Package: redcap (via r-universe)

October 28, 2024

Title R Utilities For REDCap

Version 0.2.0

Description R utilities for interacting with the REDCap API.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Depends R (>= 2.10)

Imports httr, dplyr, tidyr, purrr, rlang, readr, lubridate, chron, stringr, xml2, lifecycle, dbc

Suggests testthat, covr

Remotes epicentre-msf/dbc

URL https://github.com/epicentre-msf/redcap

BugReports https://github.com/epicentre-msf/redcap/issues

Repository https://epicentre-msf.r-universe.dev

RemoteUrl https://github.com/epicentre-msf/redcap

RemoteRef HEAD

RemoteSha 4d00109b129978472f9e3d33f572baeace085330

Contents

`delete_records` *Delete records from a REDCap project*

Description

Delete records from a REDCap project

Usage

```
delete_records(conn, records)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)
`records`	Character vector of record IDs to delete

Value

An integer, the number of records successfully deleted

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

# delete all records associated with IDs "P004" and "P007"
delete_records(conn, records = c("P004", "P007"))

## End(Not run)
```

`fetch_database` *Fetch records from multiple REDCap forms, returning separate list elements for each form*

Description

Wrapper to `fetch_records` that's vectorized over forms (i.e. instruments). Returns a list whose elements are `tibble`-style data frames corresponding to each requested form.

Usage

```
fetch_database(
  conn,
  forms = NULL,
  names_fn = function(x) x,
  records = NULL,
  records_omit = NULL,
  id_field = TRUE,
  rm_empty = TRUE,
  value_labs = TRUE,
  value_labs_fetch_raw = FALSE,
  header_labs = FALSE,
  checkbox_labs = FALSE,
  use_factors = FALSE,
  times_chron = TRUE,
  date_range_begin = NULL,
  date_range_end = NULL,
  fn_dates = parse_date,
  fn_dates_args = list(orders = c("Ymd", "dmY")),
  fn_datetimes = lubridate::parse_date_time,
  fn_datetimes_args = list(orders = c("Ymd HMS", "Ymd HM")),
  na = c("", "NA"),
  dag = TRUE,
  batch_size = 100L,
```

```
  batch_delay = 0.5,
  form_delay = 0.5,
  double_resolve = FALSE,
  double_remove = FALSE,
  double_sep = "--",
  fns = NULL
)
```

Arguments

conn	A REDCap API connection object (created with rconn)
forms	Character vector of forms (i.e. instruments) to fetch data for. Set to NULL (the default) to fetch all forms in the project.
names_fn	Function for creating custom list element names given a vector of form names. Defaults to an identity function in which case element names will correspond exactly to form names.
records	Character vector of record IDs to fetch. Set to NULL (the default) to fetch all record IDs corresponding to the selected form(s).
records_omit	Character vector of record IDs to ignore. Set to NULL (the default) to *not* ignore any records. If a given record ID appears in both argument records and records_omit, argument records_omit takes precedence and that record will not be returned.
id_field	Logical indicating whether to always include the 'record ID' field (defined in REDCap to be the first variable in the project codebook) in the API request, even if it's not specified in argument fields. Defaults to TRUE. The record ID field is defined within the first form of a REDCap project, and so API requests for other forms will not include the record ID field by default (unless it's explicitly requested with argument fields). The id_field argument is a shortcut to avoid having to always explicitly request the record ID field.
rm_empty	Logical indicating whether to remove rows for which all fields from the relevant form(s) are missing. See section **Removing empty rows**. Defaults to TRUE.
value_labs	Logical indicating whether to return value labels (TRUE) or raw values (FALSE) for categorical REDCap variables (radio, dropdown, yesno, checkbox). Defaults to TRUE to return labels.
value_labs_fetch_raw	Logical indicating whether to request raw values for categorical REDCap variables (radio, dropdown, yesno, checkbox), which are then transformed to labels in a separate step when value_labs = TRUE. Primarily used for troubleshooting issues with the REDCap API returning fewer records than expected when given certain combinations of request parameters.
header_labs	Logical indicating whether to export column names as labels (TRUE) or raw variable names (FALSE). Defaults to FALSE to return raw variable names.
checkbox_labs	Logical indicating whether to export checkbox labels (TRUE) or statuses (i.e. "Unchecked" or "Checked") (FALSE). Defaults to FALSE to export statuses. Note this argument is only relevant when value_labs is TRUE — if value_labs is FALSE checkbox variables will always be exported as raw values (usually "0"/"1").

use_factors — Logical indicating whether categorical REDCap variables (radio, dropdown, yesno, checkbox) should be returned as factors. Factor levels can either be raw values (e.g. "0"/"1") or labels (e.g. "No"/"Yes") depending on arguments value_labs and checkbox_labs. Defaults to FALSE.

times_chron — Logical indicating whether to reclass time variables using chron::times (TRUE) or leave as character HH:MM format (FALSE). Defaults to TRUE. Note this only applies to variables of REDCap type "Time (HH:MM)", and not "Time (MM:SS)".

date_range_begin — Fetch only records created or modified *after* a given date-time. Use format "YYYY-MM-DD HH:MM:SS" (e.g., "2017-01-01 00:00:00" for January 1, 2017 at midnight server time). Defaults to NULL to omit a lower time limit.

date_range_end — Fetch only records created or modified *before* a given date-time. Use format "YYYY-MM-DD HH:MM:SS" (e.g., "2017-01-01 00:00:00" for January 1, 2017 at midnight server time). Defaults to NULL to omit a lower time limit.

