RED-I Documentation

Release 0.14.2

CTS-IT

December 03, 2015

Contents

1 User Documentation **3**
- 1.1 RED-I Project . . . 3
- 1.2 RED-I Configuration . . . 6
- 1.3 RED-I Usage . . . 7
- 1.4 Add new REDCap Project and API Key . . . 9
- 1.5 Steps . . . 9
- 1.6 Clinical Data Specification . . . 13
- 1.7 Describing a REDCap Form to RED-I . . . 14
- 1.8 Mapping Clinical Components to LOINC Codes . . . 16

2 Developer Documentation **19**
- 2.1 Testing RED-I with a sample REDCap Project . . . 19
- 2.2 Code Review Checklist . . . 20
- 2.3 Setting up Travis CI for Open Source REDI project . . . 21
- 2.4 Regression Testing . . . 27
- 2.5 redi . . . 29

Python Module Index **43**

The project documentation is split into two sections

- *User Documentation*
- *Developer Documentation*

CHAPTER 1

User Documentation

1.1 RED-I Project

1.1.1 Introduction

The REDCap Electronic Data Importer (RED-I) is a tool which is used to automate the process of loading clinical data from Electronic Medical Records (EMR) systems into REDCap Study data capture systems. RED-I is a general purpose tool for REDCap data importing suitable for use on any study in any REDCap system. It uses XML lookups to translate data stored in comma separated values (CSV) files and uploads it to a REDCAP Server using the REDCap API. The tool allows study data to be securely uploaded from clinical reporting systems, error checked, and uploaded into REDCap. It provides the investigator with feedback on upload success in the form of summary reporting of the data upload process.

You can view a presentation of the RED-I tool in action on youtube.

1.1.2 How to Install RED-I

RED-I is written in Python so you will have to install it if the following comand gives you an error:

For more details on how to install python on your system please visit Downloading Python page.

1.1.3 Installation Using Source Code

We recommend to install RED-I in a Python virtual environment in order to prevent conflicts with your other packages.

The follow steps assume that you have the git version control installed on your system.

Once you are done with testing RED-I and you are satisfied with the results you can remove the virtualenv artifacts and install the RED-I package to be available system-wide.

Please refer to `redi_installation` document for more help with the installation.

1.1.4 Installation Using Binary Distribution

To uninstall the application:

See also:

http://pip.readthedocs.org/en/latest/reference/pip.html

1.1.5 Installing RED-I on Windows

- Open a command prompt by clicking on the Start menu, and typing "cmd" in the Run box.
- Install 64-bit Python 2.7.9 by running the following command in the command prompt:
- Next you need to be insure the command interpreter will be able to find the Python modules. Set

the paths to the modules by running the following commands in the command prompt:

- Make a new directory for the RED-I files by running the following command in the command prompt:
- Download the RED-I source code from: [https://github.com/ctsit/redi/archive/0.14.1.zip]
- Copy the contents of the RED-I zip file from c:Users%username%Downloadsredi-0.14.1redi-0.14.1 to c:redi
- Download the easy_install setup file from: https://bootstrap.pypa.io/ez_setup.py
- Run the easy_install setup file with the following command in the command prompt:

Note: you may need to modify the path to the ez_setup.py file if it is downloaded to a different location.

- Next, make a binary install of RED-I by running the following commands in the command prompt:
- You will need to manually install the pycrypto dependency. To avoid having to compile it with VCForPython you can

download a pre-compiled binary and install it with the following command:

- Finally, install your binary of RED-I with the following command:

1.1.6 Installing RED-I on Red Hat and Fedora

Download and install setuptools. Setuptools will aid you in installing the redi package.

Note that you must have gcc (the Gnu Compiler Collection) to build RED-I. Check that you have gcc installed:

If gcc is not installed, install it:

Install the development libxslt, libxml2, and python-devel libraries. These allow you to build the redi source.

Install redi using pip.

RED-I is now be installed.

If you get an error message while compiling pycrypto, you will need to install pycrypto separately:

- To use the example config, documentation, and other associated RED-I files, you will need to get files from the GitHub repository. You have two options:

1. Clone the repo by using Git.

Set up your install of Git to use the key on your GitHub account. Instructions are at: https://help.github.com/articles/generating-ssh-keys/

Now, clone the redi git repo:

You now have a directory called redi with the source, docs, example configuration and other RED-I files.

2. Download the zip file

- You now have a directory called redi-master with the source, docs, example configuration and other RED-I files.

