Obregon [ctb], Thomas Obadia [ctb], Marcus Lehr [ctb], Brian High [ctb],
R topics documented:

- allocationTable ........................................ 4
- armsMethods ........................................... 6
- assembleCodebook ..................................... 9
- changedRecords ......................................... 11
- checkbox_suffixes ..................................... 12
- constructLinkToRedcapForm ......................... 13
- createFileRepositoryFolder ......................... 14
- createRedcapProject ................................. 16
- dagAssignmentMethods ............................... 19
- dagMethods ............................................ 21
- deleteRecords ......................................... 23
- deprecated_redcapProjectInfo ....................... 25
- dropRepeatingNA ....................................... 27
- eventsMethods ......................................... 27
- exportBulkRecords .................................... 30
- exportBundle ........................................... 33
- exportDataQuality ................................... 34
- exportExternalCoding ................................. 35
- exportFieldNames .................................... 36
- exportFileRepositoryListing ....................... 38
- exportFilesMultiple ................................... 40
- exportInstruments ..................................... 43
- exportLogging ......................................... 44
- exportPdf ............................................... 46
- exportProjectXml ...................................... 48
- exportVersion .......................................... 50
- Extraction ............................................. 51
- fieldCastingFunctions ............................... 52
- fieldChoiceMapping ................................... 56
- fieldToVar ............................................. 56
- fieldValidationAndCasting ......................... 57
R topics documented:

fileMethods .......................................................... 62
fileRepositoryMethods ............................................. 66
fileRepositoryPath .................................................. 70
filterEmptyRow ....................................................... 70
fromFileRepositoryMethods ....................................... 71
gerProjectIdFields ................................................... 73
importFileToRecord ............................................... 74
importRecords ....................................................... 76
invalidSummary ....................................................... 79
isZeroCodedCheckField ............................................. 80
makeApiCall ......................................................... 81
mappingMethods .................................................... 84
massert ............................................................... 86
metaDataMethods .................................................... 87
missingSummary ....................................................... 90
parseBranchingLogic .............................................. 91
prepUserImportData ............................................... 92
preserveProject ..................................................... 94
projectInformationMethods ....................................... 97
purgeRestoreProject ............................................... 100
recodeCheck ........................................................ 103
reconstructFileFromExport ....................................... 104
recordsManagementMethods ...................................... 105
recordsMethods ..................................................... 107
recordsTypedMethods ............................................. 112
redcapConnection .................................................. 119
redcapDataStructures ............................................. 124
redcapError .......................................................... 125
redcapFactorFlip ................................................... 126
repeatingInstrumentMethods ..................................... 126
reviewInvalidRecords ............................................. 128
splitForms ........................................................... 130
stringCleanup ........................................................ 131
stripHTMLandUnicode .............................................. 132
surveyMethods ...................................................... 133
switchDag ............................................................ 136
syncUnderscoreCodings .......................................... 137
unlockREDCap ....................................................... 139
userMethods ........................................................ 140
userRoleAssignmentMethods ..................................... 145
userRoleMethods .................................................... 147
validateImport ....................................................... 152
vectorToApiBodyList ............................................... 153
widerRepeated ....................................................... 153
writeDataForImport ............................................... 154

Index 155
Generate Allocation Tables for the Randomization Module

Description

These methods enable the user to generate allocation tables for the REDCap randomization module. Randomization may be stratified by other (categorical) variables in the data set as well as by Data Access Group. Additionally, randomization may be blocked to ensure balanced groups throughout the allocation.

Usage

allocationTable(
  rcon,
  random,
  strata = NULL,
  group = NULL,
  dag.id = NULL,
  replicates,
  block.size,
  block.size.shift = 0,
  seed.dev = NULL,
  seed.prod = NULL,
  weights = NULL,
  ...
)

## S3 method for class 'redcapApiConnection'
allocationTable(
  rcon,
  random,
  strata = NULL,
  group = NULL,
  dag.id = NULL,
  replicates,
  block.size,
  block.size.shift = 0,
  seed.dev = NULL,
  seed.prod = NULL,
  weights = c(1, 1),
  ...
)

allocationTable_offline(
  meta_data,
  random,
  strata = NULL,
Arguments

rcon | A redcapConnection object.
random | character(1). The field name to be randomized.
strata | character. Field names by which to stratify the randomization.
group | character(1). A field name giving a group by which randomization should be stratified. This could also be listed in strata, but the argument is provided to remain consistent with the REDCap user interface.
dag.id | integerish. Data Access Group IDs.
replicates | integerish(1). The number of randomizations to perform within each stratum
block.size | integerish. Block size for the randomization. Blocking is recommended to ensure balanced groups throughout the randomization. This may be a vector to indicate variable block sizes throughout the randomization.
block.size.shift | numeric on the interval [0, 1]. A vector the same length as block.size where the first element is 0. This controls when the block size changes as a proportion of the total sample size. When block.size=c(8, 4, 2) and block.size.shift = c(0, 0.5, 0.9), the first half of the randomization is performed in blocks of 8, then the next 40 percent of the randomization is performed in blocks of 4, with the last ten percent performed in blocks of 2.
seed.dev | integerish. At least one value is required. If only one value is given, it will be converted to a vector with length equal to the number of strata. Values will be incremented by 100 to provide independent randomizations. This may also have length equal to the number of strata.
seed.prod | integerish. Same as seed.dev, but used to seed the production allocation. No pairwise elements of seed.dev and seed.prod may be equal. This guarantees that the two randomization schemes are unique.
weights | An optional vector giving the sampling weights for each of the randomization groups. There must be one number for each level of the randomization variable. If named, the names must match the group labels. If unnamed, the group labels will be assigned in the same order they appear in the data dictionary. The weights will be normalized, so they do not need to sum to 1.0. In other words, weights=c(3, 1) can indicate a 3:1 sampling ratio.
... | Arguments to pass to other methods
meta_data | character(1). For the offline method, a text string giving the location of the data dictionary downloaded from REDCap.
Details

Each element in block.size must be a multiple of the number of groups in the randomized variable. The ‘offline’ version of the function operates on the data dictionary file downloaded from REDCap. This is made available for instances where the API cannot be accessed for some reason (such as waiting for API approval from the REDCap administrator).

The value of replicates controls how many allocations are generated. It is possible to get slightly more replicates than requested if your blocking design cannot exactly match replicates. For example, if the users asks for 30 replicates in blocks of 8, a warning will be printed and 32 replicates will be returned in the randomization table.

Value

Returns a list with the elements

- dev_allocation: data.frame with the randomization allocations for the development environment.
- prod_allocation: data.frame with the randomization allocations for the production environment.
- dev_seed: The random seed values for the development environment.
- prod_seed: The random seed values for the production environment.
- blocks: Blocking scheme used to generate the randomization.
- weights: Weighting scheme for the randomization.

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

allocationTable(rcon,
                random = "treatment_assignment",
                strata = c("demographic_group", "hospital_group"),
                replicates = 12,
                block.size = 4,
                seed.dev = 12345,
                seed.prod = 54321)

## End(Not run)
```

armsMethods

Export, Import, and Delete Arms from a Project

Description

These methods enable the user to export the current arms from a project, import new arms, and modify or delete existing arms.
Usage

exportArms(rcon, ...)

importArms(rcon, data, override = FALSE, ...)

deleteArms(rcon, arms, ...)

## S3 method for class 'redcapApiConnection'
exportArms(
  rcon,
  arms = character(0),
  ..., 
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
importArms(
  rcon,
  data,
  override = FALSE,
  ..., 
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
deleteArms(
  rcon,
  arms,
  ..., 
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon
  A redcapConnection object.

arms
  character or integerish identifying the arm numbers to export or delete.

data
  A data.frame with two columns. The first column (arm_num) is an integerish value. The second (name) is a character value. For backward compatibility, this may also be passed as arms_data.

override
  logical(1). By default, data will add to or modify existing arms data. When TRUE, all the existing arms data is deleted and replaced with the contents of data.
... Arguments to pass to other methods

error_handling character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError.

config A named list. Additional configuration parameters to pass to http::POST. These are appended to any parameters in rcon$conf.

api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

Exporting arms is not supported for classical REDCap projects. If the user attempts to export arms for a classical project, a data frame will be returned with zero rows.

When importing, arms are added when the value of arm_num does not already exist in the project.

Arm names may be modified by altering the name value associated with an existing arm_num value.

Deleting arms—whether by deleteArms or importArms with override = TRUE—is a destructive act that also deletes events and records associated with the arm. This is irreversible data loss. REDCap will only permit these actions to occur in projects in Development status.

Value

exportArms returns a data.frame with columns:

- arm_num The ID number for the arm in the project.
- name The display name of the arm.

importArms invisibly returns the number of arms imported.

deleteArms invisibly returns the number of arms deleted.

Functions

- exportArms(): Export the arms defined in a project.
- importArms(): Import and modify the arms definitions in a project.
- deleteArms(): Delete arms from a project.

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Export all of the Arms
exportArms(rcon)
# Export only a subset of arms
exportArms(rcon,
    arms = c(1, 3))

# Import a new arms
# Assume arms 1, 2, and 3 exist in the project already
NewData <- data.frame(arm_num = 4,
    name = "Arm Four Name")
importArms(rcon,
    data = NewData)

# Change the name of an existing arm
NewData <- data.frame(arm_num = 1,
    name = "New Arm Name")
importArms(rcon,
    data = NewData)

# Delete all arms and replace with a new specification
NewData <- data.frame(arm_num = c(1, 2),
    name = c("Treatment Arm", "Control Arm"))
importArms(rcon,
    data = NewData,
    override = TRUE)

# Delete an existing arm
deleteArms(rcon,
    arms = 4)

# Delete multiple existing arm
deleteArms(rcon,
    arms = c(2, 3))

## End(Not run)

---

**assembleCodebook**

*Assemble Codebook From the Data Dictionary*

**Description**

This method enables the user to construct a code book similar in style to the REDCap project codebook. The codebook is similar in nature to the data dictionary, but multiple choice fields are represented with one line per coding.

**Usage**

```
assembleCodebook(  
    rcon,
    fields = NULL,
)```
forms = NULL,
drop_fields = NULL,
field_types = NULL,
include_form_complete = TRUE,
expand_check = FALSE,
...
)

## S3 method for class 'redcapConnection'
assembleCodebook(
  rcon,
  fields = NULL,
  forms = NULL,
  drop_fields = NULL,
  field_types = NULL,
  include_form_complete = TRUE,
  expand_check = FALSE,
  ...
)

## S3 method for class 'redcapCodebook'
as.list(x, ...)

Arguments

rcon  A redcapConnection object.
fields character or NULL. When character, the code book will be limited to the intersection of the fields designated by fields and forms. When NULL, all fields are included.
forms character or NULL. When character, the code book will be limited to the intersection of the fields designated by fields and forms. When NULL, all forms are included.
drop_fields character or NULL. When given, fields named will be removed from the code book.
field_types character or NULL. When given, only the field types listed will be included in the code book. This will supercede the intersection of fields and forms. Matching of field types is performed against the values in the field_type column of the meta data.
include_form_complete logical(1). When TRUE, the [form name]_complete fields will be included in the codebook.
expand_check logical(1). When FALSE, the codebook for checkbox fields will be similar to the codebook for dropdown and radio fields, with one line per user-defined option. When TRUE, each checkbox option will be represented in two fields, one each for 0 (Unchecked) and 1 (Checked).
...
Arguments to pass to other methods
x  A redcapCodebook object as returned by assembleCodebook.
Value

Returns a redcapCodebook object. This inherits the data.frame class and has the columns

- **field_name** - The name of the field.
- **form** - The name of the form on which the field is located.
- **field_type** - The field type.
- **code** - For multiple choice fields, the coding for the option.
- **label** - For multiple choice fields, the label for the option.
- **min** - For date and numeric fields, the minimum value in the validation, if any.
- **max** - For date and numeric fields, the maximum value in the validation, if any.
- **branching_logic** - For fields with branching logic, the string denoting the logic applied.
- **field_order** - The numeric order of the field in the data dictionary.
- **form_order** - The numeric order of the form in the data dictionary.

See Also

exportMetaData()

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

# codebook for the entire project
assembleCodebook(rcon)

# codebook for multiple choice fields
assembleCodebook(rcon,
                 field_types = c("dropdown", "radio", "checkbox",
                                 "yesno", "truefalse"))

## End(Not run)
```

---

**changedRecords**

returns a list of record IDs changed (adds, updates, deletes)

Description

This is a convenience function that scans logs and returns record IDs of changed records.

Usage

```r
changedRecords(rcon, ...)
```
Arguments

- `rcon` A redcapConnection object.
  ... Arguments passed to `exportLogging()`

Details

Makes a call to `exportLogging` with passed arguments. Returns filtered list or records IDs with update, delete or create events.

Value

Returns a list with the elements

- `updated` character vector of updated record IDs
- `deleted` character vector of deleted record IDs
- `created` character vector of created record IDs

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Changes in last 24 hours
changedRecords(rcon, beginTime=as.POSIXct(Sys.time()-86400))
```

## End(Not run)

---

### checkbox_suffixes

**Checkbox Suffixes**

**Description**

Checkbox variables return one vector of data for each option defined in the variable. The variables are returned with the suffix `__[option]`. `exportRecords` needs these suffixes in order to retrieve all of the variables and to apply the correct labels.

**Usage**

`checkbox_suffixes(fields, meta_data)`
constructLinkToRedcapForm

Construct a Link to a REDCap Form

Description

Uses information from the project and a record to link to the form on which a data element is recorded. This is intended to be used within the report of invalid results when exporting or importing records. It should be noted that when importing records, the records may not yet exist and the links may not work.

Usage

constructLinkToRedcapForm(rcon, form_name, record_id, event_id = NULL, ...)

## S3 method for class 'redcapApiConnection'
constructLinkToRedcapForm(rcon, form_name, record_id, event_id = NULL, ...)

## S3 method for class 'redcapOfflineConnection'
constructLinkToRedcapForm(rcon, form_name, record_id, event_id = NULL, ...)

Arguments

- **rcon** A redcapConnection object.
- **form_name** character. The name of the form on which the field name exists.
- **record_id** character. The ID of the record being linked to. If passed as a numeric value, it will be coerced to character. Must have the same length as **form_name**.
- **event_id** character or NULL. For classical projects, use either NULL or NA (NA support is permitted to assist with vectorization). For longitudinal projects, the ID of the unique event. If passed as a numeric value, it will be coerced to character.
- **...** Arguments to pass to other methods

Details

Constructing a link to a REDCap form requires knowledge of the following:

- The REDCap instance url (usually `redcap.institution.domain`).
- The REDCap instance version number.
- The REDCap project ID number
- The record ID
createFileRepositoryFolder

Create a Folder in the File Repository

**Description**

This method enables the user to create a folder in the file repository. The folder created may also be a subfolder of an existing folder.

**Usage**

```r
createFileRepositoryFolder(
  rcon,  # REDCap connection
  name,  # Form name
  folder_id = numeric(0),  # Folder ID
  dag_id = numeric(0),  # DAG ID
  role_id = numeric(0),  # Role ID
  ...  # Additional parameters
)
```

### S3 method for class `redcapApiConnection`

```r
createFileRepositoryFolder(
  rcon,  # REDCap connection
  name,  # Form name
  folder_id = numeric(0),  # Folder ID
  dag_id = numeric(0),  # DAG ID
  role_id = numeric(0),  # Role ID
  ...,  # Additional parameters
  error_handling = getOption("redcap_error_handling"),  # Error handling
  config = list(),  # Configuration options
  api_param = list()  # API parameters
)
```

- The form name
- The event ID number (if the project is longitudinal).

If any of these items is unknown, a missing value will be returned. For `redcapOfflineConnections`, the user will need to provide the version number, the project information, and the events (if the project is longitudinal) as part of the call to `offlineConnection`. Note that the REDCap User Interface does not include the event ID number with the file download for events.

**Value**

Returns a character vector the same length of `form_name`. 
createFileRepositoryFolder

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcon</td>
<td>A redcapConnection object.</td>
</tr>
<tr>
<td>name</td>
<td>character(1) The name of the folder. If a folder by this name already exists, no action will be taken.</td>
</tr>
<tr>
<td>folder_id</td>
<td>integerish(0/1). The ID of the parent folder. When length is 0, the new folder is placed in the top-level.</td>
</tr>
<tr>
<td>dag_id</td>
<td>integerish(0/1) The ID of a data access group. When provided, access to the folder will be restricted to the DAG.</td>
</tr>
<tr>
<td>role_id</td>
<td>integerish(0/1) The ID of a role. When provided, access to the folder will be restricted to users with that role.</td>
</tr>
<tr>
<td>...</td>
<td>Arguments to pass to other methods</td>
</tr>
<tr>
<td>error_handling</td>
<td>character(1). One of c(&quot;error&quot;, &quot;null&quot;). An option for how to handle errors returned by the API. see redcapError().</td>
</tr>
<tr>
<td>config</td>
<td>A named list. Additional configuration parameters to pass to http::POST(). These are appended to any parameters in rcon$config.</td>
</tr>
<tr>
<td>api_param</td>
<td>A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.</td>
</tr>
</tbody>
</table>

Value

Returns a data frame with the columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>folder_id</td>
<td>The REDCap assigned ID value for the newly created folder.</td>
</tr>
<tr>
<td>name</td>
<td>The name assigned to the folder by the user.</td>
</tr>
</tbody>
</table>

See Also

- exportFromFileRepository()
- importToFileRepository()
- deleteFromFileRepository()
- exportFileRepository()
- importFileRepository()
- deleteFileRepository()
- exportFileRepositoryListing()

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYs",
              envir = globalenv())

# Create a folder in the top-level directory
```
createRedcapProject

Description

These methods enable a user with a 64-character Super API token to create a new REDCap project.

Usage

createRedcapProject(
  rcon,
  project_title,
  purpose = REDCAP_PROJECT_PURPOSE,
  purpose_other = NULL,
  is_longitudinal = FALSE,
  surveys_enabled = FALSE,
  record_autonumbering_enabled = FALSE,
  xml = NULL,
  ...
)

## S3 method for class 'redcapApiConnection'
createRedcapProject(
  rcon,
  project_title,
  purpose = REDCAP_PROJECT_PURPOSE,
  purpose_other = NULL,
  is_longitudinal = FALSE,
  surveys_enabled = FALSE,
  record_autonumbering_enabled = FALSE,
  xml = NULL,
  ...
createRedcapProject

...,
error_handling = getOption("redcap_error_handling"),
config = list(),
api_param = list()
)

Arguments

rcon
A redcapConnection object.

project_title
character(1). Title for the new project.

purpose
class, one of c("Practice/just for fun", "Other", "Research", "Quality Improvement", "Operational Support")

purpose_other
character(1) or NULL. Ignored unless purpose = "Other", in which case this becomes a required argument.

is_longitudinal
logical(1). When TRUE the project will be set as a longitudinal project.

surveys_enabled
logical(1). When TRUE surveys are enabled for the project. (This will not add any survey instruments, only enable them).

record_autonumbering_enabled
logical(1). When TRUE if auto numbering will be enabled in the project.

xml
character(1) or NULL an XML string in CDISC ODM XML format that contains project metadata (fields, forms, events, arms) and might optionally contain data to be imported as well. When not NULL, all other arguments are ignored. See Details.

... Arguments to pass to other methods

error_handling
character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

config
A named list. Additional configuration parameters to pass to httr::POST().

These are appended to any parameters in rcon$config.

api_param
A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

The user creating the project will automatically be added to the project as a user with full user privileges and a project-level API token, which could then be used for subsequent project-level API requests.

When the project is created, it will automatically be given all the project-level defaults just as if it had been created via the web user interface, such as automatically creating a single data collection instrument seeded with a single Record ID field and Form Status field, as well as (for longitudinal projects) one arm with one event.

If the user intends to populate the project with arms and events immediately after creating the project, it is recommended that override = TRUE be used in importArms and importEvents so that the default arm and event are removed.
The `xml` argument must be in CDISC ODM XML format. It may come from a REDCap Project XML export file from REDCap itself (see `exportProjectXml()`), or may come from another system that is capable of exporting projects and data in CDISC ODM format. If the `xml` argument is used in the API request, it will use the XML to import its contents into the newly created project. This will not only create the project with the API request, but also to import all fields, forms, and project attributes (and events and arms, if longitudinal) as well as record data all at the same time.

Only users with a 64-character Super API Tokens can utilize this method (the standard API token is 32 characters). Users can only be granted a super token by a REDCap administrator (using the API Tokens page in the REDCap Control Center). Please be advised that users with a Super API Token can create new REDCap projects via the API without any approval needed by a REDCap administrator.

**Value**

Returns a character(1) the 32-character, project level API token assigned to the user that created the project. This token is intended to be used for further project configuration using the API.

**See Also**

`exportProjectXml()`

**Examples**

```r
## Not run:
# The token must be a 64-character token
super_token <- redcapConnection(url = "your_redcap_url",
                                 token = "[64-character-super-api-token]")

# Create a new project
createRedcapProject(super_token,
                     project_title = "New Project Name",
                     purpose = "Quality Improvement",
                     is_longitudinal = FALSE,
                     surveys_enabled = TRUE)

# Copy an existing project into a new project
unlockREDCap(connections = c(rcon = "token_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYs",
              envir = globalenv())

xml_file <- tempfile(file.ext = ".xml")
exportProjectXml(rcon,
                 file = xml_file)

xml_text <- paste0(readLines(xml_file), collapse = " ")
createRedcapProject(super_token,
                     xml = xml_text)
```
### Export and Import Users Assigned to Data Access Groups

**Description**

These methods enable the user to export existing assignments of users to Data Access Groups, or import new or updated assignments to the project.

**Usage**

```r
exportUserDagAssignments(rcon, ...)
importUserDagAssignments(rcon, data, ...)
```

```r
## S3 method for class 'redcapApiConnection'
exportUserDagAssignments(
  rcon,
  ...,
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
```

```r
## S3 method for class 'redcapApiConnection'
importUserDagAssignments(
  rcon,
  data,
  ...,
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
```

**Arguments**

- `rcon`  
  A `redcapConnection` object.

- `data`  
  `data.frame` with the columns `username` and `redcap_data_access_group`. The should only be one row per user name.

- `...`  
  Arguments to pass to other methods

- `error_handling`  
  `character(1)`. One of `c("error", "null")`. An option for how to handle errors returned by the API. see `redcapError()`.

- `config`  
  A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`. 
api_param

A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

When modifying existing assignments using the import method, the user must provide the unique user name and the group name. If the redcap_data_access_group column is not provided, the REDCap user will not be assigned to any group.

Value

exportUserDagAssignments method returns a data frame with two columns:

- username: The unique user name for each user in the project.
- redcap_data_access_group: The unique Data Access Group name to which the user is assigned.

importUserDagAssignments invisibly returns the number of assignments imported.

Functions

- `exportUserDagAssignments()`: Export current User-DAG Assignments
- `importUserDagAssignments()`: Import new or modified User-DAG Assignments.

See Also

- `exportDags()`, `importDags()`, `deleteDags()`, `switchDag()`

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
  url = "your_redcap_url",
  keyring = "API_KEYs",
  envir = globalenv())

# Export the current assignments
exportUserDagAssignments(rcon)

# Assign a user to a Data Access Group
ForImport <- data.frame(username = "user1",
  redcap_data_access_group = "facility_one")
importUserDagAssignments(rcon,
  data = ForImport)
```
# Assign a multiple users to a Data Access Group
ForImport <- data.frame(username = c("user1", "user2", "user3"),
    redcap_data_access_group = c("facility_one",
    "facility_one",
    "facility_two"))
importUserDagAssignments(rcon, data = ForImport)

# Remove a user from all Data Access Groups
ForImport <- data.frame(username = "user1",
    redcap_data_access_group = NA_character_)
importUserDagAssignments(rcon, data = ForImport)

## End(Not run)

dagMethods

Export, Import, Delete Data Access Groups from a Project

Description

These methods enable the user to export existing Data Access Groups, import new Data Access Groups, or delete Data Access Groups from a project.

Usage

exportDags(rcon, ...)
importDags(rcon, data, ...)
deleteDags(rcon, dags, ...)

## S3 method for class 'redcapApiConnection'
exportDags(rcon, ...,
    error_handling = getOption("redcap_error_handling"),
    config = list(),
    api_param = list() )

## S3 method for class 'redcapApiConnection'
importDags(rcon, data, ...
    error_handling = getOption("redcap_error_handling"),
    config = list(),
api_param = list()
)
## S3 method for class 'redcapApiConnection'
deleteDags(
  rcon,
  dags,
  ...,
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon  A redcapConnection object.
dags  character vector of names matching the unique_group_name.
data  A data.frame with two columns: data_access_group_name and unique_group_name.
...  Arguments to pass to other methods
error_handling  character(1). One of c("error", "null"). An option for how to handle errors returned by the API, see redcapError()
config  A named list. Additional configuration parameters to pass to http::POST(). These are appended to any parameters in rcon$config
api_param  A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

To import new data access groups, the user must provide a value for data_access_group_name with no value (NA) for unique_group_name.

To modify a group name, provide a new value for data_access_group_name with the associated unique_group_name. If unique_group_name is provided, it must match a value currently in the project.

Value

exportDags with the columns

data_access_group_name  The human readable name for the data access group.
unique_group_name      The internal unique group name.
data_access_group_id   The internal numeric identifier.

importDags invisibly returns the number of Data Access Groups imported.
deleteDags invisibly returns the number of Data Access Groups deleted.
deleteRecords

Delete Records from a Project

Description

These methods enable the user to delete records from a project.

Functions

- `exportDags()`: Export Data Access Groups from a REDCap Project
- `importDags()`: Import Data Access Groups to a project.
- `deleteDags()`: Delete Data Access Groups from a project.