fn_dates — Function to parse REDCap date variables. Defaults to parse_date, an internal wrapper to lubridate::parse_date_time. If date variables have been converted to numeric (e.g. by writing to Excel), set to e.g. lubridate::as_date to convert back to dates.

fn_dates_args — List of arguments to pass to fn_dates. Can set to empty list list() if using a function that doesn't take any arguments.

fn_datetimes — Function to parse REDCap datetime variables. Defaults to lubridate::parse_date_time.

fn_datetimes_args — List of arguments to pass to fn_datetimes. Can set to empty list list() if using a function that doesn't take any arguments.

na — Character vector of strings to interpret as missing values. Passed to readr::read_csv. Defaults to c("", "NA").

dag — Logical indicating whether to export the redcap_data_access_group field (if used in the project). Defaults to TRUE.

batch_size — Number of records to fetch per batch. Defaults to 100L. Set to Inf or NA to fetch all records at once.

batch_delay — Delay in seconds between fetching successive batches, to give the REDCap server time to respond to other requests. Defaults to 0.5.

form_delay — Delay in seconds between fetching successive forms, to give the REDCap server time to respond to other requests. Defaults to 0.5.

double_resolve — Logical indicating whether to resolve double-entries (i.e. records entered in duplicate using REDCap's Double Data Entry module), by filtering to the lowest entry number associated with each unique record.

If a project uses double-entry, the record IDs returned by an "Export Records" API request will be a concatenation of the normal record ID and the entry number (1 or 2), normally separated by "–" (e.g. "P0285–1"). To resolve double entries we move the entry number portion of the ID to its own column (entry), identify all entries belonging to the same unique record, and retain only the row with the lowest entry number for each unique record.

Unique records are identified using the record ID column (after separating the entry number portion), and any of the following columns when present (accounting for argument `header_labs`): redcap_event_name (Redcap Event), redcap_repeat_instrument (Repeat Instrument), redcap_repeat_instance (Repeat Instance).

`double_remove` Logical indicating whether to *remove* double-entries (i.e. records entered in duplicate using REDCap's Double Data Entry module), by filtering out records where the record ID field contains pattern `double_sep` (see next argument), so that only merged records remain.

`double_sep` If `double_resolve` is `TRUE`, the string separator used to split the record ID field into the record ID and entry number. Defaults to "–".

`fns` Optional list of one or more functions to apply to each list element (i.e. each form). Could be used e.g. to filter out record IDs from test entries, create derived variables, etc. Each function should take a data frame returned by `fetch_records` as its first argument.

Value

A list of `tibble`-style data frames corresponding to each of the requested forms.

Removing empty rows

Depending on the database design, an "Export Records" API request can sometimes return empty rows, representing forms for which no data has been collected. For example, if forms **F1** and **F2** are part of the same event, and participant "P001" has form data for **F2** but not **F1**, an API request for **F1** will include a row for participant "P001" where all **F1**-specific fields are empty.

If argument `rm_empty` is `TRUE` (the default), `fetch_records()` will filter out such rows. The check for empty rows is based only on fields that are specific to the form(s) specified in argument `forms` — i.e. it excludes the record ID field, and generic fields like `redcap_event_name`, `redcap_data_access_group`, etc. The check for empty rows also accounts for checkbox fields, which, if argument `checkbox_labs` is `FALSE`, will be set to "Unchecked" in an empty form (rather than missing per se).

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

fetch_database(
  conn,
  forms = c("my_form1", "my_form2", "my_form3")
)

# use a custom fn to format the 'participant_id' column of each form
# the function must take a data frame as its first argument
format_ids <- function(x) {
```

```
  x$participant_id <- toupper(x$participant_id)
  x$participant_id <- gsub("[^[:alnum:]]+", "_", x$participant_id)
  x
}

fetch_database(
  conn,
  forms = c("my_form1", "my_form2", "my_form3"),
  fns = list(format_ids)
)

## End(Not run)
```

`fetch_records` *Fetch records for a REDCap project*

Description

Execute an "Export Records" API request to fetch a `tibble`-style data frame containing records for one or more REDCap instruments.

Usage

```
fetch_records(
  conn,
  forms = NULL,
  events = NULL,
  records = NULL,
  records_omit = NULL,
  fields = NULL,
  id_field = TRUE,
  rm_empty = TRUE,
  value_labs = TRUE,
  value_labs_fetch_raw = FALSE,
  header_labs = FALSE,
  checkbox_labs = FALSE,
  use_factors = FALSE,
  times_chron = TRUE,
  date_range_begin = NULL,
  date_range_end = NULL,
  fn_dates = parse_date,
  fn_dates_args = list(orders = c("Ymd", "dmY")),
  fn_datetimes = lubridate::parse_date_time,
  fn_datetimes_args = list(orders = c("Ymd HMS", "Ymd HM")),
  na = c("", "NA"),
  dag = TRUE,
  batch_size = 100L,
```

