# PyCap Documentation

***Release 1.0.2***

**Scott Burns**

October 20, 2018

Contents

PyCap is an interface to the REDCap Application Programming Interface (API). PyCap is designed to be a minimal interface exposing all required and optional API parameters. My hope is that it makes simple things easy & hard things possible.

CHAPTER 1

# Installation

Install the latest version with pip:

```
$ pip install PyCap
```

To install the bleeding edge from the github repo, use the following:

```
$ pip install -e git+https://github.com/sburns/PyCap.git#egg=PyCap
```

The only requirement is requests which will be installed automatically for you by `pip`.

CHAPTER 2

# Philosophy

The REDCap API is pretty simple. There is no built-in search or pagination, for example. However, it does expose all the functionality required to build advanced data management services on top of the API.

In the same way, PyCap is minimal by design. It doesn't do anything fancy behind the scenes and will not prevent you from shooting yourself in the foot. However, it should be very easy to understand and mentally-map PyCap functionality to the REDCap API.

CHAPTER 3

# License

PyCap is licensed under the MIT license.

CHAPTER 4

# Citing

If you use PyCap in your research, please consider citing the software:

> Burns, S. S., Browne, A., Davis, G. N., Rimrodt, S. L., & Cutting, L. E. PyCap (Version 1.0) [Computer Software]. Nashville, TN: Vanderbilt University and Philadelphia, PA: Childrens Hospital of Philadelphia. Available from https://github.com/sburns/PyCap. doi:10.5281/zenodo.9917

Contents:

## Quickstart

PyCap makes it very simple to interact with the data stored in your REDCap projects:

```
from redcap import Project
api_url = 'https://redcap.example.edu/api/'
api_key = 'SomeSuperSecretAPIKeyThatNobodyElseShouldHave'
project = Project(api_url, api_key)
```

Export all the data:

```
data = project.export_records()
```

Import all the data:

```
to_import = [{'record': 'foo', 'test_score': 'bar'}]
response = project.import_records(to_import)
```

Import a file:

```
fname = 'something_to_upload.txt'
with open(fname, 'r') as fobj:
    project.import_file('1', 'file', fname, fobj)
```

Export a file:

```
content, headers = project.export_file('1', 'file')
with open(headers['name'], 'w') as fobj:
    fobj.write(content)
```

Delete a file:

```
try:
    project.delete_file('1', 'file')
except redcap.RedcapError:
```

```
    # Throws this if file wasn't successfully deleted
    pass
except ValueError:
    # You screwed up and gave it a bad field name, etc
    pass
```

The deep-dive into all the methods can be found in the full *Documentation*.

# Documentation

## Connecting to Projects

The main class of PyCap is `redcap.Project`. It must be instantiated with the API URL of your REDCap site and a API token:

```
from redcap import Project, RedcapError
URL = 'https://redcap.example.com/api/'
API_KEY = 'ExampleKey'
project = Project(URL, API_KEY)
```

> **note** You will have one API key per redcap project to which you have access. To communicate between projects, you would create multiple `redcap.Project` instances.

API Keys are effectively your username and password for a particular project. If you have read/write access to a project (which is most likely), anyone with your redcap URL (public knowledge) and your API key has read/write access.

Since REDCap projects are often used to store Personal Health Information (PHI), it is of the **utmost importance** to:

- **Never share your API key**.
- **Delete the key (through the web interface) after you're done using it**.

### Ignoring SSL Certificates

If you're connecting to a REDCap server whose SSL certificate can't be verified for whatever reason, you can add a `verify_ssl=False` argument in the `Project` constructor and no subsequent API calls to the REDCap server will attempt to verify the certficate.

By default though, the certificate will always be verified. Obviously, use this "feature" at your own risk. You are exposing yourself to man-in-the-middle attacks by using this.

### Using a local CA_BUNDLE

Because PyCap uses requests under the hood, you can pass a path to your own CA_BUNDLE in the `verify_ssl` argument during `Project` instantiation and it will be used.

### Project Attributes

When creating a `Project` object, PyCap goes ahead and makes some useful API calls and creates these attributes:

- `def_field`: What REDCap refers to as the unique key
- `forms`: A tuple of form names within the project

- `field_names`: A tuple of the raw fields
- `field_labels`: A tuple of the field's labels
- `events`: Unique event names (longitudinal projects)
- `arm_nums`: Unique arm numbers (longitudinal projects)
- `arm_names`: Unique arm names (longitudinal projects)
- `redcap_version`: version of REDCap the project is associated with.

For non-longitudinal projects, `events`, `arm_nums`, and `arm_names` are empty tuples.

**note** To disable calls to the API on initialization you can set `lazy=True`. This prevents much of the metadata associated with a project from being requested at *Project* initialization. Accessing these values prior to *Project.configure()* being run will result in the return of Null values. This might be useful if you're implementing PyCap into a service-based architecture. For example, On each request to the service you might want to initialize a different Project with each request to the service and use PyCap's methods to pull records without having to initialize the project's entire metadata on each service request.

