# REDCap API Guide

REDCap Data Engineering Team

2024-03-29

# Table of contents

# Welcome

This guide was developed by a epidemiologists at the Washington State Department- of Health. In our roles, we were often tasked with developing of custom data pipelines between REDCap and other agency data systems. These pipelines requires custom transformation and an in-depth understanding of the data-types inside a REDCap project. While REDCap is very widely used with ample training content on project design and customization, there is not much content specific to informatics and technical administrators. Searching for answers have even led to answer like "please consult informatics specialist in your organization". So after lots of trial and error and consulting amongst ourselves, this guide is the result. We hope to clearly illustrate the relationship between the data structures within REDCap (question type, instruments, events etc.,)to the resulting data format when exported to json or csv. Lastly, we provide samples of code in R and Python to effectively interact with all of the REDCap features that the API provides. We hope it is helpful in your data processing and analytics journey.

# Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

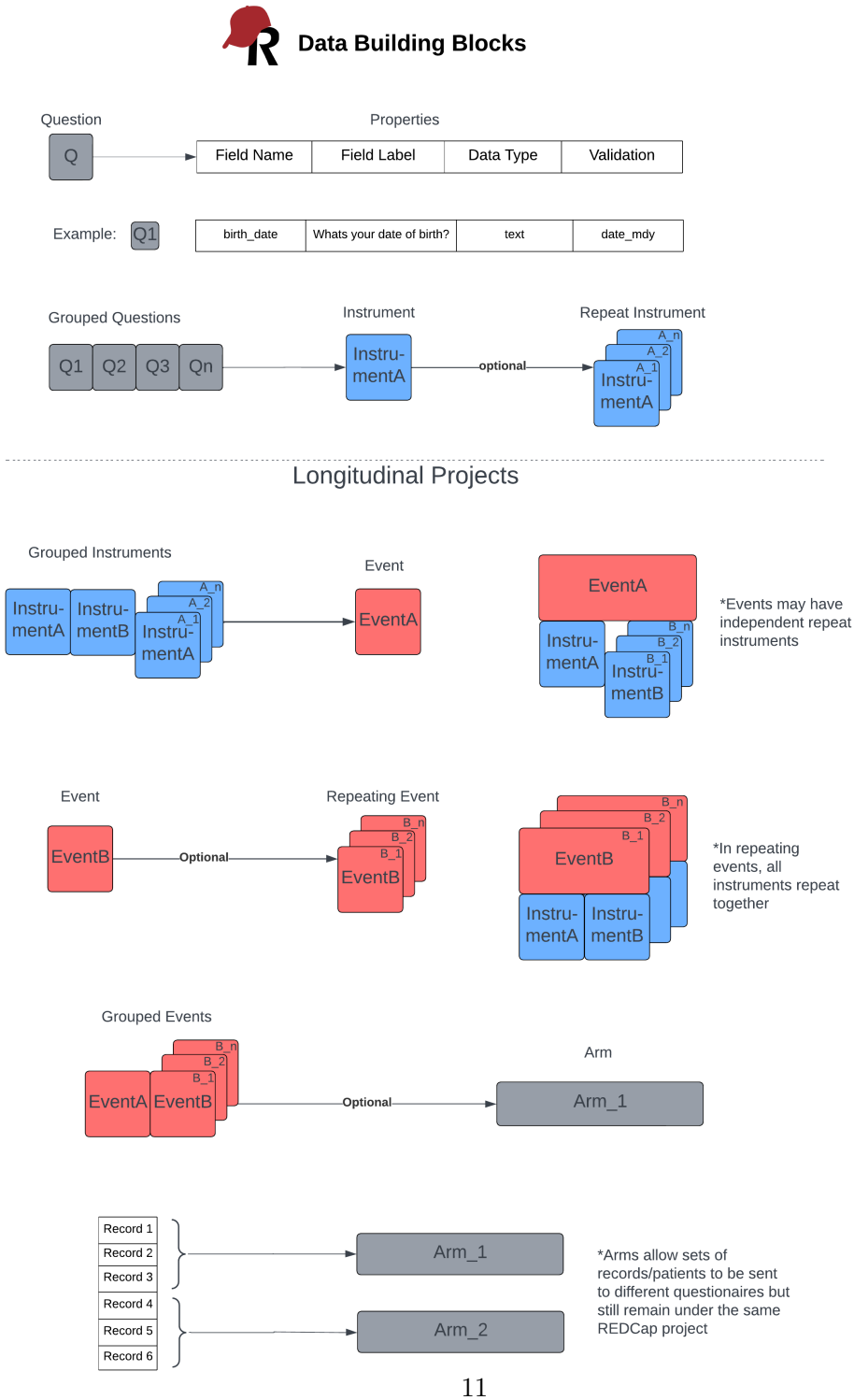
# Part I

# Introduction to REDCap

# 1 REDCap Overview

REDCap (Research Electronic Data Capture) is a secure web application developed by Vanderbilt University in 2004, designed to facilitate data collection and management for research studies. It enables researchers to build and manage online surveys and databases quickly and efficiently, offering a robust set of features including customizable data collection instruments, SMS or email notifications, API import/export capability, audit trails, and user rights management. Initially created to support academic research, REDCap has since grown into a global consortium with over 5,000 institutional partners, supporting diverse projects in clinical, translational, and behavioral research. Its purpose is to streamline data management processes, ensuring data integrity, security, and compliance with regulatory standards

# 2 REDCap Data Building Blocks



*For examples of REDCap project set-ups and exported data, see Chapter 4.*

# 3 Definitions

## 3.1 Records

Records are the set of information for a unique participant. Each record is composed of a number of fields (pieces of data), which can be spread across multiple instruments per record.

Exporting records returns all of the data entered into the record(s) of interest. By default, all records in the project will be returned. A subset of records can be specified.

The labels for the questions can be exported rather than the variable names. Similarly, the labels for the data entered can be exported instead of the corresponding raw numeric values that REDCap uses.

## 3.2 Fields

Fields are the individual places where data can be recorded (e.g., a question on a survey).

Exporting field names will capture the variable name, choice values (when applicable; e.g., for checkbox fields), and the modified export field name with the choice value appended.

## 3.3 Instruments/Forms

Instruments are a collection of fields to collect data. Instruments may be referred to as "forms" when being filled out by a project user or a "survey" when being filled out by external users (via a web link or email invitation).

Instruments may be repeating or non-repeating and can be in either longitudinal or non-longitudinal projects. Repeating instruments can be set up to repeat a defined number of times or repeat an indefinite number of times. Each repeat is called an "instance".

## 3.4 Events

Events are required when a project is enabled as longitudinal. An event may be a temporal event during your longitudinal project, such as a participant visit or a task to be performed. The default is 1 event. Once a project is enabled as longitudinal, each instrument must be associated with an event.

Events may be repeating or non-repeating. When an event is set to repeating, all the instruments in that event will be repeated together. You can also specify to have specific instrument independtly repeat within an event. Each repeat of an Event or independent instrument is called an "instance".

## 3.5 Arms

Events can be grouped into 'arms'. There may be one or more arms for a project. Arms can be though of as different groups in a clinical trial (e.g., a test group and a control group). Each arm can have as many events as you wish. Each arm can have the same events or different events with different instruments. The default is 1 arm.

## 3.6 Metadata

Metadata refers to the project's set up characteristics, including field attributes grouped by instrument assignment. Metadata can be thought of as the project's data dictionary.

## 3.7 Reports

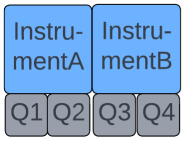
Reports are a good way to view data from multiple records at once. REDCap has two default reports (A & B) and additional custom reports can be created. Report A displays all data for all records, while Report B can be customized to display data from specified instruments or events. Data can be exported from custom reports using the auto-generated report ID number.

# 4 Data Structure

## 4.1 Example REDCap Projects and Data Structures

### Non-longitudinal Projects

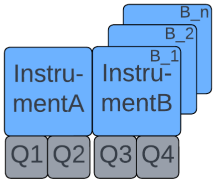
Non-longitudinal project (no events) with no repeating instruments




Unique key

| record_id | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| Record 1 | | | | |
| Record 2 | | | | |
| Record 3 | | | | |

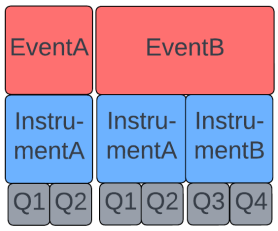
Non-longitudinal project (no events) with repeating instruments




Unique key

| record_id | repeat_instrument | repeat_instance | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|---|---|
| Record 1 | NA | NA | | | NA | NA |
| Record 1 | B | 1 | NA | NA | | |
| Record 1 | B | 2 | NA | NA | | |
| Record 1 | B | *n* | NA | NA | | |
| Record 2 | NA | NA | | | NA | NA |
| Record 2 | B | 1 | NA | NA | | |
| Record 2 | B | 2 | NA | NA | | |
| Record 2 | B | *n* | NA | NA | | |

## Longitudinal project without repeating instruments or repeating events

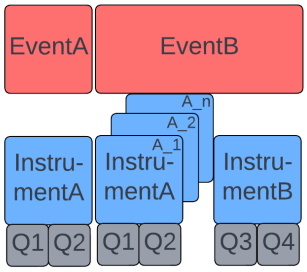



**Unique key**

| record_id | event_name* | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|---|
| Record 1 | event_A_arm_1 | | | NA | NA |
| Record 1 | event_B_arm_1 | | | | |
| Record 2 | event_A_arm_1 | | | NA | NA |
| Record 2 | event_B_arm_1 | | | | |

*Suffix "_arm_1" is added to the event name in longitudinal studies

## Longitudinal project with repeating instruments but no repeating events

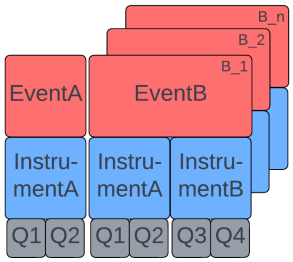



**Unique key**

| record_id | event_name* | repeat_instrument | repeat_instance | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|---|---|---|
| Record 1 | event_A_arm_1 | NA | NA | | | NA | NA |
| Record 1 | event_B_arm_1 | A | 1 | | | NA | NA |
| Record 1 | event_B_arm_1 | A | 2 | | | NA | NA |
| Record 1 | event_B_arm_1 | A | *n* | | | NA | NA |
| Record 1 | event_B_arm_1 | NA | NA | NA | NA | | |
| Record 2 | event_A_arm_1 | NA | NA | | | NA | NA |
| Record 2 | event_B_arm_1 | A | 1 | | | NA | NA |
| Record 2 | event_B_arm_1 | A | 2 | | | NA | NA |
| Record 2 | event_B_arm_1 | A | *n* | | | NA | NA |
| Record 2 | event_B_arm_1 | NA | NA | NA | NA | | |

*Suffix "_arm_1" is added to the event name in longitudinal studies

Longitudinal project with repeating events but no independently repeating instruments