1.1.7 How to Test RED-I with a Sample Project

Now that you installed the RED-I application you are probably wondering how to configure it to help you with data translation and import tasks. The good news is that you do not have to change any configuration file to test RED-I – we provide examples of working files for you:

- **`Vagrantfile`** –> allows to run a local REDCap instance
- **`settings.ini`** –> pre-configures RED-I to send data to the local REDCap instance
- **`Makefile.ini`** –> configures the make tasks from `Makefile` to simplify testing
- **`redi_sample_project_v5.7.4.sql`** –> provides the data for the sample project running in the local REDCap instance

These files make it very easy to see how RED-I imports data from a csv file into a local instance of REDCap. You just have to follow the instructions from the Testing RED-I with a sample REDCap Project document.

Note: You will need to obtain your own copy of the REDCap since the license terms prevent us from including the code in an open source project.

1.1.8 How to Configure RED-I for a New Project

To use RED-I in production you will have to edit the 'settings.ini' file with values matching your environment.

Please refer to the RED-I Configuration for more details about the meaning of each parameter in 'settings.ini' file.

Please refer to the Add new REDCap Project and API Key document for more details about new project setup.

One of the advantages of using RED-I is that it allows to be customized in order to send data to multiple types forms in REDCap projects. Please refer to Describing a REDCap Form to RED-I document for more details on how to create two of the required configuration files.

1.1.9 How to use RED-I

Please refer to the RED-I Usage for more details about all arguments supported in the command line.

1.1.10 How to Get Support

If you need any help with using RED-I please email us at ctsit@ctsi.ufl.edu

1.1.11 How to Contribute

- Fork the source-code
- Create a branch (**`git checkout -b my_branch`**)
- Commit your changes (**`git commit -am "Details about feature/bug fixes in the commit"`**)
- Push to the branch (**`git push origin my_branch`**)
- Open a pull request and we will accept it as long as it conforms to our

Code Review Checklist

1.2 RED-I Configuration

1.2.1 Required Parameters

The following parameters are required to have a value in **settings.ini**:

- raw_xml_file
- translation_table_file
- form_events_file
- research_id_to_redcap_id
- component_to_loinc_code_xml

If any of the parameters listed above is missing then the program will terminate. A detailed message about the missing parameter will be displayed to the user before the program terminates as well as written to the log file.

1.2.2 Conditional Parameters

While the parameters mentioned in the *section above* are always required, the following parameters are only required to have a value in **settings.ini** when **redi** is not performing a dry run.

- redcap_uri
- token
- redcap_support_receiver_email
- redcap_support_sender_email
- smtp_host_for_outbound_mail
- smtp_port_for_outbound_mail
- sender_email *(only required when **send_email* is set to **Y**)*
- receiver_email *(only required when **send_email* is set to **Y**)*

Note: In "dry run" mode the RED-I will not send any data to REDCap.

The following parameters are required only when `--emrdata` is specified on the command line:

- emr_sftp_server_hostname
- emr_sftp_server_username
- emr_sftp_server_password
- emr_sftp_project_name
- emr_data_file

These parameters are essential for establishing a connection with the SFTP server to obtain EMR data; so, if they are missing or do not have a value in settings.ini, then the program will terminate. As with the required parameters, a message about this will be displayed to the user before the program terminates as well as written to the log file.

1.2.3 Optional Parameters

Following parameters in settings.ini are optional:

Parameter Name	Default Value
report_file_path	report.xml
report_file_path2	report.html
input_date_format	%Y-%m-%d %H:%M:%S
output_date_format	%Y-%m-%d
project	DEFAULT_PROJECT
rate_limiter	600
batch_warning_days	13

If the above parameters are missing or do not have a value in **settings.ini** then the corresponding default value is used. Whenever a default value is used, a message about is written to the log file.

1.3 RED-I Usage

Currently the RED-I application can be only executed from the command line:

```
$ redi
```

1.3.1 Optional command-line arguments:

- -h, –help: show the help message
- -v, –verbose: increase verbosity of output

```
$ redi -v
```

- -V, –version: Show version number

```
$ redi -V
```

- -c: Specify the path to the configuration folder.

```
$ redi -c /Users/admin/redi-config
```

  Default path for config folder is **/config**. For "data directory" refer –datadir.

- -k, –keep: Prevents deletion of the output files generated during data processing.

```
$ redi -k
```

  When this parameter is provided, the output files are stored in **/out/out_<timestamp>**.

  The timestamp has the format: **YYYY_MM_DD-HH_MM_SS**.

  If the parameter is not provided, the output files are stored in a temporary folder during the execution of **redi** and then deleted along with the temporary folder once **redi** finishes execution.

- -d, –dryrun : Performs a dry run.

```
$ redi --dryrun
```

  When this parameter is provided, all data transformations are performed and execution stops after writing out the final data set to **/out/out_<timestamp>**.

The purpose of this switch is to assist developers in performing a dry run of `redi`. Data will not be written to the REDCap server nor will emails be sent.

If `-d` is used, `--keep` is implied; therefore, you do not need to specify it or provide any path for storing the output files.

By default, this parameter is disabled.