```
  batch_delay = 0.5,
  double_resolve = FALSE,
  double_remove = FALSE,
  double_sep = "--"
)
```

Arguments

conn	A REDCap API connection object (created with rconn)
forms	Character vector of forms (i.e. instruments) to fetch data for. Set to NULL (the default) to fetch all forms in the project.
events	Character vector of events to fetch. Must correspond to the selected forms. Set to NULL (the default) to fetch all events corresponding to the selected form(s).
records	Character vector of record IDs to fetch. Set to NULL (the default) to fetch all record IDs corresponding to the selected form(s).
records_omit	Character vector of record IDs to ignore. Set to NULL (the default) to *not* ignore any records. If a given record ID appears in both argument records and records_omit, argument records_omit takes precedence and that record will not be returned.
fields	Character vector of fields (i.e. variables) to fetch. Set to NULL (the default) to fetch all fields corresponding to the selected form(s).
id_field	Logical indicating whether to always include the 'record ID' field (defined in REDCap to be the first variable in the project codebook) in the API request, even if it's not specified in argument fields. Defaults to TRUE. The record ID field is defined within the first form of a REDCap project, and so API requests for other forms will not include the record ID field by default (unless it's explicitly requested with argument fields). The id_field argument is a shortcut to avoid having to always explicitly request the record ID field.
rm_empty	Logical indicating whether to remove rows for which all fields from the relevant form(s) are missing. See section **Removing empty rows**. Defaults to TRUE.
value_labs	Logical indicating whether to return value labels (TRUE) or raw values (FALSE) for categorical REDCap variables (radio, dropdown, yesno, checkbox). Defaults to TRUE to return labels.
value_labs_fetch_raw	Logical indicating whether to request raw values for categorical REDCap variables (radio, dropdown, yesno, checkbox), which are then transformed to labels in a separate step when value_labs = TRUE. Primarily used for troubleshooting issues with the REDCap API returning fewer records than expected when given certain combinations of request parameters.
header_labs	Logical indicating whether to export column names as labels (TRUE) or raw variable names (FALSE). Defaults to FALSE to return raw variable names.
checkbox_labs	Logical indicating whether to export checkbox labels (TRUE) or statuses (i.e. "Unchecked" or "Checked") (FALSE). Defaults to FALSE to export statuses. Note this argument is only relevant when value_labs is TRUE — if value_labs is FALSE checkbox variables will always be exported as raw values (usually "0"/"1").

use_factors Logical indicating whether categorical REDCap variables (radio, dropdown, yesno, checkbox) should be returned as factors. Factor levels can either be raw values (e.g. "0"/"1") or labels (e.g. "No"/"Yes") depending on arguments value_labs and checkbox_labs. Defaults to FALSE.

times_chron Logical indicating whether to reclass time variables using chron::times (TRUE) or leave as character HH:MM format (FALSE). Defaults to TRUE. Note this only applies to variables of REDCap type "Time (HH:MM)", and not "Time (MM:SS)".

date_range_begin Fetch only records created or modified *after* a given date-time. Use format "YYYY-MM-DD HH:MM:SS" (e.g., "2017-01-01 00:00:00" for January 1, 2017 at midnight server time). Defaults to NULL to omit a lower time limit.

date_range_end Fetch only records created or modified *before* a given date-time. Use format "YYYY-MM-DD HH:MM:SS" (e.g., "2017-01-01 00:00:00" for January 1, 2017 at midnight server time). Defaults to NULL to omit a lower time limit.

fn_dates Function to parse REDCap date variables. Defaults to parse_date, an internal wrapper to lubridate::parse_date_time. If date variables have been converted to numeric (e.g. by writing to Excel), set to e.g. lubridate::as_date to convert back to dates.

fn_dates_args List of arguments to pass to fn_dates. Can set to empty list list() if using a function that doesn't take any arguments.

fn_datetimes Function to parse REDCap datetime variables. Defaults to lubridate::parse_date_time.

fn_datetimes_args List of arguments to pass to fn_datetimes. Can set to empty list list() if using a function that doesn't take any arguments.

na Character vector of strings to interpret as missing values. Passed to readr::read_csv. Defaults to c("", "NA").

dag Logical indicating whether to export the redcap_data_access_group field (if used in the project). Defaults to TRUE.

batch_size Number of records to fetch per batch. Defaults to 100L. Set to Inf or NA to fetch all records at once.

batch_delay Delay in seconds between fetching successive batches, to give the REDCap server time to respond to other requests. Defaults to 0.5.

double_resolve Logical indicating whether to resolve double-entries (i.e. records entered in duplicate using REDCap's Double Data Entry module), by filtering to the lowest entry number associated with each unique record.

If a project uses double-entry, the record IDs returned by an "Export Records" API request will be a concatenation of the normal record ID and the entry number (1 or 2), normally separated by "–" (e.g. "P0285–1"). To resolve double entries we move the entry number portion of the ID to its own column (entry), identify all entries belonging to the same unique record, and retain only the row with the lowest entry number for each unique record.

Unique records are identified using the record ID column (after separating the entry number portion), and any of the following columns when present (accounting for argument header_labs): redcap_event_name (Redcap Event), redcap_repeat_instrument (Repeat Instrument), redcap_repeat_instance (Repeat Instance).

double_remove	Logical indicating whether to *remove* double-entries (i.e. records entered in duplicate using REDCap's Double Data Entry module), by filtering out records where the record ID field contains pattern double_sep (see next argument), so that only merged records remain.
double_sep	If double_resolve is TRUE, the string separator used to split the record ID field into the record ID and entry number. Defaults to "–".

Value

A tibble-style data frame containing the requested records

Removing empty rows

Depending on the database design, an "Export Records" API request can sometimes return empty rows, representing forms for which no data has been collected. For example, if forms **F1** and **F2** are part of the same event, and participant "P001" has form data for **F2** but not **F1**, an API request for **F1** will include a row for participant "P001" where all **F1**-specific fields are empty.

If argument rm_empty is TRUE (the default), fetch_records() will filter out such rows. The check for empty rows is based only on fields that are specific to the form(s) specified in argument forms — i.e. it excludes the record ID field, and generic fields like redcap_event_name, redcap_data_access_group, etc. The check for empty rows also accounts for checkbox fields, which, if argument checkbox_labs is FALSE, will be set to "Unchecked" in an empty form (rather than missing per se).

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

fetch_records(conn, forms = "my_form")

## End(Not run)
```

---

generate_queries — *Generate data validation queries for a REDCap project based on branching logic specified in the project codebook*

Description

Generates two types of data validation queries using the project codebook (see meta_dictionary) and translate_logic:

1. **Field missing**: Branching logic evaluates to TRUE (if specified), but field is missing. By default only applies to required fields (required_field == "y") (can modify with argument non_required).