See Also

`switchDag()`,
`exportUserDagAssignments()`,
`importUserDagAssignments()`

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYS",
              envir = globalenv())

exportDags(rcon)

# Import a new Data Access Group
NewData <- data.frame(data_access_group_name = "New DAG Name",
                       unique_group_name = NA_character_)
importDags(rcon,
           data = NewData)

# Modify an existing Data Access Group Name
# The user will need to match the unique_group_name to the existing DAGs
ChangeData <- data.frame(data_access_group_name = "Altered DAG Name",
                          unique_group_name = "new_dag_name")
importDags(rcon,
           data = ChangeData)

# Delete a Data Access Group
deleteDags(rcon,
           dags = "new_dag_name")
```

## End(Not run)
deleteRecords

Usage

deleteRecords(
  rcon,
  records,
  arm = NULL,
  instrument = NULL,
  event = NULL,
  repeat_instance = NULL,
  delete_logging = FALSE,
  ...
)

## S3 method for class 'redcapApiConnection'
deleteRecords(
  rcon,
  records,
  arm = NULL,
  instrument = NULL,
  event = NULL,
  repeat_instance = NULL,
  delete_logging = FALSE,
  ..., error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon         A redcapConnection object.
records      character or integerish. Record ID's to be returned.
arm          integerish. the arm number of the arm in which the record(s) should be deleted. This can only be used if the project is longitudinal with more than one arm. If the arm parameter is not provided, the specified records will be deleted from all arms in which they exist. Whereas, if arm is provided, they will only be deleted from the specified arm.
instrument   character(1) Optional instrument to delete records from.
event        character(1) Optional event to delete records from.
repeat_instance numeric(1) optional repeat instance to delete records from.
delete_logging logical. Should the logging for this record be delete as well. Default to FALSE.
...           Arguments to pass to other methods
error_handling character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().
config  A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.

api_param  A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by `redcapAPI`.

Value

deleteRecords invisibly returns a character value giving the number of records deleted.

See Also

extportRecords(),
importRecords(),
extportRecordsTyped()

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

# Delete records
deleteRecords(rcon, records = c("1", "2"))

## End(Not run)
```

deprecated_redcapProjectInfo

**Deprecated Functions**

Description

The `redcapProjectInfo` function has been deprecated to avoid confusion with the API method now executed by `exportProjectInformation`. The replacement function is `exportBundle()`.

Usage

```r
redcapProjectInfo(
  rcon,
  date = TRUE,
  label = TRUE,
  meta_data = TRUE,
```
users = TRUE,
instruments = TRUE,
events = TRUE,
arms = TRUE,
mappings = TRUE,
version = TRUE,
...
)

## S3 method for class 'redcapApiConnection'
redcapProjectInfo(
  rcon,
  date = TRUE,
  label = TRUE,
  meta_data = TRUE,
  users = TRUE,
  instruments = TRUE,
  events = TRUE,
  arms = TRUE,
  mappings = TRUE,
  version = TRUE,
  ...,
  v.number = ""
)

Arguments

rcon  A REDCap connection object as generated by redcapConnection
date  Logical. If TRUE, user expiration dates are converted to POSIXct objects.
label Logical. If TRUE, the user form permissions are converted to labeled factors.
meta_data Logical. Indicates if the meta data (data dictionary) should be exported.
users Logical. Indicates if the users table should be exported.
instruments Logical. Indicates if the instruments table should be exported.
events Logical. Indicates if the event names should be exported.
arms Logical. Indicates if the arms table should be exported.
mappings Logical. Indicates if the form-event mappings should be exported.
version Indicates if the REDCap version number should be exported. Only applicable in REDCap 6.0.0 and higher.
... Arguments to be passed to other methods
v.number A character string given the desired version number should the API method not be available.
dropRepeatingNA

**Drop Row Where Repeat Instrument Is NA**

**Description**

Drops rows where the repeat instrument is NA. Returns a data frame of records where repeat instruments have a value.

**Usage**

```r
dropRepeatingNA(Records, rcon, quiet = FALSE)
```

**Arguments**

- **Records**: A data.frame containing the records from `exportRecordsTyped()`
- **rcon**: A `redcapConnection` object.
- **quiet**: logical(1). When FALSE, a message is printed indicating how many rows were in `Records` at the start and completion of the subset.

**See Also**

- `exportRecordsTyped()`, `exportReportsTyped()`

---

**eventsMethods**

*Export, Import, and Delete Event Settings*

**Description**

These methods enable the user to export event settings, import new events, update settings for existing events, or delete events.

**Usage**

```r
exportEvents(rcon, ...)
importEvents(rcon, data, override = FALSE, ...)
deleteEvents(rcon, events = NULL, ...)
```

```r
## S3 method for class 'redcapApiConnection'
exportEvents(
  rcon,
  arms = NULL,
  ...,
```
error_handling = getOption("redcap_error_handling"),
config = list(),
api_param = list()
)

## S3 method for class 'redcapApiConnection'
importEvents(
  rcon,
  data,
  override = FALSE,
  ...
)
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()

## S3 method for class 'redcapApiConnection'
deleteEvents(
  rcon,
  events = NULL,
  ...
)
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()

Arguments

rcon A redcapConnection object.
arms character or integerish identifying the arm numbers for which event data
will be exported.
events character giving the unique event names of the events to be deleted.
data data.frame. Must have columns event_name and arm_num. To modify exist-
ing events, it must also have a column unique_event_name. It may optionally
have columns for days_offset, offset_min, offset_max. For backward com-
patibility, this argument may be passed as event_data.
override logical(1). By default, data will add to or modify existing arms data. When
TRUE, all the existing arms data is deleted and replaced with the contents of data.
... Arguments to pass to other methods
error_handling character(1). One of c("error", "null"). An option for how to handle
errors returned by the API. see redcapError().
config A named list. Additional configuration parameters to pass to http::POST().
These are appended to any parameters in rcon$config.
api_param A named list. Additional API parameters to pass into the body of the API
call. This provides users to execute calls with options that may not otherwise be
supported by redcapAPI.
Details

Exporting events is not supported for classical REDCap projects. If the user attempts to export arms for a classical project, a data frame will be returned with zero rows.

Additionally, in order for events to be exported, the project must be longitudinal, have at least one arm, and at least one event defined. When these conditions are not satisfied, `exportEvents` will return a data frame with zero rows.

To import new events, the user must provide data with the `unique_event_name` set to `NA` (REDCap assigns the unique event name automatically from the user provided `event_name`).

To modify existing events, the user must provide the `unique_event_name`. The other fields in the data provided will overwrite the current values for the matching event.

Deleting events—whether by `deleteEvents` or `importEvents` with `override = TRUE`—is a destructive act that also deletes arms and records associated with the event. This is irreversible data loss. REDCap will only permit these actions to occur in projects in Development status.

Value

`exportEvents` returns a data frame with the columns:

- `event_name` The user provided name for the event.
- `arm_num` The arm number the event is associated with.
- `unique_event_name` The REDCap generated event name.
- `custom_event_label` An optional user provided label that may be used in place of the event name.
- `event_id` REDCap’s internal event identifier.
- `days_offset` The number of days since time zero (start of the study or project period) an event is scheduled to occur. This field is only provided when the scheduling module is enabled.
- `offset_min` The number of days before the `days_offset` during which the event may occur. This field is only provided when the scheduling module is enabled.
- `offset_max` The number of days before the `days_offset` during which the event may occur. This field is only provided when the scheduling module is enabled.

`importEvents` invisibly returns the number of events added or modified.

`deleteEvents` invisibly returns the number of events deleted.

Functions

- `exportEvents()`: Export events from a REDCap project.
- `importEvents()`: Add events to a project or modify existing events.
- `deleteEvents()`: Delete events from a project.

See Also

`exportMappings()`, `importMappings()`

Examples

```r
## Not run:
```
unlockREDCap(connections = c(rcon = "project_alias"),
    url = "your_redcap_url",
    keyring = "API KEYS",
    envir = globalenv())

# Export all events
exportEvents(rcon)

# Export events for a subset of arms
exportEvents(rcon,
    arms = c(1, 3))

# Import new events
NewEvents <- data.frame(event_name = c("Event 1",
    "Event 2"),
    arm_num = c(1, 1))
importEvents(rcon,
    data = NewEvents)

# Modify existing events
UpdateEvents <- data.frame(event_name = "Event 2 New Name",
    arm_num = 1,
    unique_event_name = "event_2_arm_1",
    custom_event_label = "The second visit")
importEvents(rcon,
    data = UpdateEvents)

# Replace all events with a new set
NewEvents <- data.frame(event_name = c("Event 1",
    "Event 2",
    "Event 1"),
    arm_num = c(1, 1, 2))
importEvents(rcon,
    data = NewEvents,
    override = TRUE)

# Delete events
deleteEvents(rcon,
    events = c("event_1_arm_1", "event_1_arm_2"))

## End(Not run)

---

exportBulkRecords A helper function to export multiple records and forms using a single call.

### Description

Exports records from multiple REDCap Databases using multiple calls to `exportRecordsTyped()`.
Usage

```r
exportBulkRecords(
  lcon,
  forms = NULL,
  envir = NULL,
  sep = "_",
  post = NULL,
  ...
)
```

Arguments

- **lcon**: A named list of connections. The name is used as a prefix for data.frame names in the environment specified. It may also be used as a reference from the forms argument.
- **forms**: A named list that is a subset of `rcon`’s names. A specified `rcon` will provide a list of forms for repeated calls to `exportRecordsType`. If a connection reference is missing it will default to all forms. To override this default specify a connection’s forms with NA to just get all data.
- **envir**: A environment to write the resulting Records in as variables given by their name in `rcon` or if from a form their `rcon` named pasted to their form name joined by `sep`. If not specified the function will return a named list with the results. Will accept a number of the environment.
- **sep**: A character string to use when joining the `rcon` name to the form name for storing variables.
- **post**: A function that will run on all returned sets of Records.
- **...**: Any additional variables to pass to `exportRecordsTyped()`.

Value

Will return a named list of the resulting records if `envir` is NULL. Otherwise will assign them to the specified `envir`.

See Also

- Other records exporting functions: `exportRecordsTyped()`, `exportRecords()`, `exportReports()`
- Field validations and casting: `fieldValidationAndCasting()`, `reviewInvalidRecords()`
- Post-processing functionality: `recastRecords()`, `guessCast()`
**exportBulkRecords**

guessDate(),
castForImport(),
mChoiceCast(),
splitForms(),
widerRepeated()

**Vignettes:**
vignette("redcapAPI-offline-connection")
vignette("redcapAPI-casting-data")
vignette("redcapAPI-missing-data-detection")
vignette("redcapAPI-data-validation")
vignette("redcapAPI-faq")

**Examples**

```r
## Not run:
unlockREDCap(c(test_conn = 'TestRedcapAPI',
               sandbox_conn = 'SandboxAPI'),
            keyring = 'MyKeyring',
            envir = globalenv(),
            url = 'https://<REDCAP_URL>/api/')

# After user interaction to unlock the local encrypted keyring
# the global environment will contain the REDCap connections
# 'test_conn' and 'sandbox_conn'
#
# Next the user wants to bulk specify importing all the forms
# of interest and post process

exportBulkRecords(
  rcon = list(test = test_conn,
              sand = sandbox_conn),
  forms = list(test = c('form1', 'form2'),
              envir = globalenv(),
  post = function(Records, rcon)
  {
    Records |>
    mChoiceCast(rcon) |>
    guessDat(rcon) |>
    widerRepeated(rcon)
  }
)

# The environment now contains the data.frames: 'test.form1', 'test.form2', 'sand'.
# Each of these were retrieved, possibly using the forms argument and all were
# post processed in the same manner as specified by 'post'.

## End(Not run)
```
exportBundle  

**exportBundle**  

Perform a bundle of API calls.

**Description**

This function is deprecated and will be removed with the release of redcapAPI 3.0.0. It's functionality is built into caching of data onto the redcapConnection object.

Several of the API calls return objects that can be used to perform various validations in exportRecords, exportReports, and other methods. Using an export bundle allows the user to call these methods once and store the result instead of issuing an additional call to the API each time a method is invoked.

For example, if the user is uploading several files to the API, without an export bundle, importFiles will utilize the exportMetaData on each call in order to perform validations. Using a bundle allows the user to download the meta data once and refer to it on every subsequent call that requires the data dictionary.

**Usage**

```r
exportBundle(
  rcon,
  date = TRUE,
  label = TRUE,
  meta_data = TRUE,
  users = TRUE,
  instruments = TRUE,
  events = TRUE,
  arms = TRUE,
  mappings = TRUE,
  version = TRUE,
  ...
)
```

```r
## S3 method for class 'redcapApiConnection'
exportBundle(
  rcon,
  date = TRUE,
  label = TRUE,
  meta_data = TRUE,
  users = TRUE,
  instruments = TRUE,
  events = TRUE,
  arms = TRUE,
  mappings = TRUE,
  version = TRUE,
  ...
, return_object = TRUE
)
```
exportDataQuality

Arguments

rcon A REDCap connection object as generated by redcapConnection.
date Logical. If TRUE, user expiration dates are converted to POSIXct objects.
label Logical. If TRUE, the user form permissions are converted to labeled factors.
meta_data Logical. Indicates if the meta data (data dictionary) should be exported.
users Logical. Indicates if the users table should be exported.
instruments Logical. Indicates if the instruments table should be exported.
events Logical. Indicates if the event names should be exported.
arms Logical. Indicates if the arms table should be exported.
mappings Logical. Indicates if the form-event mappings should be exported.
version Indicates if the REDCap version number should be exported. Only applicable in REDCap 6.0.0 and higher.
... Arguments to be passed to other methods
return_object Logical. When TRUE, the exportBundle object is returned to the workspace.

Details

The project information is stored in the option redcap_project_info. If the project is not longitudinal, the events, arms, and event-form mappings elements will be assigned character vectors instead of data frames.

ExportDataQuality A helper function to export data queries from the Data Quality REDCap module.

Description

Exports Data Quality queries by record. The Data Quality module must be enabled on the Control Center of REDCap to use this function. Additionally, this module must be enabled on each project before it can be used.

Usage

exportDataQuality(rcon, prefix, ...)

Arguments

rcon A REDCap connection object as generated by redcapConnection.
prefix A string from your REDCap institutions Data Quality module url. The module prefix can be found by exporting module settings under External Modules in REDCap. At VUMC the prefix is 'vanderbilt_dataQuality'.
... additional arguments that are ignored.
Description

These methods enable redcapAPI to obtain a mapping of codes and associated labels for fields that have external dependencies. The fields include SQL fields (dependent on another project) or fields that utilize the BioPortal Ontology modules.

Usage

```
exportExternalCoding(rcon, fields, ...)
```

## S3 method for class 'redcapApiConnection'

```
exportExternalCoding(
  rcon,
  fields = NULL,
  ..., 
  batch_size = 1000,
  error_handling =getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
```

Arguments

- `rcon` A redcapConnection object.
- `fields` character or NULL. Vector of fields to be returned. If NULL, all fields are returned (unless forms is specified).
- `...` Arguments to pass to other methods
- `batch_size` integerish(1) or NULL. When NULL, all records are pulled. Otherwise, the records all pulled in batches of this size.
- `error_handling` character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().
- `config` A named list. Additional configuration parameters to pass to httr::POST(). These are appended to any parameters in rcon$config.
- `api_param` A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

These methods operate by executing two API calls to export first the coded values and then the labeled values of fields with external dependencies. The two exports are then used to generate the code-label mappings for use in casting data.
Fields of type sql are dropdown fields that are populated by a SQL query to another project.
Fields of type bioportal are text fields that have the BioPortal Ontology module enabled as the validation method.

Value

Returns a named list of named character vectors.
Each element is in the list is named for the field it maps.
The character vectors are name-value pairs where the name is the labeled data and the value is the coded data.

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
    url = "your_redcap_url",
    keyring = "API_KEYS",
    envir = globalenv())

exportExternalCoding(rcon)

## End(Not run)
```

exportFieldNames

Export the Complete Field Names for a REDCap Project

Description

This method enables the user to access the complete field names utilized during export and import methods. These are especially relevant when working with checkbox fields.

Usage

```r
exportFieldNames(rcon, ...)

## S3 method for class 'redcapApiConnection'
exportFieldNames(
    rcon,
    fields = character(0),
    ...,
    error_handling = getOption("redcap_error_handling"),
    config = list(),
    api_param = list()
)
```
exportFieldNames

Arguments

rcon
A redcapConnection object.

fields
NULL or character(1). Field name to be returned. By default, all fields are returned.

Arguments to pass to other methods

error_handling
character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

config
A named list. Additional configuration parameters to pass to httr::POST(). These are appended to any parameters in rcon$config.

api_param
A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

exportFieldNames returns a data frame of the field names the user may use when performing export and import functions. This is most useful when working with checkbox fields, which have a different field name than the one used in the Meta Data. The exported/imported field names for checkbox fields have the pattern [field_name]___[coded_checkbox_value] (there are exactly three underscores separating the field name and the coded value).

Fields of types "calc", "file", and "descriptive" are not included in the export. (Signature fields also have the "file" type and are not included)

Value

exportFieldNames returns a data frame with the columns:

original_field_name The field name as recorded in the data dictionary
choice_value represents the raw coded value for a checkbox choice. For non-checkbox fields, this will always be NA.
export_field_name The field name specific to the field. For non-checkbox fields, this is the same as original_field_name.

See Also

exportMetaData(), importMetaData(), exportInstruments(), exportMappings(), importMappings(), exportPdf()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
...
keyring = "API_KEYs",
envir = globalenv())

# Export all of the field names
exportFieldNames(rcon)

# Export MetaData for a specific field
exportFieldNames(rcon,
    fields = "checkbox_test")

## End(Not run)

---

**exportFileRepositoryListing**

*Export a Listing of Folders and Files in the File Repository*

---

**Description**

This method enables the user to export a list of folders and files saved to the File Repository. The listing may optionally include contents of subfolders.

**Usage**

```r
exportFileRepositoryListing(
    rcon,
    folder_id = numeric(0),
    recursive = FALSE,
    ...
)
```

```r
## S3 method for class 'redcapApiConnection'
exportFileRepositoryListing(
    rcon,
    folder_id = numeric(0),
    recursive = FALSE,
    ...
)
```

---

**Arguments**

- **rcon**: A redcapConnection object.
- **folder_id**: integerish(0/1). The folder ID of a specific folder in the File Repository for which a list of files and subfolders will be exported. By default, the top-level directory of the File Repository will be used.
The REDCap assigned ID value for the folder. Will be NA if the item is a file.

- **doc_id**: The REDCap assigned ID value for the file. Will be NA if the item is a folder.

- **name**: The name of the folder or file.

- **parent_folder**: The ID of the parent folder of the item. The top-level folder is represented as 0.

### Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYs",
              envir = globalenv())

# Export the top-level listing of the File Repository
exportFileRepositoryListing(rcon)

# Export the complete listing of the File Repository
exportFileRepositoryListing(rcon, recursive = TRUE)

# Export the listing of a subfolder in the File Repository
```
exportFileRepositoryListing(rcon,
    folder_id = 12345)

## End(Not run)

exportFilesMultiple  Export Multiple Files From a Project

Description

This method enables the user to export multiple files from a REDCap project with a single call. The REDCap API only allows for one file to be exported per call, and the exportFiles() methods are written to mirror that limitation. This extension allows the user to pass vectors of arguments for records, fields, events, or repeat instances. Files that can be matched to any combination of these values will be exported.

Usage

exportFilesMultiple(
    rcon,
    record,
    field,
    event = NULL,
    dir,
    file_prefix = TRUE,
    ...
)

## S3 method for class 'redcapApiConnection'
exportFilesMultiple(
    rcon,
    record,
    field,
    event = NULL,
    dir,
    file_prefix = TRUE,
    repeat_instance = NULL,
    ...,
    quiet = TRUE,
    error_handling = getOption("redcap_error_handling"),
    config = list(),
    api_param = list()
)
Arguments

- **rcon**: A `redcapConnection` object.
- **record**: character or integerish. The record ID in which the desired file is stored.
- **field**: character. The field name in which the file is stored.
- **event**: character or NULL. The event name for the file. This applies only to longitudinal projects. If the event is not supplied for a longitudinal project, the API will return an error message.
- **dir**: character(1). A directory/folder to which the file will be saved. By default, the working directory is used.
- **file_prefix**: logical(1). Determines if a prefix is appended to the file name. The prefix takes the form [record_id]-[event_name]-[file_name]. The file name is always the same name of the file as it exists in REDCap.
- **repeat_instance**: integerish or NULL. The repeat instance number of the repeating event or the repeating instrument. When available in your instance of REDCap, and passed as NULL, the API will assume a value of 1.
- **quiet**: logical(1). When TRUE, any errors encountered while exporting files will be converted to messages.
- **error_handling**: character(1). One of c("error", "null"). An option for how to handle errors returned by the API. See `redcapError()`.
- **config**: A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.
- **api_param**: A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by `redcapAPI`.

Details

`exportFilesMultiple` will construct all combinations of the `record`, `field`, `event`, and `repeat_instance` arguments and attempt to export the file associated with each combination. Should any of these combinations produce an error (for example, if a record does not have a third repeat instance), the error is captured and returned with the output.

Value

Invisibly returns a `data.frame` with the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>record</td>
<td>The record ID</td>
</tr>
<tr>
<td>field</td>
<td>The name of the field in which the file is stored.</td>
</tr>
<tr>
<td>event</td>
<td>The name of the event associated with the file.</td>
</tr>
<tr>
<td>repeat_instance</td>
<td>For repeat instances, the instance associated with the file.</td>
</tr>
<tr>
<td>is_exported</td>
<td>logical indicating if the file was successfully exported.</td>
</tr>
<tr>
<td>saved_to</td>
<td>The file path to which the file was saved.</td>
</tr>
<tr>
<td>error</td>
<td>If an error was encountered, the text of the error.</td>
</tr>
</tbody>
</table>
See Also

exportFiles()

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYS",
             envir = globalenv())

save_to_dir <- tempdir()

# Export files for multiple records
# Results are returned invisibly - saving to an object is
# helpful to be able to view the results
Export <-
  exportFilesMultiple(rcon,
                      record = 1:4,
                      field = "file_upload_field",
                      event = "event_1_arm_1",
                      dir = save_to_dir)

Export

# Export files for multiple instances
Export <-
  exportFilesMultiple(rcon,
                      record = 1,
                      field = "file_upload_field",
                      event = "event_1_arm_1",
                      repeat_instance = 1:4,
                      dir = save_to_dir)

Export

# Export files for multiple records, fields, events, and instances
Export <-
  exportFilesMultiple(rcon,
                      record = 1:10,
                      field = c("registration", "waiver"),
                      events = c("event_1_arm_1", "event_2_arm_1"),
                      repeat_instance = 1:3,
                      dir = save_to_dir)

Export

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

These methods enable the user to view the instruments defined in the project.

**Usage**

```r
exportInstruments(rcon, ...)
```

## S3 method for class 'redcapApiConnection'

```r
everyoneInstruments(
  rcon,
  ...,
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
```

**Arguments**

- `rcon` A `redcapConnection` object.
- `...` Arguments to pass to other methods
- `error_handling` character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see `redcapError()`.
- `config` A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$Config`.
- `api_param` A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by `redcapAPI`.

**Value**

Returns a data frame with the columns:

- `instrument_name` The REDCap generated instrument name.
- `instrument_label` The user provided instrument label.

**See Also**

- `exportMetaData()`,
- `importMetaData()`,
- `exportInstruments()`.
exportMappings(),
importMappings(),
exportPdf()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

exportInstruments(rcon)

## End(Not run)

---

**exportLogging**

**Export Logging Records**

Description

These methods enable the user to export the logging (audit trail) of all changes made to a project, including data exports, data changes, project metadata changes, modification of user rights, etc.

Usage

```r
exportLogging(
  rcon,
  logtype = character(0),
  user = character(0),
  record = character(0),
  dag = character(0),
  beginTime = .POSIXct(character(0)),
  endTime = .POSIXct(character(0)),
  ...,
)
```

```r
## S3 method for class 'redcapApiConnection'
exportLogging(
  rcon,
  logtype = character(0),
  user = character(0),
  record = character(0),
  dag = character(0),
  beginTime = as.POSIXct(character(0)),
  endTime = as.POSIXct(character(0)),
  ...,n
)```
error_handling = getOption("redcap_error_handling"),
    config = list(),
    api_param = list()
)

Arguments

rcon A redcapConnection object.
logtype character(0/1). The log event types to export. When the length is zero, all
    event types are exported. Otherwise, it must be one of c("export", "manage",
    "user", "record", "record_add", "record_edit", "record_delete", "lock_record",
    "page_view")
user character(0/1). Users for whom to return logs. By default logs for all users
    are returned.
record character(0/1). Record ID for which logs are to be returned. By default, logs
    are returned for all records.
dag character(0/1). Data access group ID for which to return logs. By default,
    logs are returned for all data access groups.
beginTime POSIXct(0/1). When given, only logs recorded after this time will be returned.
endTime POSIXct(0/1). When given, only logs recorded before this time will be re-
    turned.
... Arguments to pass to other methods
error_handling character(1). One of c("error", "null"). An option for how to handle
    errors returned by the API. see redcapError().
config A named list. Additional configuration parameters to pass to http::POST().
    These are appended to any parameters in rcon$config.
api_param A named list. Additional API parameters to pass into the body of the API
    call. This provides users to execute calls with options that may not otherwise be
    supported by redcapAPI.

Value

Returns a data frame with columns

timestamp The date/time of the logging record.
username The user name of the user that performed the action being logged.
action The classification of action being logged.
details Details of the action being logged.
record The record ID associated with the action being logged. When not related to a record, this will be NA

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Export all of the logging events
exportLogging(rcon)

# Export all of the events for record '2'
exportLogging(rcon,
record = "2")

# Export all of the events where a record was deleted
exportLogging(rcon,
logtype = "record_delete")

## End(Not run)

---

**exportPdf**

Export PDF file of Data Collection Instruments

**Description**

These methods allow the user to download PDF files of data collection instruments. The download may be with or without collected data; and may return a single record, multiple records, or all records.

**Usage**

```r
exportPdf(
  rcon,
  dir,
  filename = "redcap_forms_download",
  record = NULL,
  events = NULL,
  instruments = NULL,
  all_records = FALSE,
  ...
)
```

## S3 method for class 'redcapApiConnection'

```r
exportPdf(
  rcon,
  dir,
  filename = "redcap_forms_download",
  record = NULL,
  events = NULL,
  instruments = NULL,
  all_records = FALSE,
)
calculation},

   ...,
   error_handling = getOption("redcap_error_handling"),
   config = list(),
   api_param = list()
)

Arguments

rcon  A redcapConnection object.
dir   character(1). The directory into which the file should be saved.
filename character(1). The base of the file name. When record = NULL, it will be
    appended with "_blank.pdf". When record has a value, it will be appended
    with "_record_[record id].pdf"
record character(1), integerish(1), or NULL. The record id for which forms should
    be downloaded.
events character. The events for which forms should be downloaded
instruments character. The instruments for which forms should be downloaded
all_records logical(1). When TRUE forms for all records are downloaded. When TRUE, this overrides the records argument.