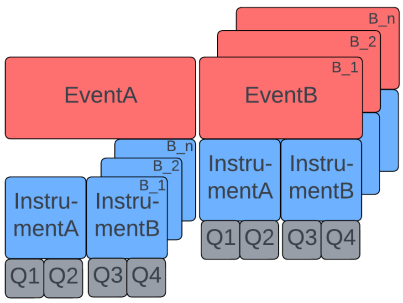



Unique key

| record_id | event_name* | repeat_instance | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|---|---|
| Record 1 | event_A_arm_1 | NA | | | NA | NA |
| Record 1 | event_B_arm_1 | 1 | | | | |
| Record 1 | event_B_arm_1 | 2 | | | | |
| Record 1 | event_B_arm_1 | *n* | | | | |
| Record 2 | event_A_arm_1 | NA | | | NA | NA |
| Record 2 | event_B_arm_1 | 1 | | | | |
| Record 2 | event_B_arm_1 | *n* | | | | |

*Suffix "_arm_1" is added to the event name in longitudinal studies

Longitudinal project with repeating events and repeating instruments




Unique key

| record_id | event_name* | repeat_instrument | repeat_instance | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|---|---|---|
| Record 1 | event_A_arm_1 | NA | NA | | | NA | NA |
| Record 1 | event_A_arm_1 | B | 1 | NA | NA | | |
| Record 1 | event_A_arm_1 | B | 2 | NA | NA | | |
| Record 1 | event_A_arm_1 | B | *n* | NA | NA | | |
| Record 1 | event_B_arm_1 | A | 1 | | | NA | NA |
| Record 1 | event_B_arm_1 | A | 2 | | | NA | NA |
| Record 1 | event_B_arm_1 | A | *n* | | | NA | NA |
| Record 1 | event_B_arm_1 | B | 1 | NA | NA | | |
| Record 1 | event_B_arm_1 | B | 2 | NA | NA | | |
| Record 1 | event_B_arm_1 | B | *n* | NA | NA | | |
| Record 2 | ... | ... | ... | | | | |

*Suffix "_arm_1" is added to the event name in longitudinal studies

The base of every unique key is always the `record_id`. In non-longitudinal projects, there may also be a `repeat_instrument` and `repeat_instance` column if these features are enabled. In longitudinal projects, there will be an `event_name` column, as well as a `repeat_instance` column in the case of repeating events and a `repeat_instrument` column in the case of independently repeating instruments.

This unique key applies for studies with and without multiple arms. Each value for the `event_name` includes the study arm as a suffix. The suffix will automatically be *"_arm_1"* for longitudinal studies without additional arms.

These special unique key fields must be appropriately filled out in the data being imported to REDCap.

## 4.2 Repeating Events and Independently Repeating Instruments

Data is exported from REDCap projects as one large table where the length of the table (number of columns) is equal to all the fields across all project instruments.

Assuming no repeating instruments or events, there is one row per record in non-longitudinal projects and one row per record-event in longitudinal projects. In the case of independently repeating instruments and repeating events, there is one additional row per repeat instance per record. Each row has all fields across all instruments, but the fields (columns) not associated with the instruments for that event (rows) will be NA.

Regardless of how you choose to export REDCap project data (directly in REDCap or using an API), the data structure will be the same. Here is an example of the data structure for the REDCap project we will be using in this tutorial.

In this example:

1. The 'close_contacts' instrument repeats independently within the 'notifications_arm_1' event (only responses to the 'close_contacts' instrument are in these rows with one row per record per instance of the instrument).
2. One other instrument is also in the 'notifications_arm_1' event but does not repeat. This data populates a seperate row where the redcap_event_name = "notifications_arm_1", but the redcap_repeat_instrument = "NA" and the redcap_repeat_instance = "NA". Non-repeating instruments in an event will have their own row, seperate from independently repeating instruments in that same event.
3. The 'case_intake_arm_1' event repeats as an entire event, so each repeat of the event per record will occupy one row with the redcap_repeat_instance variable signifying the instance number.
4. The 'personal_info_arm_1' is not repeating, nor are the instruments within this event, so it occupies one row per record-event.

**Note:** Since this is a longitudinal project example, the arm name is automatically appended as a suffix in the `redcap_event_name` column. In this project there is only one arm, so all events are exported with the "arm_1" suffix. However, if there were mutliple arms, the suffix would distinguish which arm each event is in.

Checking the instrument event map will give you a quick understanding of the project structure. See Chapter 14 for how to do this via API.

## 4.3 Notes on REDCap Data Types and API Exports

### 4.3.1 Standard Field Types

- text
- notes
- calculated field
- dropdown
- radio
- checkbox
- yes/no
- true/false
- file upload
- descriptive
- dynamic query (sql)

### 4.3.2 Non-Standard Field types

- instrument_name_complete

In addition to the standard types, each instrument (form) has a column to indicate if the instrument is complete/incomplete/unverified. The instrument_name_complete field is exported via standard API call.

### 4.3.3 API Export Records (default settings)

- text
- notes
- calculated field
- dropdown
- radio
- checkbox
- yes/no
- true/false
- file upload
- ~~descriptive~~
- dynamic query (sql)

- instrument_name_complete

Note: “descriptive” field type is NOT exported

## 4.4 Checkboxes

Checkboxes are exported as a wide data set with each checkbox option stored as its own variable. These variables will be appended with a double underscore and the number that the choice option is assigned within REDCap. Alternatively, the actual choice can be viewed if you export the dataset with labeled headers.

See below for how the checkbox variable `sympoms_exp` exports. This checkbox had 11 different options and so it will occupy 11 columns. You can also see this question is asked in the `case_intake_arm_1` event and so the value is NA in any row not associated with this event.

**Note:** When putting the exported data into a pandas dataframe in Python, any raw values for coded data that are integer strings in REDCap become floats with a decimal appended. If this exported dataset were to be re-imported into REDCap, errors will arise for all of the fields with the now float-type data when REDCap is expecting an integer. See the Python section under Section 27.4 for more.

# 5 R/Python Packages for REDCap's API

R and Python both have packages that make it easy to work with REDCap's API. We recommend REDCapR in R and PyCap in Python. We will be using each of these packages to provide examples of different import and export options throughout this guide.

# 6 Storing API tokens

Before you can make any API calls to your REDCap project, you will need to generate an API token for that project. You can do so in the REDCap web application under the 'API' page of your project.

It is recommended to save your API tokens in a separate file on your local machine that is then called into your main script. That way you can share your script and push it to the GitHub without compromising your tokens. Tokens should be treated as secrets and should not be shared with others.

To follow along with this guide, you can save your tokens for each REDCap project as a json file with the following the format:

```
{
    "prod_token": {
      "REDCap Project ID Goes Here": "API token for that project goes here",
      "REDCap Project ID Goes Here": "API token for that project goes here"
    },
    "dev_token": {
      "REDCap Project ID Goes Here": "API token for that project goes here",
      "REDCap Project ID Goes Here": "API token for that project goes here"
        },
    "qa_token": {
      "REDCap Project ID Goes Here": "API token for that project goes here",
      "REDCap Project ID Goes Here": "API token for that project goes here"
        }
}
```

**Note:** You can create a json file in any text editor by saving your file with a '.json' extention.

# 7 Project & API Setup

Install and load the necessary packages and set up the REDCap project connection.

#### 7.0.0.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R
# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}
else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 7.0.0.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

# Part II

# Exporting from REDCap

# 8 Records

## 8.1 Project Setup

Install and load the necessary packages

#### 8.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 8.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 8.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 8.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

## 8.2 Exporting Raw Data

#### 8.2.0.1 R

`redcap_read_oneshot()`

```
%%capture
%%R
records <- redcap_read_oneshot(
    redcap_uri = url,
    token = token
)$data
```

```
%%R
records_tbl<- gt(head(records))
gt::gtsave(records_tbl, filename = 'export_records1.html', path = "./files/export_files/")
```

#### 8.2.0.2 Python

`export_records()`

```
records = project.export_records(format_type='df') #all records with raw data values
records.head(10)
```

| record_id | redcap_event_ |
|---|---|
| 1 | personal_info_<br>notifications_a<br>case_intake_ar<br>notifications_a<br>notifications_a |
| 2 | personal_info_<br>notifications_a<br>case_intake_ar<br>case_intake_ar<br>notifications_a |

When `format_type = 'df'`, there is a multi-index automatically assigned including `record_id` and `redcap_event_name`.

It is not recommended to use the index automatically asigned upon export as it is not always correct. In this case, `record_id`, `redcap_event_name`, `redcap_repeat_instrument`, and `redcap_repeat_instance` combined define the unique key for this data frame. The user should asign their own index accordingly; best practice is to add is to use the `reset_index()` method on the data export to use the row number as the index.

Exporting as a CSV or JSON creates the index using the row number.

## 8.3 Exporting Labeled Data & Headers

The `raw_or_label` parameter exports raw or labeled choice values (i.e. 'male' instead of '1'), while the `raw_or_label_headers` parameter exports raw or labeled variable names (i.e. shows the actual prompt/question instead of the raw variable name).

#### 8.3.0.1 R

```
%%capture
%%R
data_labeled <- redcap_read_oneshot(
    redcap_uri = url,
    token = token,
    raw_or_label = "label",
    raw_or_label_headers = "label")$data
```

```
%%R
data_labeled_tbl <- gt(head(data_labeled)) %>% cols_width(everything() ~ px(150))
gt::gtsave(data_labeled_tbl, filename = 'export_records2.html', path = "./files/export_files/
```

#### 8.3.0.2 Python

```
data_labeled = project.export_records(raw_or_label='label', format_type='df').reset_index()
data_labeled.head(10)
```

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | first_name | la |
|---|---|---|---|---|---|---|
| 0 | 1 | Personal Info | NaN | NaN | John | D |
| 1 | 1 | Notifications | NaN | NaN | NaN | N |
| 2 | 1 | Case Intake | NaN | 1.0 | NaN | N |

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | first_name | la |
|---|---|---|---|---|---|---|
| 3 | 1 | Notifications | Close Contacts | 1.0 | NaN | N |
| 4 | 1 | Notifications | Close Contacts | 2.0 | NaN | N |
| 5 | 2 | Personal Info | NaN | NaN | Jane | D |
| 6 | 2 | Notifications | NaN | NaN | NaN | N |
| 7 | 2 | Case Intake | NaN | 1.0 | NaN | N |
| 8 | 2 | Case Intake | NaN | 2.0 | NaN | N |
| 9 | 2 | Notifications | Close Contacts | 1.0 | NaN | N |

**Note:** Exporting labeled headers only works when `format_type='csv'`.

## 8.4 Export Data In Batches (REDCapR Only)

#### 8.4.0.1 R

`redcap_read()` is almost the same as `redcap_read_oneshot()`. The only difference is that `redcap_read()` retrieves the data in quantified batches or rows, and then combines the batches to return a single data set. This function may be more appropriate than `redcap_read_oneshot()` when exporting large datasets that could tie up the server. (Source)

```
%%capture
%%R
batched_export <- redcap_read(
    redcap_uri = url,
    token = token,
    batch_size = 50L
)$data
```

In this example, the batch size was set to 50 records. The default is 100 records. The data exported using this method has the exact same format as the data exported using `redcap_read_oneshot`.

## 8.5 Exporting The Next Available Record ID

When a project is set up in REDCap it has **auto-numbering for records** enabled by default. This allows a new and unique record_id to be automatically assigned every time you enter a new record within REDCap. Before importing new records via API, you may want to know

what the next available Record ID is to ensure you are assigning new Record IDs to these new records before import (rather than overwriting an existing record).

This function is more important if using REDCapR because PyCap has a way to auto-number records on import.