- -e, –emrdata: Runs the script for fetching EMR data.

```
$ redi --emrdata
```

When this parameter us provided, a connection will be established with the sftp server mentioned in the settings.ini file in the config folder and EMR data required for the execution of **redi** will be downloaded.

Following parameters need to be set in config/settings.ini before using this option:

  - emr_sftp_server_hostname = URL of the SFTP Server
  - emr_sftp_server_username
  - emr_sftp_server_password
  - emr_sftp_project_name = folder on the SFTP server containing the EMR data
  - emr_data_file = file containing the EMR data

By default, this parameter is disabled.

- -r, –resume: Resumes a previously stopped run of `redi`.

WARNING!!! This is used in a very specific case. Use with caution.

Once **redi** has completed processing, it sends its data to configured REDCap Server. Each transaction is initially marked as *unsent* and only after a response from the REDCap Server is it changed to *sent*. If you stop **redi** from running during this time, it is possible to resume where it left off by specifying the `--resume` switch.

Do not use `--resume` for a first run; it will fail. Using `--resume` once a run has completed is unsupported, but won't do much other than send the email the report again.

The development team is looking to make this a more robust and safer feature in the future.

- –datadir: Specify path to the data directory

  $ redi –datadir /Users/admin/redi_output

  The data directory is the directory that will store the following:

  - log file. Currently up to 31 days of logs are stored, after which the file begins rotating.
  - SQLite database used for storing checksums
  - intermediate output files which are required for debugging and used by the resume logic
  - configuration directory (unless a different path for this is specified by the user)

  By default, the data directory is assumed to be the current working directory. Using this switch, one can run multiple instances of redi simultaneously.

- –skip-blanks: skip blank events when sending event data to RedCAP

  $ redi –skip-blanks

1.4 Add new REDCap Project and API Key

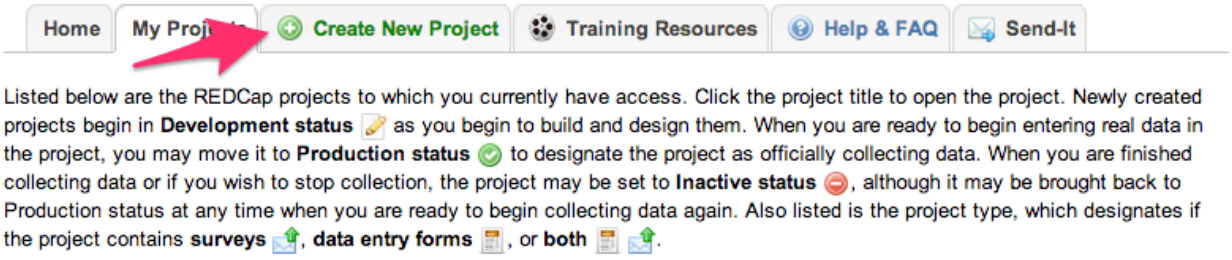
1.5 Steps

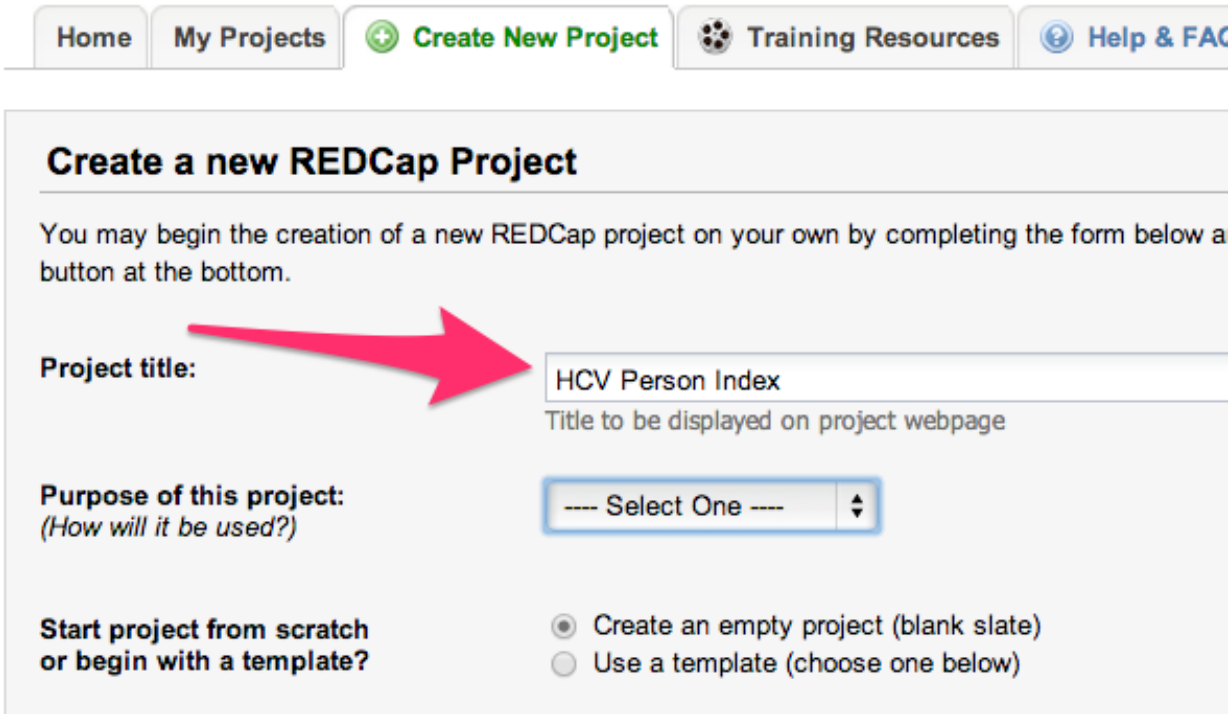
Once you have the RED-I configuration ready you can create a new REDCap project