Arguments to pass to other methods

error_handling character(1). One of c("error", "null"). An option for how to handle
errors returned by the API. see redcapError().
config A named list. Additional configuration parameters to pass to httr::POST(). These are appended to any parameters in rcon$config.
api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

These methods mimics the behavior of "Download PDF of Instruments" button on the REDCap user interface. They permit the user to export a PDF file for:

1. A single collection instrument (blank)
2. All instruments (blank)
3. A single instrument (with data from a single record)
4. All instruments (with data from a single record)
5. All instruments (with data from all records)

Value

exportPdf invisibly returns the location on the local system to which the files is saved.
See Also

`exportMetaData()`,
`importMetaData()`,
`exportFieldNames()`,
`exportInstruments()`,
`exportMappings()`,
`importMappings()`

---

**exportProjectXml**

Export Entire Project as REDCap XML File

**Description**

These methods enable the user to export a project’s settings as an XML file in CDISC ODM format. This file may be used to transfer the project to another project, REDCap instance, or any other CDISC ODM compliant database.

**Usage**

```r
exportProjectXml(
  rcon,  # redcapApiConnection
  file,
  return_metadata_only = TRUE,  # 'FALSE' or 'TRUE'
  records = NULL,             # vector of records to export
  fields = NULL,              # vector of fields to export
  events = NULL,              # vector of events to export
  survey = FALSE,             # export survey responses
  dag = FALSE,                # export the data aggregation graph
  export_files = FALSE,       # export datasets and variables
  ...  # additional arguments
)
```

## S3 method for class 'redcapApiConnection'

```r
exportProjectXml(
  rcon,  # redcapApiConnection
  file,
  return_metadata_only = TRUE,  # 'FALSE' or 'TRUE'
  records = NULL,             # vector of records to export
  fields = NULL,              # vector of fields to export
  events = NULL,              # vector of events to export
  survey = FALSE,             # export survey responses
  dag = FALSE,                # export the data aggregation graph
  export_files = FALSE,       # export datasets and variables
  ...  # additional arguments
  error_handling =getOption("redcap_error_handling"),  # character string
  config = list(),            # list of configuration settings
  api_param = list()          # list of API parameters
)
```
exportProjectXml

Arguments

rcon A redcapConnection object.

file character(1) The file to which the XML export will be saved.

return_metadata_only logical(1). When TRUE (default) only meta data values are returned. When FALSE, project records data are also exported.

records character or integerish. A vector of study id's to be returned. When NULL, all subjects are returned.

fields character. Vector of fields to be returned. When NULL, all fields are returned (unless forms is specified).

events A character. Vector of events to be returned from a longitudinal database. When NULL, all events are returned.

survey logical(1). When TRUE the survey identifier fields (e.g., redcap_survey_identifier) or survey timestamp fields (e.g., [form_name]_timestamp) will be included in the export when surveys are utilized in the project.

dag logical(1). When TRUE the redcap_data_access_group field is exported when data access groups are utilized in the project.

export_files logical(1). When TRUE will cause the XML returned to include all files uploaded for File Upload and Signature fields for all records in the project. Setting this option to TRUE can make the export very large and may prevent it from completing if the project contains many files or very large files.

... Arguments to pass to other methods

error_handling character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

config A named list. Additional configuration parameters to pass to httr::POST(). These are appended to any parameters in rcon$config.

api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

The entire project (all records, events, arms, instruments, fields, and project attributes) can be downloaded as a single XML file, which is in CDISC ODM format (ODM version 1.3.1). This XML file can be used to create a clone of the project (including its data, optionally) on this REDCap server or on another REDCap server (it can be uploaded on the Create New Project page). Because it is in CDISC ODM format, it can also be used to import the project into another ODM-compatible system.

When the return_metadata_only parameter is set to FALSE, the Data Export user rights will be applied to any data returned. For example, if the user has 'De-Identified' or 'Remove All Identifier Fields' data export rights, then some data fields might be removed and filtered out of the data set. To make sure that no data is unnecessarily filtered out of the API request, the user should have 'Full Data Set' export rights in the project.
See Also

createRedcapProject()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "token_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

xml_file <- tempfile(file.ext = ".xml")
exportProjectXml(rcon,
                 file = xml_file)

## End(Not run)
config A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.

api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by `redcapAPI`.

Value

Returns a character value giving the version number.

If this function is used in a version of REDCap that does not support the method (prior to version 6.0.0), the value "5.12.2" will be returned. This is done solely for the convenience of always returning a value that can be compared against other versions.

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

# Export the version number
exportVersion(rcon)

## End(Not run)
```

Extraction

`Extraction and Assignment for redcapFactors`

Description

Extract elements and make assignments to `redcapFactors`

Usage

```r
## S3 method for class 'redcapFactor'
x[...], drop = FALSE]

## S3 method for class 'redcapFactor'
print(x, ...)
```

Arguments

- `x` an object of class `redcapFactor`
- `...` additional arguments to pass to other methods
- `drop` logical. If TRUE, unused levels are dropped.
fieldCastingFunctions  Functions for Casting Fields After Export (Post Processing)

Description

The functions provided here allow for recasting fields after records have been exported. They generally have a similar interface to the casting strategy of `exportRecordsTyped()`, though they may not each offer all the same options.

Usage

```r
recastRecords(
  data,
  rcon,
  fields,
  cast = list(),
  suffix = "",
  warn_zero_coded = TRUE
)

castForImport(
  data,
  rcon,
  fields = NULL,
  na = list(),
  validation = list(),
  cast = list(),
  warn_zero_coded = TRUE
)

guessCast(
  data,
  rcon,
  na = isNAorBlank,
  validation,
  cast,
  quiet = FALSE,
  threshold = 0.8
)

guessDate(
  data,
  rcon,
  na = isNAorBlank,
  validation = valRx("^[0-9]{1,4}-(0?[1-9]|1[012])-(0?[1-9]|1[012]|3[01])$"),
  cast = function(x, ...) as.POSIXct(x, format = "%Y-%m-%d"),
  quiet = FALSE,
```
threshold = 0.8
)

mChoiceCast(data, rcon, style = "labelled", drop_fields = TRUE)

Arguments

data
  data.frame with the data fields to be recoded.

rcon
  A redcapConnection object.

fields
  character/logical/integerish. A vector for identifying which fields to recode. When logical, the length must match the number of columns in data (i.e., recycling not permitted). A message is printed if any of the indicated fields are not a multiple choice field; no action will be taken on such fields. For this function, yes/no and true/false fields are considered multiple choice fields. Fields of class mChoice are quietly skipped.

cast
  A named list of user specified class casting functions. The same named keys are supported as the na argument. The function will be provided the variables (x, field_name, coding). The function must return a vector of logical matching the input length. The cast should match the validation, if one is using raw_cast, then validation=skip_validation is likely the desired intent. See fieldValidationAndCasting()

suffix
  character(1). An optional suffix to provide if the recoded variables should be returned as new columns. For example, if recoding a field forklift_brand and suffix = "_labeled", the result will have one column with the coded values (forklift_brand) and one column with the labeled values (forklift_brand_labeled).

warn_zero_coded
  logical(1). Turn on or off warnings about zero coded fields. Default is TRUE.

na
  A named list of user specified functions to determine if the data is NA. This is useful when data is loaded that has coding for NA, e.g. -5 is NA. Keys must correspond to a truncated REDCap field type, i.e. date_, datetime_, date-time_seconds_, time_mm_ss, time_hh_mm_ss, time, float, number, calc, int, integer, select, radio, dropdown, yesno, truefalse, checkbox, form_complete, sql, system. The function will be provided the variables (x, field_name, coding). The function must return a vector of logicals matching the input. It defaults to isNAorBlank() for all entries.

validation
  A named list of user specified validation functions. The same named keys are supported as the na argument. The function will be provided the variables (x, field_name, coding). The function must return a vector of logical matching the input length. Helper functions to construct these are valRx() and valChoice(). Only fields that are not identified as NA will be passed to validation functions.

quiet
  Print no messages if triggered, Default=FALSE.

threshold
  numeric(1). The threshold of non-NA data to trigger casting.

style
  character. One of "labelled" or "coded". Default is "labelled"

drop_fields
  character or NULL. A vector of field names to remove from the data.
Details

recastRecords is a post-processing function motivated initially by the need to switch between codes and labels in multiple choice fields. Field types for which no casting function is specified will be returned with no changes. It will not attempt to validate the content of fields; fields that cannot be successfully cast will be quietly returned as missing values.

castForImport is written with defaults that will return data in a format ready to be imported to a project via importRecords. All fields are returned as character vectors. If any values fail to validation check, are report is returned as an attribute named invalid. This attribute may be retrieved using reviewInvalidRecords(). These are then set to NA, which will be imported as blanks through the API.

guessCast is a helper function to make a guess at casting uncast columns. It will do a type cast if a validation is met above a threshold ratio of non-NA records. It modifies the existing invalid attribute to reflect the cast. This attribute may be retrieved using reviewInvalidRecords().

guessDate is a special cast of guessCast that has defaults set for casting a date field.

mChoiceCast is a helper function that adds the Hmisc::mChoice multiple choice class. It adds a column for a multiple choice checkbox that is cast to the Hmisc::mChoice class. Requires Hmisc to be loaded.

Zero-Coded Check Fields

A zero-coded check field is a field of the REDCap type checkbox that has a coding definition of 0, [label]. When exported, the field names for these fields is [field_name]___0. As in other checkbox fields, the raw data output returns binary values where 0 represent an unchecked box and 1 represents a checked box. For zero-coded checkboxes, then, a value of 1 indicates that 0 was selected.

This coding rarely presents a problem when casting from raw values (as is done in exportRecordsTyped). However, casting from coded or labeled values can be problematic. In this case, it becomes indeterminate from context if the intent of 0 is ‘false’ or the coded value ‘0’ (‘true’)... The situations in which casting may fail to produce the desired results are

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>anything other than “0”</td>
<td>Likely to fail when casting from coded values</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Likely to fail when casting from coded or labeled values</td>
</tr>
</tbody>
</table>

Because of the potential for miscast data, casting functions will issue a warning anytime a zero-coded check field is encountered. A separate warning is issued when a field is cast from coded or labeled values.

When casting from coded or labeled values, it is strongly recommended that the function castCheckForImport() be used. This function permits the user to state explicitly which values should be recognized as checked, avoiding the ambiguity resulting from the coding.

See Also

Exporting records:
**fieldCastingFunctions**

```r
exportRecordsTyped().
exportReportsTyped().
fieldValidationAndCasting().
reviewInvalidRecords()

**Other Post Processing Functions:**
splitForms().
widerRepeated()

**Vignettes:**
vignette("redcapAPI-offline-connection", package = "redcapAPI")
vignette("redcapAPI-casting-data")
vignette("redcapAPI-missing-data-detection")
vignette("redcapAPI-data-validation")

**Examples**

```r
## Not run:
# Using recastRecords after export
Recs <-
  exportRecordsTyped(rcon) |>
  recastRecords(rcon,
    fields = "dropdown_test",
    cast = list(dropdown = castCode))

# Using castForImport
castForImport(Records, rcon)

# Using castForImport to recast zero-coded checkbox values
castForImport(Records, rcon,
  cast = list(checkbox = castCheckForImport(c("0", "Unchecked"))))

# Using guessCast
exportRecordsTyped(rcon,
  validation=skip_validation,
  cast = raw_cast) |>
  guessCast(rcon,
    validation=valRx("^[0-9]{1,4}-(0?[1-9]|1[012])-(0?[1-9]|1[2][0-9]|3[01])$"),
    cast=as.Date,
    threshold=0.6)

# Using mChoiceCast
exportRecordsTyped(rcon) |>
  mChoiceCast(rcon)
```
fieldChoiceMapping  

**Split a Field Choice Mapping Into a Two Column Matrix**

**Description**

Uses the string from the select_choices_or_calculations for the meta data to create a matrix of codes and their mapped labels.

**Usage**

```r
fieldChoiceMapping(object, field_name, ...)
```

## S3 method for class 'character'

```r
fieldChoiceMapping(object, field_name, ...)
```

## S3 method for class 'redcapApiConnection'

```r
fieldChoiceMapping(object, field_name, ...)
```

**Arguments**

- `object`: redcapConnection or character(1). When character, is matches the format of the meta data field choices (i.e. rcon$meta_data()$select_choices_or_calculations).
- `field_name`: character(1) gives the field name for which to make the choice mapping.
- `...`: Arguments to pass to other methods

**Value**

Returns a matrix with two columns, choice_value and choice_label.

---

fieldToVar  

**Convert a REDCap Data Field to an R Vector**

**Description**

Converts a field exported from REDCap into a valid R vector.

---
**fieldValidationAndCasting**

**Usage**

```r
fieldToVar(
  records,
  meta_data,
  factors = TRUE,
  dates = TRUE,
  checkboxLabels = FALSE,
  labels = TRUE,
  handlers = list(),
  mChoice = NULL,
  ...
)
```

**Arguments**

- **records**: A data frame of records returned by `exportRecords` or `exportReports`.
- **meta_data**: A data frame giving the data dictionary, as returned by `exportMetaData`.
- **factors**: Logical, determines if checkbox, radio button, dropdown and yesno variables are converted to factors.
- **dates**: Logical, determines if date variables are converted to POSIXct format.
- **checkboxLabels**: Logical, determines if checkbox variables are labeled as "Checked" or using the checkbox label. Only applicable when `factors = TRUE`.
- **labels**: Logical. Determines if the variable labels are applied to the data frame.
- **handlers**: List, Specify type conversion overrides for specific REDCap field types. E.g., `handlers=list(date_ = as.Date)`.
- **mChoice**: logical; defaults to TRUE. Convert checkboxes to mChoice if Hmisc is installed.
- **...**: additional arguments that are ignored.

**Details**

This function is called internally by `exportRecords` and `exportReports`. It is not available to the user.

**Description**

This set of functions assists in validating that the content of fields coming from REDCap match the MetaData, allowing for a validation report to provided. The cast helpers allow for transforming the REDCap data into R data types and allowing the user to customize the end product.
Usage

isNAorBlank(x, ...)

valRx(rx)

valChoice(x, field_name, coding)

valPhone(x, field_name, coding)

valSkip(x, field_name, coding)

na_values(FUN)

castLabel(x, field_name, coding)

castLabelCharacter(x, field_name, coding)

castCode(x, field_name, coding)

castCodeCharacter(x, field_name, coding)

castRaw(x, field_name, coding)

castChecked(x, field_name, coding)

castCheckedCharacter(x, field_name, coding)

castCheckLabel(x, field_name, coding)

castCheckLabelCharacter(x, field_name, coding)

castCheckCode(x, field_name, coding)

castCheckCodeCharacter(x, field_name, coding)

castCheckForImport(checked = c("Checked", "1"))

castDpNumeric(dec_symbol = "," )

castDpCharacter(n_dec, dec_symbol = "," )

castTimeHHMM(x, field_name, coding)

castTimeMMSS(x, field_name, coding)

castLogical(x, field_name, coding)

raw_cast
default_cast_no_factor

default_cast_character

skip_validation

Arguments

x character. A vector to check.
...
Consumes anything else passed to function. I.e., field_name and coding.
rx character. The regular expression pattern to check.
field_name character(1). Name of the field(s)
coding named character vector. The defined coding from the meta data.
FUN function. A function that takes a character vector.
checked character. Values to recognize as checked in a checkbox field.
dec_symbol character(1). The symbol in the field used to denote a decimal.
n_dec integerish(1). The number of decimal places permitted by the field validation.

Format

An object of class list of length 21.
An object of class list of length 25.
An object of class list of length 25.
An object of class list of length 21.

Details

Functions passed to the na, validation, and cast parameter of exportRecordsTyped() all take the form of function(x, coding, field_name). na and validation functions are expected to return a logical vector of the same length as the column processed. Helper routines are provided here for common cases to construct these functions.

Missing Data Detection:

na_values is a helper function to create a list of functions to test for NA based on field type. Useful for bulk override of NA detection for a project. The output can be directly passed to the na parameter of exportRecordsTyped(). Missing data detection is performed ahead of validation. Data that are found to be missing are excluded from validation reports.

REDCap users may define project-level missing value codes. If such codes are defined, they can be seen in Project Setup > Additional Customizations > Missing Data Codes. They will also be displayed in the project's Codebook. Project-level missing data codes cannot be accessed via the API, meaning redcapAPI is unable to assist in determining if a project has any. The most likely symptom of project-level codes is a high frequency of values failing validation (See vignette("redcapAPI-missing-data-detection")).
Validation Functions:
isNAorBlank returns TRUE/FALSE if field is NA or blank. Helper function for constructing na overrides in exportRecordsTyped().

valRx constructs a validation function from a regular expression pattern. The function returns a TRUE/FALSE if the value matches the pattern.

valChoice constructs a validation function from a set of choices defined in the MetaData. The functions returns a TRUE/FALSE if the value matches one of the choices.

valPhone constructs a validation function for (North American) phone numbers. It removes punctuation and spaces prior to validating with the regular expression.

valSkip is a function that supports skipping the validation for a field type. It returns a TRUE value for each record, regardless of its value. Validation skipping has occasional utility when importing certain field types (such as bioportal or sql) where not all of the eventual choices are available in the project yet.

skip_validation is a list of functions that just returns TRUE for all data passed in.

Casting Functions:
castLabel constructs a casting function for multiple choice variables. The field will be cast to return the choice label (generally more human readable). castLabelCharacter is an equivalent casting function that returns a character vector instead of a factor.

castCode constructs a casting function for multiple choice variables. Similar to castLabel, but the choice value is returned instead. The values are typically more compact and their meaning may not be obvious. castCodeCharacter is an equivalent casting function that returns a character vector instead of a factor.

castRaw constructs a casting function that returns the content from REDCap as it was received. It is functionally equivalent to identity. For multiple choice variables, the result will be coerced to numeric, if possible; otherwise, the result is character vector.

castUnchecked constructs a casting function for checkbox fields. It returns values in the form of Unchecked/Checked. castUncheckedCharacter is an equivalent casting function that returns a character vector instead of a factor.

castCheckLabel and castCheckCode also construct casting functions for checkbox fields. For both, unchecked variables are cast to an empty string (""), Checked variables are cast to the option label and option code, respectively. castCheckLabelCharacter and castCheckCodeCharacter are equivalent casting functions that returns a character vector instead of a factor.

castCheckForImport is a special case function to allow the user to specify exactly which values are to be considered “Checked”. Values that match are returned as 1 and all other values are returned as 0. This is motivated by the special case where the coding on a checkbox includes "0, Option". In the resulting field checkbox___0, a coded value of 0 actually implies the choice was selected. In order to perform an import on such data, it is necessary to cast it using castCheckForImport(c("0")).

castDpNumeric is a casting function for fields that use the number_ndp_comma field type (where n is the number of decimal places). This function will convert the values to numeric values for use in analysis. This is a function that returns the appropriate casting function, thus the appropriate usage when using the defaults is cast = list(number_1dp_comma = castDpNumeric()) (using the parentheses).

castDpCharacter is a casting function to return fields that use number_ndp_comma field types to character strings for import. This is a function that returns the appropriate casting function, thus
the appropriate usage when casting for one decimal place is `cast = list(number_1dp_comma = castDpCharacter(1))`.

castTimeHHMM and castTimeMMSS are casting functions to facilitate importing data. They convert time data into a character format that will pass the API requirements.

castLogical is a casting function that returns a logical vector for common, binary-type responses. It is well suited to changing true/false, yes/no, and checkbox fields into logical vectors, as it returns TRUE if the value is one of c("1", "true", "yes") and returns FALSE otherwise.

**Casting Lists:**

`raw_cast` overrides all casting if passed as the cast parameter. It is important the the validation specified matches the chosen cast. For fully raw it should be `skip_validation`.

default_cast_no_factor is a list of casting functions that matches all of the default casts but with the exception that any fields that would have been cast to factors will instead be cast to characters. It is provided for the user that prefers to work absent factors. The list `default_cast_character` is equivalent and is provided for those that prefer to describe their casting in terms of what the result is (and not what it is not).

**Value**

Validation and casting functions return the objects indicated in the following table:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Object Type Returned</th>
</tr>
</thead>
<tbody>
<tr>
<td>isNAOrBlank</td>
<td>logical</td>
</tr>
<tr>
<td>valRx</td>
<td>logical</td>
</tr>
<tr>
<td>valChoice</td>
<td>logical</td>
</tr>
<tr>
<td>valPhone</td>
<td>logical</td>
</tr>
<tr>
<td>valSkip</td>
<td>logical</td>
</tr>
<tr>
<td>castLabel</td>
<td>factor</td>
</tr>
<tr>
<td>castLabelCharacter</td>
<td>character</td>
</tr>
<tr>
<td>castCode</td>
<td>factor</td>
</tr>
<tr>
<td>castCodeCharacter</td>
<td>character</td>
</tr>
<tr>
<td>castRaw</td>
<td>character</td>
</tr>
<tr>
<td>castChecked</td>
<td>factor</td>
</tr>
<tr>
<td>castCheckedCharacter</td>
<td>character</td>
</tr>
<tr>
<td>castCheckLabel</td>
<td>factor</td>
</tr>
<tr>
<td>castCheckLabelCharacter</td>
<td>character</td>
</tr>
<tr>
<td>castCheckCode</td>
<td>factor</td>
</tr>
<tr>
<td>castCheckCodeCharacter</td>
<td>character</td>
</tr>
<tr>
<td>castCheckForImport</td>
<td>numeric</td>
</tr>
<tr>
<td>castDpNumeric</td>
<td>numeric</td>
</tr>
<tr>
<td>castDpCharacter</td>
<td>character</td>
</tr>
<tr>
<td>castTimeHHMM</td>
<td>character</td>
</tr>
<tr>
<td>castTimeMMSS</td>
<td>character</td>
</tr>
<tr>
<td>castLogical</td>
<td>logical</td>
</tr>
</tbody>
</table>

**See Also**

`fieldCastingFunctions()`,
exportRecordsTyped(),
exportReportsTyped()

Vignettes:
  vignette("redcapAPI-casting-data")
  vignette("redcapAPI-missing-data-detection")
  vignette("redcapAPI-data-validation")
  vignette("redcapAPI-faq")

Examples

## Not run:
# Make a custom function to give special treatment to a field.
# In this function, the field "field_name_to_skip" will
# be cast using 'castRaw'. All other fields will be cast
# using 'castCode'
customCastCode <- function(x, field_name, coding){
  if (field_name == "field_name_to_skip"){
    castRaw(x, field_name, coding)
  } else {
    castCode(x, field_name, coding)
  }
}

## End(Not run)

---

**fileMethods**

**Export, Import, or Delete Files to a Field in a REDCap Project**

**Description**

These methods enable to the user to export a file stored in a project field, import a file, or delete an existing file.

**Usage**

```r
exportFiles(
  rcon,
  record,
  field,
  event = NULL,
  dir = getwd(),
  file_prefix = TRUE,
  ...
)
```

```r
importFiles(
  rcon,
```
fileMethods

```r
deleteFiles(rcon, record, field, event, ...

# S3 method for class 'redcapApiConnection'
exportFiles(
  rcon, record, field, event = NULL, dir = getwd(),
  file_prefix = TRUE, repeat_instance = NULL, ...
  error_handling =getOption("redcap_errorHandling"),
  config = list(),
  api_param = list()
)

# S3 method for class 'redcapApiConnection'
importFiles(
  rcon, file, record, field, event = NULL, overwrite = TRUE,
  repeat_instance = NULL, ...
  error_handling =getOption("redcap_errorHandling"),
  config = list(),
  api_param = list()
)

# S3 method for class 'redcapApiConnection'
deleteFiles(
  rcon, record = NULL, field = NULL, event = NULL,
  repeat_instance = NULL, ...
)```
error_handling = getOption("redcap_error_handling"),
config = list(),
api_param = list()
)

Arguments

**rcon**
A redcapConnection object.

**record**
character(1) or integerish(1). The record ID in which the desired file is stored.

**field**
character(1). The field name in which the file is stored.

**event**
character(1) or NULL. The event name for the file. This applies only to longitudinal projects. If the event is not supplied for a longitudinal project, the API will return an error message

**repeat_instance**
integerish(1) or NULL. The repeat instance number of the repeating event or the repeating instrument. When available in your instance of REDCap, and passed as NULL, the API will assume a value of 1.

**file**
character(1). The file path to the file to be imported.

**overwrite**
logical(1). When FALSE, the function checks if a file already exists for that record. If a file exists, the function terminates to prevent overwriting. When TRUE, no additional check is performed.

**dir**
character(1). A directory/folder to which the file will be saved. By default, the working directory is used.

**file_prefix**
logical(1). Determines if a prefix is appended to the file name. The prefix takes the form [record_id]-[event_name]-[file_name]. The file name is always the same name of the file as it exists in REDCap.

... Arguments to pass to other methods

**error_handling**
character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

**config**
A named list. Additional configuration parameters to pass to httr::POST(). These are appended to any parameters in rcon$config.

**api_param**
A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

These functions only export, import, or delete a single file.

When exporting, the file name cannot be changed; whatever name exists in REDCap is the name that will be used. The record ID and event name may be appended as a prefix.
**Value**

exportFiles invisibly returns the file path to which the exported file was saved.

importFiles invisibly returns TRUE when successful, or throws an error if the import failed.

deleteFiles invisible returns TRUE when successful, or throws an error if the deletion failed.

**Functions**

- exportFiles(): Export a file from a REDCap project.
- importFiles(): Import a file to a REDCap project.
- deleteFiles(): Delete a file from a REDCap project.

**See Also**

exportFilesMultiple(), importFileToRecord() (can create a record to receive the file if it does yet exist)

**Examples**

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

save_to_dir <- tempdir()

# Export a file
exportFiles(rcon,
record = 1,
field = "file_upload_test",
dir = save_to_dir)

# Export a file for a specific event
exportFiles(rcon,
record = 1,
field = "file_upload_test",
event = "event_1_arm_1",
dir = save_to_dir)

# Import a file
importFiles(rcon,
file = "file_to_upload.txt"
record = 1,
field = "file_upload_test")

# Delete a file
deleteFiles(rcon,
record = 1,
field = "file_upload_test")
```
fileRepositoryMethods  Export, Import, or Delete Multiple Files from the File Repository

Description

These methods enable the user to export, import, or delete entire folders of files from the file repository. These actions may be done recursively to include subfolders as well.