2. **Field not missing**: Branching logic evaluates to FALSE but field is not missing. Applies to any field with branching logic.

Usage

```
generate_queries(
  conn,
  forms = NULL,
  dict = meta_dictionary(conn, forms = forms, expand_checkbox = FALSE),
  lang = "en",
  query_types = "both",
  non_required = FALSE,
  drop_redundant = FALSE,
  field_nchar_max = 80L,
  on_error = "warn"
)
```

Arguments

conn	A REDCap API connection object (created with rconn)
forms	Character vector of forms (i.e. instruments) to include. Set to NULL (the default) to generate queries for all forms in the project.
dict	Metadata dictionary. By default is fetched automatically with meta_dictionary, but it's included as an argument here to allow the user to modify the dictionary before passing to generate_queries (e.g. to correct bugs in branching logic). If passing a modified version, make sure it is initially fetched with argument expand_checkbox = FALSE.
lang	Query language, either English ("en") or French ("fr"). Defaults to "en".
query_types	Which type of queries to generate (see **Description** above). Options are "missing", "not missing", or "both". Defaults to "both".
non_required	Logical indicating whether to include non-required fields in queries of type "Field missing". Defaults to FALSE.
drop_redundant	Logical indicating whether to simplify expressions by removing redundant components from expressions that test both for equality and inequality with the same variable. E.g.: var == "X" & var != "" becomes var == "X"
field_nchar_max	Integer indicating the maximum number of characters to allow in field name labels before they are truncated and appended with "...". Defaults to 80L.
on_error	What to do if one or more elements of statements can't be translated into valid R expressions? Options are "ignore" (return NA for relevant elements), "warn" (return NA for relevant elements and give warning), or "fail" (throw error). Defaults to "warn".

Value

A tibble-style data frame specifying queries, with the following 7 columns:

query_id Unique query identifier based on form name and integer sequence

field_name Field name (from REDCap dictionary, see `meta_dictionary`)

form_name Form name (from REDCap dictionary)

required Is it a required field in REDCap dictionary ("y" or <NA>) ?

description Description of query (e.g. "Missing: [Signed consent forms?]")

suggestion Suggestion for query resolution. A human-readable translation of query expression (e.g. If [Is the participant 18 years or older?] is "Yes", item [Signed consent forms?] should not be missing)

branching_logic Branching logic for given field (from REDCap dictionary)

query R-style query expression (can be evaluated with `queryr::query`)

`import_records` *Import records into a REDCap project*

Description

Import records into a REDCap project

Usage

```
import_records(
  conn,
  data,
  type = c("flat", "eav"),
  overwrite = "normal",
  return = "count"
)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)
`data`	A data.frame containing record data to import into REDCap
`type`	One of: • "flat": data in wide-form with one record per row (default) • "eav": data in long-form with one row per participant/instance/field (data should have columns "record", "field_name", and "value", and if longitudinal then also "redcap_event_name" and "redcap_repeat_instance")
`overwrite`	Overwrite behaviour. Either "normal" to prevent missing values from overwriting data, or "overwrite" to allow data to be overwritten with missing values. Defaults to "normal".
`return`	What to return. Use "count" to return a count of imported records, "ids" to return a vector of the IDs that were imported, or "nothing" to return nothing. Defaults to "count".

Value

Depends on argument `return`. Either a count of imported records, a vector of record IDs, or nothing.

`meta_arms` *Export REDCap project arms*

Description

Execute an "Export Arms" API request to fetch the "arms" (number and name) associated with a REDCap project.

Usage

```
meta_arms(conn)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)

Value

A `tibble`-style data frame with 2 columns:

- `arm_num`
- `name`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_arms(conn)

## End(Not run)
```

`meta_dictionary` *Fetch variable dictionary for a REDCap project*

Description

Execute an "Export Metadata (Data Dictionary)" API request to fetch a `tibble`-style data frame containing the project codebook (field names, types, labels, choices, validation, etc.).

Usage

```
meta_dictionary(
  conn,
  forms = NULL,
  expand_checkbox = TRUE,
  add_complete = FALSE,
  cols_omit = c("section_header", "custom_alignment", "question_number",
    "matrix_group_name", "matrix_ranking")
)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)
`forms`	Character vector of forms (i.e. instruments) to include in the return. Set to `NULL` (the default) to return dictionary entries for all forms in the project.
`expand_checkbox`	Logical indicating whether to expand checkbox variables. Defaults to `TRUE`. Unlike an "Export Records" API request (see `fetch_records`), which returns 1 column for each checkbox *option*, an "Export Metadata (Data Dictionary)" request returns a single row for each *field* — including checkbox fields. Thus, the `field_name` and `field_label` entries for checkbox variables in the data dictionary will never exactly match the respective column names or values returned by `fetch_records`. When `expand_checkbox` is `TRUE`, rows for checkbox fields are expanded to 1 row per checkbox *option*, so that dictionary entries for `field_name`, `field_label`, and `choices` will always match the relevant entries returned by `fetch_records`.
`add_complete`	Logical indicating whether to add "{form}_complete" fields to the dictionary, one for each form included in the return. These will be of field_type "radio" with possible choices "0, Incomplete | 1, Unverified | 2, Complete". Defaults to `FALSE`.
`cols_omit`	Character vector of dictionary columns to omit from the return for brevity. Set to `NULL` to return all columns.