#### 8.5.0.1 R

```
%%capture --no-stdout
%%R
next_record <- redcap_next_free_record_name(
    redcap_uri = url,
    token = token,
    verbose = TRUE,
    config_options = NULL)

next_record
```

```
[1] "7"
```

#### 8.5.0.2 Python

```
project.generate_next_record_name()
```

```
'7'
```

**Note:** If Data Access Groups (DAGs) are used in the REDCap project, this method accounts for the special formatting of the record name for users in DAGs, where the unique auto-assigned DAG number is a prefix to the actual record_id (i.e. `<DAG_ID>_<record_id>`). A user assigned to a DAG with ID 1732 that already has 3 existing records will return '1732-4' as the next available record.

# 9 Reports

## 9.1 Project Setup

Install and load the necessary packages

#### 9.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 9.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

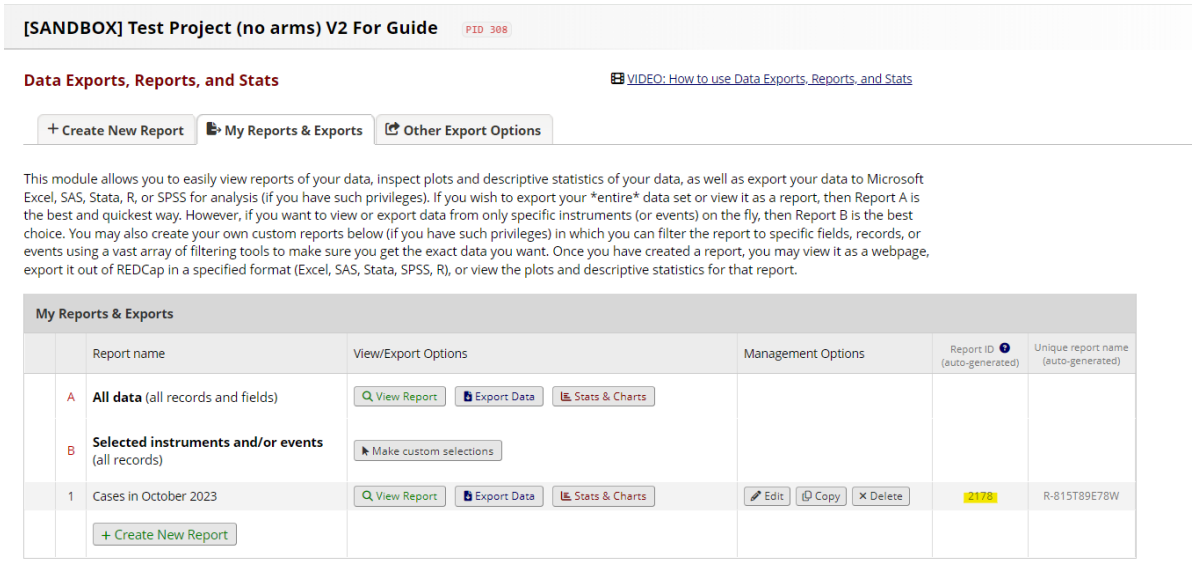
#### 9.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 9.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

To export reports via API you will need the 'Report ID' of your desired report. You can find the 'Report ID' of the report you want to export by going into your REDCap project and selecting 'Data Export, Reports, and Stats', then 'My Reports & Exports'.




## 9.2 Exporting Raw Reports

In this example, we will export the 'Cases in October 2023' report as seen in the image above.

#### 9.2.0.1 R

`redcap_report()`

```
%%capture
%%R
cases_oct_2023 <- redcap_report(
    redcap_uri = url,
    token = token,
    report_id = 2178
)$data
```

```
%%R
tbl<- gt(head(cases_oct_2023))
gt::gtsave(tbl, filename = 'export_reports1.html',  path = "./files/")
```

#### 9.2.0.2 Python

```
export_report()
```

```
project.export_report(report_id='2178', format_type='df', raw_or_label='raw').reset_index().h
```

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | first_name |
|---|---|---|---|---|---|
| 0 | 1 | notifications_arm_1 | NaN | NaN | NaN |
| 1 | 1 | personal_info_arm_1 | NaN | NaN | John |
| 2 | 1 | case_intake_arm_1 | NaN | 1.0 | NaN |
| 3 | 2 | notifications_arm_1 | NaN | NaN | NaN |
| 4 | 2 | personal_info_arm_1 | NaN | NaN | Jane |
| 5 | 2 | case_intake_arm_1 | NaN | 2.0 | NaN |
| 6 | 2 | case_intake_arm_1 | NaN | 1.0 | NaN |

## 9.3 Exporting Labeled Reports

Reports can also be exported as labeled data.

#### 9.3.0.1 R

```
%%capture
%%R
cases_oct_2023_labeled <- redcap_report(
    redcap_uri = url,
    token = token,
    report_id = 2178,
    raw_or_label = 'label'
)$data
```

```
%%R
tbl<- gt(head(cases_oct_2023_labeled))
gt::gtsave(tbl, filename = 'export_reports2.html', path = "./files/")
```

#### 9.3.0.2 Python

```
project.export_report(report_id='2178', format_type='df', raw_or_label='label').reset_index()
```

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | first_name | la |
|---|---|---|---|---|---|---|
| 0 | 1 | Notifications | NaN | NaN | NaN | N |
| 1 | 1 | Personal Info | NaN | NaN | John | D |
| 2 | 1 | Case Intake | NaN | 1.0 | NaN | N |
| 3 | 2 | Notifications | NaN | NaN | NaN | N |
| 4 | 2 | Personal Info | NaN | NaN | Jane | D |
| 5 | 2 | Case Intake | NaN | 2.0 | NaN | N |
| 6 | 2 | Case Intake | NaN | 1.0 | NaN | N |

**Note:** Exporting labeled headers only works when `format_type='csv'`. See Section 20.5 for more information.

# 10 Files

## 10.1 Project Setup

Install and load the necessary packages

#### 10.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 10.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 10.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 10.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

File uploads (attachments to individual records) are a unique field type in REDCap that accept a variety of file types, including images, pdfs, and many more. Unlike other export methods, exporting files only works for one file field from one record at a time.

If the project has repeating events (i.e. a longitudinal project), the event name that the record is in must be specified. If the file of interest is in a repeat instance, the instance number must also be specified.

In this example we will download the test file from record 1 in the `test_upload` field in `case_intake_arm_1` event, and save the file to a specified location.

#### 10.1.0.5 R

`redcap_file_download_oneshot()`

```
%%capture
%%R
redcap_file_download_oneshot(
    event = "case_intake_arm_1",
    directory = "./files/export_files/",
    file_name = "test_file_export_r.png",
    record = 1,
    field = "test_upload",
    redcap_uri = url,
    token = token,
    overwrite = TRUE
)
```

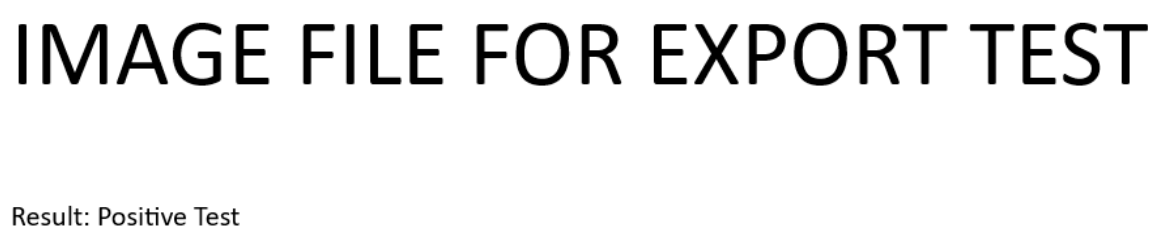
#### 10.1.0.6 Python

`export_file()`

In python we will use the `IPython.diplay` module to view the downloaded file.

```
export_file_image = project.export_file(record="1",
                                        field="test_upload",
                                        event="case_intake_arm_1")
with open("files/export_files/test_file_export_py.png","wb") as binary_file:
    binary_file.write(export_file_image[0])
```

```
from IPython.display import Image
Image("files/export_files/test_file_export_py.png", width=300)
```




Result: Positive Test

# 11 Metadata

## 11.1 Project Setup

Install and load the necessary packages

#### 11.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 11.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 11.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 11.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

REDCap metadata can be exported via the API.

The REDCap metadata will contain all the fields in your REDCap project *excluding*:

1) Any survey timestamp or survey identifier fields (automatically generated when a form is enabled as a survey).
2) Any fields that are part of the unique key aside from the `record_id` (`redcap_event_name`, `redcap_repeat_instrument` and `redcap_repeat_instance`).
3) Any `<form_name>_complete` fields.
4) Checkbox fields are exported as one field per checkbox (not in the wide format as they appear in the data export).

In REDCap the `record_id` field can be renamed. The first field in the exported metadata will always be `record_id` or the 'record_id' equivalent field.

#### 11.1.0.5 R

`redcap_metadata_read()`

```
%%capture
%%R
metadata <- redcap_metadata_read(
    redcap_uri = url,
    token = token
)$data
```

```
%%R
metadata_tbl <- gt(head(metadata))
gt::gtsave(metadata_tbl, filename = 'metadata.html', path = "./files/export_files/")
```

#### 11.1.0.6 Python

`export_metadata()`

```
project.export_metadata(format_type="df").head(10)
```

| field_name | form_name | section_header | field_type | field_label | select_choices_or_calcula |
|---|---|---|---|---|---|
| record_id | demographics | NaN | text | Record ID | NaN |
| first_name | demographics | Personal Information | text | First Name | NaN |
| last_name | demographics | NaN | text | Last Name | NaN |
| phone_num | demographics | NaN | text | Phone Number | NaN |
| zip_code | demographics | NaN | text | ZIP Code | NaN |

| field_name | form_name | section_header | field_type | field_label | select_choices_or_calcula |
|---|---|---|---|---|---|
| dob | demographics | NaN | text | Date of birth | NaN |
| age | demographics | NaN | calc | Age (years) | rounddown(datediff([dob], |
| ethnicity | demographics | NaN | radio | Ethnicity | 0, Hispanic or Latino\|1, N |
| race | demographics | NaN | dropdown | Race | 0, American Indian/Alask |
| gender | demographics | NaN | radio | Gender | 0, Female\|1, Male\|2, Othe |

# 12 Field Names

## 12.1 Project Setup

Install and load the necessary packages

#### 12.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 12.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 12.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 12.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

Using the export field names method *will* export all fields including the `<form_name>_complete` fields and the checkbox fields in a wide format, unlike when exporting the project's metadata.

It will *exclude*:

1. Any survey timestamp or survey identifier fields (automatically generated when a form is enabled as a survey).
2. Any fields that are part of the unique key aside from the `record_id` (i.e. `redcap_event_name`, `redcap_repeat_instrument` and `redcap_repeat_instance`).