### Metadata

Every Project object has an attribute named metadata. This is a list of dicts with the following keys (in no particular order):

- `field_name`: The raw field name
- `field_type`: The field type (text, radio, mult choice, etc.)
- `field_note`: Any notes for this field
- `required_field`: Whether the field is required
- `custom_alignment`: (Web-only) Determines how the field looks visually
- `matrix_group_name`: The matrix name this field belongs to
- `field_label`: The field label
- `section_header`: Under which section in the form page this field belongs
- `text_validation_min`: Minimum value for validation
- `branching_logic`: Any branching logic that REDCap uses to show or hide fields
- `select_choices_or_calculations`: For radio fields, the choices
- `question_number`: For survey fields, the survey number
- `form_name`: Form under which this field exists
- `text_validation_type_or_show_slider_number`: Validation type
- `identifier`: Whether this field has been marked as containing identifying information
- `text_validation_max`: Maximum value for validation

You can export the metadata on your own using the `export_metadata` method on `Project` objects.

## Exporting Data

Exporting data is very easy with PyCap:

```
data = project.export_records()
```

`data` is a list of `dicts` with the raw field names as keys.

We can request slices to reduce the size of the transmitted data, which can be useful for large projects:

```
# Known record identifiers
ids_of_interest = ['1', '2', '3']
subset = project.export_records(records=ids_of_interest)
# Contains all fields, but only three records

# Known fields of interest
fields_of_interest = ['age', 'test_score1', 'test_score2']
subset = project.export_records(fields=fields_of_interest)
# All records, but only three columns

# Only want the first two forms
forms = project.forms[:2]
subset = project.export_records(forms=forms)
# All records, all fields within the first two forms
```

Note, no matter which fields or forms are requested, the `project.def_field` key will always be in the returned dicts.

Finally, you can tweak the how the data is labeled or formatted:

```
# Same data, but keys will the field labels
data = project.export_records(raw_or_label='label')

# You can also get the data in different formats
csv_data = project.export_records(format='csv') # or format='xml'

# quickly make a pandas.DataFrame
data_frame = project.export_records(format='df')
other_df = project.export_records(format='df', df_kwargs={'index_col': project.field_names[1]})

# export checkbox field labels as values (necessary in REDCap >= 6.0 to retrieve checkbox labels)
data = project.export_records(raw_or_label='label', export_checkbox_labels=True)  # note you will st
```

When you request a `DataFrame`, PyCap exports the data as csv and passes it to the `pandas.read_csv` function. The `df_kwargs` dict can be used to guide the conversion from csv to `DataFrame`.

Previously, PyCap enforced a strict intersection between the passed fields and `project.field_names` but that requirement was dropped in PyCap v0.5:

```
non_fields = ['foo', 'bar', 'bat']
response = project.export_records(fields=non_fields)
# response will contain dicts with only the def_field
```

### Dealing with large exports

**note** If your databases are smaller than about 1 million cells (X records x Y columns), you can safely ignore this section.

Exporting large projects will fail on REDCap's backend and PyCap will throw a `redcap.RedcapError`. The threshold for failure seems to be around 1 million cells but I haven't studied this empirically. So for large projects, the export call with default values will fail:

```
>>> project = Project(url, 'TokenToALargeProject')
>>> try:
>>>     data = project.export_records()
>>> except RedcapError:
>>>     print "Failure"
Failure
```

Here's an exporting function that trades speed for robustness:

```
def chunked_export(project, chunk_size=100):
    def chunks(l, n):
        """Yield successive n-sized chunks from list l"""
        for i in xrange(0, len(l), n):
            yield l[i:i+n]
    record_list = project.export_records(fields=[project.def_field])
    records = [r[project.def_field] for r in record_list]
    try:
        response = []
        for record_chunk in chunks(records, chunk_size):
            chunked_response = project.export_records(records=record_chunk)
            response.extend(chunked_response)
    except RedcapError:
        msg = "Chunked export failed for chunk_size={:d}".format(chunk_size)
        raise ValueError(msg)
    else:
        return response
```

The gist of the function:

- Define a sub-function that will yield successive n-sized chunks from a list.
- Export only the record identifiers. If this times out because you have a million records in your project, you effectively can't interact with the project through the API. Sorry.
- **Build a list of just the record identifiers and iterate on the chunks:**
  - Export the data for just this chunk of identifiers.
  - Extend an ongoing list of responses with this list of data.
- If any `export_records` call fails during the loop, a `ValueError` is raised. You should try again with a smaller chunk size. Otherwise, the list of responses is returned.

Caveats:

- You can do this with json responses because each chunked response is a list of dictionaries with no structure between records. This becomes much more difficult if you want csv or xml as there is much more structure in these responses.
- You could also do this with `pandas.DataFrame` but you'll want to `.append` the chunked dataframe, not extend.

I'm hesitant to include this as a method on `Project` because of these issues. I'm also not sure how often this is encountered in the real world. But feel free to use this function if you need it.

Regardless, you should remember that the REDCap instance you're working with is most likely a shared resource and you should always try to limit your API export requests to just the information you need at that point in time.

## Importing Data

PyCap aims to make importing as easy as exporting:

```
# toy
def increment_score(record):
    record['score'] += 5

data = project.export_records(fields=['score'])
map(increment_score, data)
response = project.import_records(data)
# response['count'] is the number of records successfully updated

# import other formats too
response = project.import_records(csv_string, format='csv')

# PyCap will convert a DataFrame to csv and import it automatically
response = project.import_records(df)
```

### Date String Formatting

If the REDCap server you're working with is older than version 5.9 (look at the footer on the main page of your site to find your version), date strings to be imported can be formatted as either `'YYYY-MM-DD'` **or** `'MM/DD/YYYY'`. Beginning with v5.9, the API will **only** accept `'YYYY-MM-DD'` formatting unless you specify the `date_format` parameter in the `import_records` call. Possible values are `'YMD'` (default), `'DMY'` or `'MDY'`:

```
to_import = [{'record': '1', 'date_of_birth': '02/14/2000'}]
response = project.import_records(to_import, date_format='MDY')
```

## Working with Files

You can download files in a REDCap project (exporting) and upload local files (import) to a REDCap project. You can also delete them but there is no undo button for this operation.