Value

A tibble-style data frame containing the project dictionary. Note that some of the returned column names are shortened versions of the original column names returned by the API:

Original	Returned
select_choices_or_calculations	choices
text_validation_type_or_show_slider_number	validation
text_validation_min	validation_min
text_validation_max	validation_max

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_dictionary(conn)

## End(Not run)
```

---

meta_events *Fetch event names and labels for a REDCap project*

Description

Execute an "Export Events" API request to fetch a tibble-style data frame containing event names and labels. Note that this request type is not available for 'classic projects', from which event details cannot be exported.

Usage

```
meta_events(conn, on_error = "fail")
```

Arguments

conn	A REDCap API connection object (created with rconn)
on_error	How to handle errors returned by the API (e.g. events cannot be exported for classic projects). Set to "fail" to halt execution and return the API error message, or "null" to ignore the error and return NULL. Defaults to "fail".

Value

A tibble-style data frame with 7 columns:

- event_name
- arm_num
- day_offset
- offset_min
- offset_max
- unique_event_name
- custom_event_label

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_events(conn)

## End(Not run)
```

---

meta_factors *Fetch field option values and labels for factor-type variables in a REDCap project*

Description

Converts field options and labels from the metadata dictionary (see meta_dictionary), which has 1 row per variable and field options in compact string-form, to long form with 1 row per option. E.g.:

Dictionary version (1 row per variable):

field_name	form_name	field_type	choices
consented	enrollment	radio	0, No | 1, Yes

Long format (1 row per option):

field_name	form_name	field_type	value	label
consented	enrollment	radio	0	No
consented	enrollment	radio	1	Yes

Usage

```
meta_factors(
  conn,
  forms = NULL,
  expand_checkbox = TRUE,
  add_complete = FALSE,
  types = c("radio", "yesno", "dropdown", "checkbox")
)
```

Arguments

conn	A REDCap API connection object (created with rconn)
forms	Character vector of forms (i.e. instruments) to include. Set to NULL (the default) to return field options for all forms in the project.
expand_checkbox	Logical indicating whether to expand checkbox variables. Defaults to TRUE. Unlike an "Export Records" API request (see fetch_records), which returns 1 column for each checkbox *option*, an "Export Metadata (Data Dictionary)" request returns a single row for each *field* — including checkbox fields. Thus, the field_name and field_label entries for checkbox variables in the data dictionary will never exactly match the respective column names or values returned by fetch_records. When expand_checkbox is TRUE, rows for checkbox fields are expanded to 1 row per checkbox *option*, so that dictionary entries for field_name, field_label, and choices will always match the relevant entries returned by fetch_records.
add_complete	Logical indicating whether to add "{form}_complete" fields to the dictionary, one for each form included in the return. These will be of field_type "radio" with possible choices "0, Incomplete | 1, Unverified | 2, Complete". Defaults to FALSE.
types	Character vector of variable types to return field options for. Defaults to c("radio", "yesno", "dropdown", "checkbox").

Value

A tibble-style data frame with 6 columns:

- field_name
- form_name
- field_type
- field_label
- value
- label

See Also

meta_dictionary

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_factors(conn)

## End(Not run)
```

`meta_fields` *Fetch exported field names for a REDCap project*

Description

Execute an "Export List of Export Field Names" API request to fetch a `tibble`-style data frame containing field (i.e. variable) names as listed in the project codebook (column `original_field_name`) and the corresponding exported variable name(s) (column `export_field_name`).

Original and exported field names will be identical except in the case of checkbox-type variables. A given checkbox variable (e.g. "patient_status") will have a single entry in the codebook (i.e. `field_name` = "patient_status"), but will be exported as multiple variables — one for each possible choice value.

original_field_name	choice_value	export_field_name
patient_status	1	patient_status___1
patient_status	2	patient_status___2
patient_status	3	patient_status___3
patient_status	88	patient_status___88

Usage

```
meta_fields(conn)
```

Arguments

conn	A REDCap API connection object (created with `rconn`)

Value

A `tibble`-style data frame with 3 columns:

- `original_field_name`
- `choice_value`
- `export_field_name`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_fields(conn)

## End(Not run)
```

`meta_forms` *Fetch instrument names and labels for a REDCap project*

Description

Execute an "Export Instrument (Data Entry Forms)" API request to fetch a `tibble`-style data frame containing instrument names and labels.

Usage

```
meta_forms(conn)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)

Value

A `tibble`-style data frame with 2 columns:

- `instrument_name`
- `instrument_label`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_forms(conn)

## End(Not run)
```

`meta_mapping` *Fetch instrument-event mappings for a REDCap project*

Description

Execute an "Export Instrument-Event Mappings" API request to fetch a `tibble`-style data frame containing the mapping between instruments and events. Note that this request type is not available for 'classic projects', from which event details cannot be exported.