#### 12.1.0.5 R

`redcap_variables()`

```
%%capture
%%R
field_names <- redcap_variables(
    redcap_uri = url,
    token = token,
    verbose = TRUE,
    config_options = NULL
)$data
```

```
%%R
field_names_tbl <- gt(head(field_names,15))
gt::gtsave(field_names_tbl, filename = 'field_names.html', path = "./files/export_files/")
```

#### 12.1.0.6 Python

`export_field_names()`

```
project.export_field_names(format_type="df").head(15)
```

| original_field_name | choice_value | export_field_name |
|---|---|---|
| record_id | NaN | record_id |
| first_name | NaN | first_name |
| last_name | NaN | last_name |
| phone_num | NaN | phone_num |
| zip_code | NaN | zip_code |
| dob | NaN | dob |
| age | NaN | age |
| ethnicity | NaN | ethnicity |
| race | NaN | race |

| original_field_name | choice_value | export_field_name |
|---|---|---|
| gender | NaN | gender |
| demographics_complete | NaN | demographics_complete |
| symptoms_yesno | NaN | symptoms_yesno |
| symptom_onset | NaN | symptom_onset |
| symptoms_exp | 1.0 | symptoms_exp____1 |
| symptoms_exp | 2.0 | symptoms_exp____2 |

# 13 Forms/Instruments

## 13.1 Project Setup

Install and load the necessary packages

#### 13.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 13.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 13.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 13.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

## 13.2 Instrument Names and Labels

You can export a list of instrument names and their corresponding instrument labels.

#### 13.2.0.1 R

```
redcap_instruments()
```

```
%%capture
%%R
instruments <- redcap_instruments(
    redcap_uri = url,
    token = token,
    verbose = TRUE,
    config_options = NULL
)$data
```

```
%%R
instruments_tbl <- gt(head(instruments))
gt::gtsave(instruments_tbl, filename = 'instruments.html', path = "./files/export_files/")
```

```
UsageError: Cell magic `%%R` not found.
```

#### 13.2.0.2 Python

```
export_instruments()
```

```
project.export_instruments(format_type = "df")
```

| | instrument_name | instrument_label |
|---|---|---|
| 0 | demographics | Demographics |
| 1 | symptoms | Symptoms |
| 2 | test_information | Test Information |
| 3 | close_contacts | Close Contacts |
| 4 | work_information | Work Information |

**Note:** Use the value under `instrument_name` (not `instrument_label`) when specifying specific instruments as arguments in the REDCapR and PyCap API functions.

## 13.3 Download PDF of Instruments

These functions will download the instrument specified and all of the fields as a cleanly formatted questionnaire pdf file.

#### 13.3.0.1 R

```
redcap_instrument_download()
```

```
%%capture
%%R
redcap_instrument_download(
    instrument = "symptoms",
    directory = "./files/instruments/",
    file_name = "symptoms_instrument.pdf",
    redcap_uri = url,
    overwrite = TRUE,
    token = token
)
```

#### 13.3.0.2 Python

```
export_pdf()
```

```
close_contact = project.export_pdf(instrument = 'close_contacts')
with open("files/instruments/close_contacts.pdf","wb") as binary_file:
    binary_file.write(close_contact[0])
```

```
from IPython.display import IFrame
IFrame("files/instruments/close_contacts.pdf", width=750, height=500)
```

```
<IPython.lib.display.IFrame at 0x1728efb76b0>
```

# 14 Instrument/Event Map

## 14.1 Export the Instrument Event Mapping

This shows which instruments are associated with each event in longitudinal projects.

## 14.2 Project Setup

Install and load the necessary packages

#### 14.2.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 14.2.0.2 Python

```
# import pycap
import redcap
import json
import pandas as pd
# import requests
```

Assign your project URL and Token

#### 14.2.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 14.2.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

#### 14.2.0.5 R

```
redcap_event_instruments()
```

```
%%capture
%%R
final_mapping <- redcap_event_instruments(
    redcap_uri = url,
    token,
    verbose = TRUE,
    config_options = NULL
)$data
```

```
%%R
final_mapping_tbl <- gt(head(final_mapping))
gt::gtsave(final_mapping_tbl, filename = 'export_instrument_event.html', path = "./files/expo
```

#### 14.2.0.6 Python

```
export_instrument_event_mappings()
```

```
project.export_instrument_event_mappings(format_type='df')
```

| | arm_num | unique_event_name | form |
|---|---|---|---|
| 0 | 1 | personal_info_arm_1 | demographics |
| 1 | 1 | case_intake_arm_1 | symptoms |
| 2 | 1 | case_intake_arm_1 | test_information |
| 3 | 1 | notifications_arm_1 | close_contacts |
| 4 | 1 | notifications_arm_1 | work_information |

**Note:** Events with multiple instruments (AKA 'form') will occupy multiple rows, with one row per instrument.

## 14.3 Export the Repeated Instrument/Event Mapping (PyCap Only)

#### 14.3.0.1 Python

In REDCap projects instruments can be set to repeating and in the case of longitudinal projects, events can be set to repeating. You can use the `export_repeating_instruments_events` function to get a list of repeating events/instruments in your REDCap project.

```
project.export_repeating_instruments_events(format_type = 'df')
```

| | event_name | form_name | custom_form_label |
|---|---|---|---|
| 0 | case_intake_arm_1 | NaN | NaN |
| 1 | notifications_arm_1 | close_contacts | NaN |

Note in the data export above that `form_name` is NaN for the 'case_intake_arm_1' event. This means that the entire 'case_intake_arm_1' is repeating. Whereas, in the 'noticications_arm_1' event, the 'close_contacts' form is independently repeating (meaning it can endlessly repeat on it's own within that event). For more informatino on repeating events and isntruments refer to Chapter 2, Chapter 3 and Chapter 4.

If a project is not longitudinal, there will be no `event_name` exported. If the project has no repeating instruments or events, the API call will fail and the error string will say: "does not contain any repeating instruments and events"

# 15 Users and User Roles

## 15.1 Project Setup

Install and load the necessary packages

#### 15.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 15.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 15.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 15.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

REDCap allows strict control of user rights for each project. These rights include a user's ability to edit, export and view data, add or edit reports, import data, create records, etc. Privileges to view and export data are specified for each instrument.

User roles can be set with predefined privileges and users can then be assigned to these user role groups.

Both REDCapR and Pycap have functions for exporting user rights.

#### 15.1.0.5 R

`redcap_users_export()`

```
%%capture --no-display --no-stdout
%%R
users_data <- redcap_users_export(redcap_uri = url, token = token)
names(users_data)
```

```
[1] "data_user"       "data_user_form"  "success"         "status_code"
[5] "outcome_message" "elapsed_seconds" "raw_text"
```

**Extract the `data_user` table to view who has access to the REDCap project:**

`redcap_users_export` output provides a list with several elements. The two most useful elements are the `data_user` and the `data_user_form`.

```
%%capture
%%R
users <- users_data$data_user
```

```
%%R
users_tbl <- gt(head(users))
gt::gtsave(users_tbl, filename = 'export_users1.html', path = "./files/export_files/")
```

There is no way to export which user is in which named user role group with REDCapR or the native API. When you export this data, if a user has been assigned to a user role group, then it will return the user with the role's defined privileges.

**Extract the `data_user_form` table to view which forms each users has access to:**

```
%%capture
%%R
users_forms <- users_data$data_user_form
```

```
%%R
users_forms_tbl <- gt(users_forms)
gt::gtsave(users_forms_tbl, filename = 'export_users2.html', path = "./files/export_files/")
```

#### 15.1.0.6 Python

`export_users()`, `export_user_roles()`, `export_user_role_assignment()`

**Use `export_users` to view who has access to the REDCap project and what their rights are:**

```
project.export_users(format_type='df')
```

| | username | email | firstname | lastname | expiration | data_acces |
|---|---|---|---|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | Alexey.Gilman@doh.wa.gov | Alexey | Gilman | NaN | records1_2 |
| 1 | caitlin.drover@doh.wa.gov | Caitlin.Drover@doh.wa.gov | Caitlin | Drover | NaN | NaN |
| 2 | emily.pearman@doh.wa.gov | emily.pearman@doh.wa.gov | Emily | Pearman | NaN | NaN |

**Use `export_user_roles` to view the defined user rold groups for that project and their permissions:**

```
project.export_user_roles(format_type='df')
```

| | unique_role_name | role_label | design | alerts | user_rights | data_access_groups | reports | stats_and |
|---|---|---|---|---|---|---|---|---|
| 0 | U-5354FA3HYL | Admin | 1 | 1 | 1 | 1 | 1 | 1 |

**Use `export_user_role_assignment` to view which user is in in which role group and data access group:**

```
project.export_user_role_assignment(format_type='df')
```

| | username | unique_role_name | data_access_group |
|---|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | NaN | records1_2 |
| 1 | caitlin.drover@doh.wa.gov | U-5354FA3HYL | NaN |
| 2 | emily.pearman@doh.wa.gov | NaN | NaN |

You can use the `unique_role_name` field from the `export_user_roles` output to see which user is assigned to which user role in the ouput of `export_user_role_assignment`.

The user rights tables also include information on which Data Access Group each user is assigned to. For more information on Data Access Groups see Chapter 16

# 16 Data Access Groups (DAGs)

## 16.1 Project Setup

Install and load the necessary packages

#### 16.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 16.1.0.2 Python

```
import redcap
import json
import pandas as pd
```

Assign your project URL and Token

#### 16.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 16.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

In addition to user roles for access control, DAGs control which records each user can access. Users assigned to a DAG can only access records assigned to that particular DAG, and are

blinded to records outside of their group. DAGs are particularly useful in multi-site or multi-jurisdictional projects to restrict sites from viewing records from other sites.

Users can be assigned to a DAG after being added to the project. Users can be in multiple DAGs. Users not assigned to any DAG have global access and can see all records in the project.

Records are assigned to DAG's using the 'Record Status Dashboard' in REDCap. In this project, one DAG group was created called 'records1_2' that contains Record IDs 1 & 2.

#### 16.1.0.5 R

`redcap_dag_read()`

**View the DAGs and their unique `data_access_group_id`:**

```
%%capture
%%R
dag <- redcap_dag_read(redcap_uri = url, token)$data
```

```
%%R
dag_tbl <- gt(head(dag))
gt::gtsave(dag_tbl, filename = 'export_dag1.html', path = "./files/export_files/")
```

**Use `redcap_read_oneshot` to view record DAG assignments:**

```
%%capture
%%R
data_dag <- redcap_read_oneshot(
    redcap_uri = url,
    token = token,
    export_data_access_groups = TRUE,
    fields = "record_id")$data
```

```
%%R
data_dag_tbl <- gt(head(data_dag))
gt::gtsave(data_dag_tbl, filename = 'export_dag2.html', path = "./files/export_files/")
```

For the REDCapR package, refer to the `data_user` table printed in Chapter 15 to see user DAG assignments.

#### 16.1.0.6 Python

`export_dags()`, `export_user_dag_assignment()`

**View the DAGs and their unique `data_access_group_id`:**

```
project.export_dags(format_type='df')
```

| | data_access_group_name | unique_group_name | data_access_group_id |
|---|---|---|---|
| 0 | Limited Access | limited_access | 2707 |
| 1 | records1_2 | records1_2 | 2716 |

**View users' DAG assignments:**

```
project.export_user_dag_assignment(format_type='df') #exports users and their assigned DAGs
```

| | username | redcap_data_access_group |
|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | records1_2 |
| 1 | caitlin.drover@doh.wa.gov | NaN |
| 2 | emily.pearman@doh.wa.gov | NaN |

**Use `export_records` to view which records belong to which data access group.**

```
dag = project.export_records(format_type = 'df', export_data_access_groups = True, fields =
dag.head()
```

| record_id | redcap_event_ |
|---|---|
| 1 | personal_info_<br>notifications_a<br>case_intake_ar |
| 2 | personal_info_<br>notifications_a |

**Note:** Even though only `record_id` was specified for export under the `fields` argument, PyCap will automatically include all the variables that make the unique_key (`record_id`, `redcap_event_name`, `redcap_repeat_instrument`, `redcap_repeat_instance`).