> **note** Unlike exporting and importing data, exporting/importing/deleting files can only be done for a single record at a time.

Generally, you will be given bytes from the file export method so binary-formatted data can be written properly and you are expected to pass an open file object for file importing. Of course, you should open a file you wish to import with a well-chosen mode.

The REDCap API doesn't send any return message for file methods. Therefore, it's important to watch out for `redcap.RedcapError` exceptions that may occur when a request fails on the server. If this isn't thrown, you can assume your request worked:

```
try:
    file_content, headers = project.export_file(record='1', field='file')
except RedcapError:
    # file_content will actually contain an error message now that might be useful to look at.
    pass
else:
    # Note, you may want to change the mode in which you're opening files
    # based on the header['name'] value, but that is completely up to you.
    mode = 'wb' if headers['name'].endswith('.pdf') else 'w'
    with open(headers['name'], mode) as f:
        f.write(file_content)
```

```
existing_fname = 'to_upload.pdf'
fobj = open(existing_fname, 'rb')
field = 'data_file'
# In the REDCap UI, the link to download the file will be named the fname you pass as the ``fname`` p
try:
    response = project.import_file(record='1', field=field, fname=existing_fname, fobj=fobj)
except RedcapError:
    # Your import didn't work
    pass
finally:
    fobj.close()

# And deleting...
try:
    project.delete_file('1', field)
except RedcapError:
    # The file wasn't deleted
    pass
else:
    # It's gone
    pass

# Attempting to do any file-related operation on a non-file field will raise a ValueError quickly
try:
    project.import_file(record='1', field='numeric_field', fname, fobj)
except ValueError:
    # Bingo
```

## Exporting Users

You can also export data related to the fellow users of your REDCap project:

```
users = project.export_users()
for user in users:
    assert 'firstname' in user
    assert 'lastname' in user
    assert 'email' in user
    assert 'username' in user
    assert 'expiration' in user
    assert 'data_access_group' in user
    assert 'data_export' in user
    assert 'forms' in user
```

So each dict in the exported users list contains the following key, value pairs:

- `firstname`: First name of the user
- `lastname`: Last name of the user
- `email`: Email address for the user
- `username`: The username of the user
- `expiration`: The user's access expiration date (empty if no expiration)
- `data_access_group`: Data access group of the user

- `data_export`: An integer where 0 means they have no access, 2 means they get a De-Identified data, and 1 means they can export the full data set
- `forms`: A list of dicts, each having one key (the form name) and an integer value, where 0 means they have no access, 1 means they can view records/responses and edit records (survey responses are read-only), 2 means they can only read surveys and forms, and 3 means they can edit survey responses as well as forms

You can also specify the `format` argument to `project.export_users` to be `'csv'` or `'xml'` and get strings in those respective formats, though `json` is default and will return the decoded objects.

### Exporting Form-Event Mappings

Longitudinal projects have a mapping of what forms are available to collect data within each event. These mappings can be exported from the `Project`:

```
fem = project.export_fem()
# Only ask for particular arms
subset = project.export_fem(arms=['arm1'])

# You can also get a DataFrame of the FEM
fem_df = project.export_fem(format='df')
```

### Full API

Full API documentation can be found in the *API* docs.

## API

PyCap is structured into two layers: a high level that exposes `redcap.Project` and a low-level, `redcap.request`. Users (like yourself) should generally only worry about working the `redcap.Project` class.

### JSON Handling

For any request with a `format='json'` argument, the API will respond with a JSON-formatted string representation of the response. This is automatically decoded by PyCap into a list of python dictionaries. This is the default format for all requests.

#### High-Level

The `redcap.Project` class is the high-level object of the module. Generally you'll only need this class.

**class** `redcap.project.`**`Project`**(*url*, *token*, *name=''*, *verify_ssl=True*, *lazy=False*)

Main class for interacting with REDCap projects

**`__init__`**(*url*, *token*, *name=''*, *verify_ssl=True*, *lazy=False*)

**Parameters** **url** : str

API URL to your REDCap server

**token** : str

API token to your project

**name** : str, optional

name for project

**verify_ssl** : boolean, str

Verify SSL, default True. Can pass path to CA_BUNDLE.

**backfill_fields**(*fields*, *forms*)

Properly backfill fields to explicitly request specific keys. The issue is that >6.X servers *only* return requested fields so to improve backwards compatiblity for PyCap clients, add specific fields when required.

**Parameters** **fields: list** :

requested fields

**forms: list** requested forms

**Returns:** :

new fields, forms

**delete_file**(*record*, *field*, *return_format='json'*, *event=None*)

Delete a file from REDCap

**Parameters** **record** : str

record ID

**field** : str

field name

**return_format** : ('json'), 'csv', 'xml'

return format for error message

**event** : str

If longitudinal project, event to delete file from

**Returns** **response** : dict, str

response from REDCap after deleting file

### Notes

There is no undo button to this.