Usage

```
meta_mapping(conn, on_error = "fail")
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)
`on_error`	How to handle errors returned by the API (e.g. events cannot be exported for classic projects). Set to "fail" to halt execution and return the API error message, or "null" to ignore the error and return `NULL`. Defaults to "fail".

Value

A `tibble`-style data frame with 3 columns:

- `arm_num`
- `unique_event_name`
- `form`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_mapping(conn)

## End(Not run)
```

`meta_repeating` *Fetch repeating instrument names and labels for a REDCap project*

Description

Execute an "Export Repeating Instruments and Events" API request to fetch a `tibble`-style data frame containing repeating instrument names and labels.

Usage

```
meta_repeating(conn, on_error = "fail")
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)
`on_error`	How to handle errors returned by the API (e.g. events cannot be exported for classic projects). Set to "fail" to halt execution and return the API error message, or "null" to ignore the error and return `NULL`. Defaults to "fail".

Value

A `tibble`-style data frame with 2 columns:

- `instrument_name`
- `instrument_label`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

meta_repeating(conn)

## End(Not run)
```

parse_logging *Convert a REDCap project log file to a tidy data frame*

Description

[Experimental]

REDCap project log files have a complicated format where multiple types of information are contained in a single column. For example the column `action` contains the relevant record ID, the type of action that was taken (e.g. Updated / Created / Deleted), and sometimes further details about the source of the action (e.g. API / Import / Automatic field calculation). The `details` column contains a string of variable/value combinations describing any changes (e.g. "var1 = '0', var2 = '1', var3(1) = checked"), and may also contain the relevant repeat instance number (e.g. "[instance = 3]").

The parse_logging() function tidies up the log file by splitting the record ID, action, action type, and repeat instance into separate columns. Optionally, the string of variable/value changes in the `details` column may be further transformed to long format to yield a single row for each combination of variable and value.

Note that this function only deals with log entries of type Created / Deleted / Updated Record. All other log entries (e.g. Data Export, Manage/Design, Edited Role, User Assigned to Role) will be filtered out.

Usage

```
parse_logging(x, format_long = FALSE, dict = NULL)
```

Arguments

`x`	REDCap project log file (data frame), e.g. returned by `project_logging`
`format_long`	Logical indicating whether to transform the log file to long format, with one row per variable-value combination. Defaults to `FALSE`.
`dict`	A REDCap metadata dictionary (data frame), e.g. returned by `meta_dictionary`. Only needed when argument `format_long` is `TRUE`.

Value

A `tibble`-style data frame with 8 columns:

`rowid` Row number based on original log file. There may be gaps for rows that have been excluded from the output because they reflected an action type other than create / delete / update.

`timestamp` unchanged from original log file

`username` unchanged from original log file

`action` One of "Created Record", "Deleted Record", or "Updated Record", extracted from original `details` column

`action_type` Parenthetical details, if any, extracted from original `action` column (e.g. "(API)", "(import)", "(Auto calculation)")

`record_id` Record ID, extracted from original `action` column

`redcap_repeat_instance` Instance number (integer), extracted from original `details` column. Note that 1st instances are not explicitly specified in the log file and will appear as NA

`details` String of variable value pairs (e.g. "var1 = '0', var2 = '1', var3(1) = checked"), reflecting the data that was modified

If argument `format_long` is `TRUE` the `details` column will be replaced with three other columns:

`form_name` Form name, joined from metadata dictionary based on variable `field_name`. Will be `<NA>` in cases where field name has been changed or removed and therefore doesn't appear in the dictionary, or for fields not associated with a specific form like `redcap_data_access_group`.

`field_name` Field name, extracted from original `details` column

`value` Value, extracted from original `details` column

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

parse_logging(project_logging(conn))

## End(Not run)
```

`parse_xml` *Convert a REDCap project XML file to a tidy data frame of project records*

Description

Extract records from a REDCap project XML file (e.g. returned by `project_xml`) and assemble into a tidy long-form data frame, with one row for each combination of record x field x event x instance.

Usage

```
parse_xml(x)
```

Arguments

`x`	A REDCap project XML obect of class `xml_document`, e.g. returned by `project_xml`

Value

A tibble-style data frame with 6 columns:

- record_id
- form
- redcap_event
- redcap_repeat_instance
- field
- value

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

parse_xml(project_xml(conn))

## End(Not run)
```

---

project_dags *Export REDCap user information*

Description

Execute an "Export Data Access Groups (DAGs)" API request to fetch the DAGs (labels and code names) associated with a REDCap project.

Usage

```
project_dags(conn)
```

Arguments

conn	A REDCap API connection object (created with rconn)

Value

A tibble-style data frame with 2 columns:

- data_access_group_name
- unique_group_name

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

project_dags(conn)

## End(Not run)
```

`project_info` *Export REDCap Project information*

Description

Execute an "Export Project Info" API request to fetch project-related details (e.g. title, creation time, production status, language, etc.) corresponding to a REDCap project.

Usage

```
project_info(conn)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)

Value

A `tibble`-style data frame containing the columns returned by an "Export Project Info" API request

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

project_info(conn)

## End(Not run)
```

`project_logging` *Export project logging (audit trail) for a REDCap project*

Description

Execute an "Export Logging" API request to export the logging (audit trail) of all changes made to the project, including data exports, data changes, project metadata changes, modification of user rights, etc.

Usage