Usage

exportFileRepository(  
rcon,  
folder_id,  
dir = getwd(),  
dir_create = FALSE,  
recursive = FALSE,  
...  
)

importFileRepository(rcon, dir, folder_id = numeric(0), ...)  

deleteFileRepository(rcon, folder_id, recursive = FALSE, ...)

## S3 method for class 'redcapApiConnection'
exportFileRepository(  
rcon,  
folder_id = numeric(0),  
dir = getwd(),  
dir_create = FALSE,  
recursive = FALSE,  
...,
  error_handling = getOption("redcap_error_handling"),  
  config = list(),  
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
importFileRepository(  
rcon,  
dir,  
folder_id = numeric(0),  
dag_id = numeric(0),  
role_id = numeric(0),  
recursive = FALSE,
## S3 method for class 'redcapApiConnection'
deleteFileRepository(
  rcon,
  folder_id,
  recursive = FALSE,
  ..., confirm = c("ask", "no", "yes"),
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

### Arguments

- **rcon**: A `redcapConnection` object.
- **folder_id**: `integerish(0/1)`. The folder ID with the files to download. If length 0, defaults to the top-level directory.
- **dir**: `character(1)`. A directory on the local system to which the files are to be saved. Defaults to the working directory.
- **dir_create**: `logical(1)`. When `TRUE` create the directory `dir` if it does not already exist. Defaults to `FALSE`. If `dir` does not exist and `dir_create = FALSE`, an error is thrown.
- **dag_id**: `integerish(0/1)`. The ID of a data access group. When provided, access to the folder will be restricted to the DAG.
- **role_id**: `integerish(0/1)`. The ID of a role. When provided, access to the folder will be restricted to users with that role.
- **recursive**: `logical(1)`. When `TRUE`, export all subfolders and their files as well.
- **confirm**: `character`. One of `c("ask", "no", "yes")`. When "ask", user will be prompted to confirm the deletion. When "no", the function will terminate with no action. When "yes", the function will proceed without confirmation (useful for automated processes).
- **...**: Arguments to pass to other methods
- **error_handling**: `character(1)`. One of `c("error", "null")`. An option for how to handle errors returned by the API. see `redcapError()`.
- **config**: A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.
- **api_param**: A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by `redcapAPI`.
Details

deleteFileRepository will only delete files and cannot delete folders.
Deleted files will remain in the recycling bin for up to 30 days.

Value

elexportFileRepository returns a data frame with the locations to which the files were saved on
the local system. It has the columns:

- directory   The directory in which the file is saved.
- filename    The name of the saved file.

importFileRepository returns a data frame with the locations to which the files were saved on
the local system. It has the columns:

- directory   The directory in which the file is saved.
- filename    The name of the saved file.

deleteFileRepository returns a data frame listing the files that were deleted from the file repository. It has the columns:

- folder_id   The REDCap assigned ID number for the folder. This will be NA for files.
- doc_id       The REDCap assigned ID number for the file.
- name         The filename of the deleted files.
- parent_folder The folder ID of parent folder.

Functions

- `exportFileRepository()`: Export multiple files from the File Repository.
- `importFileRepository()`: Import multiple files to the File Repository.
- `deleteFileRepository()`: Delete multiple files from the File Repository.

See Also

- `exportFromFileRepository()`.
- `importToFileRepository()`.
- `deleteFromFileRepository()`.
- `exportFileRepositoryListing()`.
- `createFileRepositoryFolder()`
Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

save_location <- tempdir()

# Export the top-level file repository folder
exportFileRepository(rcon,
    folder_id = numeric(0),
    dir = save_location)

# Export the entire repository
exportFileRepository(rcon,
    folder_id = numeric(0),
    dir = save_location,
    recursive = TRUE)

# Export a file repository folder below the top-level
exportFileRepository(rcon,
    folder_id = 12345,
    dir = save_location)

# Import the files from a folder to the top-level file repository
importFileRepository(rcon,
    dir = "path/to/folder")

# Import the files from a folder to sub folder of the file repository
importFileRepository(rcon,
    dir = "path/to/folder",
    folder_id = 12345)

# Import the files from a folder and assign to a specific Data Access Group
importFileRepository(rcon,
    dir = "path/to/folder",
    dag_id = 789)

# Delete files from the top-level folder of the file repository
deleteFileRepository(rcon,
    folder_id = numeric(0))

# Delete all the files from the file repository
deleteFileRepository(rcon,
    folder_id = numeric(0),
    recursive = TRUE)
```

## End(Not run)
fileRepositoryPath  

Reconstruct the file repository path

Description
Reconstruct the file repository path

Usage
fileRepositoryPath(doc_id = numeric(0), folder_id = numeric(0), fileRepo)

Arguments
- doc_id: integerish(0/1). The document ID for which the file path should be returned. Only one of doc_id or folder_id should be specified.
- folder_id: integerish(0/1). The folder ID for which the file path should be returned. Only one of doc_id or folder_id should be specified.
- fileRepo: data.frame with the file repository listing. Typically provided by rcon$fileRepository()

filterEmptyRow  

Remove Rows Containing Only Missing Values

Description
Evaluates each row of a data frame for missingness. If all fields (excluding the identifying fields) are missing, the row is removed from the data. For the purpose of this function, redcap_data_access_group is considered an identifying field.

Usage
filterEmptyRow(data, rcon)

Arguments
- data: A data.frame to be filtered.
- rcon: A redcapConnection object.

See Also
exportRecordsTyped(), exportReportsTyped()
Description

These methods enable the user to export, import, or delete individual files from a REDCap project’s file repository.

Usage

```r
exportFromFileRepository(rcon, doc_id, dir = getwd(), dir_create = FALSE, ...)  
importToFileRepository(rcon, file, folder_id = numeric(0), ...)  
deleteFromFileRepository(rcon, doc_id, ...)
```

```r
## S3 method for class 'redcapApiConnection'
exportFromFileRepository(
  rcon,
  doc_id,
  dir = getwd(),
  dir_create = FALSE,
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
```

```r
## S3 method for class 'redcapApiConnection'
importToFileRepository(
  rcon,
  file,
  folder_id = numeric(0),
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
```

```r
## S3 method for class 'redcapApiConnection'
deleteFromFileRepository(
  rcon,
  doc_id,
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
)
api_param = list()

Arguments

rcon               A redcapConnection object.
doc_id             integerish(1). The document ID to be downloaded.
folder_id          integerish(0/1). The ID of the folder into which the file is to be imported. If
                   length is zero, it is imported to the top-level folder.
file               character(1). A file on the local system to be imported to the File Repository.
dir                character(1). A directory on the local system to which the file is to be saved.
                   Defaults to the working directory.
dir_create         logical(1). Create the directory dir if it does not already exist. Defaults to
                   FALSE. If dir does not exist and create = FALSE, an error is thrown.
...                 Arguments to pass to other methods
error_handling     character(1). One of c("error", "null"). An option for how to handle
                   errors returned by the API. see redcapError().
config             A named list. Additional configuration parameters to pass to http::POST().
                   These are appended to any parameters in rcon$config.
api_param          A named list. Additional API parameters to pass into the body of the API
                   call. This provides users to execute calls with options that may not otherwise be
                   supported by redcapAPI.

Details

When a file is deleted, the file will remain in the Recycle Bin folder for up to 30 days.

Value

exportFromFileRepository, importToFileRepository, and deleteFromFileRepository each
return a data frame with the columns:

   directory The directory in which the file is saved.
   filename  The name of the saved file.

Functions

• exportFromFileRepository(): Export a file from the file repository.
• importToFileRepository(): Import a file to the file repository.
• deleteFromFileRepository(): Delete a file from the file repository.

See Also

exportFileRepository().
getProjectIdFields

Return a vector of the Project ID Fields

Description

Returns a character vector listing the project ID fields. This will be at most a vector of length two. The first element will be the first field in the meta data. The second, if provided, will be the name of the secondary unique field specified in the project.
importFileToRecord

Usage

getProjectIdFields(rcon)

Arguments

rcon  A redcapConnection object.

Value

Returns a character vector with the field names that uniquely identify an experimental unit.

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

getProjectIdFields(rcon)

## End(Not run)

importFileToRecord  Import a File With Option to Create A Record to Receive the File

Description

There are times when the user may desire to create a record and import a file as part of a single action. For example, a study consent form may have been collected and needs to be stored with the data of the new study participant. importFileToRecord extends importFiles to allow the concurrent creation of the record in which the file will be stored.

Usage

importFileToRecord(
  rcon,
  file,
  record = NULL,
  field,
  event,
  overwrite = TRUE,
  repeat_instance = NULL,
  ...)
)
importFileToRecord

Arguments

- **rcon**: A redcapConnection object.
- **file**: character(1). The file path to the file to be imported.
- **record**: character(1) or integerish(1) or NULL. The record ID in which the desired file is stored. When NULL, an attempt will be made to create a new record for the file. See 'Details'
- **field**: character(1). The field name in which the file is stored.
- **event**: character(1) or NULL. The event name for the file. This applies only to longitudinal projects. If the event is not supplied for a longitudinal project, the API will return an error message
- **overwrite**: logical(1). When FALSE, the function checks if a file already exists for that record. If a file exists, the function terminates to prevent overwriting. When TRUE, no additional check is performed.
- **repeat_instance**: integerish(1) or NULL. The repeat instance number of the repeating event or the repeating instrument. When available in your instance of REDCap, and passed as NULL, the API will assume a value of 1.

... Arguments to pass to other methods

Details

The behavior of importFileToRecord depends on

1. whether record auto numbering has been enabled in the project,
2. if the record is specified by the user
3. if the record specified by the user exists.

The following table details the actions taken based on these conditions. (force_auto_number is an argument to importRecords()).

<table>
<thead>
<tr>
<th>Autonumbering enabled</th>
<th>record</th>
<th>Record Exists</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>NULL</td>
<td>No</td>
<td>Create a new record (using force_auto_number = TRUE) and import the file</td>
</tr>
<tr>
<td>Yes</td>
<td>Specified</td>
<td>Yes</td>
<td>Import the file to the existing record</td>
</tr>
<tr>
<td>No</td>
<td>Specified</td>
<td>No</td>
<td>Create a new record (using force_auto_number = TRUE) and import the file</td>
</tr>
<tr>
<td>No</td>
<td>NULL</td>
<td>No</td>
<td>Error: record must be provided when auto numbering is not enabled</td>
</tr>
<tr>
<td>No</td>
<td>Specified</td>
<td>Yes</td>
<td>Import the file to the existing record</td>
</tr>
<tr>
<td>No</td>
<td>Specified</td>
<td>No</td>
<td>Create the record (using force_auto_number = FALSE) and import the file</td>
</tr>
</tbody>
</table>

See Also

importFiles(), importRecords()

Examples

```r
## Not run:
```
importRecords

unlockREDCap(connections = c(rcon = "project_alias"),
    url = "your_redcap_url",
    keyring = "API_KEYs",
    envir = globalenv())

importFileToRecord(rcon,
    file = "file_to_upload.txt"
    record = NULL,
    field = "file_upload_test")

## End(Not run)

importRecords

Import Records to a Project

Description

These methods enable the user to import new records or update existing records to a project.

Usage

importRecords(
    rcon,
    data,
    overwriteBehavior = c("normal", "overwrite"),
    returnContent = c("count", "ids", "nothing", "auto_ids"),
    returnData = FALSE,
    logfile = "",
    ...
)

## S3 method for class 'redcapApiConnection'
importRecords(
    rcon,
    data,
    overwriteBehavior = c("normal", "overwrite"),
    returnContent = c("count", "ids", "nothing", "auto_ids"),
    returnData = FALSE,
    logfile = "",
    force_auto_number = FALSE,
    ...
    batch.size = -1,
    error_handling = getOption("redcap_error_handling"),
    config = list(),
    api_param = list()
)
Arguments

**rcon**  A redcapConnection object.

**data**  A data.frame to be imported to the project.

**overwriteBehavior**  character(1). One of c("normal", "overwrite"). "normal" prevents blank fields from overwriting populated fields. "overwrite" causes blanks to overwrite data in the database.

**returnContent**  character(1). One of c("count", "ids", "nothing", "auto_ids"). 'count' returns the number of records imported; 'ids' returns the record ids that are imported; 'nothing' returns no message; 'auto_ids' returns a list of pairs of all record IDs that were imported. If used when force_auto_number = FALSE, the value will be changed to 'ids'.

**returnData**  logical(1). When TRUE, prevents the REDCap import and instead returns the data frame that would have been given for import. This is sometimes helpful if the API import fails without providing an informative message. The data frame can be written to a csv and uploaded using the interactive tools to troubleshoot the problem.

**logfile**  character(1). An optional filepath (preferably .txt) in which to print the log of errors and warnings about the data. When "", the log is printed to the console.

**force_auto_number**  logical(1). If record auto-numbering has been enabled in the project, it may be desirable to import records where each record’s record name is automatically determined by REDCap (just as it does in the user interface). When TRUE, the record names provided in the request will not be used (although they are still required in order to associate multiple rows of data to an individual record in the request); instead those records in the request will receive new record names during the import process. It is recommended that the user use returnContent = "auto_ids" when force_auto_number = TRUE.

**batch.size**  integerish(1). Specifies the number of subjects to be included in each batch of a batched export or import. Non-positive numbers export/import the entire operation in a single batch. Batching may be beneficial to prevent tying up smaller servers. See Details.

**error_handling**  character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

**config**  A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.

**api_param**  A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

*importRecords* prevents the most common import errors by testing the data before attempting the import. Namely
importRecords

1. Check that all variables in data exist in the REDCap data dictionary.
2. Check that the record id variable exists
3. Force the record id variable to the first position in the data frame (with a warning)
4. Remove calculated fields (with a warning)
5. Verify that REDCap date fields are represented in the data frame as either character, POSIXct, or Date class objects.
6. Determine if values are within their specified validation limits.

See the documentation for validateImport() for detailed explanations of the validation.

A ‘batched’ import is one where the export is performed over a series of API calls rather than one large call. For large projects on small servers, this may prevent a single user from tying up the server and forcing others to wait on a larger job.

**BioPortal Fields:**

Text fields that are validation enabled using the BioPortal Ontology service may be imported by providing the coded value. Importing the coded value does not, however, guarantee that the labeled value will be immediately available. Labels for BioPortal values are cached on the REDCap server in a process that occurs when viewing data in the user interface. Thus, if the label has not be previously cached on the server, the code will be used to represent both the code and the label.

**Value**

importRecords, when returnData = FALSE, returns the content from the API response designated by the returnContent argument.

importRecords, when returnData = TRUE, returns the data frame that was internally prepared for import. This data frame has values transformed from R objects to character values the API will accept.

**See Also**

exportRecords(),
deleteRecords(),
exportRecordsTyped()

**Examples**

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

# Import records
NewData <- data.frame(record_id = c(1, 2, 3),
                      age = c(27, 43, 32),
                      date_of_visit = rep(Sys.Date(), 3))
importRecords(rcon,
              data = NewData)
```
# Import records and save validation info to a file
NewData <- data.frame(record_id = c(1, 2, 3),
                        age = c(27, 43, 32),
                        date_of_visit = rep(Sys.Date(), 3))
importRecords(rcon,
              data = NewData,
              logfile = "import-validation-notes.txt")

## End(Not run)

invalidSummary

**Helper functions for formatting validation failure report**

**Description**

`exportRecordsTyped()` may have an `invalid` attribute if validations fail. This data has some routines which help locate the failing records.

**Usage**

```r
## S3 method for class 'invalid'
format(x, ...)
```

```r
## S3 method for class 'invalid'
print(x, ...)
```

```r
## S3 method for class 'invalid'
summary(object, ...)
```

**Arguments**

- `x` The `invalid` class object.
- `...` additional arguments to `print`
- `object` The `invalid` class object.

**Examples**

```r
## Not run:
rec <- exportRecordsTyped(rcon)
attr(rec, "invalid")
## End(Not run)
```
isZeroCodedCheckField  Identify Check Fields with a Zero Coded Option

Description
Check fields that have 0 as a coding option can confuse certain data processing steps because it can be difficult to differentiate if a 0 value represents an unchecked or checked option. Identifying these fields is important to handling them correctly.

Usage

\[
\text{isZeroCodedCheckField}(\text{field\_name})
\]
\[
\text{warnOfZeroCodedCheckCasting}(\text{field\_name}, x)
\]
\[
\text{warnZeroCodedFieldPresent}(\text{field\_names}, \text{warn\_zero\_coded})
\]

Arguments
- field_name character(1) The name of a field to be tested.
- x atomic object.
- field_names character vector of field names.
- warn_zero_coded logical(1). Turn on or off warnings about zero coded fields. Defaults to TRUE.

Value

isZeroCodedCheckField returns a logical(1)
warnOfZeroCodedCheckCasting has no return and issues a warning if the field name appears to be zero-coded.
warnZeroCodedFieldPresent has no return and issues a warning if any of the fields passed appear to be zero-coded.

Zero-Coded Check Fields

A zero-coded check field is a field of the REDCap type checkbox that has a coding definition of 0, [label]. When exported, the field names for these fields is [field\_name]___0. As in other checkbox fields, the raw data output returns binary values where 0 represent an unchecked box and 1 represents a checked box. For zero-coded checkboxes, then, a value of 1 indicates that 0 was selected.

This coding rarely presents a problem when casting from raw values (as is done in exportRecordsTyped). However, casting from coded or labeled values can be problematic. In this case, it becomes indeterminate from context if the intent of 0 is ‘false’ or the coded value ‘0’ (‘true’) ...

The situations in which casting may fail to produce the desired results are

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
<th>Result</th>
</tr>
</thead>
</table>


Because of the potential for miscast data, casting functions will issue a warning anytime a zero-coded check field is encountered. A separate warning is issued when a field is cast from coded or labeled values.

When casting from coded or labeled values, it is strongly recommended that the function `castCheckForImport()` be used. This function permits the user to state explicitly which values should be recognized as checked, avoiding the ambiguity resulting from the coding.

**Examples**

```r
## Not run:
isZeroCodedCheckField("check_field___x")

isZeroCodedCheckField("check_field___0")

x <- factor(c(1, 0, 1, 0, 0),
           levels = 0:1)
warnOfZeroCodedCheckCasting(field_name = "check_field___0",
                          x = x)

warnZeroCodedFieldPresent(c("check_field___x", "check_field___0"), TRUE)

## End(Not run)
```

### Description

Constructs and executes API calls to the REDCap API. These are left deliberately abstract in order to be flexible enough to support the redcapAPI functions, but also allow users to execute calls for new REDCap features that are not yet implemented.

### Usage

```r
makeApiCall(rcon, body = list(), config = list(), url = NULL)
```

### Arguments

- **rcon**: A redcapConnection object.
- **body**: list List of parameters to be passed to `httr::POST()`’s body argument
- **config**: list A list of options to be passed to `httr::POST()`. These will be appended to the config options included in the rcon object.
- **url**: character(1) A url string to hit. Defaults to rcon$url.
Details

The intent of this function is to provide an approach to execute calls to the REDCap API that is both consistent and flexible. Importantly, this provides a framework for making calls to the API using features that the R package does not yet support (redcapAPI will always lag behind when REDCap adds new features).

The API call consists of two components: the "body" and the "config." The body of the call contains all of the arguments being passed to the API. When building body components, be sure to review the documentation. options to the API that require an array need to be built using vectorToApiBodyList; options that are not an array can be entered directly (see examples).

The config list is a list of parameters to pass to `httr::POST()`. Refer to documentation there for details.

Using the settings stored in the redcapConnection object, a response code of 408 (Request Time-out), 500 (Internal Server Error), 502 (Bad Gateway), 503 (Service Unavailable), or 504 (Gateway Timeout) will prompt reattempts at calling the API. See `redcapConnection()` for details. If the API reaches its attempt limit without resolving to any other code, the last response is returned. If any other response code is returned at any point in the retry loop, the loop breaks and returns that response.

Examples

```r
## Not run:
url <- "Enter your API URL here"
token <- "Enter your API token here"

rcon <- redcapConnection(url = url,
token = token)

MetaData <-
makeApiCall(rcon = rcon,
body = list(content = "metadata",
format = "csv",
returnFormat = "csv"))

MetaData <-
utils::read.csv(text = as.character(MetaData),
stringsAsFactors = FALSE,
na.strings = "")

# Call to export Meta Data (Data Dictionary) for specific fields

fields <- vectorToApiBodyList(vector = c("row_purpose",
"prereq_radio"),
parameter_name = "fields")

MetaData <-
makeApiCall(rcon = rcon,
body = c(list(content = "metadata",
format = "csv",
returnFormat = "csv"),
fields))

MetaData <- read.csv(text = as.character(MetaData),
stringsAsFactors = FALSE,
na.strings = "")
```
# Basic call to export records

```r
Records <- makeApiCall(rcon = rcon,
                        body = list(content = "record",
                                    format = "csv",
                                    returnFormat = "csv",
                                    type = "flat"))
Records <- read.csv(text = as.character(Records),
                    stringsAsFactors = FALSE,
                    na.strings = "")
```

# Call to export records for a single form.
# Note that even though we are interested in a single form, the
# API requires an array, so we use vectorToApiBodyList

```r
export_form <- vectorToApiBodyList("branching_logic",
                                    parameter_name = "forms")
Records <- makeApiCall(rcon = rcon,
                        body = c(list(content = "record",
                                      format = "csv",
                                      returnFormat = "csv",
                                      type = "flat"),
                                 export_form))
Records <- read.csv(text = as.character(Records),
                    stringsAsFactors = FALSE,
                    na.strings = "")
```

# Call to export records with a pipe delimiter.

```r
Records <- makeApiCall(rcon = rcon,
                        body = list(content = "record",
                                     format = "csv",
                                     returnFormat = "csv",
                                     type = "flat",
                                     csvDelimiter = "|")
Records <- read.csv(text = as.character(Records),
                    stringsAsFactors = FALSE,
                    na.strings = "",
                    sep = "|")
```

# Call to export records created/modified after 25 Dec 2022 14:00.

```r
Records <- makeApiCall(rcon = rcon,
                        body = list(content = "record",
                                     format = "csv",
                                     returnFormat = "csv",
                                     type = "flat")
Records <- read.csv(text = as.character(Records),
                    stringsAsFactors = FALSE,
                    na.strings = "",
                    sep = "|")
```
mappingMethods

Export and Import Instrument-Event Mappings

Description

These methods enable the user to export and add/modify the mappings between instruments and events. The information provided with the methods corresponds to what is provided in the 'Designate Instruments for My Events' page in the user interface.

Usage

exportMappings(rcon, arms, ...)
importMappings(rcon, data, ...)

## S3 method for class 'redcapApiConnection'
exportMappings(
  rcon,
  arms = NULL,
  ...,  
  error_handling =getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
importMappings(
  rcon,
  data,
  ...,  
  error_handling =getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

format = "csv",
returnFormat = "csv",
type = "flat",
dateRangeBegin = "2022-12-25 14:00:00")

Records <- read.csv(text = as.character(Records),
  stringsAsFactors = FALSE,
  na.strings = "")

## End(Not run)
mappingMethods

Arguments

rcon  A redcapConnection object.
arms  integerish or character. A vector of arm numbers. When given, mappings are only exported for the given arms.
data  data.frame with columns arm_num, unique_event_name, and form. See Details
...  Arguments to pass to other methods
error_handling  character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().
config  A named list. Additional configuration parameters to pass to http::POST(). These are appended to any parameters in rcon$conf.
api_param  A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

These methods are only applicable to longitudinal projects. If the project information reports that the project is not longitudinal, a data frame with 0 rows is returned without calling the API.

Value

exportMappings returns a data frame with the columns:

<table>
<thead>
<tr>
<th>arm_num</th>
<th>The arm number for the unique event mapped to the instrument.</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_event_name</td>
<td>The unique event name to which the instrument is assigned.</td>
</tr>
<tr>
<td>form</td>
<td>The REDCap assigned instrument name mapped to the event.</td>
</tr>
</tbody>
</table>

importMappings invisible returns the number of mappings added or edited.

Functions

• exportMappings(): Export instrument-event mappings.
• importMappings(): Import instrument-event mappings to the project.

See Also

exportFieldNames(), exportInstruments(), exportMetaData(), importMetaData(), exportPdf()
Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

# Export all mappings
exportMappings(rcon)

# Export mappings for a specific arm
exportMappings(rcon,
               arms = 1)

# Import mappings
NewMapping <-
  data.frame(arm_num = c(1, 1, 2),
             unique_event_name = c("event_1_arm_1",
                                   "event_2_arm_1",
                                   "event_1_arm_2"),
             form = c("registration",
                      "follow_up",
                      "registration"))

importMapping(rcon,
              data = NewMapping)

## End(Not run)
```

massert

Conduct Multiple Assertions

Description

This documentation attempts to describe arguments to make assertions on arguments. In order to prevent confusion, it is imperative to develop some terminology up front. We will use function argument to refer to an argument of the function for which we are conducting assertions. We will use assertion argument to refer to arguments to pass to the assertion function being applied to a function argument. Lastly, we will use massert argument to refer to arguments to massert

Usage

```r
massert(formula, fun, ..., fixed = list())
```

Arguments

- `formula`: A one sided formula naming the arguments on which the assertion will be performed.
fun

Additional lists. Each argument provided is a named list of assertion arguments. The name of each element in a list should match the name of a function argument. lower = list(var1 = 0, var2 = 10) sets the assertion argument lower = 0 for function argument var1, and sets the assertion argument lower = 10 for function argument var2. The massert arguments in ... may themselves be named or unnamed.

fixed

A named list of arguments that are fixed across all assertions.