**export_fem**(*arms=None*, *format='json'*, *df_kwargs=None*)

Export the project's form to event mapping

**Parameters** **arms** : list

Limit exported form event mappings to these arm numbers

**format** : ('json'), 'csv', 'xml'

Return the form event mappings in native objects, csv or xml, 'df'' will return a `pandas.DataFrame`

**df_kwargs** : dict

Passed to pandas.read_csv to control construction of returned DataFrame

**Returns** **fem** : list, str, `pandas.DataFrame`

form-event mapping for the project

**export_file**(*record*, *field*, *event=None*, *return_format='json'*)
Export the contents of a file stored for a particular record

**Parameters** **record** : str

record ID

**field** : str

field name containing the file to be exported.

**event: str** :

for longitudinal projects, specify the unique event here

**return_format: ('json'), 'csv', 'xml'** :

format of error message

**Returns** **content** : bytes

content of the file

**content_map** : dict

content-type dictionary

#### Notes

Unlike other export methods, this works on a single record.

**export_metadata**(*fields=None*, *forms=None*, *format='json'*, *df_kwargs=None*)
Export the project's metadata

**Parameters** **fields** : list

Limit exported metadata to these fields

**forms** : list

Limit exported metadata to these forms

**format** : (`'json'`), `'csv'`, `'xml'`, `'df'`

Return the metadata in native objects, csv or xml. `'df'` will return a `pandas.DataFrame`.

**df_kwargs** : dict

Passed to `pandas.read_csv` to control construction of returned DataFrame. by default `{'index_col': 'field_name'}`

**Returns** **metadata** : list, str, `pandas.DataFrame`

metadata sttructure for the project.

**export_records**(*records=None*, *fields=None*, *forms=None*, *events=None*, *raw_or_label='raw'*, *event_name='label'*, *format='json'*, *export_survey_fields=False*, *export_data_access_groups=False*, *df_kwargs=None*, *export_checkbox_labels=False*, *filter_logic=None*)
Export data from the REDCap project.

**Parameters** **records** : list

array of record names specifying specific records to export. by default, all records are exported

**fields** : list

array of field names specifying specific fields to pull by default, all fields are exported

**forms** : list

array of form names to export. If in the web UI, the form name has a space in it, replace the space with an underscore by default, all forms are exported

**events** : list

an array of unique event names from which to export records

**note** this only applies to longitudinal projects

**raw_or_label** : (`'raw'`), `'label'`, `'both'`

export the raw coded values or labels for the options of multiple choice fields, or both

**event_name** : (`'label'`), `'unique'`

export the unique event name or the event label

**format** : (`'json'`), `'csv'`, `'xml'`, `'df'`

Format of returned data. `'json'` returns json-decoded objects while `'csv'` and `'xml'` return other formats. `'df'` will attempt to return a `pandas.DataFrame`.

**export_survey_fields** : (`False`), True

specifies whether or not to export the survey identifier field (e.g., "redcap_survey_identifier") or survey timestamp fields (e.g., form_name+"_timestamp") when surveys are utilized in the project.

**export_data_access_groups** : (`False`), `True`

specifies whether or not to export the `"redcap_data_access_group"` field when data access groups are utilized in the project.

**note** This flag is only viable if the user whose token is being used to make the API request is *not* in a data access group. If the user is in a group, then this flag will revert to its default value.

**df_kwargs** : dict

Passed to `pandas.read_csv` to control construction of returned DataFrame. by default, `{'index_col': self.def_field}`

**export_checkbox_labels** : (`False`), `True`

specify whether to export checkbox values as their label on export.

**filter_logic** : string

specify the filterLogic to be sent to the API.

**Returns** **data** : list, str, `pandas.DataFrame`

exported data

**export_survey_participant_list**(*instrument*, *event=None*, *format='json'*)

Export the Survey Participant List

The passed instrument must be set up as a survey instrument.

**instrument: str** Name of instrument as seen in second column of Data Dictionary.

**event: str** Unique event name, only used in longitudinal projects

**format: (json, xml, csv), json by default** Format of returned data

**export_users**(*format='json'*)

Export the users of the Project

**Parameters** **format** : ('json'), 'csv', 'xml'

response return format

**Returns** **users: list, str** :

list of users dicts when 'format'='json', otherwise a string

#### Notes

Each user will have the following keys:

- 'firstname' : User's first name
- 'lastname' : User's last name
- 'email' : Email address
- 'username' : User's username
- 'expiration' : Project access expiration date
- 'data_access_group' : data access group ID
- 'data_export' : (0=no access, 2=De-Identified, 1=Full Data Set)
- **'forms'** [a list of dicts with a single key as the form name and] value is an integer describing that user's form rights, where: 0=no access, 1=view records/responses and edit records (survey responses are read-only), 2=read only, and 3=edit survey responses,

**filter**(*query*, *output_fields=None*)

Query the database and return subject information for those who match the query logic

**Parameters** **query: Query or QueryGroup** :

Query(Group) object to process

**output_fields: list** :

The fields desired for matching subjects

**Returns** **A list of dictionaries whose keys contains at least the default field** :

**and at most each key passed in with output_fields, each dictionary** :

**representing a surviving row in the database.** :

**filter_metadata**(*key*)

Return a list of values for the metadata key from each field of the project's metadata.