Note: The instructions below assume you have the permissions to create a new REDCap project, create a user, set user permissions, and create an API Token. Depending on how your REDCap server is configured you may or may not have the permissions to do these tasks yourself. Please consult with your local REDCap managers for assistance if you do not see these options.

1.5.1 1. Create new Project

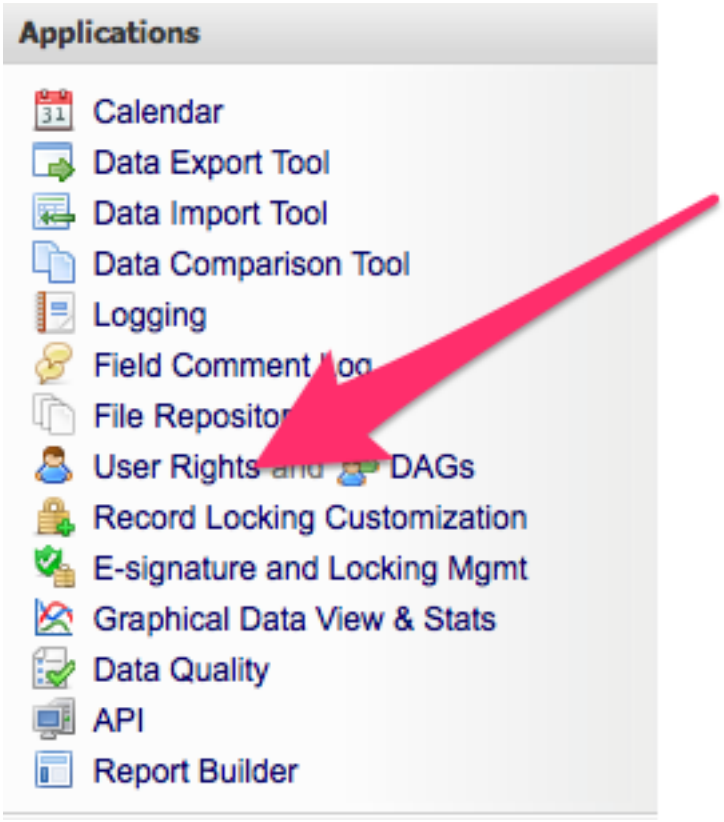
Open in the browser the REDCap url. Select the `Create New Project` tab. Enter `<your_project_name>` for the project title. Please check below images for reference.





1.5.2 2. Authorize People

To adjust user rights, access the User Rights tool via the menu on the left side of the REDCap screen.



or click on `User Rights` button in the `Project Setup`

In REDCap User Rights, set `Data Entry Rights` as per your needs.Please check below image