Details

Only one assert function may be utilized in each call to massert. This allows for all numeric variables to be checked in one call, all logical variables to be checked in a subsequent call, etc.

metaDataMethods

Export and Import the Project Meta Data (Data Dictionary)

Description

These methods provide the user access to a REDCap project’s data dictionary. The data dictionary may be exported or altered via the import.

Usage

exportMetaData(rcon, ...)

importMetaData(rcon, data, ...)

## S3 method for class 'redcapApiConnection'
exportMetaData(
  rcon,
  fields = character(0),
  forms = character(0),
  ...,
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
importMetaData(
  rcon,
  data,
  ...,
  field_types = REDCAP_METADATA_FIELDTYPE,
  validation_types = REDCAP_METADATA_VALIDATION_TYPE,
  error_handling = getOption("redcap_error_handling"),
  ...,
config = list(),
api_param = list()
)

Arguments

rcon A redcapConnection object.
fields character vector of field names for which the metadata is to be retrieved.
forms character vector of forms for which the metadata is to be retrieved. If a form
name is given, all of the fields on that form will be returned, regardless of
whether it is included in fields or not. Form names should match those in
the second column of the data dictionary, and not the display names shown on
the web interface.
data data.frame with the Meta Data to import.
... Arguments to pass to other methods
field_types character giving the acceptable field types when validating the field_type
column. This
validation_types character giving the acceptable values for the text_validation_or_show_slider_number
column.
error_handling character(1). One of c("error", "null"). An option for how to handle
errors returned by the API. see redcapError()
config A named list. Additional configuration parameters to pass to http::POST().
These are appended to any parameters in rcon$config.
api_param A named list. Additional API parameters to pass into the body of the API
call. This provides users to execute calls with options that may not otherwise be
supported by redcapAPI.

Details

When importing meta data, the following conditions apply:
Field names may start with a letter, have any number of letters, numbers, or underscores, and end
in either a letter or a number. All letters must be lowercase (the function will coerce them to lower
before checking for duplicate field names).
Form names may start with a letter, have any number of letters, numbers, or underscores, and end
in either a letter or a number. All letters must be lowercase (the function will coerce them to lower
before checking for duplicate field names).
Field types may be one of REDCAP_METADATAFIELDTYPE. In the event that a new field type is
added to REDCap and redcapAPI is not yet updated, the user may add additional values via
c(REDCAP_METADATAFIELDTYPE, "new_type").
Validationtypes may be one of REDCAP_METADATAVALIDATION_TYPE or NA. As with field types,
additional values can be appended if necessary. Only fields that have a field type of "text" or "slider"
should have a validation type. "slider" fields should be either NA (do not display the selected number)
or "number".
For multiple choice fields, the selection choices take the format of "code1, label1 | ... | coden, labeln". For slider fields, the format is "left_value | mid_value | right_value". Any of those values may be an empty character, but the two pipes are required, nonetheless.

For calculated fields, the values in "select_choices_or_calculations" are currently unvalidated.

All of the values between brackets in the branching logic must be either a field name or an existing unique event name (such as "event_1_arm_1")

Value

exportMetaData returns a data frame. Not all 18 (or more) columns are documented here, but the most commonly used within redcapAPI are (these may appear in a different order in the data frame):

- field_name: The name of a field in the project.
- filed_label: The human-readable form of the field name.
- form_name: The name of the form on which the field is found.
- field_type: One of two fields used to determine how a field is transformed into an R object.
- select_choices_or_calculations: The second field used to determine how a field is translated into an R object.
- text_validation_type_or_show_slider_number: Describes how fields are validated. For slider fields, it gives the limits and center point to display.
- field_annotation: Contains annotations such as units of measures. Also contains action tags.

importMetaData invisibly returns the number of fields that were imported.

Functions

- exportMetaData(): Export the Meta Data (Data Dictionary) of a REDCap Project
- importMetaData(): Import New Meta Data (Data Dictionary) Definitions

See Also

exportFieldNames(), exportInstruments(), exportMappings(), importMappings(), exportPdf()

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYs",
              envir = globalenv())

# Export the MetaData from REDCap
exportMetaData(rcon)
```
# Export MetaData for select fields only (returns two rows)
exportMetaData(rcon,
    fields = c("dropdown_test", "radio_test"))

# Export MetaData for select forms
exportMetaData(rcon,
    forms = c("first_form", "second_form"))

# MetaData my be exported for a combination of fields and forms
exportMetaData(rcon,
    fields = c("dropdown_test", "radio_test"),
    forms = c("first_form", "second_form"))

# Alter and import new MetaData (change the record ID label)
Meta <- exportMetaData(rcon)

Meta$field_label[1] <- "A better description of the Record ID"
importMetaData(rcon,
    data = Meta)

## End(Not run)

---

missingSummary  Report of Missing Values

**Description**

Returns a data frame of subject events with missing values.

**Usage**

missingSummary(rcon, excludeMissingForms = TRUE, ...)

## S3 method for class 'redcapApiConnection'
missingSummary(rcon, excludeMissingForms = TRUE, ...)

missingSummary_offline(records, meta_data, excludeMissingForms = TRUE)

**Arguments**

- **rcon**  
  A redcapConnection object.

- **excludeMissingForms**  
  logical(1) When TRUE, forms where all fields are missing are assumed to be deliberately missing data and are excluded from the count of missing values. An example when this is desirable is if a patient did not experience an adverse event; the adverse event form would contain no data and the empty fields should not be considered missing data.
parseBranchingLogic

Description

Branching logic from the REDCap Data Dictionary is parsed into R Code and returned as expressions. These can be evaluated if desired and allow the user to determine if missing values are truly missing or not required because the branching logic prevented the variable from being presented.
prepUserImportData

Prepare User Data for Import

Description

Prepares a data frame for import via the API. Allows for data to be passed in either the raw format or the labeled data received from exportUsers.

Usage

prepUserImportData(data, rcon, consolidate = TRUE, user_role = FALSE)
prepUserImportData

Arguments

data: data.frame with the structure of `redcapAPI:::REDCAP_USER_STRUCTURE`. It may also have additional columns for the form and export access of each of the instruments.

rcon: A `redcapConnection` object.

consolidate: logical(1) If TRUE, the form and data export access values will be read from the expanded columns. Otherwise, the consolidated values (as provided by the API export) are utilized.

user_role: logical(1) If TRUE, the code will treat the data as if it is being prepared for importing User Roles.

Value

Returns a data.frame with user settings that will be accepted by the API for import.

See Also

`importUsers()`, `importUserRoles()`

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Prep user data
NewData <- data.frame(username = "target_user",
                      design = 1,
                      api_export = "Access",
                      api_import = "No Access",
                      surveys_enabled = 0)
prepUserImportData(data = NewData,
                    rcon = rcon)

# Prep user role data
NewData <- data.frame(unique_role_name = "target_user",
                      design = 1,
                      api_export = "Access",
                      api_import = "No Access",
                      surveys_enabled = 0)
prepUserImportData(data = NewData,
                    rcon = rcon)

## End(Not run)
```
**Preserve Project Data Locally**

**Description**

The methods enable the user to export a project data and meta data into local memory. For convenience, options are provided to save the objects to files on the local machine. Files may be saved as either .Rdata files or .csv files.

**Usage**

```r
preserveProject(rcon, ..., save_as = NULL, dir, dir_create = TRUE)
```

```r
def S3 method for class 'redcapApiConnection'
preserveProject(
  rcon,
  ..., 
  save_as = NULL,
  dir,
  dir_create = TRUE,
  error_handling = getOption("redcap_error_handling"),
  config = list()
)
```

```r
readPreservedProject(x, ...)
```

```r
def S3 method for class 'list'
readPreservedProject(x, ..., version = NULL, url = NULL)
```

```r
def S3 method for class 'character'
readPreservedProject(x, ..., version = NULL, url = NULL)
```

**Arguments**

- `rcon` A redcapConnection object.
- `...` arguments to pass to other methods
- `save_as` character(1) or NULL. When "Rdata", the data objects will be saved to an .Rdata file. When "csv", the data objects will be written to files at `dir`. Any other character value will prompt an error.
- `dir` character(1). The path to a directory in which the data objects (or files) will be saved. Must be provided if `save_as` is not NULL.
- `dir_create` logical(1). When TRUE, an attempt will be made to create the directory at `dir` if it does not already exist. When FALSE, and the directory does not exist, an error is returned.
- `error_handling` character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see `redcapError()`.


**preserveProject**

config A list to be passed to `httr::POST()`. This allows the user to set additional configurations for the API calls, such as certificates, SSL version, etc. For the majority of users, this does not need to be altered.

x list or character. If a list, the list returned (or saved) by `preserveProject`. If character, the directory to which the CSV files are saved by `preserveProject`.

version character(1) giving the instance’s REDCap version number.

url character(1). URL for the user’s REDCap database API.

**Details**

The options to save files to local files provide the user a convenient tool set for providing other users with the ability to work with data offline. See the examples for suggestions on how to read data into an `offlineConnection`.

When saving to an `.Rdata` file, the data are saved in a list named RedcapList. The list has the same elements in the list returned when `save_as = NULL` and is suitable for creating an `offlineConnection`. The file name it is saved to follows the pattern "project-[project_id]-RedcapList.Rdata".

When saving to a `.csv` file, each element of the data is saved to a file with the pattern "project-[project_id]-[data type].csv".

`readPreservedProject` is a function of convenience for users who need to work using offline connections. If given a list, it must be in the format returned by `preserveProject`. If given a character, it must be the directory in which the CSV files were saved by `preserveProject`. If any of the file names have been changed, `readPreservedProject` will fail to execute. Refer to vignette("redcapAPI-offline-connection", package = "redcapAPI") for more details.

**Value**

`preserveProject`:

When `save_as = NULL`, returns a list is returned with the elements

- project_information
- arms
- events
- meta_data
- mappings
- repeating_instruments
- users
- user_roles
- user_role_assignments
- dags
- dag_assignments
- records

When `save_as` is not NULL, the logical TRUE is invisibly returned to provide an indication that the save operation(s) are complete.

`readPreservedProject`:

Returns a `redcapOfflineConnection` object.
See Also

vignette("redcapAPI-offline-connection", package = "redcapAPI").
offlineConnection()
purgeProject().
restoreProject()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Save a project to the current session
projectData <- preserveProject(rcon)

# Save a project to an Rdata file
save_to_dir <- tempdir()
preserveProject(rcon,
    save_as = "Rdata",
    dir = save_to_dir)

# Create an offline connection from the Rdata file
load(file.path(save_to_dir,
    "project-[project_id]-RedcapList.Rdata"))
off_conn <- readPreservedProject(RedcapList,
    version = "[redcap_api_version]",
    url = "[redcap_api_url]"

# Save a project to CSV files
save_to_dir <- tempdir()
preserveProject(rcon,
    save_as = "csv",
    dir = save_to_dir)

# Create an offline connection from the CSV files
off_con <-
    readPreservedProject(save_to_dir,
        version = "[redcap_api_version]",
        url = "[redcap_api_url]"

## End(Not run)
Export and Import Project Settings

Description

These methods enable the user to export or update project level settings, such as the project title, if it is longitudinal, if surveys are enabled, etc.

Usage

```r
exportProjectInformation(rcon, ...)  
importProjectInformation(rcon, data, ...)
```

### S3 method for class 'redcapApiConnection'
```r
exportProjectInformation(
  rcon,  
  ...,  
  error_handling = getOption("redcap_error_handling"),  
  config = list(),  
  api_param = list()
)
```

### S3 method for class 'redcapApiConnection'
```r
importProjectInformation(
  rcon,  
  data,  
  ...,  
  error_handling = getOption("redcap_error_handling"),  
  config = list(),  
  api_param = list()
)
```

Arguments

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rcon</td>
<td>A <code>redcapConnection</code> object.</td>
</tr>
<tr>
<td>data</td>
<td><code>data.frame</code> with only one row and any subset of allowable fields to be updated. See Details.</td>
</tr>
<tr>
<td>...</td>
<td>Arguments to pass to other methods</td>
</tr>
<tr>
<td>error_handling</td>
<td>character(1). One of c(&quot;error&quot;, &quot;null&quot;). An option for how to handle errors returned by the API. see <code>redcapError()</code>.</td>
</tr>
<tr>
<td>config</td>
<td>A named list. Additional configuration parameters to pass to <code>httr::POST()</code>. These are appended to any parameters in <code>rcon$session$config</code>.</td>
</tr>
</tbody>
</table>
api_param: A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

When importing, fields that are not editable will be quietly removed prior to import. This allows the user to use the result of exportProjectInformation as a template for the import.

For any values that are boolean, they should be represented as either a '0' (no/false) or '1' (yes/true).

It is not required for data to have all of the fields, but only the fields the user wishes to update (see examples).

The following project attributes can be updated:

- project_title
- project_language
- purpose
- purpose_other
- project_notes
- custom_record_label
- secondary_unique_field
- is_longitudinal
- surveys_enabled
- scheduling_enabled
- record_autonumbering_enabled
- randomization_enabled
- project_irb_number
- project_grant_number
- project_pi_firstname
- project_pi_lastname
- display_today_now_button
- bypass_branching_erase_field_prompt

Value

exportProjectInformation returns a data frame with the columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>project_id</td>
<td>The internal ID number assigned to the project.</td>
</tr>
<tr>
<td>project_title</td>
<td>The project title given to the project.</td>
</tr>
<tr>
<td>creation_time</td>
<td>The date/time the project was created.</td>
</tr>
<tr>
<td>production_time</td>
<td>The date/time the project was moved into production status.</td>
</tr>
<tr>
<td>in_production</td>
<td>Boolean value indicating if the project is in production status.</td>
</tr>
<tr>
<td>project_language</td>
<td>The language associated with the project.</td>
</tr>
</tbody>
</table>
purpose  
An integerish value identifying the purpose of the project. 0 = 'Practice/Just for fun', 1 = 'Other', 2 = 'Research', 3 = 'Quality Improvement', 4 = 'Operational Support'.

purpose_other  
The user supplied character value given when the project purpose is 'Other'.

project_notes  
The user supplied notes about the project.

custom_record_label  
The user provided custom label for the record identifier field.

secondary_unique_field  
The name of the secondary unique field, if this has been configured.

is_longitudinal  
Boolean value indicating if the project is a longitudinal project.

has_repeating_instruments_or_events  
Boolean value indicating if the repeating instruments or events module has been enabled.

surveys_enabled  
Boolean value indicating if the surveys module has been enabled.

scheduling_enabled  
Boolean value indicating if the scheduling module has been enabled.

record_autonumbering_enabled  
Boolean value indicating if the record autonumbering feature has been enabled.

randomization_enabled  
Boolean value indicating if the randomization module has been enabled.

ddp_enabled  
Boolean value indicating if dynamic data pull has been enabled for a project (may only be enabled by a REDCap administrator).

project_irb_number  
The user provided IRB number for the project.

project_grant_number  
The user provided grant number for the project.

project_pi_firstname  
The first name of the principal investigator.

project_pi_lastname  
The last name of the principal investigator.

display_today_now_button  
Boolean value indicating if the today/now button is displayed for date/time fields in the UI.

missing_data_codes  
Character value giving the missing data codes enabled for the project. They are given in the format 

<table>
<thead>
<tr>
<th>code</th>
<th>label</th>
</tr>
</thead>
</table>

, with each coding separated by a pipe character.

external_modules  
Character value listing the external modules enabled.

bypass_branching_erase_field_prompt  
Boolean value indicating if the box for 'Prevent branching logic from hiding fields' is checked.

importProjectInformation invisibly returns the number of fields updated.

Functions

• exportProjectInformation(): Export project settings.

• importProjectInformation(): Import project settings.

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())

# Export Project Information
exportProjectInformation(rcon)

# Import a new project title
NewData <- data.frame(project_title = "New Title Name")
importProjectInformation(rcon,
                         data = NewData)

# Enable surveys in the project
NewData <- data.frame(surveys_enabled = 1)
importProjectInformation(rcon,
                         data = NewData)
```
# Change multiple fields in the project settings
NewData <- data.frame(project_irb_number = "IRB-12345",
                      display_today_now_button = 0,
                      scheduling_enabled = 1)
importProjectInformation(rcon,
                         data = NewData)

## End(Not run)

---

### purgeRestoreProject

**Purge and Restore Project Data**

#### Description

These functions are primarily intended to assist with testing features of `redcapAPI`. Purging and restoring project data permits us to perform tests on different project structures without having to manage multiple projects or API tokens.

When purging project data, many of these actions may only be performed with a project in development status, as they are potentially destructive and may result in data loss. It is a good practice to back up your data and project structure before purging a project.

#### Usage

```
purgeProject(object, ...)

## S3 method for class 'redcapApiConnection'
purgeProject(
  object,
  arms = FALSE,
  events = FALSE,
  users = FALSE,
  user_roles = FALSE,
  dags = FALSE,
  records = FALSE,
  purge_all = FALSE,
  flush = TRUE,
  ...
)
```

```
restoreProject(object, ...)

## S3 method for class 'redcapApiConnection'
restoreProject(
  object,
  ...)```
project_information = NULL,
arms = NULL,
events = NULL,
meta_data = NULL,
mappings = NULL,
repeating_instruments = NULL,
users = NULL,
user_roles = NULL,
user_role_assignments = NULL,
dags = NULL,
dag_assignments = NULL,
records = NULL,
flush = TRUE,
...
error_handling = getOption("redcap_error_handling"),
config = list()
)

## S3 method for class 'list'
restoreProject(
  object,
  rcon,
  ...
  error_handling = getOption("redcap_error_handling"),
  config = list()
)

Arguments

object, rcon  A redcapConnection object. Except in restoreProject.list, where object
is a list of data frames to use in restoring the project.
...
Arguments to pass to other methods
arms  Either logical(1) indicating if arms data should be purged from the project; or
a data.frame for restoring arms data via importArms.
events  Either logical(1) indicating if events data should be purged from the project;
or a data.frame for restoring events data via importEvents
users  Either logical(1) indicating if users data should be purged from the project;
or a data.frame for restoring users data via importUsers. NOT YET IMPLEMENTED
user_roles  Either logical(1) indicating if user roles data should be purged from the project;
or a data.frame for restoring user roles data via importUserRoles. NOT YET IMPLEMENTED
dags  Either logical(1) indicating if DAG data should be purged from the project;
or a data.frame for restoring DAGs data via importDags. NOT YET IMPLEMENTED
records  Either logical(1) indicating if records data should be purged from the project;
or a data.frame for restoring records data via importRecords
purgeRestoreProject

purge_all logical(1). A shortcut option to purge all data elements from a project.
flush logical(1). When TRUE, all caches in the connection object will be flushed after completing the operation. This is highly recommended.
error_handling character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see `redcapError()`.
config A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.

project_information data.frame for restoring data. Provides the project settings to load via `importProjectInformation`.
meta_data A data.frame for restoring metadata data via `importMetaData`. The API does not support deleting metadata, but an import replaces the existing metadata.
mappings A data.frame for restoring instrument-event mappings via `importMappings`. The API does not support deleting mappings, but an import replaces the existing mappings.
repeating_instruments A data.frame for restoring repeating instruments configuration via `importRepeatingInstrumentsEvents()`. The API does not support deleting repeating instruments, but an import replaces the existing instruments. NOT YET IMPLEMENTED
user_role_assignments A data.frame for restoring user-role assignments via `importUserRoleAssignments`. The API does not support deleting assignments, but an import replaces the existing assignments. NOT YET IMPLEMENTED.
dag_assignments A data.frame for restoring DAG assignments via `importDagAssignments`. The API does not support deleting assignments, but an import replaces the existing assignments. NOT YET IMPLEMENTED.

Details

When restoring a project, all arguments are optional. Any argument that is NULL will result in no import being made. The order of reconstructing the project is (purging data occurs in the reverse order):

1. Update project information
2. Import Arms Data
3. Import Events Data
4. Import Meta Data
5. Import Mappings
6. Import Repeating Instruments
7. Import Users
8. Import User Roles
9. Import User-Role Assignments
10. Import Data Access Groups
11. Import Data Access Group Assignments
12. Import Records
Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
  url = "your_redcap_url",
  keyring = "API_KEYs",
  envir = globalenv())

# Preserve a project
preserveProject(rcon)

# Purge a project
purgeProject(rcon,
  purge_all = TRUE)

# Restore a project
restoreProject(rcon)

## End(Not run)
```

---

**recodeCheck**

*Change labeling of checkbox variables*

Description

Rewrites the labeling of checkbox variables from Checked/Unchecked to Yes/No (or some other user specified labeling).

Usage