**Parameters** **key: str** :

A known key in the metadata structure

**Returns** **filtered :** :

attribute list from each field

**import_file**(*record*, *field*, *fname*, *fobj*, *event=None*, *return_format='json'*)
Import the contents of a file represented by fobj to a particular records field

**Parameters** **record** : str

record ID

**field** : str

field name where the file will go

**fname** : str

file name visible in REDCap UI

**fobj** : file object

file object as returned by *open*

**event** : str

for longitudinal projects, specify the unique event here

**return_format** : ('json'), 'csv', 'xml'

format of error message

**Returns** **response :** :

response from server as specified by `return_format`

**import_records**(*to_import*, *overwrite='normal'*, *format='json'*, *return_format='json'*, *return_content='count'*, *date_format='YMD'*, *force_auto_number=False*)
Import data into the RedCap Project

**Parameters** **to_import** : array of dicts, csv/xml string, `pandas.DataFrame`

**note** If you pass a csv or xml string, you should use the `format` parameter appropriately.

**note** Keys of the dictionaries should be subset of project's, fields, but this isn't a requirement. If you provide keys that aren't defined fields, the returned response will contain an `'error'` key.

**overwrite** : ('normal'), 'overwrite'

`'overwrite'` will erase values previously stored in the database if not specified in the to_import dictionaries.

**format** : ('json'), 'xml', 'csv'

Format of incoming data. By default, to_import will be json-encoded

**return_format** : ('json'), 'csv', 'xml'

Response format. By default, response will be json-decoded.

**return_content** : ('count'), 'ids', 'nothing'

By default, the response contains a 'count' key with the number of records just imported. By specifying 'ids', a list of ids imported will be returned. 'nothing' will only return the HTTP status code and no message.

**date_format** : ('YMD'), 'DMY', 'MDY'

Describes the formatting of dates. By default, date strings are formatted as 'YYYY-MM-DD' corresponding to 'YMD'. If date strings are formatted as 'MM/DD/YYYY' set this parameter as 'MDY' and if formatted as 'DD/MM/YYYY' set as 'DMY'. No other formattings are allowed.

**force_auto_number** : ('False') Enables automatic assignment of record IDs

of imported records by REDCap. If this is set to true, and auto-numbering for records is enabled for the project, auto-numbering of imported records will be enabled.

**Returns** **response** : dict, str

response from REDCap API, json-decoded if `return_format == 'json'`

**is_longitudinal**()

**Returns** **boolean :** :

longitudinal status of this project

**metadata_type**(*field_name*)
If the given field_name is validated by REDCap, return it's type

**names_labels**(*do_print=False*)
Simple helper function to get all field names and labels

**exception** redcap.**RedcapError**
This is thrown when an API method fails. Depending on the API call, the REDCap server will return a helpful message. Sometimes it won't :(

### Low-Level

The `Project` class makes all HTTP calls to the REDCap API through the `redcap.request.RCRequest` class. You shouldn't need this in day-to-day usage.

**exception** redcap.request.**RCAPIError**
Errors corresponding to a misuse of the REDCap API

**class** redcap.request.**RCRequest**(*url*, *payload*, *qtype*)
Private class wrapping the REDCap API. Decodes response from redcap and returns it.

#### References

https://redcap.vanderbilt.edu/api/help/

Users shouldn't really need to use this, the Project class is the biggest consumer.

**__init__**(*url*, *payload*, *qtype*)
Constructor

**Parameters** **url** : str

REDCap API URL

**payload** : dict

key,values corresponding to the REDCap API

**qtype** : str

Used to validate payload contents against API

**execute**(***kwargs*)
Execute the API request and return data

**Parameters** **kwargs :** :

passed to requests.post()

**Returns** **response** : list, str

data object from JSON decoding process if format==’json’, else return raw string (ie format==’csv’|’xml’)

**expect_empty_json**()
Some responses are known to send empty responses

**get_content**(*r*)
Abstraction for grabbing content from a returned response

**raise_for_status**(*r*)
Given a response, raise for bad status for certain actions

Some redcap api methods don’t return error messages that the user could test for or otherwise use. Therefore, we need to do the testing ourself

Raising for everything wouldn’t let the user see the (hopefully helpful) error message

**validate**()
Checks that at least required params exist

# API Change Log

All changes to the REDCap API to be considered for addition into PyCap are listed below by REDCap version number (oldest first).

These changes contain new methods and parameters introduced in REDCap 6.X.X which we hope to add support for in PyCap in the future.

Note: The API Change Log assumes PyCap 1.0 is up-to-date as of REDCap 6.0.0, however all changes made from REDCap 6.X are included for sake of completeness.

## New API methods, new parameters, and significant changes

**Note:** while some “significant changes” listed may not directly influence PyCap methods and parameters, implementation procedures and/or output may be effected such that they were listed as a precautionary measure.

### (6.0.0)

- **Data Export rights - Significant changes** - New user rights privilege for Data Export rights: “Remove all tagged Identifier fields” - If a user is given this data export privilege setting, then it will be applied to *all* data export files in all modules where data is exported (e.g., PDFs, reports, API). In reports and in API data exports, any fields that have been tagged as Identifier fields will simply be removed from the export file. However, in PDF exports, it will include the Identifier fields in the PDF, but it will remove and replace any saved data values with the text *DATA REMOVED* for such fields.
- **Export REDCap Version - New API method** - Returns the current REDCap version number as plain text (e.g., 4.13.18, 5.12.2, 6.0.0). Set the parameter content=”version” in the API request, and it will return the REDCap version number.