1.5.3 3. Create an API Token

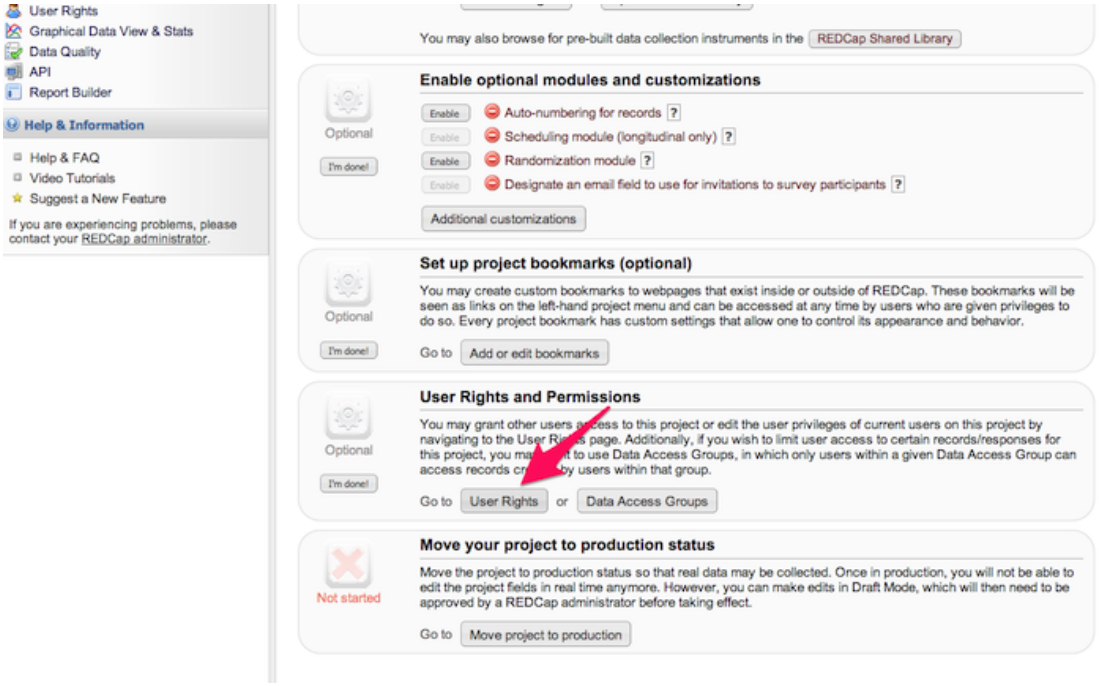
For the data in your project to be used by programs, those programs will need access through API interface. You will need to create an API Token to allow those programs to authenticate and get the correct permissions on your project.

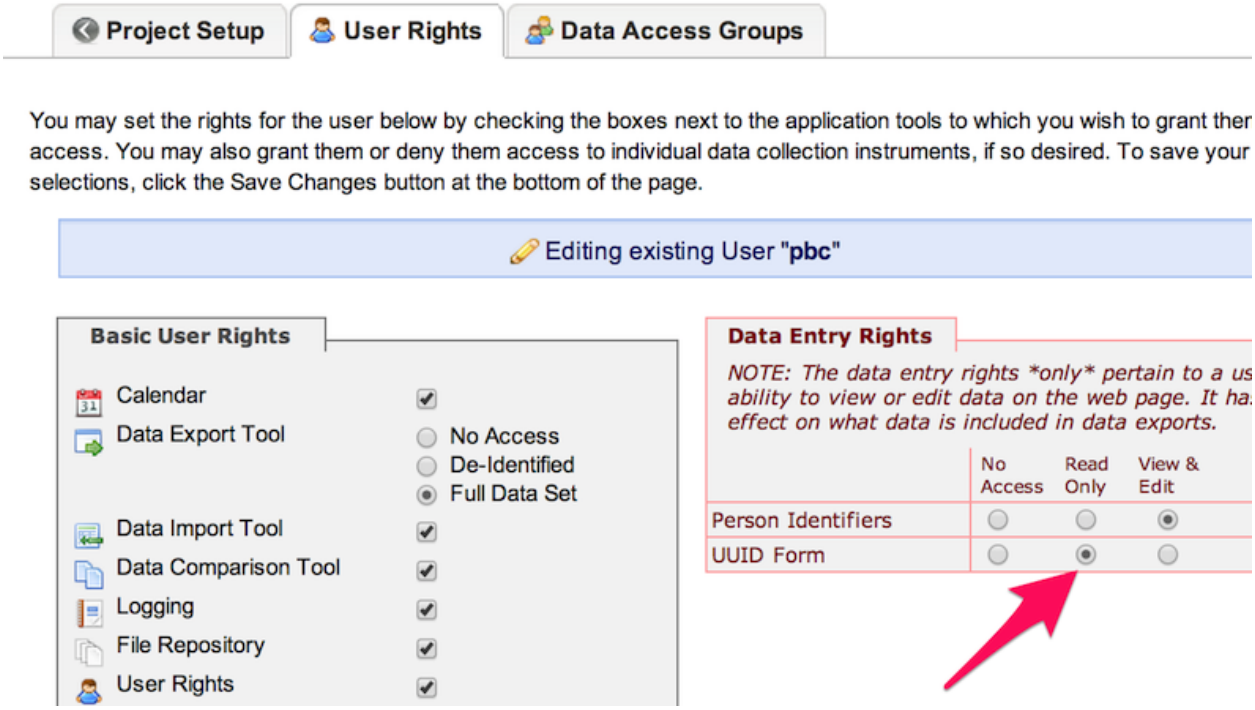
This token can be created on any account, but for automated processes a service account will provide a more reliable authentication. Add a user in this REDCap project with the permissions shown below:

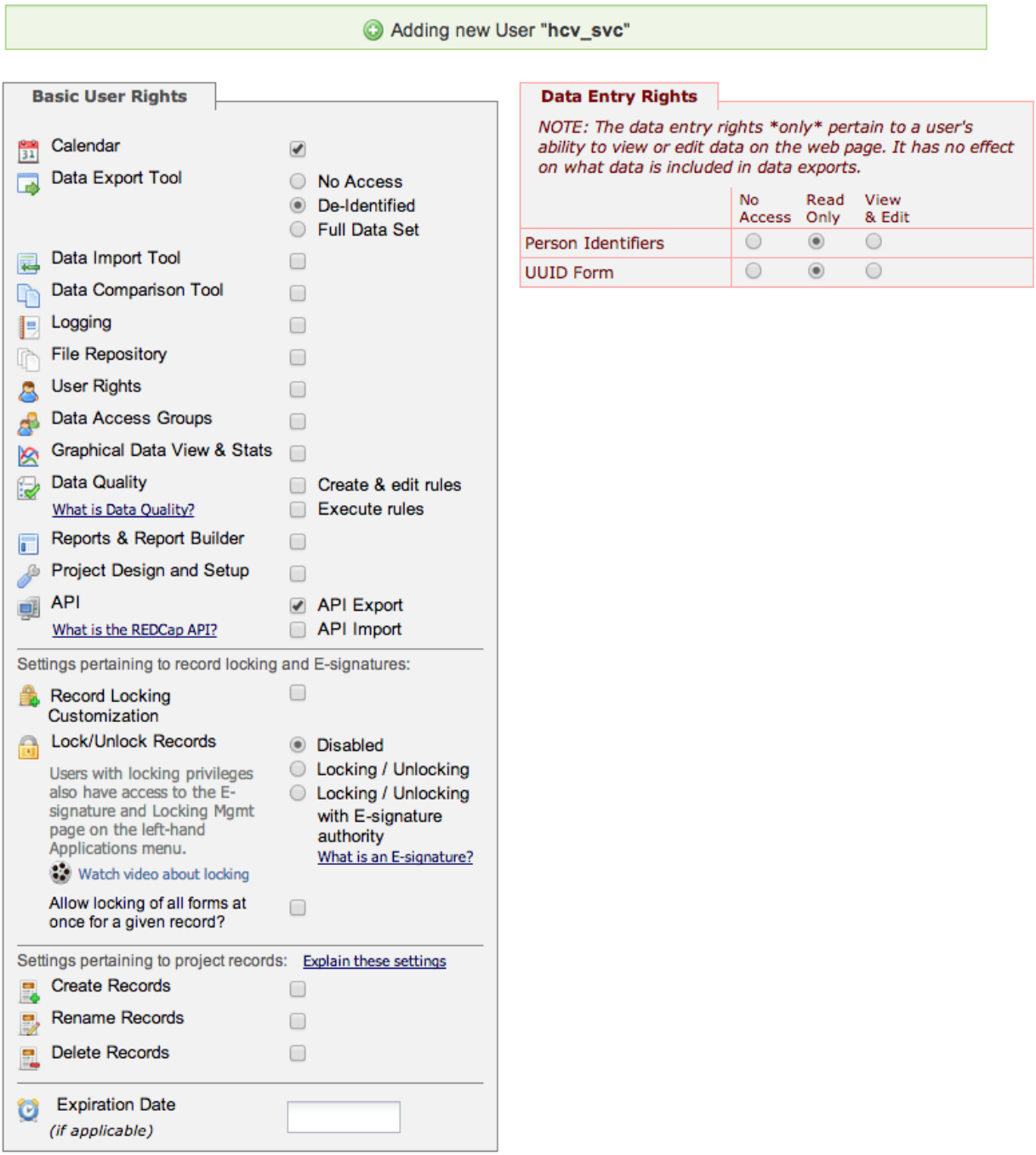
After you have created the new user, login as that user and request an Read-only API button on the left hand toolbar.