# 17 Logging

The REDCap log provides an audit trail to track record creation, deletion, update and export. It also tracks changes made to survey design and user rights.

## 17.1 Project Setup

Install and load the necessary packages

#### 17.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 17.1.0.2 Python

```
# import pycap
import redcap
import json
import pandas as pd
# import requests
```

Assign your project URL and Token

#### 17.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 17.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

#### 17.1.0.5 R

`redcap_log_read()`

The `redcap_log_read` function is only available on REDCapR Version 1.1.9 and later.

```
%%capture
%%R
log <- redcap_log_read(redcap_uri = url, token)$data
```

```
%%R
log_tbl <- gt(head(log))
gt::gtsave(log_tbl, filename = 'export_log.html', path = "./files/export_files/")
```

#### 17.1.0.6 Python

`export_logging()`

```
log = project.export_logging(format_type='df')
log.head()
```

| | timestamp | username | action | details |
|---|---|---|---|---|
| 0 | 2024-04-12 17:23 | caitlin.drover@doh.wa.gov | Manage/Design | Export Logging (API) |
| 1 | 2024-04-12 17:22 | caitlin.drover@doh.wa.gov | Manage/Design | Export instrument-event mappings (A |
| 2 | 2024-04-12 17:22 | caitlin.drover@doh.wa.gov | Data export (API) | export_format: CSV, rawOrLabel: ra |
| 3 | 2024-04-12 17:22 | caitlin.drover@doh.wa.gov | Manage/Design | Download data dictionary (API) |
| 4 | 2024-04-12 17:22 | caitlin.drover@doh.wa.gov | Manage/Design | Export User-DAG assignments (API) |

# 18 Survey Link (REDCapR Only)

## 18.1 Project Setup

Install and load the necessary packages

#### 18.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 18.1.0.2 Python

```
# import pycap
import redcap
import json
import pandas as pd
# import requests
```

Assign your project URL and Token

#### 18.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 18.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

#### 18.1.0.5 R

`redcap_survey_link_export_oneshot()`

To export the survey link, the instrument must be enabled as a survey in REDCap. If it is a longitudinal survey, you must specify the event name (and instance if repeating) along with the instrument name. This export can only return a survey link for already existing records.

```
%%capture
%%R
link <- redcap_survey_link_export_oneshot(
    record = 6,
    instrument = "test_information",
    redcap_uri = url,
    token = token,
    event = "case_intake_arm_1"
)
result$survey_link
```

```
%%R
link$survey_link
```

```
[1] "https://dev-redcap.doh.wa.gov/surveys/?s=k4z7SZWpuKBi4mQt"
```

# 19 REDCap Version

## 19.1 Project Setup

Install and load the necessary packages

#### 19.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 19.1.0.2 Python

```
# import pycap
import redcap
import json
import pandas as pd
# import requests
```

Assign your project URL and Token

#### 19.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 19.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

REDCap version can be viewed using API calls.

#### 19.1.0.5 R

```
redcap_version()
```

```
%%capture --no-stdout
%%R
redcap_version(
  url,
  token,
  verbose = TRUE,
  config_options = NULL
)
```

```
[1] '14.0.18'
```

#### 19.1.0.6 Python

```
export_version()
```

```
project.export_version()
```

```
Version('14.0.18')
```

# 20 Appendix

## 20.1 Project Setup

Install and load the necessary packages

#### 20.1.0.1 R

```
%%capture
%%R

library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

version <- packageVersion("REDCapR")
version
```

```
[1] '1.1.9005'
```

In this project, we will use the bleeding edge version of REDCapR available on Github

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
```

```
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] "REDCapR package up to date"
```

#### 20.1.0.2 Python

```
# import pycap
import redcap
import json
import pandas as pd
# import requests
```

Assign your project URL and Token

#### 20.1.0.3 R

```
%%R
path = paste0("C:/Users/", Sys.getenv("USERNAME"), '/json_api_data.json')
token <- jsonlite::fromJSON(path)$dev_token$'308'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 20.1.0.4 Python

```
path_to_json = f"C:/Users/{os.environ.get('USERNAME')}/json_api_data.json"
api_key = json.load(open(path_to_json))
api_token = api_key['dev_token']['308']
api_url = api_key['dev_url']
project = redcap.Project(api_url, api_token)
```

## 20.2 Filter Data During Export

REDCapR and PyCap functions have options for filtering data upon export. If a REDCap project has a large amount of data that is slow to export, then we recommend using REDCapR/PyCap functions to filter the data during export when applicable.

Creating custom reports within REDCap and then exporting those filtered reports is another option for filtering the data before export for projects with a large amount of data. See Chapter 9.

### 20.2.1 Filter By Record ID

Export data for Record IDs 1 and 2.

#### 20.2.1.1 R

```
%%capture
%%R
data_by_record <- redcap_read_oneshot(
    records = c(1,2),
    redcap_uri = url,
    token = token
)$data
```

```
%%R
data_by_record_tbl <- gt(head(data_by_record))
gt::gtsave(data_by_record_tbl, filename = 'export_records_filtered_1.html', path = "./files/e
```

#### 20.2.1.2 Python

```
project.export_records(records=['1','2'],
                       raw_or_label='label',
                       format_type='df').reset_index()
```

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | first_name | |
|---|---|---|---|---|---|---|
| 0 | 1 | Personal Info | NaN | NaN | John | |
| 1 | 1 | Notifications | NaN | NaN | NaN | |
| 2 | 1 | Case Intake | NaN | 1.0 | NaN | |

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | first_name | |
|---|---|---|---|---|---|---|
| 3 | 1 | Notifications | Close Contacts | 1.0 | NaN | |
| 4 | 1 | Notifications | Close Contacts | 2.0 | NaN | |
| 5 | 2 | Personal Info | NaN | NaN | Jane | |
| 6 | 2 | Notifications | NaN | NaN | NaN | |
| 7 | 2 | Case Intake | NaN | 1.0 | NaN | |
| 8 | 2 | Case Intake | NaN | 2.0 | NaN | |
| 9 | 2 | Notifications | Close Contacts | 1.0 | NaN | |
| 10 | 2 | Notifications | Close Contacts | 2.0 | NaN | |

### 20.2.2 Filter By Date

Data exports can be filtered by the date the record was added or modified. In the following example, we will view all records that were modified or added after April 9, 2024. For more information on how these records were modified, refer to the logging section of this guide. See Chapter 17.

#### 20.2.2.1 R

```
%%capture
%%R
data_by_date <- redcap_read_oneshot(
    redcap_uri = url,
    token = token,
    datetime_range_begin = as.POSIXct("2024/04/09",
                                      format = "%Y/%m/%d")
)$data
```

**Note:** Need to specify date format as seen above using as.POSIXct

```
%%R
data_by_date_tbl <- gt(data_by_date)
gt::gtsave(data_by_date_tbl, filename = 'export_records_filtered_2.html', path = "./files/exp
```

#### 20.2.2.2 Python

```
from datetime import datetime
project.export_records(date_begin=datetime.fromisoformat("2024-04-09"),
                       format_type='df')
```

| record_id | redcap_event_ |
|---|---|
| 1 | personal_info_<br>notifications_a<br>case_intake_ar<br>notifications_a<br>notifications_a |
| 2 | personal_info_<br>notifications_a<br>case_intake_ar<br>case_intake_ar<br>notifications_a<br>notifications_a |
| 3 | personal_info_<br>notifications_a<br>case_intake_ar<br>notifications_a<br>notifications_a |

### 20.2.3 Filter By Field Value

**Export records that reported 'female' for gender.**

#### 20.2.3.1 R

```
%%capture
%%R
data_by_gender <- redcap_read_oneshot(
    redcap_uri = url,
    token = token,
    filter_logic = "[gender] = '0'"
)$data %>%
 select(c(record_id, gender))
```

```
%%R
data_by_gender_tbl <- gt(head(data_by_gender))
gt::gtsave(data_by_gender_tbl, filename = 'export_records_filtered_3.html', path = "./files/e
```

**Note:** When filtering on REDCap's multiple choice variables, yes/no variables, and checkboxes, you must put quotes around the coded value when using the `filter_logic` argument, otherwise REDCap will not perform the filtering correctly.

#### 20.2.3.2 Python

```
project.export_records(filter_logic="[gender] = 0",
                       format_type='df'
                      ).reset_index()[['record_id', 'gender']]
```

| | record_id | gender |
|---|---|---|
| 0 | 2 | 0 |
| 1 | 4 | 0 |
| 2 | 6 | 0 |

**Export records that reported an age greater than 20.**

#### 20.2.3.3 R

```
%%capture
%%R
data_by_age <- redcap_read_oneshot(
    redcap_uri = url,
    token = token,
    filter_logic = "[age] > 20 "
)$data%>%
 select(c(record_id, age))
```

```
%%R
data_by_age_tbl <- gt(head(data_by_age))
gt::gtsave(data_by_age_tbl, filename = 'export_records_filtered_4.html', path = "./files/exp
```

**Note:** Because age is a numeric field in REDCap, it does not need quotes around the number 20.

#### 20.2.3.4 Python

```
project.export_records(filter_logic="[age] > 20",
                       format_type='df'
                      ).reset_index()[['record_id', 'age']]
```

| | record_id | age |
|---|---|---|
| 0 | 2 | 29 |
| 1 | 4 | 28 |
| 2 | 5 | 34 |
| 3 | 6 | 25 |

## 20.3 Export Selected Fields

**Export first and last name**

#### 20.3.0.1 R

```
%%capture
%%R
#specifying record_id automatically also pulls the event, instrument and instance columns (wh
field_subset_1 <- redcap_read_oneshot(
    fields = c("record_id","first_name","last_name"),
    redcap_uri = url,
    token = token
)$data
```

```
%%capture
%%R
field_subset_1_tbl<- gt(head(field_subset_1))
gt::gtsave(field_subset_1_tbl, filename = 'export_records5.html', path = "./files/export_file
```

**Note:** if `record_id` is not specified, no identifier fields will be exported. By including `record_id` in the `fields` argument, all variables that make up the unique key are automatically exported.

#### 20.3.0.2 Python

```
project.export_records(records=['3','4'],
                       fields=["first_name","last_name"],
                       format_type='df')
```

| record_id | redcap_event_ |
|---|---|
| 3 | personal_info_<br>notifications_a<br>case_intake_ar |
| 4 | personal_info_<br>notifications_a<br>case_intake_ar<br>case_intake_ar |

**Note:** if `record_id` is not specified, all fields that make up the unique key will still be exported.

## 20.4 Export Specific Instruments

See Chapter 13 to get a list of all the instrument names in your project. Specifying instruments to export will still export all rows of the project data (including rows not relevant to the desired instrument). However, it is useful because it will only export the fields (columns) in that instrument. Use the `filter_logic` argument and the `<form>_complete` variable (automatically created by REDCap for each form) to get the desired output.

In this example, we use the `symptoms_complete` field to export the 'symptoms' form and all associated data.