- **Export Reports - New API method** - Any report created by a user can be exported via the API in JSON, XML, or CSV format. Needed for the API request is the report ID number, which is listed next to the report name on the "Data Exports, Reports, and Stats" page (only displayed there if the user has API Export privileges). In order to export a report via the API, the user must have been given access to that report, must not have "No Access" data export rights, and must have API Export rights. More details regarding the API Export Reports method can be found on the API Documentation page.
- **Export Records and Export Reports - New parameters** - "rawOrLabelHeaders" parameter for "Export Records" and "Export Reports" API methods. Possible values: "raw" (default) or "label". Parameter is valid for "csv" format only (and only works for "flat" type output). Allows one to have the API return the CSV headers in the data export either as the variable/field names (rawOrLabelHeaders=raw, or if rawOrLabelHeaders is not specified) or instead as the field labels (rawOrLabelHeaders=label).
- **Export Records and Export Reports - New parameters** - "exportCheckboxLabels" parameter for "Export Records" and "Export Reports" API methods. Possible values: "true" or "false" (default). Parameter is only valid for "flat" type output. Allows one to set the format of checkbox field values when specifically exporting the data as labels (i.e., when rawOrLabel=true). When exporting labels, by default (without providing the exportCheckboxLabel flag or if exportCheckboxLabel=false), all checkboxes will either have a value "Checked" if they are checked or "Unchecked" if not checked. But if exportCheckboxLabel is set to true, it will instead export the checkbox value as the checkbox option's label (e.g., "Choice 1") if checked or it will be blank/empty (no value) if not checked.
- **Export Records - Significant change** - For the "Export Records" API method when using "flat" type output while exporting the data as labels (i.e., when rawOrLabel=label), it now no longer defaults to exporting checkbox field values as their choice label (i.e., "Choice 1") as it did in previous versions, but instead it returns "Checked" or "Unchecked" as the checkbox values by default. So if any users depend upon checkbox values being exported as their choice label, they should add the new parameter "exportCheckboxLabels" to their script as exportCheckboxLabels=true to maintain continuity after the upgrade to REDCap 6.0 or higher.
- **Data Export user rights - Significant change** - Data Export user rights are now applied to API data exports. In previous versions, the data export rights had no effect on API data exports. Now if a user has No Access export rights, they will not be able to export data from the API (this includes the "Export Records" API method, the new "Export Reports" API method, and also the "Export a File" API method). And if the user has either "De-Identified" export rights or "Remove all tagged Identifier fields" export rights, then fields will be removed from the export data set accordingly in the API export request. If a user attempts to use the "Export a File" API method on a File Upload field that has been tagged as an Identifier when that user has either "De-Identified" export rights or "Remove all tagged Identifier fields" export rights, then it will return an error.
- **Export Records - Significant change** - For the "Export Records" API method, "both" will no longer be an option for the "rawOrLabel" parameter. The only valid options for rawOrLabel will be "raw" and "label". This is also true for the new "Export Reports" API method.
- **Data Exports - Significant change** - "eventName" will no longer be used as a parameter in API data exports. In previous versions, users could manually set whether an API data export would return the event name for longitudinal projects as the event label text or instead as the unique event name (generated automatically from the event label). It now simply returns the event label text if rawOrLabel=label, whereas it will return the unique event name if rawOrLabel=raw (or if rawOrLabel is not provided). If "eventName" is provided in the API request, it will be ignored.
- **Data Exports - Significant change** - When doing an API data export where a list of fields or forms are explicitly provided in the API request to REDCap, the record ID field will now only be included in the exported result if the record ID field is explicitly specified in the list of fields or if its form is explicitly specified in the list of forms. In previous versions, if *any* form or field was explicitly provided in the API request, it would return the record ID field. So for users who depend upon the Record ID field to be returned in the exported data set, they should go ahead and add the Record ID to the "fields" array parameter in their API request to maintain continuity after the upgrade to REDCap 6.0 or higher.

#### (6.2.0)

- **Signature fields - Significant change** - Allows a person to draw their signature on a survey or data entry form using a mouse, pen, or finger (depending on whether using a desktop computer or mobile device). Once captured, the signature will be displayed as an inline image on the survey page or data entry form. While this option appears as a "Signature" field type in the Online Designer, it is specified in the Data Dictionary as a "file" type field with validation type of "signature". Thus, it is essentially a special type of File Upload field. Note: The signature image for Signature fields cannot be imported via the API, although they can be downloaded or deleted via the API using the "Export a File" and "Delete a File" API methods, respectively.

#### (6.3.0)

- **Server-side auto-calculations - Significant change** - When performing a data import (via Data Import Tool or API), REDCap will now perform the calculations for any calculated fields that are triggered by the values being imported. For example, if you have a BMI field whose calculation is based off of a height field and a weight field, then if you perform a data import of height and weight values, it will automatically calculate the BMI for each record that is imported and also save those calculations (and thus log them too on the Logging page). Auto-calculations are now also triggered when using cross-form calculations in the case where the calculated field exists on a different instrument than the fields being entered that are used in the calculation. So while in previous versions users would have to go to the instrument where the calculated field existed and would have to click Save to store the calculation, users now no longer have to do that because the calculation is performed and saved automatically at the time when the trigger fields are initially entered or changed. So essentially, users never have to worry that calculations are not being performed or saved in certain situations. They should expect that calculations are now always being saved silently in the background.