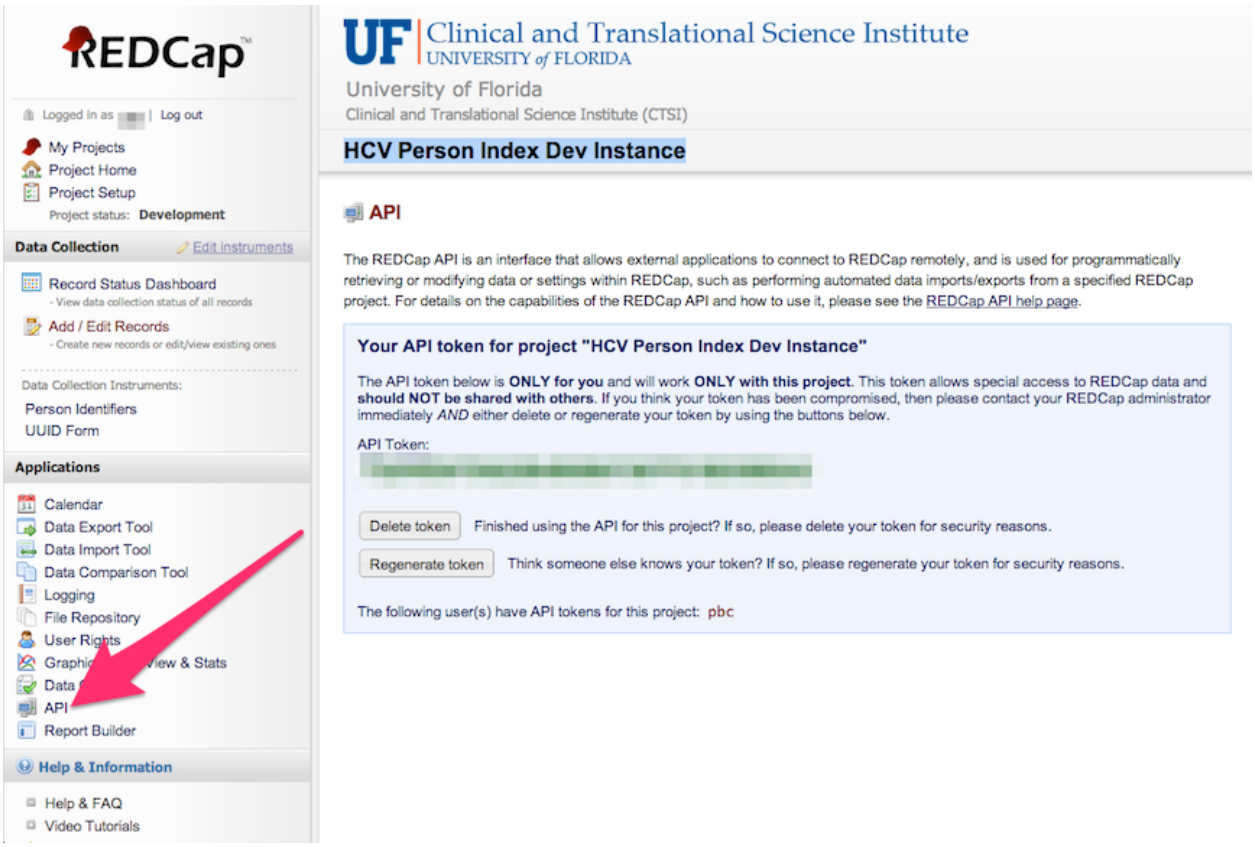
1.5.4 4. Export Data

If you have data in this project that needs to be preserved, you can export it using the steps listed in the section `2` above.


User Rights
Graphical Data View & Stats
Data Quality
API
Report Builder
Help & Information
Help & FAQ
Video Tutorials
Suggest a New Feature
If you are experiencing problems, please contact your REDCap administrator.
You may also browse for pre-built data collection instruments in the REDCap Shared Library
Enable optional modules and customizations
Optional
I'm done!
Enable
Auto-numbering for records
Scheduling module (longitudinal only)
Randomization module
Designate an email field to use for invitations to survey participants
Additional customizations
Set up project bookmarks (optional)
You may create custom bookmarks to webpages that exist inside or outside of REDCap. These bookmarks will be seen as links on the left-hand project menu and can be accessed at any time by users who are given privileges to do so. Every project bookmark has custom settings that allow one to control its appearance and behavior.
Go to
Add or edit bookmarks
User Rights and Permissions
You may grant other users access to this project or edit the user privileges of current users on this project by navigating to the User Rights page. Additionally, if you wish to limit user access to certain records/responses for this project, you may opt to use Data Access Groups, in which only users within a given Data Access Group can access records created by users within that group.
Go to
User Rights
or
Data Access Groups
Move your project to production status
Not started
Move the project to production status so that real data may be collected. Once in production, you will not be able to edit the project fields in real time anymore. However, you can make edits in Draft Mode, which will then need to be approved by a REDCap administrator before taking effect.
Go to
Move project to production


Project Setup
User Rights
Data Access Groups
You may set the rights for the user below by checking the boxes next to the application tools to which you wish to grant ther access. You may also grant them or deny them access to individual data collection instruments, if so desired. To save your selections, click the Save Changes button at the bottom of the page.
Editing existing User "pbc"
Basic User Rights
Calendar
Data Export Tool
No Access
De-Identified
Full Data Set
Data Import Tool
Data Comparison Tool
Logging
File Repository
User Rights
Data Entry Rights
NOTE: The data entry rights *only* pertain to a us
ability to view or edit data on the web page. It ha
effect on what data is included in data exports.
No Access
Read Only
View & Edit
Person Identifiers
UUID Form


