

Data Validation with `exportRecordsTyped`

2023-12-16

Contents

Introduction	1
Data Validation	2
Default Validation Behavior	2
Customizing Data Validation For a Field	2
Failed Validations from REDCap project ‘Test redcapAPI (Sandbox)’	3
Appendix	4
Data Validation Field Types	4
Default Validation Settings	4

Introduction

The addition of `exportRecordsTyped` opened a great deal of flexibility and potential for customization when exporting data from REDCap and preparing them for analysis. The tasks of preparing data are broadly categorized into three phases

1. Missing Value Detection
2. Field Validation
3. Casting Data

This document will focus on data validation and how the user can customize validation to support their data analysis.

```
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>
```

Data Validation

Data validation operates in a similar manner to casting data (see `vignette("redcapAPI-casting-data", package = "redcapAPI")`). When data are exported from REDCap, the field type of each field is determined and an appropriate validation function is applied to the field to determine which, if any, values do not adhere to the expected format. This section describes the default behavior of data validation and discusses how to customize these behaviors.

Default Validation Behavior

When data are exported from REDCap, each field is compared against the expected format for the field type. A record of values that fail validation is recorded and available for review using `reviewInvalidRecords`.

Validation is performed after missing data detection. Values that are determined to be missing are not included in the validation report.

Customizing Data Validation For a Field

The default validation functions are designed to match the format expected by REDCap for each data type. Users can customize validation to unique circumstances by writing custom functions and/or regular expressions. Functions must return logical values and may utilize arguments `x`, `field_name`, and `coding` (the last argument is used when validating multiple choice fields).

Consider an scenario where researchers are recording ICD-10 diagnosis codes. ICD-10 codes are 3 - 7 character, alpha-numeric codes where the first character is any letter except U, the next two characters are numbers, and the remaining (up to) four characters are numbers or letters. A regular expression to match this pattern is `[A-T,V-Z][0-9]{2}\\.[A-Z,0-9]{0-4}`.

The user may write a function to validate the ICD-10 codes as follows:

```
validateICD10 <- function(x, field_name, ...){
  regex <- "[A-T,V-Z][0-9]{2}\\.[A-Z,0-9]{0-4}"

  if (field_name == "icd10"){
    valRx(regex)(toupper(x), ...)
  } else {
    rep(TRUE, length(x))
  }
}

Rec <- exportRecordsTyped(rcon,
  fields = c("icd10"),
  validation = list(text = validateICD10))
```

```
## Warning in .castRecords_attachInvalid(rcon = rcon, Records = Records, Raw =
## Raw, : Some records failed validation. Use 'reviewInvalidRecords' to review the
## failures.
```

After noticing the warning about failed validations, the listing of invalid data can be reviewed by calling `reviewInvalidRecords`.

```
reviewInvalidRecords(Rec)
```

Failed Validations from REDCap project ‘Test redcapAPI (Sandbox)’

Sat Dec 16 00:00:00 2023
Package redcapAPI version 2.8.3
REDCap version 14.0.2

- Field[text] ‘icd10’ on form ‘validation’ has 1 failure
 - Row 2, Record Id ‘2’, Value ‘U93.401A’ link

```
Rec
```

```
##  record_id  icd10
## 1          1 S93.401A
## 2          2 U93.401A
```

Appendix

Data Validation Field Types

- **bioportal**: Text fields that are validated using the BioPortal Ontology service.
- **calc**: Calculated fields.
- **checkbox**: Checkbox fields.
- **date_**: Text fields with the “Date” validation type.
- **datetime_**: Text fields with the “Datetime” validation type.
- **datetime_seconds_**: Text fields with the “Datetime with seconds” validation type.
- **dropdown**: Drop down multiple choice fields.
- **float**: Text fields with the “Number” validation type.
- **form_complete**: Fields automatically added by REDCap indicating the completion status of the form.
- **int**: Text fields with the “Integer” validation type. This appears to be a legacy type, and integer appears to be used by more recent version of REDCap.
- **integer**: Text fields with the “Integer” validation type.
- **number**: Text fields with the “Number” validation type.
- **number_1dp**: Text fields with the “number (1 decimal place)” validation type.
- **number_1dp_comma_decimal**: Text fields with the “number (1 decimal place - comma as decimal)” validation type.
- **number_2dp**: Text fields with the “number (2 decimal place)” validation type.
- **number_2dp_comma_decimal**: Text fields with the “number (2 decimal place - comma as decimal)” validation type.
- **radio**: Radio button fields.
- **select**: Possible alias for dropdown or radio.
- **sql**: Fields that use a SQL query to make a drop down tools from another project.
- **system**: Fields automatically provided by REDCap for the project. These include `redcap_event_name`, `redcap_data_access_group`, `redcap_repeat_instrument`, and `redcap_repeat_instance`.
- **time_mm_ss**: Text fields with the “Time (MM:SS)” validation type.
- **time_hh_mm_ss**: Text fields with the “Time (HH:MM:SS)” validation type.
- **truefalse**: True - False fields.
- **yesno**: Yes - No fields.

Default Validation Settings

The values of the `REGEX_*` regular expressions can be quite long and several would not fit on the page. Users who wish to review the definitions of the regular expressions can retrieve them using, for example, `redcapAPI::REGEX_DATETIME`.

```
.default_validate <- list(  
  date_           = valRx(REGEX_DATE),  
  datetime_       = valRx(REGEX_DATETIME),  
  datetime_seconds_ = valRx(REGEX_DATETIME_SECONDS),  
  time_mm_ss      = valRx(REGEX_TIME_MMSS),  
  time_hh_mm_ss   = valRx(REGEX_TIME_HHMMSS),  
  time            = valRx(REGEX_TIME),  
  float           = valRx(REGEX_FLOAT),  
  number          = valRx(REGEX_NUMBER),  
  calc            = valRx(REGEX_CALC),  
  int             = valRx(REGEX_INT),  
  integer         = valRx(REGEX_INTEGER),  
  yesno           = valRx(REGEX_YES_NO),  
  truefalse       = valRx(REGEX_TRUE_FALSE),
```

```
checkbox          = valRx(REGEX_CHECKBOX),
form_complete  = valRx(REGEX_FORM_COMPLETE),
select        = valChoice,
radio          = valChoice,
dropdown       = valChoice,
sql            = valChoice,
biportal       = valChoice
)
```