#### (6.4.0)

- **Export PDF file of Data Collection Instruments - New API method** - Returns a PDF file of one or all instruments in the project, either with no data (blank), with a single record's data, or with all records from the project.
- **Export a Survey Link for a Participant - New API method** - Returns a unique survey link (i.e., a URL) in plain text format for a specified record and data collection instrument (and event, if longitudinal) in a project.
- **Export a Survey Queue Link for a Participant - New API method** - Returns a unique Survey Queue link (i.e., a URL) in plain text format for the specified record in a project that is utilizing the Survey Queue feature.
- **Export a Survey Return Code for a Participant - New API method** - Returns a unique Return Code in plain text format for a specified record and data collection instrument (and event, if longitudinal) in a project with surveys that are utilizing the "Save & Return Later" feature.
- **Export a Survey Participant List - New API method** - Returns the list of all participants for a specific survey instrument (and for a specific event, if a longitudinal project).
- **Export List of Export Field Names - New API method** - Returns a list of the export/import-specific version of field names for all fields (or for one field, if desired) in a project. This is mostly used for checkbox fields because during data exports and data imports, checkbox fields have a different variable name used than the exact one defined for them in the Online Designer and Data Dictionary, in which *each checkbox option* gets represented as its own export field name in the following format: field_name + triple underscore + converted coded value for the choice.
- **All data import methods - Significant change** - Negative values can now be used as the raw coded values for checkbox fields with regard to their usage in data exports and data imports. In previous versions, negative values for checkbox choices would save successfully on surveys and data entry forms, but due to certain limitations, they would not work when importing values for those choices using the Data Import Tool or using the API

data import. In the same regard, they would also cause problems when exporting data into a statistical analysis package. Now negative signs can be used for checkbox options, in which the negative sign will be replaced by an underscore in the export/import-specific version of the variable name (e.g., for a checkbox named "meds", its choices "2" and "-2" would export as the fields "meds_2" and "meds2", respectively).

#### (6.4.3)

- **Export PDF - Significant change** - The "Export PDF" API method's optional parameter "allrecords" has been changed to "!allRecords" to be more consistent with API parameter naming conventions. Note: To be backward compatible, the older version "allrecords" will still work the same as before if it is used.

#### (6.5.0)

- **Export Project Information - New API method** - Exports some of the basic attributes of a given REDCap project, such as the project's title, if it is longitudinal, if surveys are enabled, the time the project was created and moved to production, etc. See the official API documention/help page in 6.5.0 for all the details.
- **Export Users - Significant change** - "Export Users" will now return two new attributes for each user: "mobile_app" and "mobile_app_download_data". If mobile_app's value is "1", then the user has privileges to use the REDCap Mobile App for that project. If "0", then not. If mobile_app_download_data's value is "1", then the user has the ability to download all records from the project to the mobile app, but if "0", then the user will not have the option in the app to download any records to the app.

#### (6.7.0)

- **Export Project Information - Significant Change** - Improvement: New project-level attributes are now included in the "Export Project Information" API method. The following attributes were added: "project_irb_number", "project_grant_number", "project_pi_firstname", and "project_pi_lastname".

#### (6.8.1)

- **Server-side auto-calculations - Significant change** - Administrators may disable the auto-calculation functionality for a given project on the "Edit a Project's Settings" page in the Control Center. If left as enabled (default), server-side auto-calculations (introduced in REDCap 6.3.0) will be performed for calc fields when data is imported (via Data Import Tool or API) or when saving a form/survey containing cross-form or cross-event calculations. If auto-calculations are disabled, then calculations will only be done after being performed via JavaScript (client-side) on the data entry form or survey page on which they are located, and they will not be done on data imports. Tip: This setting should *only* be disabled if the auto-calculations are causing excessive slowdown when saving data. If disabled, then some calculations might not get performed, and if so, must then be fixed with Data Quality rule H.

#### (6.9.5)

- **All API tokens - Signficant change** - The API is now more strict with regard to the validation of API tokens sent in API requests. In previous versions, if the token was longer than 32 characters, it would truncate the token to 32 characters (which is the expected length). It no longer truncates the token if longer than expected but merely returns an error message.

### (6.10.0)

- **All data import methods - Significant change** - When importing data in CSV format via API or Data Import Tool, all blank rows will now be ignored instead of returning an error. This is to avoid the common mistake by users of leaving some lines as blank in the CSV file since most users assume the blank line would be ignored anyway.