```
project_logging(
  conn,
  type = NULL,
  user = NULL,
  record = NULL,
  time_start = NULL,
  time_end = NULL,
  timestamp_to_posix = TRUE
)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)
`type`	Type of logging to return. Defaults to NULL to return all types. Specific logging types include: • "export": Data export • "manage": Manage/Design • "user": User or role created-updated-deleted • "record": Record created-updated-deleted • "record_add": Record created (only) • "record_edit": Record updated (only) • "record_delete": Record deleted (only) • "lock_record": Record locking & e-signatures • "page_view": Page Views
`user`	REDCap username to fetch logs for. Defaults to NULL to fetch logs relating to all users. Note that, in the current API version (10.8.5), it's not possible to pass a character vector with multiple usernames (i.e. one user per request, or NULL for all).
`record`	Record ID to fetch logs for. Defaults to NULL to fetch logs relating to all record IDs. Note that, in the current API version (10.8.5), it's not possible to pass a character vector with multiple record IDs (i.e. one record per request, or NULL for all).
`time_start`	Fetch logs from *after* a given date-time. Use format "YYYY-MM-DD HH:MM". Defaults to NULL to omit a lower time limit.

time_end — Fetch logs from *before* a given date-time. Use format "YYYY-MM-DD HH:MM". Defaults to NULL to omit an upper time limit.

timestamp_to_posix — Logical indicating whether to convert the `timestamp` column to class POSIXct using `lubridate::as_datetime`. Defaults to TRUE.

Value

A `tibble`-style data frame with 4 columns:

- `timestamp`
- `username`
- `actions`
- `details`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

project_logging(conn)

## End(Not run)
```

---

project_users *Export REDCap user information*

Description

Execute an "Export Users" API request to fetch user-related details (e.g. username, email, access permissions, etc.) corresponding to a REDCap project.

Usage

```
project_users(conn)
```

Arguments

conn — A REDCap API connection object (created with `rconn`)

Value

A `tibble`-style data frame containing the columns returned by an "Export Users" API request

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

project_users(conn)

## End(Not run)
```

`project_users_dags` *Export REDCap mapping between users and Data Access Groups (DAGs)*

Description

Execute an "Export User-DAG Assignments" API request to fetch the mapping between users and Data Access Groups (DAGs) associated with a REDCap project.

Usage

```
project_users_dags(conn)
```

Arguments

`conn`	A REDCap API connection object (created with `rconn`)

Value

A `tibble`-style data frame with 2 columns:

- `username`
- `redcap_data_access_group`

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

project_users_dags(conn)

## End(Not run)
```

project_xml *Export REDCap Project XML file*

Description

Execute an "Export Entire Project as REDCap XML File" API request to fetch all metadata (and optionally also data records) corresponding to a REDCap project.

Usage

```
project_xml(
  conn,
  meta_only = FALSE,
  records = NULL,
  fields = NULL,
  events = NULL,
  filter_logic = NULL,
  export_dag = FALSE,
  export_survey = FALSE,
  export_files = FALSE
)
```

Arguments

conn	A REDCap API connection object (created with rconn)
meta_only	Logical indicating whether to fetch only the project metadata (if TRUE) or both the metadata and data records (if FALSE). Defaults to FALSE to fetch both metadata and data.
records	Optional character vector of specific record IDs to fetch record data for. Only used when meta_only = FALSE.
fields	Optional character vector of specific fields to fetch record data for. Only used when meta_only = FALSE.
events	Optional character vector of specific events to fetch record data for. Only used when meta_only = FALSE.
filter_logic	Optional character string containing a REDCap-style expression used to filter records returned by the API (e.g. "[age] > 30")
export_dag	Logical indicating whether to export the redcap_data_access_group field. Defaults to FALSE.
export_survey	Logical indicating whether to export survey identifier or timestamp fields, if surveys are used in the project. Defaults to FALSE.
export_files	Logical indicating whether to export uploaded files. Note this may lead to large exports. Defaults to FALSE.

Value

An object of class xml_document

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

project_xml(conn)

## End(Not run)
```

`rconn` *Connection to a REDCap Database*

Description

Creates an object of class "rconn" containing the URL and token used to access a REDCap project API.

Usage

```
rconn(url, token, config = httr::config())
```

Arguments

`url`	URL for a REDCap database API
`token`	REDCap project API token (good practice to set using an environmental variable, e.g. with `Sys.getenv`).
`config`	Optional configuration settings passed to `httr::POST`. Defaults to `httr::config()`.

Value

An object of class "rconn", to be passed as the first argument to most other `redcap` functions.

Examples

```
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)
```

reclass *Reclass columns of a data frame to match classes specified in a metadata dictionary*

Description

Reclass columns of a data frame to match classes specified in a metadata dictionary

Usage

```
reclass(
  x,
  dict,
  use_factors = FALSE,
  value_labs = TRUE,
  header_labs = FALSE,
  times_chron = TRUE,
  fn_dates = parse_date,
  fn_dates_args = list(orders = c("Ymd", "dmY")),
  fn_datetimes = lubridate::parse_date_time,
  fn_datetimes_args = list(orders = c("Ymd HMS", "Ymd HM"))
)
```

Arguments

x	A data frame representing a REDCap form
dict	A metadata dictionary
use_factors	Logical indicating whether categorical REDCap variables (radio, dropdown, yesno, checkbox) should be returned as factors. Factor levels can either be raw values (e.g. "0"/"1") or labels (e.g. "No"/"Yes") depending on arguments value_labs and checkbox_labs. Defaults to FALSE.
value_labs	Logical indicating whether to return value labels (TRUE) or raw values (FALSE) for categorical REDCap variables (radio, dropdown, yesno, checkbox). Defaults to TRUE to return labels.
header_labs	Logical indicating whether to export column names as labels (TRUE) or raw variable names (FALSE). Defaults to FALSE to return raw variable names.
times_chron	Logical indicating whether to reclass time variables using chron::times (TRUE) or leave as character HH:MM format (FALSE). Defaults to TRUE. Note this only applies to variables of REDCap type "Time (HH:MM)", and not "Time (MM:SS)".
fn_dates	Function to parse REDCap date variables. Defaults to parse_date, an internal wrapper to lubridate::parse_date_time. If date variables have been converted to numeric (e.g. by writing to Excel), set to e.g. lubridate::as_date to convert back to dates.
fn_dates_args	List of arguments to pass to fn_dates. Can set to empty list list() if using a function that doesn't take any arguments.