#### 20.4.0.1 R

```
%%capture
%%R

records_dem <- redcap_read_oneshot(redcap_uri = url,
                           fields = "record_id",
                           forms = "symptoms",
                           filter_logic = '[symptoms_complete] <> ""',
```

```
                            token = token
                                        )$data

records_dem_tbl <- gt(head(records_dem))
gt::gtsave(records_dem_tbl, filename = 'export_records6.html', path = "./files/export_files/
```

```
%%capture
%%R

records_dem_tbl <- gt(records_dem)
gt::gtsave(records_dem_tbl, filename = 'export_records6.html', path = "./files/export_files/
```

**Note:** You must add `record_id` to the `fields` argument for the data to export with the `record_id` and associated unique key attached.

#### 20.4.0.2 Python

```
project.export_records(forms='symptoms',
                       filter_logic ="[symptoms_complete] <>''",
                       format_type='df')
```

| record_id | redcap_event_ |
|---|---|
| 1 | case_intake_ar |
| 2 | case_intake_ar<br>case_intake_ar |
| 3 | case_intake_ar |
| 4 | case_intake_ar<br>case_intake_ar |
| 5 | case_intake_ar |
| 6 | case_intake_ar |

**Note:** if `record_id` is not specified, all fields that make up the unique key will still be exported.

## 20.5 Export Data as CSV (PyCap Only)

#### 20.5.0.1 Python

In Section 8.3 and Section 9.3, it was noted that to export labeled headers, the data needs to be exported as a csv. See the example below on how to do this.

```
from io import StringIO

data_csv = StringIO(project.export_records(records='2',
                        raw_or_label='label',
                        raw_or_label_headers='label',
                        format_type='csv'))
df_csv = pd.read_csv(data_csv, sep=',')
df_csv
```

| | Record ID | Event Name | Repeat Instrument | Repeat Instance | First Name | Last Name | Phone Num |
|---|---|---|---|---|---|---|---|
| 0 | 2 | Personal Info | NaN | NaN | Jane | Doe | (999) 999-9 |
| 1 | 2 | Notifications | NaN | NaN | NaN | NaN | NaN |
| 2 | 2 | Case Intake | NaN | 1.0 | NaN | NaN | NaN |
| 3 | 2 | Case Intake | NaN | 2.0 | NaN | NaN | NaN |
| 4 | 2 | Notifications | Close Contacts | 1.0 | NaN | NaN | NaN |
| 5 | 2 | Notifications | Close Contacts | 2.0 | NaN | NaN | NaN |

## 20.6 Creating a Reference Class (REDCapR only)

#### 20.6.0.1 R

This Reference Class represents a REDCap project. Once some values are set that are specific to a REDCap project (such as the URI and token), later calls are less verbose (such as reading and writing data).

**First, define the project:**

```
%%capture
%%R
project <- REDCapR::redcap_project$new(redcap_uri=url, token=token)
ds_all <- project$read()
```

```
%%capture
%%R
data <- project$read(fields = c("record_id", "gender", "first_name"))$data
```

```
%%capture
%%R

data_tbl <- gt(data)
gt::gtsave(data_tbl, filename = 'export_records7.html', path = "./files/export_files/")
```

**Pull the `record_id` for all 'female' records:**

```
%%R

record_of_females <- data$record_id[data$gender=='0']
record_of_females
```

```
 [1] NA NA  2 NA NA NA NA NA  4 NA NA NA NA NA  6 NA NA
```

## 20.7 Clean Checkbox Choices (REDCapR Only)

#### 20.7.0.1 R

REDCapR has a `checkbox_choices` function that can be used to neatly list all answer options for a checkbox field.

First pull the metadata:

```
%%capture
%%R

metadata <- redcap_metadata_read(
    redcap_uri = url,
    token = token
)$data
```

Select the checkbox field that you would like to view:

```
%%R
symptoms_exp <- metadata[metadata$field_name == "symptoms_exp",]$select_choices_or_calculati
symptoms_exp_list <- REDCapR::checkbox_choices(select_choices=symptoms_exp)
symptoms_exp_list
```

```
# A tibble: 11 x 2
   id    label
   <chr> <chr>
 1 1     Sore Throat
 2 2     Cough
 3 3     Shortness of Breath
 4 4     Chest Pain
 5 5     Headache
 6 6     Runny Nose
 7 7     Congestion
 8 8     Fever
 9 9     Body Aches
10 10    Nausea/Vomiting
11 11    Diarrhea
```

## 20.8 REDCap Constants (REDCapR Only)

#### 20.8.0.1 R

You can quickly search for the numerical values of several 'constants' within REDCap. For example, when exporting data from REDCap, all instruments end with a variable called `<form_name>_complete` that when exported as raw data will take the values of 0, 1 or 2. You can see what each numerical value means by using `constant("form_incomplete")` and `constant("form_complete")`.

Other constants across all REDCap projects include the values for the various user rights settings. For a full list of constants available, refer to the following documentation.

**View the `<form_name>_complete` REDCap constant values:**

```
%%R
REDCapR::constant(c(
  "form_incomplete",
  "form_complete",
  "form_unverified"
))
```

```
[1] 0 2 1
```

# Part III

# Importing to REDCap

# 21 Records

#### 21.0.0.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 21.0.0.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

#### 21.0.0.3 R

`redcap_write_oneshot()` and `redcap_write()`

Records can be imported into a REDCap project from a dataframe in R using `redcap_write_oneshot()` to write a records all at once, or using `redcap_write()` which can batch the records to be imported so the server is not overwhelmed in the case of large imports. These methods will accept either an R dataframe or tibble containing the data to be imported.

If the `record_id`(s) being imported already exists in the REDCap project, the imported data will overwrite the previously existing data for that record. Using a `record_id` that does not already exist will create a new record. See Section 8.5 on how to use the API to find the next available `record_id`.

The `overwrite_with_blanks` argument is set to 'FALSE' by default; under this setting, if blank values are imported for fields on existing REDCap records and that data is not missing in REDCap, these values will not be overwritten as missing. If you want to overwrite existing data as missing, be sure to use `overwrite_with_blanks = 'TRUE'`.

```
%%capture --no-display
%%R
# Define data to import
df1 <- data.frame(record_id = c(7,8),
                  first_name = c("John","Jane"),
```

```
                  last_name = c("Doe","Doe")
                  )
```

```
%%capture --no-stdout
%%R
redcap_write_oneshot(df1, redcap_uri=url, token=token)
```

```
$success
[1] TRUE

$status_code
[1] 200

$outcome_message
[1] "2 records were written to REDCap in 0.6 seconds."

$records_affected_count
[1] 2

$affected_ids
[1] "7" "8"

$elapsed_seconds
[1] 0.634845

$raw_text
[1] ""
```

```
%%R
df2 <- data.frame(record_id = 9,
                  first_name = "John",
                  last_name = "Doe"
                  )
```

```
%%capture --no-stdout
%%R
redcap_write(df2, redcap_uri=url, token=token)
#optional argument: batch_size = 100 (default)
```

```
$success
```

```
[1] TRUE

$status_code
[1] "200"

$outcome_message
[1] "1 records were written to REDCap in 0.4 seconds."

$records_affected_count
[1] 1

$affected_ids
[1] "9"

$elapsed_seconds
[1] 0.934711
```

#### 21.0.0.4 Python

`import_records()`

Data can be imported as a pandas dataframe, json, csv, or xml, specified by the `import_format` argument (default is json).

If the `record_id`(s) being imported already exists in the REDCap project, the imported data will overwrite the previously existing data for that record. Using a `record_id` that does not already exist will create a new record. The `force_auto_number = 'True'` argument will automatically reassign existing `record_ids` to new `record_ids` during import. If set to 'False' and your Record ID's to import already exist in REDCap, they will overwrite the existing REDCap records during import. You can also see Section 8.5 on how to use the API to find the next available `record_id`.

The `overwrite` argument is set to 'normal' by default; under this setting, if blank values are imported for fields on existing REDCap records and that data is not missing in REDCap, these values will not be overwritten as missing. If you want to overwrite existing data as missing, be sure to use `overwrite = 'overwrite'`.

```
df_py = [{'record_id': 7,
  'redcap_event_name': 'personal_info_arm_1',
  'redcap_repeat_instrument': '',
  'redcap_repeat_instance': None,
  'first_name': 'John',
  'last_name': 'Doe'},
```

```
 {'record_id': 8,
  'redcap_event_name': 'personal_info_arm_1',
  'redcap_repeat_instrument': '',
  'redcap_repeat_instance': None,
  'first_name': 'Jane',
  'last_name': 'Doe'}]

project.import_records(df_py, force_auto_number=True)
```

```
{'count': 2}
```

**Note:** For troubleshooting import errors, please thoroughly review Chapter 27. This chapter goes into to detail about the limitations to importing and provides more detailed import examples.

# 22 Files

Files are optional attachments to individual records.

File uploads are a unique field type in REDCap that accept a variety of file types, including images and other documents. Unlike other export methods, importing files only works for one file field for one record at a time.

If the project has repeating events (i.e. a longitudinal project), the event name that the record is in must be specified. If the file field of interest is in a repeat instance, the instance number must also be specified.

### 22.0.1 Project Setup

#### 22.0.1.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}
```

```
else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 22.0.1.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

#### 22.0.1.3 R

```
redcap_file_upload_oneshot()
```

```
%%capture --no-stdout
%%R
redcap_file_upload_oneshot(file_name='./files/test_file.png', record=7, field='test_upload',
```

```
$success
[1] TRUE

$status_code
[1] 200

$outcome_message
[1] "file uploaded to REDCap in 1.0 seconds."

$records_affected_count
[1] 1

$affected_ids
[1] "7"

$elapsed_seconds
[1] 1.024144

$raw_text
[1] ""
```

#### 22.0.1.4 Python

```
import_file()
```

```
tmp_file = tempfile.TemporaryFile()
project.import_file(record="7",
                    field="test_upload",
                    file_name="./files/test_file.png",
                    file_object=tmp_file,
                    event="case_intake_arm_1")
```

```
[{}]
```

The output is a list of an empty JSON object, as expected for a successful file import using this method.

# 23 Metadata

Metadata refers to the project's set up characteristics, including field attributes grouped by instrument assignment. Metadata can be thought of as the project's data dictionary.

In these examples, we will export the project metadata and re-import it so that no changes are made to the project.

### 23.0.1 Project Setup

#### 23.0.1.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 23.0.1.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

#### 23.0.1.3 R

```
redcap_metadata_write()
```

```
%%R
metadata <- redcap_metadata_read(redcap_uri=url, token=token)$data
```

```
R[write to console]: The data dictionary describing 30 fields was read from REDCap in 0.7 se
```

```
%%R
tbl<- gt(head(metadata))
gt::gtsave(tbl, filename = 'import_metadata.html', path = "./files/")
```

```
%%capture --no-stdout
%%R
redcap_metadata_write(metadata, redcap_uri=url, token=token)
```

```
$success
[1] TRUE

$status_code
[1] 200

$outcome_message
[1] "30 fields were written to the REDCap dictionary in 0.7 seconds."

$field_count
[1] 30

$elapsed_seconds
[1] 0.688035

$raw_text
[1] ""
```

#### 23.0.1.4 Python

```
import_metadata()
```

```
metadata = project.metadata
metadata[0]
```

```
{'field_name': 'record_id',
 'form_name': 'demographics',
 'section_header': '',
 'field_type': 'text',
 'field_label': 'Study ID',
 'select_choices_or_calculations': '',
 'field_note': '',
 'text_validation_type_or_show_slider_number': '',
 'text_validation_min': '',
 'text_validation_max': '',
 'identifier': '',
```

```
 'branching_logic': '',
 'required_field': '',
 'custom_alignment': '',
 'question_number': '',
 'matrix_group_name': '',
 'matrix_ranking': '',
 'field_annotation': ''}
```

```
project.import_metadata(to_import=metadata)
```

```
30
```

# 24 Instrument Event Map

This shows which instruments are associated with each event in longitudinal projects.