```r
recodeCheck(
  df,
  vars = NULL,
  old = c("Unchecked", "Checked"),
  new = c("No", "Yes"),
  reverse = FALSE
)
```

Arguments

- **df**: A data frame, presumably retrieved from REDCap, though not a strict requirement.
- **vars**: Optional character vector of variables to convert. If left missing, all of the variables in df that are identified as checkbox variables are relabeled. See 'Details' for more about identifying checkbox variables.
- **old**: A character vector to be passed to factor. This indicates the levels to be replaced and their order.
new

A character vector of labels to replace the values in `levels`. The first value becomes the reference value.

reverse

For convenience, if the user would prefer to reverse the order of the elements in `levels` and `labels`, simply set this to `TRUE`.

Details

Checkbox variables are not identified using the metadata from the REDCap database. Instead, variables are scanned, and those variables in which every value is in `levels` are assumed to be checkbox variables.

Realistically, this could be used to relabel any set of factors with identical labels, regardless of the data source. The number of labels is not limited, but `levels` and `labels` should have the same length.

The actual code to perform this is not particularly difficult (`df[,checkbox] <- lapply(df[,checkbox], factor, levels=levels, labels=labels)`), but checkbox variables are common enough in REDCap (and the Checked/Unchecked scheme so unpalatable) that a quick way to replace the labels was highly desirable.
filename character(0/1) An optional filename. This is used in the case where a filename is being provided. If this has length 0, the filename will be extracted from the API response.

See Also

exportFiles(),
exportFromFileRepository(),
exportFileRepository(),
exportPdf()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
    url = "your_redcap_url",
    keyring = "API_KEYs",
    envir = globalenv())

response <- makeApiCall(rcon,
    body = list(content = 'file',
        action = 'export',
        record = '1',
        field = 'file_upload_test',
        event = 'event_1_arm_1'))
reconstituteFileFromExport(response,
    dir = tempdir())

## End(Not run)
error_handling = getOption("redcap_error_handling"),
config = list(),
api_param = list()
)

## S3 method for class 'redcapApiConnection'
renameRecord(
  rcon,
  record_name,
  new_record_name,
  arm = NULL,
  ...
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon A redcapConnection object.
record_name character(1) or integerish(1). The name of an existing record in the project.
new_record_name character(1) or integerish(1). The new name to give to the record.
arm character(1) or NULL, an optional arm number. If NULL, then all records with
same name across all arms on which it exists (if longitudinal with multiple arms)
will be renamed to new record name, otherwise it will rename the record only in
the specified arm.
...
error_handling character(1). One of c("error", "null"). An option for how to handle
errors returned by the API. see redcapError().
config A named list. Additional configuration parameters to pass to http::POST().
These are appended to any parameters in rcon$config.
api_param A named list. Additional API parameters to pass into the body of the API
call. This provides users to execute calls with options that may not otherwise be
supported by redcapAPI.

Value

exportNextRecordName returns an integerish value. The value is determined by looking up the
highest record ID number in the project and incrementing it by 1.
renameRecord invisibly returns a logical value that indicates if the operation was successful. Other-
wise, an error is thrown.

Functions

- exportNextRecordName(): Get the ID number for the next record to be created.
- renameRecord(): Rename an existing record.
See Also

exportRecords()
exportReports()
importRecords()
deleteRecords()
exportRecordsTyped()
exportReportsTyped()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Get the next record name
exportNextRecordName(rcon)

# Rename an existing record
renameRecord(rcon,
record_name = "1",
new_record_name = "42")

## End(Not run)
checkboxLabels = FALSE,
colClasses = character(0),
...
)

exportRecords_offline(
dataFile,
metaDataFile,
factors = TRUE,
fields = NULL,
forms = NULL,
labels = TRUE,
dates = TRUE,
checkboxLabels = FALSE,
colClasses = NA,
...,  
meta_data
)

exportReports(
  rcon,
  report_id,
  factors = TRUE,
  labels = TRUE,
  dates = TRUE,
  drop = NULL,
  checkboxLabels = FALSE,
  ...
)

## S3 method for class 'redcapApiConnection'
expor...
config = list(),
api_param = list(),
form_complete_auto = TRUE
)

## S3 method for class 'redcapApiConnection'
exportReports(
  rcon,
  report_id,
  factors = TRUE,
  labels = TRUE,
  dates = TRUE,
  drop = NULL,
  checkboxLabels = FALSE,
...
,  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon
A redcapConnection object.

report_id
integerish(1). Gives the report id of the desired report. This is located on the Report Builder page of the user interface.

factors
logical(1). When TRUE, multiple choice fields will be returned as factors. Otherwise, they are returned as character values. See 'Exporting Records' for more on how this interacts with the checkboxLabels argument.

labels
character. Fields to be returned. When NULL, all fields are returned.

forms
character. Forms to be returned. When NULL, all forms are returned.

records
character or integerish. Record ID’s to be returned. When NULL, all records are returned.

events
character. Events to be returned from a longitudinal database. When NULL, all events are returned.

labels
logical(1). When TRUE, field labels are attached to each column as an attribute.

dates
logical(1). When TRUE, date variables are converted to POSIXct objects.

drop
character. An optional vector of REDCap field names to remove from the dataset. Ignored when NULL. Any fields in this argument that do not exist in the project will be ignored.

survey
logical(1). specifies whether or not to export the survey identifier field (redcap_survey_identifier) or survey timestamp fields ([form_name]_timestamp) when surveys are utilized in the project.

dag
logical(1). When TRUE, the system field redcap_data_access_group is included in the export. This option is only viable if the user whose token is being
used to make the API request is not in a data access group. If the user is in a
group, then this flag will revert to FALSE.

checkboxLabels  logical(1). When FALSE labels are applied as "Unchecked"/"Checked". When
TRUE, they are applied as ""/[field_label] where [field_label] is the label
assigned to the level in the data dictionary.

form_complete_auto  logical(1). When TRUE (default), the [form].complete fields for any form
from which at least one variable is requested will automatically be retrieved.
When FALSE, these fields must be explicitly requested.

colClasses  Named character vector. Column classes passed to \texttt{utils::read.csv()} calls.
Useful to force the interpretation of a column in a specific type and avoid an
unexpected recast.

batch.size  integerish(1). Specifies the number of subjects to be included in each batch
of a batched export or import. Non-positive numbers export/import the entire
operation in a single batch. Batching may be beneficial to prevent tying up
smaller servers. See Details.

dataFile  character(1). Gives the location of the dataset downloaded from REDCap.
This should be the raw (unlabeled) data set.

metaDataFile  character(1). Gives the location of the data dictionary downloaded from
REDCap.

... Arguments to pass to other methods

error_handling  character(1). One of c("error", "null"). An option for how to handle
errors returned by the API. see \texttt{redcapError()}. These are appended to any parameters in rcon$\texttt{config}$.

config  A named list. Additional configuration parameters to pass to \texttt{httr::POST()}. A named list. Additional API parameters to pass into the body of the API
call. This provides users to execute calls with options that may not otherwise be
supported by redcapAPI.

meta_data  Deprecated version of metaDataFile.

Details

It is unnecessary to include "redcap_event_name" in the fields argument. This field is automati-
cally exported for any longitudinal database. If the user does include it in the fields argument, it is
removed quietly in the parameter checks.

There are four ways the data from checkbox variables may be represented depending on the values
of factors and checkboxLabels. The most common are the first and third rows of the table below.
When checkboxLabels = TRUE, either the coded value or the labeled value is returned if the box is
checked, or an empty string if it is not.

<table>
<thead>
<tr>
<th>factors</th>
<th>checkboxLabels</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>0 / 1</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>&quot;&quot; / code</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>Unchecked / Checked</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>&quot;&quot; / label</td>
</tr>
</tbody>
</table>
The 'offline' version of `exportReports` operates on the raw (unlabeled) data file downloaded from REDCap along with the data dictionary. This is made available for instances where the API cannot be accessed for some reason (such as waiting for API approval from the REDCap administrator).

A `batched` export (or import) is one where the export is performed over a series of API calls rather than one large call. For large projects on small servers, this may prevent a single user from tying up the server and forcing others to wait on a larger job. The batched export is performed by first calling the API to export the record identifier field (the first field in the meta data). The unique ID's are then assigned a batch number with no more than `batch.size` ID's in any single batch. The batches are exported from the API and stacked together.

In longitudinal projects, `batch.size` may not necessarily be the number of records exported in each batch. If `batch.size` is ten and there are four records per patient, each batch will consist of 40 records. Thus, if the user is concerned about tying up the server with a large, longitudinal project, it would be prudent to use a smaller `batch.size`.

**Value**

`exportRecords` returns a data frame with the project data. The data will be formatted consistent with the meta data and the arguments provided by the user.

`exportReports` returns a data frame with the data from the requested report. The data will be formatted consisted with the meta data and the arguments provided by the user.

**Functions**

- `exportRecords()`: Export records from a project.
- `exportRecords_offline()`: Format records from REDCap data file exports.
- `exportReports()`: Export data via a report.

**See Also**

- `exportRecordsTyped()`,
- `exportReportsTyped()`,
- `importRecords()`,
- `deleteRecords()`,
- `exportNextRecordName()`,
- `renameRecord()`

**Examples**

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYs",
              envir = globalenv())

# Export records
exportRecords(rcon)

# Export records in batches of one hundred IDs
exportRecords(rcon,
```
batch.size = 100)

# Export records without factors
exportRecords(rcon,
              factors = FALSE)

# Export a report
exportReports(rcon,
              report_id = 12345)

# Export raw data
exportRecordsTyped(rcon,
                   validation = skip_validation,
                   cast = raw_cast)

## End(Not run)

---

**recordsTypedMethods** Export Records or Reports From a Project

**Description**

These methods enable the user to export records from a database or from a report. These methods have more control for casting fields to R objects than `exportRecords`.

**Usage**

```r
exportRecordsTyped(
  rcon,
  fields = NULL,
  drop_fields = NULL,
  forms = NULL,
  records = NULL,
  events = NULL,
  ...
)
exportReportsTyped(rcon, report_id, ...)
```

## S3 method for class 'redcapApiConnection'

```r
exportRecordsTyped(
  rcon,
  fields = NULL,
  drop_fields = NULL,
  forms = NULL,
)```
records = NULL,
events = NULL,
survey = TRUE,
dag = FALSE,
date_begin = NULL,
date_end = NULL,
na = list(),
validation = list(),
cast = list(),
assignment = list(label = stripHTMLandUnicode, units = unitsFieldAnnotation),
filter_empty_rows = TRUE,
warn_zero_coded = TRUE,
...,
config = list(),
api_param = list(),
csv_delimiter = ",",
batch_size = NULL,
errorHandling = getOption("redcap_error_handling")
)

## S3 method for class 'redcapOfflineConnection'
exportRecordsTyped(
  rcon,
  fields = NULL,
  drop_fields = NULL,
  forms = NULL,
  records = NULL,
  events = NULL,
  na = list(),
  validation = list(),
  cast = list(),
  assignment = list(label = stripHTMLandUnicode, units = unitsFieldAnnotation),
  warn_zero_coded = TRUE,
  ...
)

## S3 method for class 'redcapApiConnection'
exportReportsTyped(
  rcon,
  report_id,
  drop_fields = NULL,
  na = list(),
  validation = list(),
  cast = list(),
  assignment = list(label = stripHTMLandUnicode, units = unitsFieldAnnotation),
  warn_zero_coded = TRUE,
  ...
)

config = list(),
api_param = list(),
csv_delimiter = ",,"
)

Arguments

rcon A redcapConnection object.
report_id integerish(1). The ID number of the report to be exported.
fields character or NULL. Vector of fields to be returned. If NULL, all fields are returned (unless forms is specified).
drop_fields character or NULL. A vector of field names to remove from the export.
forms character or NULL. Vector of forms to be returned. If NULL, all forms are returned (unless fields is specified).
records character or integerish. A vector of study ID's to be returned. If NULL, all subjects are returned.
events A character vector of events to be returned from a longitudinal database. If NULL, all events are returned. When using a redcapOfflineConnection object, this argument is unvalidated, and only rows that match one of the values given are returned; misspellings may result in unexpected results.
survey logical(1). When TRUE, the survey identifier field (e.g., redcap_survey_identifier) and survey timestamp fields (e.g., [form_name]_timestamp) will be exported (relevant only when surveys are utilized in the project).
dag logical(1). When TRUE the redcap_data_access_group field will be included in the export \ when data access groups are utilized in the project. This flag is only viable if the user whose token is being used to make the API request is not in a data access group. If the user is in a group, then this flag will revert to its default value. Data Access Groups privilege is required when creating/renaming/deleting DAGs and when importing/exporting user-DAG assignments. Therefore, the default for this flag is FALSE. To export DAG information set this flag to TRUE.
date_begin POSIXct(1) or NULL. Ignored if NULL (default). Otherwise, records created or modified after this date will be returned.
date_end POSIXct(1) or NULL. Ignored if NULL (default). Otherwise, records created or modified before this date will be returned.
na A named list of user specified functions to determine if the data is NA. This is useful when data is loaded that has coding for NA, e.g. -5 is NA. Keys must correspond to a truncated REDCap field type, i.e., date_, datetime_, date-time_seconds_, time_mm_ss, time_hh_mm_ss, time, float, number, calc, int, integer, select, radio, dropdown, yesno, truefalse, checkbox, form_complete, sql, system. The function will be provided the variables (x, field_name, coding). The function must return a vector of logicals matching the input. It defaults to isNAorBlank() for all entries.
validation A named list of user specified validation functions. The same named keys are supported as the na argument. The function will be provided the variables (x, field_name, coding). The function must return a vector of logical matching the
input length. Helper functions to construct these are `valRx()` and `valChoice()`. Only fields that are not identified as NA will be passed to validation functions.

cast  
A named list of user specified class casting functions. The same named keys are supported as the na argument. The function will be provided the variables (x, field_name, coding). The function must return a vector of logical matching the input length. The cast should match the validation, if one is using raw_cast, then validation=skip_validation is likely the desired intent. See `fieldValidationAndCasting()`

assignment  
A named list of functions. These functions are provided, field_name, label, description and field_type and return a list of attributes to assign to the column. Defaults to creating a label attribute from the stripped HTML and UNICODE raw label and scanning for units={"UNITS"} in description

filter_empty_rows  
logical(1). Filter out empty rows post retrieval. Defaults to TRUE.

csv_delimiter  
character. One of c("","\	","","\\","","\^""). Designates the delimiter for the CSV file received from the API.

batch_size  
integerish(1) or NULL. When NULL, all records are pulled. Otherwise, the records all pulled in batches of this size.

warn_zero_coded  
logical(1). Turn on or off warnings about potentially problematic zero coded fields. Defaults to TRUE.

...  
Arguments to pass to other methods

error_handling  
character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see `redcapError()`.

config  
A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `redcapAPI`.

api_param  
A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

The ‘offline’ method operates on the raw (unlabeled) data file downloaded from REDCap along with the data dictionary. This is made available for instances where the API cannot be accessed for some reason (such as waiting for API approval from the REDCap administrator).

When validating data for `offlineRedcapConnection` objects, links to invalid data forms will not work if the user does not provide the url, version, project_info, and events arguments (if the project is longitudinal). For the project_info, the values project_id and is_longitudinal are required. The user may be able to provide as little as `project_info = data.frame(project_id = [id], is_longitudinal = [long])`. The user should be aware that the REDCap User Interface download for events does not include the event ID. To include the event ID, the user must construct a data frame to pass to `offlineConnection`.

Record Identifier (System) Fields:

In all calls, the project’s ID fields will be included–there is no option provided to prevent this. Additionally, if the project has a secondary unique field specified, it will also be included. Inclusion of these fields is necessary to support some post-processing functions.
By default, the system fields `redcap_event_name`, `redcap_repeat_instrument`, and `redcap_repeat_instance` are exported (when they are appropriate to the project). These are automatically included by the API. However, if the user omits any of these in `fields` or designates one in `drop_fields`, the final result will honor those conditions. Excluding any of these identifiers may cause problems with some post-processing functions that operate on repeating instrument data.

The combination of the project ID field, secondary unique field, and the system fields are what uniquely identify an experimental unit. In nearly all cases, it is desirable to have them all included.

System fields are cast to labelled values by default. They may be cast to their coded values using the override `cast = list(system = castRaw)`. The fields affected by the system override are `redcap_event_name`, `redcap_repeat_instrument`, and `redcap_data_access_group`.

**BioPortal Fields:**

Text fields that are validation enabled using the BioPortal Ontology service may be cast to labeled values so long as the labels have been cached on the REDCap server. Caching is performed when the field is viewed in a form on the web interface. However, labels are not cached when data are imported via the API. In cases where labels are not cached, the coded value is treated as both the code and the label.

**Record Batching:**

A 'batched' export is one where the export is performed over a series of API calls rather than one large call. For large projects on small servers, this may prevent a single user from tying up the server and forcing others to wait on a larger job. The batched export is performed by first calling the API to export the subject identifier field (the first field in the meta data). The unique ID's are then assigned a batch number with no more than `batch_size` ID's in any single batch. The batches are exported from the API and stacked together.

In longitudinal projects, `batch_size` may not necessarily be the number of records exported in each batch. If `batch_size` is ten and there are four records per patient, each batch will consist of 40 records. Thus, if the user is concerned about tying up the server with a large, longitudinal project, it would be prudent to use a smaller batch size.

**Inversion of Control:**

The final product of calling this is a `data.frame` with columns that have been type cast to most commonly used analysis class (e.g. factor). This version allows the user to override any step of this process by specifying a different function for each of the stages of the type casting. The algorithm is as follows:

1. Detect NAs in returned data (`na` argument).
2. Run `validate` functions for the `field_types`.
3. On the fields that are not NA and pass `validate` do the specified cast.

It is expected that the `na` and `validate` overrides should rarely be used. Their exposure via the function parameters is to future proof against possible bugs in the defaults, and allows for things that higher versions of REDCap add as possible field types. I.e., the overrides are for use to continue using the library when errors or changes to REDCap occur.

The cast override is one where users can specify things that were controlled by an ever increasing set of flags before. E.g., `dates=as.Date` was an addition to allow dates in the previous version to be overridden if the user wanted to use the Date class. In this version, it would appear called as `cast=list(_date=as.Date)`). See `fieldValidationAndCasting()` for a full listing of package provided cast functions.
Value

exportRecordsTyped returns a data frame with the formatted data.
exportReportsTyped returns a data frame with the formatted data.

Functions

- exportRecordsTyped(): Export records with type casting.
- exportReportsTyped(): Export reports with type casting.
- exportRecordsTyped(redcapOfflineConnection): Export records without API access.

Zero-Coded Check Fields

A zero-coded check field is a field of the REDCap type checkbox that has a coding definition of \( 0, \ [\text{label}] \). When exported, the field names for these fields is \([\text{field}\_\text{name}]\_0\). As in other checkbox fields, the raw data output returns binary values where 0 represent an unchecked box and 1 represents a checked box. For zero-coded checkboxes, then, a value of 1 indicates that 0 was selected.

This coding rarely presents a problem when casting from raw values (as is done in exportRecordsTyped). However, casting from coded or labeled values can be problematic. In this case, it becomes indeterminate from context if the intent of 0 is ‘false’ or the coded value ’0’ (‘true’) ...

The situations in which casting may fail to produce the desired results are ...

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>anything other than &quot;0&quot;</td>
<td>Likely to fail when casting from coded values</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Likely to fail when casting from coded or labeled values</td>
</tr>
</tbody>
</table>

Because of the potential for miscast data, casting functions will issue a warning anytime a zero-coded check field is encountered. A separate warning is issued when a field is cast from coded or labeled values.

When casting from coded or labeled values, it is strongly recommended that the function castCheckForImport() be used. This function permits the user to state explicitly which values should be recognized as checked, avoiding the ambiguity resulting from the coding.

See Also

Other records exporting functions:
- exportRecords()
- exportReports()
- exportBulkRecords()

Field validations and casting:
- fieldValidationAndCasting()
- reviewInvalidRecords()

Post-processing functionality:
recastRecords(),
guessCast(),
guessDate(),
castForImport(),
mChoiceCast(),
splitForms(),
widerRepeated()

Vignettes:

vignette("redcapAPI-offline-connection")
vignette("redcapAPI-casting-data")
vignette("redcapAPI-missing-data-detection")
vignette("redcapAPI-data-validation")
vignette("redcapAPI-faq")

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Export records with default settings
exportRecordsTyped(rcon)

# Export records with no factors
exportRecordsTyped(rcon,
    cast = default_cast_character)

# Export records for specific records
exportRecordsTyped(rcon,
    records = 1:3)

# Export records for specific instruments
exportRecordsTyped(rcon,
    forms = c("registration", "visit_1", "medications"))

# Export records using filterLogic, an API parameter not provided
# in the exportRecordsTyped function signature
exportRecordsTyped(
    rcon,
    records = 1:3,
    api_param = list(filterLogic = "[age_at_enrollment] > 25")
)

# Export a report
exportReports(rcon,
    report_id = 12345)
redcapConnection

Connect to a REDCap Database

Description

These methods enable the user to create a connection object used to access the database.

Usage

redcapConnection(
  url = getOption("redcap_api_url"),
  token,
  config = httr::config(),
  retries = 5,
  retry_interval = 2^(seq_len(retries)),
  retry_quietly = TRUE
)

## S3 method for class 'redcapApiConnection'
print(x, ...)

offlineConnection(
  meta_data = NULL,
  arms = NULL,
  events = NULL,
  instruments = NULL,
  field_names = NULL,
  mapping = NULL,
  repeat_instrument = NULL,
  users = NULL,
)
user_roles = NULL,
user_role_assignment = NULL,
dags = NULL,
dag_assignment = NULL,
project_info = NULL,
version = NULL,
file_repo = NULL,
records = NULL,
url = NULL,
external_coding = list()
)

## S3 method for class 'redcapOfflineConnection'
print(x, ...)

Arguments

url character(1). URL for the user’s REDCap database API.
token character(1) REDCap API token
config A list to be passed to `httr::POST()`. This allows the user to set additional configurations for the API calls, such as certificates, SSL version, etc. For the majority of users, this does not need to be altered.
retries integerish(1). Sets the number of attempts to make to the API if a timeout error is encountered. Must be a positive value.
retry_interval numeric. Sets the intervals (in seconds) at which retries are attempted. By default, set at a $2^r$ where $r$ is the $r$th retry (ie, 2, 4, 8, 16, ...). For fixed intervals, provide a single value. Values will be recycled to match the number of retries.
retry_quietly logical(1). When FALSE, messages will be shown giving the status of the API calls. Defaults to TRUE.
x redcapConnection object to be printed
... arguments to pass to other methods
meta_data Either a character giving the file from which the metadata can be read, or a data.frame.
arms Either a character giving the file from which the arms can be read, or a data.frame.
events Either a character giving the file from which the events can be read, or a data.frame.
instruments Either a character giving the file from which the instruments can be read, or a data.frame.
field_names Either a character giving the file from which the field names can be read, or a data.frame.
mapping Either a character giving the file from which the Event Instrument mappings can be read, or a data.frame.
Either a character giving the file from which the Repeating Instruments and Events settings can be read, or a data.frame. Note: The REDCap GUI does not offer a download file of these settings (at the time of this writing).

Either a character giving the file from which the User settings can be read, or a data.frame.

Either a character giving the file from which the User Roles can be read, or a data.frame.

Either a character giving the file from which the User-Role Assignments can be read, or a data.frame.

Either a character giving the file from which the Data Access Groups can be read, or a data.frame.

Either a character giving the file from which the Data Access Group Assignments can be read, or a data.frame.

Either a character giving the file from which the Project Information can be read, or a data.frame. See Details.

Either a character giving the file from which the File Repository Listing can be read, or a data.frame.

Either a character giving the file from which the Records can be read, or a data.frame. This should be the raw data as downloaded from the API, for instance. Using labeled or formatted data is likely to result in errors when passed to other functions.

Named list of named character vectors or a character giving the file from which the external coding may be read. The list is generally obtained from the API using `exportExternalCoding()`. The name of the list element should be a field name in the data that is of type `bioportal` or `sql`. The named vectors are code-label pairings where the value of the vector is the code and the name is the label. If passing a file name, it should be a file with the list saved via `dput`.

**Details**

`redcapConnection` objects will retrieve and cache various forms of project information. This can make metadata, arms, events, etc. available directly from the `redcapConnection` object. The retrieval of these objects uses the default values of the respective export functions (excepting the file repository, which uses `recursive = TRUE`).

For each of these objects, there are four methods that can be called from the `redcapConnection` object:

<table>
<thead>
<tr>
<th>Function type</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[info_type]</code></td>
<td>Returns the information from the connection object</td>
<td><code>rcon$metadata()</code></td>
</tr>
<tr>
<td><code>has_[info_type]</code></td>
<td>Returns a boolean indicating if the information is cached</td>
<td><code>rcon$has_metadata()</code></td>
</tr>
<tr>
<td><code>flush_[info_type]</code></td>
<td>Purges the information from the connection object</td>
<td><code>rcon$flush_metadata()</code></td>
</tr>
<tr>
<td><code>refresh_[info_type]</code></td>
<td>Replaces the information with a new call to the API</td>
<td><code>rcon$refresh_metadata()</code></td>
</tr>
</tbody>
</table>
Information is cached for

- metadata
- arms
- events
- instruments
- fieldnames
- mapping (field-event mappings)
- repeatInstrumentEvent
- users
- user_roles
- user_role_assignment
- dags
- dag_assignment
- projectInformation
- version
- fileRepository
- externalCoding

There is also a flush_all and refresh_all method that will purge the entire cache and refresh the entire cache, respectively.

The externalCoding elements relate to the code-label mappings of text fields with the external validation types (such as sql fields or text fields with BioPortal Ontology modules enabled).

**Specific to API Connections:**
The redcapApiConnection object also stores the user preferences for handling repeated attempts to call the API. In the event of a timeout error or server unavailability, these settings allow a system pause before attempting another API call. In the event all of the retries fail, the error message of the last attempt will be returned. These settings may be altered at any time using the methods rcon$set_retries(r), rcon$set_retry_interval(ri), and rcon$set_retry_quietly(rq). The argument to these functions have the same requirements as the corresponding arguments to redcapConnection.

Tokens are specific to a project, and a token must be created for each project for which the user wishes to use the API.

Additional Curl option can be set in the config argument. See the documentation for httr::config() and http::http_options() for more Curl options.

**Specific to Offline Connections:**
"Offline connections" are a tool designed to provide the users without API privileges with at least a subset of the functionality available to API users. The offline connections are typically constructed from the comma separated value (CSV) files downloaded from the REDCap user interface. Alternatively, data frames may be provided with the necessary data.

Not all of the components of an offline connection are needed for most operations. Rather, the object was built to accept the same components available to the redcapApiConnection in order to provide a consistent interface and simplify future development.
The metadata will be required for nearly all operations. For validating and casting data, the records data must be provided, and works best if the data are the raw, unlabeled data downloaded from the REDCap user interface.

Other components that may prove useful when casting records are the url, version, events (if the project is longitudinal), and a subset of the project information. The user is encouraged to review the vignette for working with offline connections for more details.

With offline connections, the refresh methods have an important difference. The user may pass the refresh method a file path or data frame which will be used to replace the existing component. See examples.

**See Also**

For establishing connections using secure token storage.

unlockREDCap()

vignette("redcapAPI-getting-started-connecting", package = "redcapAPI")

For working with offline connections. vignette("redcapAPI-offline-connection", package = "redcapAPI")

To prepare data for an offline user, see **preserveProject()** and **readPreservedProject()**.

**Examples**

```r
## Not run:
 rcon <- redcapConnection(url = [YOUR_REDCAP_URL],
 token = [API_TOKEN])

exportRecords(rcon)

# Get the complete metadata for the project
 rcon$metadata()

# Get the fieldnames for a project
 rcon$fieldnames()

# remove a cached value for fieldnames
 rcon$flush_fieldnames()
 rcon$has_fieldnames()

# Using offline connections

meta_data_file <- "path/to/meta_data_file.csv"
records_file <- "path/to/records_file.csv"
events_file <- "path/to/events_file.csv"

ProjectInfo <- data.frame(project_id = 12345,
 is_longitudinal = 1)

off_conn <- offlineConnection(meta_data = meta_data_file,
 records = records_file,
 events = events_file,
 ProjectInfo = ProjectInfo)
```

redcapDataStructures

Description

Utilities for recognizing and validating data structures for use with the REDCap API

Usage

validateRedcapData(data, redcap_data)

Arguments

data data.frame User provided data to be compared against the established REDCap data structure.
redcap_data data.frame A data set from the redcapAPI package to use a reference for comparing to expected data structure.
redcapError

Format

An object of class character of length 4.
An object of class character of length 5.
An object of class character of length 11.
An object of class character of length 23.
An object of class data.frame with 0 rows and 3 columns.

redcapError  Handle Errors from the REDCap API

Description

Determine the proper way to handle errors returned from the API. Not all errors should be fatal. See Details for more

Usage

redcapError(x, error_handling = getOption("redcap_error_handling"))

Arguments

x  Object returned by http::POST.

error_handling  character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError.

Details

Maintaining consistent functionality for all types of REDCap projects requires that errors be handled delicately. It is not always desirable for an error from the API to terminate the program. One example of such a case is when a user executes the exportEvents function for a classic project; doing so returns an error from the API that events cannot be exported for classic projects. In REDCap versions earlier than 6.5.0, there is no way to determine if a project is classic or longitudinal without attempting to export the events.

For this reason, it is often preferable to have these kinds of errors return NULL so that the program does not crash if it does not need to (one such instance where it does not need to crash is when exportEvents is called within exportRecords; the events argument is irrelevant to a classic project and the error can safely be ignored.

The other common type of error that does not need to be fatal is when a redcapAPI method is sent to a REDCap instance that does not support the method. For example, the exportVersion method is not supported in REDCap instances earlier than 6.0.0. In these cases, we may prefer not to cast a hard error.

These two types of errors may be handled in one of two ways. When the error handler is set to "null", a NULL is returned. When the error handler is set to "error", the error is returned. The option is set globally using options(redcap_error_handler = "null") and is set to "null" by default.
repeatingInstrumentMethods

---

**redcapFactorFlip**  
*Convert REDCap factors between labeled and coded*

---

**Description**

Factors exported from REDCap can be toggled between the coded and labeled values with the use of the attributes assigned to the factors during export.

**Usage**

redcapFactorFlip(v)

**Arguments**

- **v**  
  A factor exported from REDCap. The REDCap type may be radio, dropdown, check, yesno, etc.

**Details**

Each factor type variable in REDCap is given the attributes `redcapLabels` and `redcapLevels`. With these attached to the vector, switching between the coded and labeled values can be done with ease. This may be helpful when the coded value has importance, such as 0/1 for death, or if a yes is worth six points (instead of one).

---

**repeatingInstrumentMethods**  
*Export or Import Repeating Instrument and Events Settings*

---

**Description**

These methods enable the user to export the existing repeating instrument and event settings, or import new settings to the project.

**Usage**

exportRepeatingInstrumentsEvents(rcon, ...)

importRepeatingInstrumentsEvents(rcon, data, ...)

