Frequently Asked Questions

2024-01-12

Contents


```
library(redcapAPI)
url <- "https://redcap.vanderbilt.edu/api/" # Our institutions REDCap instance

unlockREDCap(c(rcon = "Sandbox"),
             envir = .GlobalEnv,
             keyring = "API_KEYs",
             url = url)
```

```
## <environment: R_GlobalEnv>
```

Exporting Records

Which is preferred `exportRecordsTyped` or `exportBulkRecords`?

This depends on ones preferred use case. It's important to understand the difference in the two and their relationship.

`exportRecordsTyped` exports a single `data.frame` of all the data requested.

`exportBulkRecords` call `exportRecordsTyped` to create a `data.frame` for each form in the project, or just those requested via the `forms` argument. Additional arguments are all passed to `exportRecordsTyped`. Thus the documentation on validation and casting is the same for both.

If one is starting a new project, we would recommend using `exportBulkRecords` as the subsetting and filtering of empty rows is taken care of, and the user is left with doing the required joins to the data. If one has existing code they are converting that used `exportRecords`, then `exportRecordsTyped` is the recommendation. This is usually followed by code to subset into forms, filtering and then the same joins between these. Thus new projects can save some code by starting with `exportBulkRecords`.

What is the warning about zero-coded check fields?

The `redcapAPI` development team strongly advises against the use of zero-coded check fields in project databases.

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

Code	Label	Result
0	anything other than "0"	Likely to fail when casting from coded values
0	0	Likely to fail when casting from coded or labeled values

Examples of problematic coding are

```
0, Stegosaurus     (likely to fail when casting from coded values)
1, Triceratops
2, Brachiosaurus
```

and

```
0, 0               (likely to fail when casting for coded or labeled values)
1, 1
2, 2
```

When it is necessary to cast a zero-coded check field from coded or labeled values, the `castCheckForImport` casting function is the best option, as it provides the user full control over what values are to be considered "Checked."

`redcapAPI` is noisy (creates lots of warnings) about the presence of zero-coded check fields. The potential for loss of data integrity is serious and users need to be aware of that potential. The user may disable these warnings in `exportRecordsTyped` by setting the argument `warn_zero_coded = FALSE`.

Casting Records

How do I stop casting fields to factors?

I used to be able to set `factors = FALSE` to prevent categorical values from being returned as factors. How do I do that with `exportRecordsTyped`?

Users may substitute an alternate casting list specification within the call to `exportRecordsTyped`. `redcapAPI` provides two lists for this purpose: `default_cast_character` and `default_cast_no_factor`. These two lists are identical and may be used interchangeably.

```
exportRecordsTyped(rcon,
                   cast = default_cast_character)

exportRecordsTyped(rcon,
                   cast = default_cast_no_factor)
```

Aside from not casting factors, all other settings in this list are identical to the default casting.

How do I control the casting of `redcap_event_name`?

In earlier versions of `redcapAPI`, the `redcap_event_name` field commonly returned the values such as `event_1_arm_1`, `event_2_arm_1`, etc. It now returns "fancy" values. How do I get the original behavior?

The `redcap_event_name` field is one of the fields referred to as a "system" field. These fields are not part of the project's data dictionary, and are automatically returned by the API based on the configuration of the project.

By default, `exportRecordsTyped` returns the "labeled" values of the event names.

```
exportRecordsTyped(rcon,
                   fields = "redcap_event_name",
                   records = 1:3)
```

```
##        redcap_event_name
## 1 Event 1 (Arm 1: Arm 1)
## 2 Event 1 (Arm 1: Arm 1)
## 3 Event 1 (Arm 1: Arm 1)
```

This behavior can be changed using the `system` casting override (this will also affect the casting of other system fields).

```
exportRecordsTyped(rcon,
                   fields = "redcap_event_name",
                   records = 1:3,
                   cast = list(system = castRaw))
```

```
##   redcap_event_name
## 1     event_1_arm_1
## 2     event_1_arm_1
## 3     event_1_arm_1
```

Concerns Over Invalid Data Being Marked NA

Users have expressed concern that marking data that fails validation is not desired, as NA(not in REDCap) is not the same as NA(Unable to Cast). While this is true, there is no means to easily differentiate the two in the same `data.frame`. The problem can be demonstrated with a simple example as follows.

Say that one needs to write a general function that given a vector of strings in R and turn this into a `Date` object; in type theory this is $f : \texttt{Text} \rightarrow \texttt{Date}$. What if the string presented is "yyz"? There is no date that can be assigned in R and the core date routines in R will throw an error and halt processing. R is somewhat unique in that *all* it's core data types exist inside the `Maybe` monad, i.e. `Maybe a = Just a | NA`. Thus the true function type is $f : \texttt{Text} \rightarrow \texttt{Maybe Date}$. This allows for `NA` or "not available" values. R makes no judgement on the cause of the lack of availability. To continue processing the only safe choice is to assign "yyz" to NA, $f(\text{“}yyz'') = NA$.This was a big driving reason behind the design of `exportRecordsTyped`. The "Typed" portion of the name referring to this. Thus the algorithm:

- Assign NA to all values that are NA by definition from the data collection source.
- Assign NA to all values that cannot be cast into their target types and record this in the `invalidRecords` attribute.
- Perform the final type casting on the values that have not been assigned NA.

Via inversion of control, the user can override any choices the library team has made in `exportRecordsTyped` via the `na`, `validation` or `cast` arguments. It should be noted that the `validation` and `cast` arguments need to remain consistent with one another for each type.

Because NA has 2 meanings, the scanning of the resulting `reviewInvalidRecords` report is a crucial and important step in ensuring data quality of preparing data for analysis and reporting.

Missing Data Detection

Change the Default Missing Data Detection for All Field Types

How do I change the default missing data detection for all field types?

`redcapAPI` has an obscure function that will create a list of overrides for every field type. Use the `na_values` function to create the override list as illustrated below. (Yes, `na_values` takes a function as an argument)

```
customMissingDetection <- function(x, ...){
  is.na(x) | x == "" | x %in% c(-98, -99)
}

Rec <- exportRecordsTyped(rcon,
                          fields = c("days_between",
                                     "days_between_duplicate",
                                     "dropdown_example",
                                     "dropdown_example_duplicate"),
                          na = na_values(customMissingDetection))
Rec
```

```
##   record_id days_between days_between_duplicate dropdown_example
## 1         1           10                     10         One week
## 2         2           22                     22      Three weeks
## 3         3           NA                     NA             <NA>
## 4         4           NA                     NA             <NA>
##   dropdown_example_duplicate
## 1                   One week
## 2                Three weeks
## 3                       <NA>
## 4                       <NA>
```