### 24.0.1 Project Setup

#### 24.0.1.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 24.0.1.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

#### 24.0.1.3 R

Cannot be imported using REDCapR. *See Section 27.1 for more information.*

#### 24.0.1.4 Python

`import_instrument_event_mappings()`

In this example, we will export the project's instrument-event mapping and re-import it so that no changes are made to the project.

```
instrument_event_mappings = project.export_instrument_event_mappings(format_type='df')
instrument_event_mappings
```

| | arm_num | unique_event_name | form |
|---|---|---|---|
| 0 | 1 | personal_info_arm_1 | demographics |

| | arm_num | unique_event_name | form |
|---|---|---|---|
| 1 | 1 | case_intake_arm_1 | symptoms |
| 2 | 1 | case_intake_arm_1 | test_information |
| 3 | 1 | notifications_arm_1 | close_contacts |
| 4 | 1 | notifications_arm_1 | work_information |

```
project.import_instrument_event_mappings(instrument_event_mappings, import_format='df')
```

```
5
```

# 25 Users and User Roles

### 25.0.1 Project Setup

#### 25.0.1.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 25.0.1.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

REDCap allows strict control of user rights for each project. These rights include a user's ability to edit, export and view data, add or edit reports, import data, create records, etc. Privileges to view and export data are specified for each instrument.

User roles can be set with predefined privileges and users can then be assigned to these user role groups.

## 25.1 Users

#### 25.1.0.1 R

Cannot be imported with REDCapR. *See Section 27.1 for more information.*

#### 25.1.0.2 Python

`import_users()`

In this example, we will export the project's users and re-import it so that no changes are made to the project.

```
users = project.export_users(format_type='df')
users
```

| | username | email | firstname | lastname | expiration | data_acces |
|---|---|---|---|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | Alexey.Gilman@doh.wa.gov | Alexey | Gilman | NaN | NaN |
| 1 | caitlin.drover@doh.wa.gov | Caitlin.Drover@doh.wa.gov | Caitlin | Drover | NaN | NaN |
| 2 | emily.pearman@doh.wa.gov | emily.pearman@doh.wa.gov | Emily | Pearman | NaN | NaN |

```
project.import_users(users, import_format='df')
```

3

**Note:** attemping to import a user already assigned to a user role will result in an error.

## 25.2 User Roles

#### 25.2.0.1 R

Cannot be imported with REDCapR. *See Section 27.1 for more information.*

#### 25.2.0.2 Python

Roles can be imported using `import_user_roles()` and assigned to a project user using `import_user_role_assignment()`.

**Importing User Roles**

```
user_roles = project.export_user_roles(format_type='df')
user_roles
```

| | unique_role_name | role_label | design | alerts | user_rights | data_access_groups | reports | stats_ |
|---|---|---|---|---|---|---|---|---|
| 0 | U-1564393FT9 | Limited Role | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | U-5354FA3HYL | Admin | 1 | 1 | 1 | 1 | 1 | 1 |

**Note:** the `unique_role_name` is automatically generated by REDCap.

```
project.import_user_roles(user_roles, import_format='df')
```

```
3
```

### Importing User Role Assignments

```
user_role_assign = project.export_user_role_assignment(format_type='df')
user_role_assign
```

| | username | unique_role_name | data_access_group |
|---|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | NaN | NaN |
| 1 | caitlin.drover@doh.wa.gov | NaN | NaN |
| 2 | emily.pearman@doh.wa.gov | NaN | NaN |

```
user_role_assign['unique_role_name'].astype(str).replace('nan', np.NaN)
user_role_assign.loc[0,'unique_role_name'] = 'U-5354FA3HYL'
user_role_assign
```

| | username | unique_role_name | data_access_group |
|---|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | U-5354FA3HYL | NaN |
| 1 | caitlin.drover@doh.wa.gov | NaN | NaN |
| 2 | emily.pearman@doh.wa.gov | NaN | NaN |

```
project.import_user_role_assignment(user_role_assign, import_format='df')
```

```
3
```

```
user_role_assign.loc[0,'unique_role_name'] = np.NaN
user_role_assign
```

| | username | unique_role_name | data_access_group |
|---|---|---|---|
| 0 | alexey.gilman@doh.wa.gov | NaN | NaN |
| 1 | caitlin.drover@doh.wa.gov | NaN | NaN |
| 2 | emily.pearman@doh.wa.gov | NaN | NaN |

```
project.import_user_role_assignment(user_role_assign, import_format='df')
```

```
3
```

# 26 Data Access Groups (DAGs)

### 26.0.1 Project Setup

#### 26.0.1.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 26.0.1.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

In addition to user roles for access control, DAGs control which records each user can access. Users assigned to a DAG can only access records assigned to that particular DAG, and are blinded to records outside of their group. DAGs are particularly useful in multi-site or multi-jurisdictional projects to restrict sites from viewing records from other sites.

Users can be assigned to a DAG after being added to the project. Users can be in multiple DAGs. Users not assigned to any DAG have global access and can see all records in the project.

Records are assigned to DAG's using the 'Record Status Dashboard' in REDCap.

#### 26.0.1.3 R

Cannot be imported with REDCapR. *See Section 27.1 for more information.*

#### 26.0.1.4 Python

DAGs can be imported using `import_dags()` and assigned using `import_user_dag_assignment()`. If the API user is assigned to multiple DAGs, they can be switched between DAGs using `switch_dag()`.

View the current DAGs by exporting them:

```
dags = project.export_dags(format_type='df')
dags
```

| | data_access_group_name | unique_group_name | data_access_group_id |
|---|---|---|---|
| 0 | Full Access | full_access | 2708 |
| 1 | Limited Access | limited_access | 2709 |

Create and import a new DAG:

```
new_dag = [{"data_access_group_name": "Test DAG", "unique_group_name": ""}]
project.import_dags(new_dag)
```

```
1
```

**Note:** the `unique_group_name` field must be left blank as this is auto-generated by REDCap from the `data_access_group_name`.

The newly created DAG can now be seen and assigned to.

```
dags = project.export_dags(format_type='df')
dags
```

| | data_access_group_name | unique_group_name | data_access_group_id |
|---|---|---|---|
| 0 | Full Access | full_access | 2708 |
| 1 | Limited Access | limited_access | 2709 |
| 2 | Test DAG | test_dag | 2721 |

```
dag_mapping = [{"username": 'alexey.gilman@doh.wa.gov', "redcap_data_access_group": "test_dag
project.import_user_dag_assignment(dag_mapping)
```

```
1
```

**Note:** the `redcap_data_access_group` name when importing is the same as `unique_group_name` when exporting DAGs.

```
project.delete_dags(["test_dag"])
```

```
1
```

# 27 Appendix

### 27.0.1 Project Setup

#### 27.0.1.1 R

```
%%capture --no-display
%%R
library("dplyr")
library("jsonlite")
library("tidyr")
library("REDCapR")
library("knitr")
library("remotes")
library("gt")
```

```
%%capture --no-display --no-stdout
%%R

# Detach REDCapR if already loaded, and download the latest version
if (version!='1.1.9005') {
    detach("package:REDCapR", unload=TRUE)
    remotes::install_github("OuhscBbmc/REDCapR")
    library("REDCapR")
    print(packageVersion("REDCapR"))
}

else {
    print("REDCapR package up to date")
}
```

```
[1] '1.1.9005'
```

```
%%R
# Load API tokens from the json file
token <- jsonlite::fromJSON('./../../json_api_data.json')$dev_token$'309'
url <- "https://dev-redcap.doh.wa.gov/api/"
```

#### 27.0.1.2 Python

```
import redcap
import json
import csv
import pandas as pd
import numpy as np
import requests
import tempfile
```

```
# Load API tokens from the json file
key = json.load(open('./../../json_api_data.json'))
token = key['dev_token']['309']
url = key['dev_url']

project = redcap.Project(url, token)
```

## 27.1 Limitations to Importing

#### 27.1.0.1 R

**Field Names**
- Field names cannot be imported using REDCapR.
- Can be exported using `redcap_variables()`.

**Forms/Instruments** - Forms and instruments cannot be imported using REDCapR. Can be uploaded as a ZIP file in the REDCap web application under the 'Designer' page of the REDCap Project.
- Can be downloaded as pdf using `redcap_instrument_download()`.

**Instrument/Event Map** - Instrument/event mapping cannot be imported using REDCapR. Can be imported with REDCap's native API or uploaded as a CSV in the REDCap web application under the 'Designate Instruments for My Events' on the 'Project Setup' page of the REDCap project. - Can be exported using `redcap_event_instruments()`.

**Reports** - Cannot be imported using REDCapR. - Can be exported using `redcap_report()`.

**Users** - Cannot be imported using REDCapR. Can be imported with REDCap's native API or uploaded as a CSV in the REDCap web application under the 'User Rights' page of the REDCap Project. - Can be exported using `redcap_users_export()`.

**User Roles** - Cannot be imported using REDCapR. Can be imported with REDCap's native API or uploaded as a CSV in the REDCap web application under the 'User Rights' page of the REDCap Project. - Can be exported using `redcap_users_export()`.

**Data Access Groups (DAGs)** - Cannot be imported using REDCapR. Can be imported with REDCap's native API or uploaded as a CSV inthe REDCap web application under the 'DAGs' page of the REDCap Project. - Can be exported using `redcap_dag_read()`.

**Logging** - Cannot be imported. - Can be exported using `redcap_log_read()`.

#### 27.1.0.2 Python

**Field Names** - Field names alone cannot be imported using PyCap.
- Can be exported using `export_field_names()`.

**Forms/Instruments** - Forms and instruments cannot be imported using PyCap. Can be uploaded as a ZIP file in the REDCap web application under the 'Designer' page of the REDCap Project. - Can be exported using `export_instruments()` and `export_repeating_instruments_events()` for the settings.

**Reports** - Reports cannot be imported using PyCap.
- Can be exported using `export_records()`.

**Logging** - Logging cannot be imported using PyCap.
- Can be exported using `export_logging()`.

## 27.2 General Import notes (regardless of package used)

### 27.2.1 REDCap Validations

**Identifiers**

- No duplicates in the unique key.
  - Note: Duplicates in the unique key will be automatically dropped during import (only one occurence of the duplicate will be uploaded) without any warning or error message. The output message only lists the number of unique ids that were imported.

- If `redcap_repeat_instrument` is part of the unique key its value must be valid and associated with the correct event.
  - Exception for longitudinal projects with repeating events and no repeating instruments: Data exported form these projects will have the 'redcap_repeat_instrument' column with all values set to NA. This column is technically not needed in this type of project and API import will be accepted with (as long as all values are NA) or without it.

**Data Structure**

- That non-missing fields are in the correct instrument/event row (for longitudinal projects or projects with repeating instruments).
- That 'redcap_repeat_instance' is filled when required and missing when required (for projects with repeating events or instruments).
- That there are no extra columns in the data being imported that are not fields in the REDCap project.