```r
## S3 method for class 'redcapApiConnection'
exportRepeatingInstrumentsEvents(
  rcon,
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
)```

---
importRepeatingInstrumentsEvents(
    rcon,
    data,
    ...
)  

Arguments

**rcon**  
A redcapConnection object.

**data**  
data.frame. For classical projects, it must have the columns `form_name` and `custom_form_label`. Longitudinal projects also require a column for `event_name`.

**...**  
Arguments to pass to other methods

**error_handling**  
character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

**config**  
A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.

**api_param**  
A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

Repeating events (as opposed to repeating instruments) are provided as a row of data where the `form_name` column is NA.

It is not possible to update the `has_repeating_instruments_or_events` property of the project through `importProjectInformation`. Enabling of repeating instruments and events must be done through the GUI.

Although the API does not provide a delete method, it is possible to remove settings by doing an import that excludes the settings that are to be deleted. All settings can be cleared by executing `importRepeatingInstrumentsEvents(rcon, REDCAP_REPEAT_INSTRUMENT_STRUCTURE)`.

Value

`exportRepeatingInstrumentsEvents` returns a data frame with the columns:

- **event_name**: The unique event name.
- **form_name**: The form name, as given in the second column of the Meta Data
- **custom_form_label**: A custom display string for the repeating instrument/event
importRepeatingInstrumentsEvents invisibly returns the number of rows imported.

Functions

- exportRepeatingInstrumentsEvents(): Export repeating instruments and events.
- importRepeatingInstrumentsEvents(): Import repeating instruments and events.

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Export repeating instruments and events
exportRepeatingInstrumentsEvents(rcon)

# Import repeating instruments and events
NewData <- data.frame(event_name = c("event_1_arm_1", "event_2_arm_1"),
                      form_name = c("field_observation", "self_assessment"),
custom_form_label = c("Instructor led field observation", "Trainee self assessment"))

importRepeatingInstrumentsEvents(rcon,
data = NewData)
```

reviewInvalidRecords

Review Invalid Records Following Field Validation

Description

This function retrieves a summary of data elements that failed validation during field validation and casting.

Usage

```r
reviewInvalidRecords(data, quiet = TRUE)
```

Arguments

data: data.frame. The result of a function that performed field validation.
quiet: logical(1). When TRUE, a message will be printed if the invalid attribute is not found on data. Otherwise, the message is suppressed.
Details

When discussing field validation and invalid data, it is helpful to establish the following terminology:

A *Records data frame* is a data frame returned by a function where the fields (columns) in the data frame have been cast for subsequent analysis.

Some casting function also perform field validation and return an *Invalid data frame*, which is a listing of data elements that have failed validation. The Invalid data frame is attached as an attribute to the Records data frame. If no data elements fail the validation, the Invalid data frame will have zero rows. If at least one data element fails validation, a warning is printed to notify the user so that the user may review the Invalid data frame and mitigate the failed validations.

The Invalid data frame has an additional class (c("invalid", "data.frame")) and comes with a print method. The print.invalid method displays the content of the Invalid data frame neatly in both the console and within reports utilizing markdown.

Value

If data has the "invalid" attribute, an object with class c("invalid", "data.frame") is returned. (NULL will be returned if data does not have the attribute).

The columns in the Invalid data frame are

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The row number from the Records data frame for which validation failed.</td>
</tr>
<tr>
<td>record_id</td>
<td>The record ID for the failed validation.</td>
</tr>
<tr>
<td>field_name</td>
<td>The field name (column) of the failed validation.</td>
</tr>
<tr>
<td>field_type</td>
<td>The field type of the failed validation.</td>
</tr>
<tr>
<td>value</td>
<td>The original value that failed validation. It will be replaced with NA in the Records data.</td>
</tr>
</tbody>
</table>

The Invalid data frame has additional attributes

- time - The date/time at which the validation was performed.
- version - The REDCap version number (as retrieved by exportVersion).
- project - The title of the REDCap project (as retrieved by exportProjectInformation).

See Also

`exportRecordsTyped()`, `exportReportsTyped()`, `castForImport()`, `guessCast()`

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYS",
```
splitForms

Split a Data Frame into its Forms

Description
Separates a data frame from REDCap into a list of data frames where each form constitutes an element in the list.

Usage
splitForms(Records, rcon, envir = NULL, base = NULL, post = NULL)

Arguments
- **Records**: data.frame such as one generated by `exportRecords` or `exportRecordsTyped`
- **rcon**: `redcapConnection` object.
- **envir**: environment. The target environment for the resulting list of data.frames. Defaults to `NULL` which returns the a list. Use `globalenv` to assign the global environment. Will accept a number of the environment.
- **base**: character(1) giving the start of the naming scheme for the elements of the list. By default, the names of the list will be the form names. If this value is provided, it will follow the format `base.form_name`.
- **post**: function to apply to each element of form data after separating them, must be of signature `function(data, rcon)`. 

envir = globalenv()

# Review the Invalid data frame after export
Records <- exportRecordsTyped(rcon)
reviewInvalidRecords(Records)

# Review Invalid data frame before import
Records <- castForImport(rcon)
reviewInvalidRecords(Records)

# Access the Invalid data frame the attributes
Records <- exportRecordsTyped(rcon)
attr(Records, "invalid")
attributes(Records)$invalid

## End(Not run)
stringCleanup

See Also

Other post-processing functions:

recastRecords(), guessCast(),
guessDate(),
castForImport(),
mcchoiceCast(),
widerRepeated()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

Records <- exportRecordsTyped(rcon)

splitForms(Records, rcon)

## End(Not run)

---

stringCleanup  Remove Undesired Characters From Strings

Description

These functions are utilities to clear undesired characters from REDCap output.

Usage

stripHTMLTags(
  x,
  tags = c("p", "br", "div", "span", "b", "font", "sup", "sub"),
  ignore.case = TRUE
)

stripUnicode(x)

Arguments

x character, vector of content to be cleaned.
tags character, vector of HTML tags to remove from x
ignore.case logical(1), should cases be ignored when matching patterns?  Defaults to TRUE.
stripHTMLandUnicode

Value

stripHTMLTags returns a character vector.
stripUnicode returns a character vector.

Examples

stripHTMLTags("<p>Text in a paragraph <b>tag</b> with bold formatting </p>")
stripUnicode("\U00B5 = 0")

Description

These functions assist in setting attributes for columns of the resulting type cast data.frame.

Usage

stripHTMLandUnicode(field_name, field_label, field_annotation)
unitsFieldAnnotation(field_name, field_label, field_annotation)

Arguments

field_name character. Name of the fields.
field_label character. Labels from meta data.
field_annotation character. Annotations from meta_data.

Details

Functions passed into the assignment argument list of exportRecordsTyped() construct attributes on a column. They are expected to have a signature of function(field_name, field_label, field_annotation) and return the attribute to assign or NA. They must be vectorized.

Useful utilities are provided in stringCleanup()

stripHTMLandUnicode strips both HTML and UNICODE from the field_label.
unitsFieldAnnotation pulls a units string from the field_annotation. An example of the form searched for is units=\"meters\"

Value

stripHTMLandUnicode returns a character vector.
unitsFieldAnnotation returns a character vector.
See Also

exportRecordsTyped(),
exportReportsTyped(),
stripHTMLTags(),
stripUnicode()

Examples

## Not run:
stripHTMLandUnicode("field_name", "<b>Field label</b>", "field annotation")

unitsFieldAnnotation("field", "label", "units=("meters")")

## End(Not run)

Description

These methods enable the user to export information relating to survey participants.

Usage

exportSurveyParticipants(rcon, instrument, event, ...)

exportSurveyLink(rcon, record, instrument, event, repeat_instance = 1, ...)

exportSurveyQueueLink(rcon, record, ...)

exportSurveyReturnCode(
  rcon,
  record,
  instrument,
  event,
  repeat_instance = 1,
  ...
)

## S3 method for class 'redcapApiConnection'
exportSurveyParticipants(
  rcon,
  instrument = NULL,
  event = NULL,
  ...
  error_handling = getOption("redcap_error_handling"),
)
config = list(),
api_param = list()
)

## S3 method for class 'redcapApiConnection'
exportSurveyLink(
  rcon,
  record,
  instrument,
  event = NULL,
  repeat_instance = 1,
  ...
,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
exportSurveyQueueLink(
  rcon,
  record,
  ...
,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
exportSurveyReturnCode(
  rcon,
  record,
  instrument,
  event = NULL,
  repeat_instance = 1,
  ...
,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon A redcapConnection object.
record character(1) or integerish(1). The record ID of a survey participant.
imstrument character(1). The name of a survey instrument.
event character(1) The event name of the event for which participant information should be exported.
repeat_instance
integerish(1). The repeat instance if the instrument is designated as a repeating instrument. Default value is 1.

Arguments to pass to other methods

error_handling character(1). One of c("error","null"). An option for how to handle errors returned by the API. see `redcapError()`.

config A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.

api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by `redcapAPI`.

Value

`exportSurveyParticipants` returns a data frame with the columns:

- `email` The e-mail address of the participant.
- `email_occurrence` The number of times the invitation has been sent (after the next invite).
- `identifier` Participant identifier (if it exists) to match the survey response to a participant.
- `record` Record ID of the participant.
- `invitation_sent_status` Boolean value indicating if a survey invitation has been sent.
- `invitation_send_time` Date/time the survey invitation was sent.
- `response_status` Boolean value indicating if the participant has responded.
- `survey_access_code` The participant’s survey access code.
- `survey_link` The participant’s survey link.
- `survey_queue_link` The participants’ survey queue link.

`exportSurveyLink` returns a character(1) giving the link for the user to access the survey form.

`exportSurveyQueueLink` returns a character(1) giving the survey queue link for the user.

`exportSurveyReturnCode` returns a character(1) giving the survey return code for the user.

Functions

- `exportSurveyParticipants()`: Export survey participants for a survey instrument.
- `exportSurveyLink()`: Export a survey participant’s survey instrument link.
- `exportSurveyQueueLink()`: Export a survey participant’s survey queue link.
- `exportSurveyReturnCode()`: Export a survey participant’s instrument return code.

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
             url = "your_redcap_url",
             keyring = "API_KEYs",
             envir = globalenv())
```
switchDag

Switch Data Access Group Assignment for the Current User

Description

This method enables the current API user to switch (assign/reassign/unassign) their current Data Access Group assignment if they have been assigned to multiple DAGs via the DAG Switcher page in the project.

Usage

switchDag(rcon, dag, ...)

## S3 method for class 'redcapApiConnection'
switchDag(
  rcon,
  dag,
  ...
)
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)
Arguments

- **rcon**
  A redcapConnection object.
- **dag**
  character(1) A unique data access group to which to assign the current user. Use NA to leave the user unassigned.
- **error_handling**
  character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see `redcapError()`.
- **config**
  A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in `rcon$config`.
- **api_param**
  A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Value

Invisibly returns TRUE when the call is completed successfully. Otherwise an error is thrown.

See Also

- `exportDags()`,
- `importDags()`,
- `deleteDags()`,
- `exportUserDagAssignments()`,
- `importUserDagAssignments()`

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYS",
envir = globalenv())

# Switch the current user to the DAG "Facility Two"
switchDag(rcon,
dag = "facility_two")

## End(Not run)
```

**syncUnderscoreCodings**  Synchronize coding of checkbox variables between meta data and records field names.
Description

Due to a bug in the REDCap export module, underscores in checkbox codings are not retained in the suffixes of the field names in the exported records. For example, if variable `chk` is a checkbox with a coding `"a_b, A and B"`, the field name in the data export becomes `chk___ab`. The loss of the underscore causes `fieldToVar` to fail as it cannot match variable names to the meta data. `syncUnderscoreCodings` rectifies this problem by searching the suffixes and meta data for underscores. If a discrepancy is found, the underscores are removed from the metadata codings, restoring harmony to the universe. This bug was fixed in REDCap version 5.5.21 and this function does not apply to that and later versions.

Usage

```r
syncUnderscoreCodings(records, meta_data, export = TRUE)
```

Arguments

- `records`: The data frame object returned from the API export prior to applying factors, labels, and dates via the `fieldToVar` function.
- `meta_data`: Metadata export from `exportMetaData`
- `export`: Logical. Specifies if data are being synchronized for import or export

Details

`syncUnderscoreCodings` performs a series of evaluations. First, it determines if any underscores are found in the checkbox codings. If none are found, the function terminates without changing anything.

If the checkbox codings have underscores, the next evaluation is to determine if the variable names suffixes have matching underscores. If they do, then the function terminates with no changes to the meta data.

For data exports, if the prior two checks find underscores in the meta data and no underscores in the suffixes, the underscores are removed from the meta data and the new meta data returned.

For data imports, the meta data are not altered and the `checkbox_field_name_map` attribute is used to synchronize field names to the meta data and the expectations of REDCap (for import, REDCap expects the underscore codings to be used).

Backward Compatibility

In retrospect, we realize that the way `syncUnderscoreCodings` is written is backward. We should have altered the field names in the records data frame. Any scripts that make use of `syncUnderscoreCodings` and were written prior to version 5.5.21 will fail because the underscores in the codings will now be present where they were not before.

For backward compatibility of `redcapAPI`, we continue to alter the codings in the meta data. We do not anticipate many problems, as most people do not use underscores in the checkbox codings.

If your scripts were written under REDCap 5.5.21 or higher, there will be no backward compatibility problems related to this issue.
unlockREDCap

Open REDCap connections using cryptolocker for storage of API_KEYs.

Description

Opens a set of connections to REDcap from API_KEYs stored in an encrypted keyring. If the keyring does not exist, it will ask for password to this keyring to use on later requests. Next it will ask for the API_KEYs specified in connections. If an API_KEY does not work, it will request again. On later executions it will use an open keyring to retrieve all API_KEYs or for a password if the keyring is currently locked.

Usage

unlockREDCap(
  connections,
  url,
  keyring,
  envir = NULL,
  passwordFUN = .default_pass(),
  ...
)

Arguments

connections character vector. A list of strings that define the connections with associated API_KEYs to load into environment. Each name should correspond to a REDCap project for traceability, but it can be named anything one desires. The name in the returned list is this name.

url character. The url of one's institutional REDCap server api.

keyring character. Potential keyring, not used by default.

envir environment. The target environment for the connections. Defaults to NULL which returns the keys as a list. Use globalenv() to assign in the global environment. Will accept a number such a '1' for global as well.

passwordFUN function. Function to get the password for the keyring. Usually defaults getPass::getPass. On MacOS it will use rstudioapi::askForPassword if available.

Details

If one forgets the password to this keyring, or wishes to start over: keyring::keyring_delete("<NAME_OF_KEY_RING_HERE>")

Consistent behavior requires options(keyring_backend=keyring::backend_file) to be set. It is recommended to place this in ~/.Rprofile.

For production servers where the password must be stored in a readable plain text file, it will search for ../<basename>_.yml. DO NOT USE this unless one is a sysadmin, as this defeats the security and purpose of a local encrypted file. The expected structure of this yaml file is as follows:
other-config-stuff1: blah blah
redcapAPI:
  keys:
    intake: THIS_IS_THE_INTAKE_DATABASE_APIKEY
    details: THIS_IS_THE_DETAILS_DATABASE_APIKEY
other-config-stuff2: blah blah
other-config-stuff3: blah blah

For production servers the use ofENVvariables is also supported. The connection string is converted to upper case for the search ofENV. If a YAML and ENV variable both exist, the YAML will take precedence.

IMPORTANT: Make sure that R is set to NEVER save workspace to .RData as this is writing the API_KEY to a local file in clear text because connection objects contain the unlocked key in memory. Tips are provided in vignette("redcapAPI-best-practices").

Value
If envir is NULL returns a list of opened connections. Otherwise connections are assigned into the specified envir.

See Also

  redcapConnection()

Vignettes:

  vignette("redcapAPI-best-practices").
  vignette("redcapAPI-getting-started-connecting")

Examples

```r
## Not run:
options(keyring_backend=keyring::backend_file) # Put in .Rprofile

unlockREDCap(c(test_conn = 'TestRedcapAPI',
                sandbox_conn = 'SandboxAPI'),
             keyring = '<NAME_OF_KEY_RING_HERE>',
             envir = globalenv(),
             url = 'https://<INSTITUTIONS_REDCAP_DOMAIN>/api/')

## End(Not run)
```

userMethods  

Export, Import, or Delete Users and User Permissions

Description

These methods enable the user to add and remove users from a project. They also enable the user to modify the permissions granted to each user within the project.
userMethods

Usage

exportUsers(rcon, ...)  
importUsers(rcon, data, ...)  
deleteUsers(rcon, users, ...)

## S3 method for class 'redcapApiConnection'
exportUsers(
  rcon,
  dates = TRUE,
  labels = TRUE,
  form_rights = TRUE,
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
importUsers(
  rcon,
  data,
  consolidate = TRUE,
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
deleteUsers(
  rcon,
  users,
  ...,  
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon A redcapConnection object.
dates logical(1). When TRUE, expiration dates are converted to a POSIXct object.
labels logical(1). When TRUE the data export and form access rights are converted to factor objects.
form_rights logical(1). When TRUE, the form rights will be transformed to one column per form. The API-provided character string is always returned with the format
[form_name]:[access_code] and a comma separating each form.

users character. Vector of unique user names to be deleted.

data data.frame. Provides the user data for import. It must have a column titled username. All other columns are optional.

consolidate logical(1). When TRUE, the form and data export access values will be read from the expanded columns. Otherwise, the consolidated values (as provided by the API export) are utilized.

... Arguments to pass to other methods

error_handling character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().

config A named list. Additional configuration parameters to pass to httr::POST(). These are appended to any parameters in rcon$config.

api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Details

User project access fields (those not related to forms or exports) are mapped between coded and labeled values as:

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>Access</td>
</tr>
</tbody>
</table>

Form access fields are mapped as:

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>View records/responses and edit records (survey responses are read-only)</td>
</tr>
<tr>
<td>2</td>
<td>Read Only</td>
</tr>
<tr>
<td>3</td>
<td>Edit survey responses</td>
</tr>
</tbody>
</table>

Form export permission fields are mapped as:

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>Full Data Set</td>
</tr>
<tr>
<td>2</td>
<td>De-Identified</td>
</tr>
<tr>
<td>3</td>
<td>Remove Identifier Fields</td>
</tr>
</tbody>
</table>

Importing Users/User Roles:

It is not required that the user provide a data frame with all of the fields available for modification.
Only fields that are provided will be modified. The only required field for imports is the username field.

When setting permissions for a user project access fields, form access, and form export permissions, the user may provided any of the coded or labeled values above. The user data is passed through `prepUserImportData()` before sending it to the API; text values will be converted to the numeric value.

It is also permissible to use a column for each form individually, as can be exported via `exportUsers()`. With `consolidate = TRUE`, these settings will be consolidated into the text string expected by the API.

The REDCap API does not natively allow for modifying the rights of a user that is part of a User Role. When an attempt to modify the rights of a user in a User Role is made with this package, the user will be removed from the User Role, the rights modified, and then the User Role restored. This is done silently: be aware that modifications to a user’s rights may not have an impact while the User Role is assigned.

**Limitations:**
When importing via CSV, (as `redcapAPI` does by default) it appears that the form access rights are imported but may not always be reflected in the exported values. The form export rights do not appear to be imported when using the CSV format. We may be able to resolve this in the future using a JSON format.

**Value**

`exportUsers` returns a data frame with the columns:

- **username**
  - The unique username for a user that can access the project.
- **email**
  - The e-mail address associated with the user in the REDCap system.
- **firstname**
  - The user’s first name.
- **lastname**
  - The user’s last name.
- **expiration**
  - The date at which the user’s access to the project will expire.
- **data_access_group**
  - The text name of the Data Access Group to which the user is assigned.
- **data_access_group_id**
  - The REDCap assigned unique identifier of the Data Access Group.
- **design**
  - Boolean flag indicating if the user has permissions to utilize the project design modules.
- **alerts**
  - Boolean flag indicating if the user has permissions to utilize the alerts tools.
- **user_right**
  - Boolean flag indicating if the user has permissions to modify user rights.
- **data_access_groups**
  - Boolean flag indicating if the user has permission to assign user to Data Access Groups.
- **reports**
  - Boolean flag indicating if the user has permissions to design reports.
- **stats_and_charts**
  - Boolean flag indicating if the user has permissions to view the Statistics and Charts module.
- **manage_survey_participants**
  - Boolean flag indicating if the user has permissions to manage survey participants.
- **calendar**
  - Boolean flag indicating if the user has permissions to utilize the project calendar.
- **data_import_tool**
  - Boolean flag indicating if the user has permissions to use the data import tool.
- **data_comparison_tool**
  - Boolean flag indicating if the user has permissions to use the data comparison tool.
- **logging**
  - Boolean flag indicating if the user has permissions to view the project logs (audit trail).
- **file_repository**
  - Boolean flag indicating if the user has permissions to access the project file repository.
- **data_quality_create**
  - Boolean flag indicating if the user has permission create new data quality rules.
- **data_quality_execute**
  - Boolean flag indicating if the user has permission to execute data quality rules.
- **api_export**
  - Boolean flag indicating if the user has API export privileges.
- **api_import**
  - Boolean flag indicating if the user has API import privileges.
When `form_rights = TRUE`, additional columns are created that give the form access and form export rights in an individual column for each form. Form access rights columns have the naming pattern `[form_name]_access` and the form export rights columns have the naming pattern `[form_name]_export_access`.

`importUsers` invisibly returns the number of users that were added or modified.

`deleteUsers` invisibly returns the number of users that were deleted.

**Functions**

- `exportUsers()`: Export users affiliated with a project.
- `importUsers()`: Add users or modify user permissions in a project.
- `deleteUsers()`: Remove users from a project.

**See Also**

- `exportUserRoles()`, `importUserRoles()`, `deleteUserRoles()`, `exportUserRoleAssignments()`, `importUserRoleAssignments()`

**Examples**

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
              url = "your_redcap_url",
              keyring = "API_KEYs",
              envir = globalenv())

# Export users
exportUsers(rcon)

# Export users without additional form access variables
```
Export or Import User-Role Assignments

Description

These methods enable the user to export the user-role assignments, add assignments, or modify existing assignments.

Usage

```r
exportUserRoleAssignments(rcon, ...)

importUserRoleAssignments(rcon, data, ...)