fn_datetimes	Function to parse REDCap datetime variables. Defaults to lubridate::parse_date_time.
fn_datetimes_args	List of arguments to pass to fn_datetimes. Can set to empty list list() if using a function that doesn't take any arguments.

---

recode_labels *Convert between values and labels for factor-type variables (e.g. yes/no, radio, dropdown, checkbox)*

Description

Convert between values and labels for factor-type variables (e.g. yes/no, radio, dropdown, checkbox)

Usage

```
recode_labels(
  x,
  conn,
  dict = redcap::meta_dictionary(conn, add_complete = TRUE),
  convert_to = c("labels", "values"),
  types = c("radio", "yesno", "dropdown", "checkbox"),
  header_labs = FALSE
)
```

Arguments

x	Data frame representing a REDCap form (e.g. from a previous export using fetch_records)
conn	A REDCap API connection object (created with rconn)
dict	REDCap metadata dictionary. Defaults to fetching the current version with meta_dictionary
convert_to	Convert values to labels ("labels") or labels to values ("values")
types	Types of REDCap variables to convert, based on column "field_type" in the metadata dictionary. Defaults to c("radio", "yesno", "dropdown", "checkbox").
header_labs	Logical indicating whether column names of x are labels (TRUE) or raw variable names (FALSE). Default assumes header has raw variable names (i.e. FALSE).

redcap_version *Fetch the REDCap database version used for a particular project*

Description

Execute an "Export REDCap version" API request

Usage

```
redcap_version(conn)
```

Arguments

conn	A REDCap API connection object (created with rconn)

Value

A character string

Examples

```
## Not run:
conn <- rconn(
  url = "https://redcap.msf.fr/api/",
  token = Sys.getenv("MY_REDCAP_TOKEN")
)

redcap_version(conn)

## End(Not run)
```

translate_logic *Translate REDCap branching logic into R expressions*

Description

Translate REDCap branching logic into expressions evaluable in R. E.g.

Translation step	Example expression
0. Initial REDCap logic	`[over_18]='1' and [signed_consent]<>''`
1. Translate to R	`over_18 == '1' & signed_consent != ''`
2. (Optional) Swap values/labels	`over_18 == 'Yes' & signed_consent != ''`
3. (Optional) Use is.na	`over_18 == 'Yes' & !is.na(signed_consent)`
4. (Optional) Use %in%	`over_18 %in% 'Yes' & !is.na(signed_consent)`

Usage

```
translate_logic(
  x,
  use_value_labs = TRUE,
  use_header_labs = FALSE,
  use_is_na = TRUE,
  use_in = TRUE,
  drop_redundant = FALSE,
  field_nchar_max = 80L,
  meta_factors = NULL,
  meta_dictionary = NULL,
  on_error = "warn"
)
```

Arguments

x	A character vector of REDCap branching logic statements
use_value_labs	Logical indicating whether to replace factor option values with labels (based on the mapping defined in meta_factors). E.g.: y == '1' becomes y == 'Yes'
use_header_labs	Logical indicating whether to use labels instead of variable names as column names (based on the mapping defined in meta_dictionary). E.g.: age >= 18 becomes "Participant's age" >= 18
use_is_na	Logical indicating whether to replace REDCap-style tests for missingness with is.na. E.g.: y == "" becomes is.na(y) y != "" becomes !is.na(y)
use_in	Logical indicating whether to replace instances of == and != associated with factor-type variables (as defined in meta_factors) with %in%. E.g.: y == 'Yes' becomes y %in% 'Yes' y != 'Yes' becomes !y %in% 'Yes'
drop_redundant	Logical indicating whether to simplify expressions by removing redundant components from expressions that test both for equality and inequality with the same variable. E.g.: var == "X" & var != "" becomes var == "X"
field_nchar_max	Integer indicating the maximum number of characters to allow in field name labels before they are truncated and appended with "...". Defaults to 80L.
meta_factors	A data frame containing variable names (column field_name) and corresponding values (column value) and labels (column label) for factor-type variables. Fetch with meta_factors. Only needed if either use_value_labs or use_in are TRUE.
meta_dictionary	A data frame containing variable names (column field_name) and labels (column field_label). Fetch with meta_dictionary. Only needed if use_header_labs is TRUE.

on_error — What to do if one or more elements of statements can't be translated into valid R expressions? Options are "ignore" (return NA for relevant elements), "warn" (return NA for relevant elements and give warning), or "fail" (throw error). Defaults to "warn".

Value

A character vector of R-style expressions

Examples

```
# normally would fetch factor metadata with redcap::meta_factors(), but here
# we'll create a simple example by hand
df_factors <- data.frame(
  field_name = c("head_household", "head_household"),
  value = c("0", "1"),
  label = c("No", "Yes"),
  stringsAsFactors = FALSE
)

redcap_logic <- "head_household=1 and age<>\"\""
translate_logic(redcap_logic, meta_factors = df_factors)
```

Index