### (6.11.0)

- **Arm import/delete - New API method** - for longitudinal projects only; requires API Import privileges and Project privileges
- **Event import/delete - New API method** - for longitudinal projects only; requires API Import privileges and Project privileges
- **Import instrument-event mappings - New API method** - for longitudinal projects only; requires API Import privileges and Project privileges
- **Import metadata, i.e. data dictionary - New API method** - available only in development status; requires API Import privileges and Project privileges
- **Import users - New API method** - (import new users into a project while setting their user privileges, or update the privileges of existing users in the project.) - requires API Import privileges and User Rights privileges
- **Create project - New API method**
  - Allows a user to create a new REDCap project while setting some project attributes, such as project title, project purpose, enable/disable record auto-numbering, enable the project as longitudinal, and enable surveys in the project.
  - This method requires a Super API Token that must be granted to a user by a REDCap administrator on the API Tokens page in the Control Center.
  - After the super token has been granted, the user can view the super token on their My Profile page.
- **Export Records - New parameter** - A new optional API parameter named "filterLogic" was API method "Export Records". filterLogic should be a string of logic text (e.g., age 30) for filtering the data to be returned by this API method, in which the API will only return the records (or record-events, if a longitudinal project) where the logic evaluates as TRUE. This parameter is blank/null by default unless a value is supplied. Please note that if the filter logic contains any incorrect syntax, the API will respond with an error message.
- **Export Users - Significant change** - Change: For the API method "Export Users", many more user privilege rights are included in the response. The following is the full header list:
  - username,email,firstname,lastname,expiration,data_access_group,data_access_group_id,design, user_rights,data_access_groups,data_export,reports,stats_and_charts,manage_survey_participants, calendar,data_import_tool,data_comparison_tool,logging,file_repository,data_quality_create, data_quality_execute,api_export,api_import,mobile_app,mobile_app_download_data,record_create, record_rename,record_delete,lock_records_all_forms,lock_records,lock_records_customization,forms
- **Export Users - Significant change** - Change: For the API method "Export Users", when requesting a response in CSV format, form-level rights are returned in a different format in order to prevent possible duplication of other new user privileges that are returned, in which all form rights will now be consolidated into a single column named "forms" (whereas in previous versions each form was represented as an individual column). The last column of the CSV string returned will have "forms" as the header, and the value will be each unique form name and its numerical value as a colon-separated pair with all the form value pairs strung together as a single comma-separated string (e.g. "demographics:1,visit_data:3,baseline:1"). See a full CSV example below of two users exported from a project.

  - username,email,firstname,lastname,expiration,data_access_group,data_access_group_id,design, user_rights,data_access_groups,data_export,reports,stats_and_charts,manage_survey_participants, calendar,data_import_tool,data_comparison_tool,logging,file_repository,data_quality_create, data_quality_execute,api_export,api_import,mobile_app,mobile_app_download_data,record_create, record_rename,record_delete,lock_records_all_forms,lock_records,lock_records_customization,forms harrispa,baseline_data:1,visit_lab_data:1,patient_morale_questionnaire:1,visit_blood_workup:1, completion_data:1,completion_project_questionnaire:1,visit_observed_behavior:,baseline_data:1, visit_lab_data:1,patient_morale_questionnaire:1,visit_blood_workup:1,completion_data:1, completion_project_questionnaire:1,visit_observed_behavior:

- **Export Users - Significant change** - Change: For the API method "Export Users", when requesting a response in XML format, the main parent tags at the beginning and end of the response will no longer be <records> but instead will be <users> to be less confusing (since "records" often denotes something else in REDCap) and also to be more consistent with how other API methods return XML items.
- **Export Users - Significant change** - Change: For the API method "Export Users", the new "data_access_group_id" field was added, in which it returns the numerical group ID number that the "data_access_group" field used to return in previous versions. And now, the unique group name of a user's Data Access Group is returned for the "data_access_group" field rather than the numerical group ID number.
- **Export Instrument-Event Mappings - Significant change** - Change: The API method "Export Instrument-Event Mappings" now returns a different structure if exporting as JSON or XML (however, the CSV format will remain the same). It will now export with "arm_num", "unique_event_name", and "form" as attributes of each item/mapping, as seen in the JSON/XML examples below.
- **Export Project Information - Signficant change** - For "Export Project Information" API method, the following two project attributes were added:
  - secondary_unique_field - The variable name of the secondary unique field defined in the project (if applicable).
  - display_today_now_button - Value will be "0" or "1" (i.e. False or True). If "0", then do NOT display the today/now button next to date/datetime fields on data entry forms and surveys. If "1" (default), display them.

### (6.12.0)

- **Export Project XML - New API method** - Returns the contents of an entire project (records, events, arms, instruments, fields, and project attributes - even uploaded files and Descriptive field attachments) as a single XML file, which is in CDISC ODM format.
- **Import Records - New parameter** - for data format now accepts value of "odm" to import data in CDISC ODM format. This only returns data (not the project structure/metadata).
- **Create Project - New parameter** - named "odm" can be used to pass the ODM XML string of an entire project's structure (the same as output by the Export Project XML method) when creating a new project using a Super API Token. This will allow you not only to create the project with the API request, but also to import all fields, forms, and project attributes (and events and arms, if longitudinal) as well as record data all at the same time.
- **Export Records - New Parameter** - Parameter for data format now accepts value of "odm" to export data in CDISC ODM format. This only returns data (not the project structure/metadata).

### (6.12.1)

- **Export Project XML - New parameter** - exportFiles (boolean) parameter was added to the API method. The parameter, which defaults to FALSE, specifies whether or not the resulting XML will include all files (base64 encoded) that were uploaded for File Upload and Signature fields for all records in the project. Please note that

while the previous version (6.12.0) exported all files in the resulting XML by default, it no longer does that and must now be specified explicitly.

## Issues & Contributing

If you have an issue with PyCap or the REDCap API, please raise an issue on PyCap's issues page. I'll do my best to help where I can.

PyCap follows the Fork-Pull workflow method for accepting contributions. If you'd like to contribute code to PyCap, please use the following workflow:

1. If you don't already have an account on GitHub, please make one.
2. Fork my repo to your own account.
3. Checkout a branch & commit your changes. Tests are definitely appreciated!
4. Push those changes to your repo & submit a Pull-Request to my repository.

If any of these steps are unclear, please peruse the helpful GitHub Guide on Forking or reach out to me on Twitter.

## r

## r

## Symbols



## B



## D



## E



## F



## G



## I



## M



## N



## P



## R



## V