## S3 method for class 'redcapApiConnection'
exportUserRoleAssignments(
  rcon,  
  ...,  
  error_handling = getOption("redcap_error_handling"),  
  config = list(),  
  api_param = list()  
)

## S3 method for class 'redcapApiConnection'
```
importUserRoleAssignments(
  rcon,
  data,
  ...,
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

rcon A redcapConnection object.
data data.frame with columns username and unique_role_name. Each username must be unique. Users without a unique_role_name will not be assigned to a user role.
...
Arguments to pass to other methods
error_handling character(1). One of c("error", "null"). An option for how to handle errors returned by the API. see redcapError().
config A named list. Additional configuration parameters to pass to http::POST(). These are appended to any parameters in rcon$config.
api_param A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.

Value

exportUserRoleAssignments returns a data frame with the columns:

- username Username of a user in the project.
- unique_role_name The unique role name to which the user is assigned.
- data_access_group The Data Access Group to which the user is assigned.

importUserRoleAssignments invisibly returns the number of user roles assignments added or modified.

Functions

- exportUserRoleAssignments(): Export user-role assignments from a project.
- importUserRoleAssignments(): Import user-role assignments to a project.

See Also

exportUsers(),
importUsers(),
deleteUsers().
userRoleMethods

exportUserRoles(),
importUserRoles(),
deleteUserRoles()

Examples

## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
    url = "your_redcap_url",
    keyring = "API_KEYs",
    envir = globalenv())

# Export user-role assignments
exportUserRoleAssignments(rcon)

# Import/modify a user-role assignment
NewData <- data.frame(username = "desired_user_name",
    unique_role_name = "KN3430U")
importUserRolesAssignments(rcon,
    data = NewData)

## End(Not run)

userRoleMethods

Export, Import, or Delete User Roles in a Project

Description

These methods enable the user to export user roles, add user roles, or remove user roles from a project. They also enable the user to modify the permissions granted to a user.

Usage

exportUserRoles(rcon, ...)

importUserRoles(rcon, data, ...)

deleteUserRoles(rcon, user_roles, ...)

## S3 method for class 'redcapApiConnection'
exportUserRoles(
    rcon,
    labels = TRUE,
    form_rights = TRUE,
    ...,
    error_handling = getOption("redcap_error_handling"),
    config = list(),
    api_param = list()
userRoleMethods

## S3 method for class 'redcapApiConnection'
importUserRoles(
  rcon,
  data,
  consolidate = TRUE,
  ...
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

## S3 method for class 'redcapApiConnection'
deleteUserRoles(
  rcon,
  user_roles,
  ...
  error_handling = getOption("redcap_error_handling"),
  config = list(),
  api_param = list()
)

Arguments

- **rcon**: A redcapConnection object.
- **labels**: logical(1). When TRUE the data export and form access rights are converted to factor objects.
- **form_rights**: logical(1). When TRUE, the form rights will be transformed to one column per form. The API-provided character string is always returned with the format [form_name]:[access_code] and a comma separating each form.
- **user_roles**: character. Unique role names to be deleted from the project.
- **data**: data.frame. Provides the user data for import. It must have a column titled unique_role_name. All other columns are optional.
- **consolidate**: logical(1). When TRUE, the form and data export access values will be read from the expanded columns. Otherwise, the consolidated values (as provided by the API export) are utilized.
- **...**: Arguments to pass to other methods
- **error_handling**: character(1). One of c("error", "null"). An option for how to handle errors returned by the API. See redcapError().
- **config**: A named list. Additional configuration parameters to pass to `httr::POST()`. These are appended to any parameters in rcon$ config.
- **api_param**: A named list. Additional API parameters to pass into the body of the API call. This provides users to execute calls with options that may not otherwise be supported by redcapAPI.
Details

User project access fields (those not related to forms or exports) are mapped between coded and labeled values as:

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>Access</td>
</tr>
</tbody>
</table>

Form access fields are mapped as:

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>View records/responses and edit records (survey responses are read-only)</td>
</tr>
<tr>
<td>2</td>
<td>Read Only</td>
</tr>
<tr>
<td>3</td>
<td>Edit survey responses</td>
</tr>
</tbody>
</table>

Form export permission fields are mapped as:

<table>
<thead>
<tr>
<th>Code</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Access</td>
</tr>
<tr>
<td>1</td>
<td>Full Data Set</td>
</tr>
<tr>
<td>2</td>
<td>De-Identified</td>
</tr>
<tr>
<td>3</td>
<td>Remove Identifier Fields</td>
</tr>
</tbody>
</table>

Importing Users/User Roles:

It is not required that the user provide a data frame with all of the fields available for modification. Only fields that are provided will be modified. The only required field for imports is the `username` field.

When setting permissions for a user project access fields, form access, and form export permissions, the user may provide any of the coded or labeled values above. The user data is passed through `prepUserImportData()` before sending it to the API; text values will be converted to the numeric value.

It is also permissible to use a column for each form individually, as can be exported via `exportUsers()`. With `consolidate = TRUE`, these settings will be consolidated into the text string expected by the API.

The REDCap API does not natively allow for modifying the rights of a user that is part of a User Role. When an attempt to modify the rights of a user in a User Role is made with this package, the user will be removed from the User Role, the rights modified, and then the User Role restored. This is done silently: be aware that modifications to a user’s rights may not have an impact while the User Role is assigned.

Limitations:

When importing via CSV, (as redcapAPI does by default) it appears that the form access rights are imported but may not always be reflected in the exported values. The form export rights do
not appear to be imported when using the CSV format. We may be able to resolve this in the future using a JSON format.

**Value**

`exportUserRoles` returns a data frame with the columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique_role_name</td>
<td>The REDCap assigned unique role name.</td>
</tr>
<tr>
<td>role_label</td>
<td>The user provided label describing the role.</td>
</tr>
<tr>
<td>design</td>
<td>Boolean flag indicating if the user has permissions to utilize the project design modules.</td>
</tr>
<tr>
<td>alerts</td>
<td>Boolean flag indicating if the user has permissions to utilize the alerts tools.</td>
</tr>
<tr>
<td>user_right</td>
<td>Boolean flag indicating if the user has permissions to modify user rights.</td>
</tr>
<tr>
<td>data_access_groups</td>
<td>Boolean flag indicating if the user has permission to assign user to Data Access Groups.</td>
</tr>
<tr>
<td>reports</td>
<td>Boolean flag indicating if the user has permission to design reports.</td>
</tr>
<tr>
<td>stats_andCharts</td>
<td>Boolean flag indicating if the user has permissions to view the Statistics and Charts module.</td>
</tr>
<tr>
<td>manage_survey_participants</td>
<td>Boolean flag indicating if the user has permissions to manage survey participants.</td>
</tr>
<tr>
<td>calendar</td>
<td>Boolean flag indicating if the user has permissions to utilize the project calendar module.</td>
</tr>
<tr>
<td>data_import_tool</td>
<td>Boolean flag indicating if the user has permissions to use the data import tool.</td>
</tr>
<tr>
<td>data_comparison_tool</td>
<td>Boolean flag indicating if the user has permissions to use the data comparison tool.</td>
</tr>
<tr>
<td>logging</td>
<td>Boolean flag indicating if the user has permissions to view the project logs (audit trail).</td>
</tr>
<tr>
<td>file_repository</td>
<td>Boolean flag indicating if the user has permissions to access the project file repository.</td>
</tr>
<tr>
<td>data_quality_create</td>
<td>Boolean flag indicating if the user has permission create new data quality rules.</td>
</tr>
<tr>
<td>data_quality_execute</td>
<td>Boolean flag indicating if the user has permission to execute data quality rules.</td>
</tr>
<tr>
<td>api_export</td>
<td>Boolean flag indicating if the user has API export privileges.</td>
</tr>
<tr>
<td>api_import</td>
<td>Boolean flag indicating if the user has API import privileges.</td>
</tr>
<tr>
<td>mobile_app</td>
<td>Boolean flag indicating if the user has permissions to use the mobile app.</td>
</tr>
<tr>
<td>mobile_app_download_data</td>
<td>Boolean flag indicating if the user has permissions to download data on the mobile app.</td>
</tr>
<tr>
<td>record_create</td>
<td>Boolean flag indicating if the user has permission to create new records.</td>
</tr>
<tr>
<td>record_rename</td>
<td>Boolean flag indicating if the user has permission to rename existing records.</td>
</tr>
<tr>
<td>record_delete</td>
<td>Boolean flag indicating if the user has permission to delete records.</td>
</tr>
<tr>
<td>lock_records_all_forms</td>
<td>Boolean flag indicating if the user has permission to lock records across all forms.</td>
</tr>
<tr>
<td>lock_records</td>
<td>Boolean flag indicating if the user has permission to lock a records on individual forms.</td>
</tr>
<tr>
<td>lock_records_customization</td>
<td>Boolean flag indicating if the user has permission to customize record locking.</td>
</tr>
<tr>
<td>random_setup</td>
<td>Boolean flag indicating if the user has permission to set up randomization rules.</td>
</tr>
<tr>
<td>random_dashboard</td>
<td>Boolean flag indicating if the user has permission to view the randomization dashboard.</td>
</tr>
<tr>
<td>random_perform</td>
<td>Boolean flag indicating if the user has permission to perform record randomization.</td>
</tr>
<tr>
<td>forms</td>
<td>Character string listing form access rights for each form.</td>
</tr>
<tr>
<td>forms_export</td>
<td>Character string listing the form export rights for each form.</td>
</tr>
</tbody>
</table>

When `form_rights = TRUE`, additional columns are created that give the form access and form export rights in an individual column for each form. Form access rights columns have the naming pattern `[form_name]_access` and the form export rights columns have the naming pattern `[form_name]_export_access`.

`importUserRoles` invisibly returns the number of user roles that were added or modified.

`deleteUserRoles` invisibly returns the number of user roles that were deleted.
userRoleMethods

Functions

- exportUserRoles(): Export user roles from a project.
- importUserRoles(): Import user roles to a project.
- deleteUserRoles(): Delete user roles from a project.

See Also

exportUsers(), importUsers(), deleteUsers(),
exportUserRoleAssignments(), importUserRoleAssignments()

Examples

```r
## Not run:
unlockREDCap(connections = c(rcon = "project_alias"),
url = "your_redcap_url",
keyring = "API_KEYs",
envir = globalenv())

# Export users-roles
exportUserRoles(rcon)

# Export user-roles without additional form access variables
exportUsersRoles(rcon,
form_rights = FALSE)

# Export users as raw data
exportUserRoles(rcon,
labels = FALSE)

# Import new permissions
NewData <- data.frame(unique_role_name = "KN439U",
design = 0,
api_export = 1,
api_import = "No Access")
importUserRoles(rcon,
data = NewData)

# Remove a user from a project
deleteUserRoles(rcon,
user_roles = "KN439U")

## End(Not run)
```
validateImport

Validate Data Frames for Import

Description

Validates the variables in a data frame prior to attempting an import to REDCap.

Usage

validateImport(data, meta_data, logfile = "")

Arguments

data data.frame being prepared for import to REDCap.
meta_data REDCap database meta data.
logfile A character string giving the filepath to which the results of the validation are printed. If "", the results are printed in the console.

Details

validateImport is called internally by importRecords and is not available to the user.

Each variable is validated by matching the type of variable with the type listed in the REDCap database.

Although the log messages will indicate a preference for dates to be in mm/dd/yyyy format, the function will accept mm/dd/yy, yyyy-mm-dd, yyyy/mm/dd, and yyyymmdd formats as well. When possible, pass dates as Date objects or POSIXct objects to avoid confusion. Dates are also compared to minimum and maximum values listed in the data dictionary. Records where a date is found out of range are allowed to import and a message is printed in the log.

For continuous/numeric variables, the values are checked against the minimum and maximum allowed in the data dictionary. Records where a value is found out of range are allowed to import and a message is printed in the log.

ZIP codes are tested to see if they fit either the five digit or five digit + four format. When these conditions are not met, the data point is deleted and a message printed in the log.

YesNo fields permit any of the values 'yes', 'no', '0', '1' to be imported to REDCap with 0=No, and 1=Yes. The values are converted to lower case for validation, so any combination of lower and upper case values will pass (ie, the data frame is not case-sensitive).

TrueFalse fields will accept 'TRUE', 'FALSE', 0, 1, and logical values and are also not case-sensitive.

Radio and dropdown fields may have either the coding in the data dictionary or the labels in the data dictionary. The validation will use the meta data to convert any matching values to the appropriate coding before importing to REDCap. Values that cannot be reconciled are deleted with a message printed in the log. These variables are not case-sensitive.

Checkbox fields require a value of "Checked", "Unchecked", "0", or "1". These are currently case-sensitive. Values that do not match these are deleted with a warning printed in the log.
Phone numbers are required to be ten digit numbers. The phone number is broken into three parts: 1) a three digit area code, 2) a three digit exchange code, and 3) a four digit station code. The exchange code must start with a number from 2-9, followed by 0-8, and then any third digit. The exchange code starts with a number from 2-9, followed by any two digits. The station code is four digits with no restrictions.

E-mail addresses are considered valid when they have three parts. The first part comes before the @ symbol, and may be number of characters from a-z, A-Z, a period, underscore, percent, plus, or minus. The second part comes after the @, but before the period, and may consist of any number of letters, numbers, periods, or dashes. Finally, the string ends with a period then anywhere from two to six letters.

---

**vectorToApiBodyList**  
*Convert R Vector To List for the API Call Body*

**Description**

Converts an R vector to a list that will be suitable for makeApiCall.

**Usage**

```r
vectorToApiBodyList(vector, parameter_name)
```

**Arguments**

- `vector`  
  An atomic vector.

- `parameter_name`  
  character(1). The REDCap API parameter name.

**Examples**

```r
## Not run:
vectorToApiBodyList(1:3, "records")
## End(Not run)
```

---

**widerRepeated**  
*Transform Data Into Wide Format*

**Description**

Converts a dataframe into wide format given a single REDCap form. This function assumes that the Records argument is the result of `exportRecordsTyped`, and that all empty values have been previously dropped. This will only widen data frames that have a unique identification variable (e.g. `record_id`), "redcap_event_name" and "redcap_repeat_instrument" in the fields. Otherwise, the data passed in will be returned unchanged.
writeDataForImport

Prepare a Data Frame for Import Through the API

Description

Converts a dataframe into a character value in the format of a CSV for import through the API.

Usage

writeDataForImport(data)

Arguments

data data.frame to be imported to the API
## Index

* **datasets**
  - fieldValidationAndCasting, 57
  - redcapDataStructures, 124
  - [.redcapFactor (Extraction), 51

allocationTable, 4
allocationTable_offline
  - (allocationTable), 4
armsMethods, 6
as.list.redcapCodebook
  - (assembleCodebook), 9
assembleCodebook, 9

castCheckCode
  - (fieldValidationAndCasting), 57
castCheckCodeCharacter
  - (fieldValidationAndCasting), 57
castChecked
  - (fieldValidationAndCasting), 57
castCheckedCharacter
  - (fieldValidationAndCasting), 57
castCheckForImport
  - (fieldValidationAndCasting), 57
castCheckForImport()
  - (fieldValidationAndCasting), 57
castCheckLabel
  - (fieldValidationAndCasting), 57
castCheckLabelCharacter
  - (fieldValidationAndCasting), 57
castLogical
  - (fieldValidationAndCasting), 57
castRaw
  - (fieldValidationAndCasting), 57
castTimeHHMM
  - (fieldValidationAndCasting), 57
castTimeMMSS
  - (fieldValidationAndCasting), 57
castCheckForImport()
  - (fieldValidationAndCasting), 57
castLabel
  - (fieldValidationAndCasting), 57
castLabelCharacter
  - (fieldValidationAndCasting), 57
castLogical
  - (fieldValidationAndCasting), 57
castRaw
  - (fieldValidationAndCasting), 57
castTimeHHMM
  - (fieldValidationAndCasting), 57
castTimeMMSS
  - (fieldValidationAndCasting), 57
castCheckForImport()
  - (fieldValidationAndCasting), 57
castLabel
  - (fieldValidationAndCasting), 57
castLabelCharacter
  - (fieldValidationAndCasting), 57
castLogical
  - (fieldValidationAndCasting), 57
castRaw
  - (fieldValidationAndCasting), 57
castTimeHHMM
  - (fieldValidationAndCasting), 57
castTimeMMSS
  - (fieldValidationAndCasting), 57
dagAssignmentMethods, 19
dagMethods, 21
default_cast_character
  - (fieldValidationAndCasting), 57
default_cast_no_factor
  - (fieldValidationAndCasting), 57
deleArms (armsMethods), 6
deleteDags (dagMethods), 21
deleteDags()
  - (dagMethods), 20, 137
deleteEvents (eventsMethods), 27
deleteFileRepository
  - (fileRepositoryMethods), 66
deleteFileRepository()
  - (fileRepositoryMethods), 66
deleteFiles (fileMethods), 62
deleteFromFileRepository
  - (fromFileRepositoryMethods), 71
deleteFromFileRepository()
  - (fromFileRepositoryMethods), 71
deleteRecords, 23
deleteRecords()
  - (eventsMethods), 78, 107, 111
deleteUserRoles (userRoleMethods), 147
deleteUserRoles()
  - (userRoleMethods), 144, 147
deleteUsers (userMethods), 140
deleteUsers()
  - (userMethods), 146, 151
exportMetaData(), 18
exportProjectXml(). 18
exportReports (recordsMethods), 107
exportReports(), 25, 31, 78, 107, 117
exportReports_offline (recordsMethods), 107
exportReportsTyped
(recordsTypedMethods), 112
exportReportsTyped(), 25, 27, 30, 31, 52, 55, 59, 60, 62, 70, 78, 79, 107, 111, 129, 132, 133, 154
exportRepeatingInstrumentsEvents
(repeatingInstrumentMethods), 126
exportSurveyLink (surveyMethods), 133
exportSurveyParticipants
(surveyMethods), 133
exportSurveyQueueLink (surveyMethods), 133
exportSurveyReturnCode (surveyMethods), 133
exportUserDagAssignments
(dagAssignmentMethods), 19
exportUserDagAssignments(). 23, 137
exportUserRoleAssignments
(userRoleAssignmentMethods), 145
exportUserRoleAssignments(). 144, 151
exportUserRoleAssignments(userRoleMethods), 147
exportUserRoles (userRoleMethods), 144, 147
exportUserRoles(). 144, 147
exportUsers (userMethods), 140
exportUsers(). 143, 146, 149, 151
exportVersion, 50
Extraction, 51
fieldCastingFunctions, 52
fieldCastingFunctions(). 61
fieldChoiceMapping, 56
fieldToVar, 56
fieldValidationAndCasting, 57
fieldValidationAndCasting(). 31, 53, 55, 115–117
fileMethods, 62
fileRepositoryMethods, 66
fundamentalRedcapProjectInfo, 25
fileRepositoryMethods, 43
fileMethods, 40, 42, 105
exportFiles(), 40, 42, 105
exportFilesMultiple, 40
exportFilesMultiple(). 65
exportFromFileRepository
(fromFileRepositoryMethods), 71
exportFromFileRepository(), 15, 39, 68, 105
eventsMethods, 27
eventMethods, 37, 43, 48, 85, 89
eventLogging, 44
eventLogging(). 12
exportMappings (mappingMethods), 84
exportMappings(), 29, 37, 44, 48, 89
exportMetaData (metaDataMethods), 87
exportMetaData(), 11, 37, 43, 48, 85
eventNextRecordName
(recordsManagementMethods), 105
eventNextRecordName(), 111
exportPdf, 46
exportPdf(), 37, 44, 85, 89, 105
exportProjectInformation
(projectInformationMethods), 97
exportProjectXml(), 18
deprecated_redcapProjectInfo, 25
dropRepeatingNA, 27
eventMethods, 27
exportArms (armsMethods), 6
exportBulkRecords, 30
exportBulkRecords(). 117
exportBundle, 33
exportBundle(), 25
exportDags (dagMethods), 21
exportDags(), 20, 137
exportDataQuality, 34
exportEvents (eventsMethods), 27
exportExternalCoding, 35
exportExternalCoding(), 121
exportFieldNameNames, 36
exportFieldNameNames(), 48, 85, 89
exportFieldNameNamesArgs
(exportFieldNameNames), 36
exportFileRepository
(fileRepositoryMethods), 66
exportFileRepository(), 15, 39, 72, 105
exportFileRepositoryListing, 38
exportFileRepositoryListing(), 15, 68, 73
exportFiles (fileMethods), 62
exportFiles(), 40, 42, 105
exportFilesMultiple, 40
exportFilesMultiple(). 65
exportFromFileRepository
(fromFileRepositoryMethods), 71
exportFromFileRepository(), 15, 39, 68, 105
eventsMethods, 27
exportFields (fileMethods), 37, 43, 48, 85, 89
eventLogging, 44
eventLogging(), 12
exportMappings (mappingMethods), 84
exportMappings(), 29, 37, 44, 48, 89
exportMetaData (metaDataMethods), 87
exportMetaData(), 11, 37, 43, 48, 85
exportNextRecordName
(recordsManagementMethods), 105
exportNextRecordName(), 111
exportPdf, 46
exportPdf(), 37, 44, 85, 89, 105
exportProjectInformation
(projectInformationMethods), 97
exportProjectXml(), 18
exportRecords (recordsMethods), 107
exportRecords(), 25, 31, 78, 107, 117
exportRecords_offline (recordsMethods), 107
exportRecordsTyped
(recordsTypedMethods), 112
exportRecordsTyped(), 25, 27, 30, 31, 52, 55, 59, 60, 62, 70, 78, 79, 107, 111, 129, 132, 133, 154
exportRepeatingInstrumentsEvents
(repeatingInstrumentMethods), 126
exportReports (recordsMethods), 107
exportReports(), 31, 107, 117
exportReportsTyped
(recordsTypedMethods), 112
exportReportsTyped(), 27, 55, 62, 70, 107, 111, 129, 133
exportSurveyLink (surveyMethods), 133
exportSurveyParticipants
(surveyMethods), 133
exportSurveyQueueLink (surveyMethods), 133
exportSurveyReturnCode (surveyMethods), 133
exportUserDagAssignments
(dagAssignmentMethods), 19
exportUserDagAssignments(). 23, 137
exportUserRoleAssignments
(userRoleAssignmentMethods), 145
exportUserRoleAssignments(). 144, 151
exportUserRoleAssignments(userRoleMethods), 147
exportUserRoleAssignments(userRoleMethods), 144, 147
exportUserRoles (userRoleMethods), 140
exportUserRoles(). 143, 146, 149, 151
exportVersion, 50
Extraction, 51
fieldCastingFunctions, 52
fieldCastingFunctions(), 61
fieldChoiceMapping, 56
fieldToVar, 56
fieldValidationAndCasting, 57
fieldValidationAndCasting(). 31, 53, 55, 115–117
fileMethods, 62
fileRepositoryMethods, 66
fileRepositoryPath, 70
filterEmptyRow, 70
format.invalid(invalidSummary), 79
fromFromFileRepositoryMethods, 71
getprojectIdFields, 73
globalenv(), 139
guessCast(fieldCastingFunctions), 52
guessDate(), 31, 118, 129, 131, 154
guessDate(fieldCastingFunctions), 52
guessDate(), 32, 118, 131, 154

httr::config(), 122
httr::httr_options(), 122
httr::POST(), 8, 15, 17, 19, 22, 25, 28, 35, 37, 39, 41, 43, 45, 47, 49, 51, 64, 67, 72, 77, 81, 82, 85, 88, 95, 97, 102, 106, 110, 115, 120, 125, 127, 135, 137, 142, 146, 148

importArms(armsMethods), 6
importDags(dagMethods), 21
importDags(), 20, 137
importEvents(eventsMethods), 27
importFileRepository
  (fileRepositoryMethods), 66
importFileRepository(), 15, 39, 73
importFiles(fileMethods), 62
importFiles(), 75
importFileToRecord, 74
importFileToRecord(), 65
importMappings(mappingMethods), 84
importMappings(), 29, 37, 44, 48, 89
importMetaData(metaDataMethods), 87
importMetaData(), 37, 43, 48, 85
importProjectInformation
  (projectInformationMethods), 97
importRecords, 76
importRecords(), 25, 75, 107, 111
importRepeatingInstrumentsEvents
  (repeatingInstrumentEvents), 126
importRepeatingInstrumentsEvents(), 102
importToFileRepository
  (fromFileRepositoryMethods), 71
importToFileRepository(), 15, 39, 68
importUserDagAssignments
  (dagAssignmentMethods), 19
importUserDagAssignments(), 23, 137
importUserRoleAssignments
  (userRoleAssignmentMethods), 145
importUserRoleAssignments(), 144, 151
importUserRoleAssignments(userRoleMethods), 147
importUserRoleAssignments(), 93, 144, 147
importUserRoleAssignments(userMethods), 140
importUserRoleAssignments(), 93, 146, 151
invalidSummary, 79
isNAOrNull
  (fieldValidationAndCasting), 57
isNAOrNull(), 53, 114
isZeroCodedCheckField, 80

makeApiCall, 81
mChoiceCast(fieldCastingFunctions), 52
mChoiceCast(), 32, 118, 131, 154
metaDataMethods, 87
metaDataMethodsArgs(metaDataMethods), 87
missingSummary, 90
missingSummary(), 92
missingSummary_offline
  (missingSummary), 90
na_values(fieldValidationAndCasting), 57
offlineConnection(redcapConnection), 119
offlineConnection(), 96
parseBranchingLogic, 91
prepUserImportData, 92
prepUserImportData(), 143, 149
preserveProject, 94
preserveProject(), 123
print.invalid(invalidSummary), 79
print.redcapApiConnection
  (redcapConnection), 119
print.redcapFactor (Extraction), 51
print.redcapOfflineConnection
  (redcapConnection), 119
projectInformationMethods, 97
purgeProject(purgeRestoreProject), 100
purgeProject(), 96
purgeRestoreProject, 100
raw_cast (fieldValidationAndCasting), 57
readPreservedProject (preserveProject), 94
readPreservedProject(), 123
recastRecords (fieldCastingFunctions), 52
recastRecords(), 31, 118, 131, 154
recastCheck, 103
reconstituteFileFromExport, 104
recordsManagementMethods, 105
recordsMethods, 107
recordsTypedMethods, 112
REDCAP_METADATA_FIELDTYPE
 (redcapDataStructures), 124
REDCAP_METADATA_VALIDATION_TYPE
 (redcapDataStructures), 124
REDCAP_PROJECT_PURPOSE
 (redcapDataStructures), 124
REDCAP_REPEAT_INSTRUMENT_STRUCTURE
 (redcapDataStructures), 124
redcapConnection, 119
redcapConnection(), 82, 139, 140
redcapDataStructures, 124
redcapError, 155
redcapError(), 8, 15, 17, 19, 22, 24, 28, 35,
37, 39, 41, 43, 45, 47, 49, 50, 64, 67,
72, 77, 85, 88, 94, 97, 102, 106, 110,
113, 125, 127, 135, 137, 142, 146, 148
redcapFactorFlip, 126
redcapProjectInfo
 (deprecated_redcapProjectInfo), 25
renameRecord
 (recordsManagementMethods), 105
renameRecord(), 111
repeatingInstrumentEventMethods
 (repeatingInstrumentMethods), 126
repeatingInstrumentMethods, 126
restoreProject (purgeRestoreProject), 100
restoreProject(), 96
reviewInvalidRecords, 128
reviewInvalidRecords(), 31, 54, 55, 117
skip_validation
 (fieldValidationAndCasting), 57
splitForms, 130
splitForms(), 32, 55, 118, 154
stringCleanup, 131
stringCleanup(), 132
stripHTMLandUnicode, 132
stripHTMLTags (stringCleanup), 131
stripHTMLTags(), 133
stripUnicode (stringCleanup), 131
stripUnicode(), 133
summary_invalid (invalidSummary), 79
surveyMethods, 133
switchDag(), 20, 23
switchDagArgs (switchDag), 136
syncUnderscoreCodings, 137

unitsFieldAnnotation
 (stripHTMLandUnicode), 132
unlockREDCap, 139
unlockREDCap(), 123
userMethods, 140
userRoleAssignmentMethods, 145
userRoleMethods, 147
utils::read.csv(), 110

valChoice (fieldValidationAndCasting), 57
valChoice(), 53, 115
validateImport, 152
validateImport(), 78
validateRedcapData
 (redcapDataStructures), 124
valPhone (fieldValidationAndCasting), 57
valRx (fieldValidationAndCasting), 57
valRx(), 53, 115
valSkip (fieldValidationAndCasting), 57
vectorToApiBodyList, 153

warnOfZeroCodedCheckCasting
 (isZeroCodedCheckField), 80
warnZeroCodedFieldPresent
 (isZeroCodedCheckField), 80
widerRepeated, 153
widerRepeated(), 32, 55, 118, 131
writeDataForImport, 154