**Values**

- Radio and Dropdown Fields
  - That values are within the set of `select_choices_or_calculations` as defined in the metadata.
- Text Fields
  - That only numeric values are present in numeric validation fields.
  - That only integer values are present in integer validation fields.
  - That date and datetime fields are in the YMD format (can be YYYY/MM/DD or YYYY-MM-DD).
  - That email fields are formatted as 'something' + @ + 'something' + . + 'something' (does not check for valid domain names; multiple '.somethings' are accepted after @ symbol (e.g., email@doh.wa.gov)).
  - That phone numbers have 10 digits and that the area code starts with digits 2-9 (can have ###-###-#### format or just ##########).
  - That zipcode fields have either 5 or 9 digits, and if 9 digits there is a hyphen after the 5th digit.

  - That 'alpha only' text validation fields contain only letters (no spaces, numbers, or punctuation).
  - That there are only 10 digits in the fields with MRN 10 digit validation fields.
  - That there are only digits and '-' or '_' in fields with MRN generic validation fields.
- Slider Fields
  - That slider field values are within the `text_validation_min` and `text_validation_max` range as defined in the metadata.
- Checkbox, True/False, Yes/No Fields
  - Values must be 0 or 1 or missing.
- Text Box (no validation) and Notes box
  - Values are less than 65,000 characters.
- Calculated Fields
  - Regardless of the value in the data attempting to be imported, REDCap will auto-calculate these fields (assuming all of the inputs to the calculation are available). The values being imported are ignored.
- Files and Signature Fields
  - Regardless of the value in the data attempting to be imported, REDCap will ignore these fields when importing records. All file-type field imports are done through a separate process.
- Dynamic Query Fields
  - That values are within the dynamic query options (value options not exported in metadata).

**NOT Validated**

- A second instance of a repeating event or instrument can be uploaded without a first instance exisiting (in import data or existing project data).
- Data can be imported outside of the specified validation range for text fields with the following validation types: dates, times, datetimes, integers, and numbers. Recommendation: run Data Quality rule D after import.
- Missing required fields can be imported. Recommendation: run Data Quality rule B after upload.
- Incorrect calculated field values can be imported. Recommendation: run Data Quality rule H after upload.

- Files and Signature fields can only be imported through the file import method, any data attempting to be imported using the record import method is ignored.
- Field values that violate the REDCap project branching logic can be imported via the API without issues, however, when you go to open that record in REDCap it will alert you to the invalid logic.

**REDCap Log Behavior**

- If data for an existing record is imported via the API but the incoming fields for that records are the same values as the fields already stored for that record in REDCap (no changes made), the API import will run, however, there will be no 'Update record (API)' action logged for this record.
- If a row containing a valid combination of unique fields but NA accross all other fields is imported via API, the import will run. However, this 'blank' record will not be created and there will be no action logged in the REDCap log for this record..

## 27.3 Data Validation Tools

#### 27.3.0.1 R

REDCapR has a few data validation functions that can be used to check your data before importing it to your REDCap project. These validations will not be specific to your paricular REDCap project but are general validations that apply to all REDCap projects.

For example, you can check if you have any boolean values (True/False) since REDCap will only accept a raw data import of 0/1 integers. You can also check for duplicates and unique IDs. You can view more details on these data validation functions here.

## 27.4 Example: Uploading Records from a CSV

In this example, we have a csv named "data_to_import.csv" with records to upload.

#### 27.4.0.1 R

```
%%R
df_to_import <- read.csv("./files/data_to_import.csv")
```

```
%%R
tbl <- gt(head(df_to_import))
gt::gtsave(tbl, filename = 'import_csv_data.html', path = "./files/")
```

There are multiple rows per record because this project is longitudinal with repeat instruments and events.

```
%%R
# view which record_id's are currently being used in the data set to import.
unique(df_to_import$record_id)
```

```
[1] 3 4 5 6
```

In the dataframe we will import, the record IDs are 3-6. However, these record IDs already exist in the REDCap project and importing this data would overwrite the exisiting record IDs 3-6. If we want to import these as new records, we will need to renumber the record IDs.

```
%%R
# start by getting the next available record_id
next_record <- redcap_next_free_record_name(redcap_uri=url, token=token)
```

```
R[write to console]: The next free record name in REDCap was successfully determined in 1.1 s
```

```
%%R
### sequence the df_to_import records starting at one
df_to_import <- df_to_import[order(df_to_import$record_id), , drop = FALSE]
df_to_import$seq <- as.numeric(factor(df_to_import$record_id))
```

```
%%capture
%%R
sequencing <- df_to_import %>% group_by(record_id, seq) %>% summarize(n=n())
```

```
%%R
head(sequencing)
```

```
# A tibble: 4 x 3
# Groups:   record_id [4]
  record_id   seq     n
      <int> <dbl> <int>
```

```
1         3     1     5
2         4     2     6
3         5     3     4
4         6     4     5
```

```
%%R
tbl <- gt(sequencing)
gt::gtsave(tbl, filename = 'sequencing.html', path = "./files/")
```

```
%%R
# Adjust record IDs to start at the next available record_id
df_to_import$record_id <- as.numeric(df_to_import$seq) + (as.numeric(next_record)-1)
unique(df_to_import$record_id)
```

```
[1] 10 11 12 13
```

The record IDs have been changed to new record IDs that don't already exist in the REDCap project.

```
%%R
# Remove the seq var that was created above
df_to_import <- df_to_import %>% select(-seq)
```

### Formatting Date Fields

Date fields in REDCap are character fields with a designated date validation added. There are many different types of date validations/formats that can be chosen for a date field. All date fields must be imported to REDCap formatted as `YYYY-MM-DD`, regardless of the specific date format designated for this field in the REDCap project. (PyCap has an import records argument to change the default YMD format, but REDCapR does not have this option.) Below is an example on how to use the project metadata to isolate and format all date fields before importing data.

```
%%R
# Export metadata
metadata <- redcap_metadata_read(redcap_uri = url, token = token)$data
```

```
R[write to console]: The data dictionary describing 30 fields was read from REDCap in 0.2 se
```

```
%%R
tbl <- gt(head(metadata))
gt::gtsave(tbl, filename = 'import_metadata.html', path = "./files/")
```

Note that the 'text_validation_type_or_show_slider_number' field in the metadata is where the date format is specified.

```
%%R
unique(metadata$text_validation_type_or_show_slider_number)
```

```
[1] NA          "phone"    "integer"  "date_mdy" "email"
```

```
%%R
# Isolate all field_names in the metadata that have any date validation
date_fields <- metadata %>% filter(grepl("date", text_validation_type_or_show_slider_number))
```

```
%%R
# Make a list of all the date fields
date_list <- (date_fields$field_name)
date_list
```

```
[1] "dob"                "symptom_onset"      "test_positive_date"
[4] "prior_covid_date"   "cc_date"            "work_date"
```

```
%%R
# mutate across all date fields to get the desired Y-M-D format.
df_to_import2 <- df_to_import %>%
  mutate(across(all_of(date_list), ~as.Date(., "%m/%d/%Y" )))
```

```
%%R
unique(df_to_import2$test_positive_date)
```

```
[1] NA           "2023-10-10" "2023-10-12" "2021-06-07" "2023-10-03"
```

Now import the new records.

```
%%capture --no-stdout
%%R
redcap_write(df_to_import2, redcap_uri=url, token=token)
```

```
$success
[1] TRUE

$status_code
[1] "200"

$outcome_message
[1] "4 records were written to REDCap in 1.8 seconds."

$records_affected_count
[1] 4

$affected_ids
[1] "10" "11" "12" "13"

$elapsed_seconds
[1] 2.354373
```

```
%%R
records_to_delete <- unique(df_to_import$record_id)
redcap_delete(records_to_delete, arm_of_records_to_delete = 1L, redcap_uri=url, token=token)
```

```
R[write to console]:
R[write to console]: indexing file4ea843144da8 [=====================================] ?,

R[write to console]:
R[write to console]: The 4 records were deleted from REDCap in 0.3 seconds. The http status

$success
[1] TRUE

$status_code
[1] 200

$outcome_message
[1] "The 4 records were deleted from REDCap in 0.3 seconds. The http status code was 200."
```

```
$records_affected_count
[1] 4

$elapsed_seconds
[1] 0.279567

$raw_text
[1] ""
```

#### 27.4.0.2 Python

When reading a csv as a pandas dataframe, Python will take any numeric column with missing data and convert them to float with NaN inserted in the blank cells. In longitudinal projects, we expect many blank cells since the data is wide, as only columns relevant to that event/instrument field are filled out in each row. Many of REDCap's field types (checkbox, yes/no, radio, and form_complete variables) are integers. Pandas will convert these columns to float variables with a decimal place added (e.g. 1.0 instead of 1 for 'Yes' in a yes/no field) and importing this to REDCap will fail.

```
# Read and view data to import
df_to_import = pd.read_csv("./files/data_to_import.csv")
df_to_import.head()
```

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | redcap_surve |
|---|---|---|---|---|---|
| 0 | 3 | personal_info_arm_1 | NaN | NaN | NaN |
| 1 | 3 | notifications_arm_1 | NaN | NaN | NaN |
| 2 | 3 | case_intake_arm_1 | NaN | 1.0 | NaN |
| 3 | 3 | notifications_arm_1 | close_contacts | 1.0 | NaN |
| 4 | 3 | notifications_arm_1 | close_contacts | 2.0 | NaN |

Notice how the redcap_repeat_instance, close_contacts_complete, and work_inperson_yesno are some of the many fields that were converted to float with an added decimal. Importing this dataset as-is will produce errors.

**Solution:** Convert all floats to Int64 Pandas datatype.

- Int64 is a unique pandas datatype that allows numeric fields to contain missing values. For more information, read the documentation here.

- Note: Before applying this solution, ensure that there are no numeric fields in your REDCap project that should have decimals (you will not want to convert these variables to int64 since they would lose their decimal places). Make sure you are familiar with your project's metadata. All radio, checkboxes, yes/no, redcap_repeat_instance, and form_complete variables need to be integers. In REDCap, actual numeric fields are stored as text fields with optional validation. Any text field in REDCap with no validation or with 'numeric' as their validation type will accept numbers with decimal places. Any text fields with other validations types (i.e. zip code, phone number, integer) will not accept decimals.

```
float_list = df_to_import.select_dtypes(include=[np.float64]).columns.values.tolist()
print(float_list)
```

```
['redcap_repeat_instance', 'redcap_survey_identifier', 'demographics_timestamp', 'zip_code',
```

At this point, if needed, you can remove any variables from this list that you need to keep as a float.

```
df_to_import[float_list] = df_to_import[float_list].apply(lambda x: x.astype("Int64"))
df_to_import.head()
```

| | record_id | redcap_event_name | redcap_repeat_instrument | redcap_repeat_instance | redcap_surve |
|---|---|---|---|---|---|
| 0 | 3 | personal_info_arm_1 | NaN | <NA> | <NA> |
| 1 | 3 | notifications_arm_1 | NaN | <NA> | <NA> |
| 2 | 3 | case_intake_arm_1 | NaN | 1 | <NA> |
| 3 | 3 | notifications_arm_1 | close_contacts | 1 | <NA> |
| 4 | 3 | notifications_arm_1 | close_contacts | 2 | <NA> |

You can now see the `redcap_repeat_instance` and `close_contacts_complete` fields are integers. The `<NA>` seen in the blank cells will not interfere with data import. Now you can make any other edits necessary and import the data successfully.

```
# Import data
project.import_records(df_to_import, date_format = 'MDY', import_format = 'df')
```

```
{'count': 4}
```

# References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.