Adding new User "hcv_svc"
Basic User Rights
Calendar
Data Export Tool
No Access
De-Identified
Full Data Set
Data Import Tool
Data Comparison Tool
Logging
File Repository
User Rights
Data Access Groups
Graphical Data View & Stats
Data Quality
What is Data Quality?
Create & edit rules
Execute rules
Reports & Report Builder
Project Design and Setup
API
What is the REDCap API?
API Export
API Import
Settings pertaining to record locking and E-signatures:
Record Locking Customization
Lock/Unlock Records
Users with locking privileges also have access to the E-signature and Locking Mgmt page on the left-hand Applications menu.
Watch video about locking
Disabled
Locking / Unlocking
Locking / Unlocking with E-signature authority
What is an E-signature?
Allow locking of all forms at once for a given record?
Settings pertaining to project records:
Explain these settings
Create Records
Rename Records
Delete Records
Expiration Date
(if applicable)
Data Entry Rights
NOTE: The data entry rights *only* pertain to a user's ability to view or edit data on the web page. It has no effect on what data is included in data exports.
No Access
Read Only
View & Edit
Person Identifiers
UUID Form



1.5.5 5. Backup Data

If you would like to backup this project along with other `REDCap` projects, please follow the procedures listed in the section `3`. If you want to initialize the project with no data in it, follow the procedures in section `2`.

1.5.6 6. Document the Existence of the Project

Please update the README-projects.md document with a detailed decription of the new project.

1.6 Clinical Data Specification

Clinical data from an Epic Clarity system should conform to this specification. The specification was designed to adhere to default Epic Clarity field names and data formats wherever possible.

- All data will be returned in the UTF8 Character set
- The first row of the file will be a header row showing the name of each column, enclosed in double quotes and separated by commas.
- The field names in the header row will be those shown below. The case of each column name should be that shown below.
- Subsequent rows will show lab result values. All values will be enclosed in double quotes and separated by commas.
- Each row will show one lab result value.

These fields may be specified in the data file:

```
Field Name       Field Required?  Field Description
STUDY_ID             yes          Identifier of a person within a study
NAME                 yes          Name of lab component
COMPONENT_ID         yes          Numeric identifier of lab component
ORD_VALUE            yes          Result value for lab component
REFERENCE_LOW        no           Lowest expected value for ORD_VALUE
REFERENCE_HIGH       no           Highest expected value for ORD_VALUE
REFERENCE_UNIT       yes          Units for ORD_VALUE
SPECIMN_TAKEN_TIME   yes          Date and time specimen was taken from the patient/study subject. Dat
```

1.7 Describing a REDCap Form to RED-I

RED-I needs to know certain information about REDCap forms and events so that it can correctly translate and insert clinical data into them.

The description of the forms is done in two XML files called **'Form Events <#form-events>'__** and **'Translation Table <#translation-table>'__**. The location of these files is specified in the `settings.ini` configuration file.

See: *How to Add a Simple REDCap Form to RED-I*

1.7.1 Form Events

- Format: **XML**
- Configuration:
- Key: **form_events_file**
- Default: **formEvents.xml**

Sample

```
<?xml version="1.0" encoding="UTF-8"?>
<redcapProject>
    <name>Project</name>
    <form>
        <name>cbc</name>

        <formDateField>cbc_date</formDateField>

        <formCompletedFieldName>cbc_done</formCompletedFieldName>
        <formCompletedFieldValue>2</formCompletedFieldValue>

        <formImportedFieldName>cbc_imported</formImportedFieldName>
        <formImportedFieldValue>Y</formImportedFieldValue>

        <event>
            <name>1_arm_1</name>
        </event>
        <event>
            <name>2_arm_1</name>
        </event>
    </form>
</redcapProject>
```

Root Element

Form Elements

The follow are children elements of each `redcap/form` element.

Event Elements

The following are children elements of each `redcap/form/event` element.

Required?	Name or Path	Description	Max
Required	`name`	Name of the **REDCap Event**	1

1.7.2 Translation Table

- Format: **XML**
- Configuration:
- Key: **translation_table_file**
- Default: **translationTable.xml**

Sample

```
<rediFieldMap>

    <clinicalComponent>
        <loinc_code>34714-6</loinc_code>
        <clinicalComponentName>INR</clinicalComponentName>

        <redcapFormName>inr</redcapFormName>

        <redcapFieldNameValue>inr_lab_result</redcapFieldNameValue>
        <redcapFieldNameValueDescriptiveText>INR</redcapFieldNameValueDescriptiveText>

        <redcapStatusFieldName>inr_lab_status</redcapStatusFieldName>
        <redcapStatusFieldValue>NOT_DONE</redcapStatusFieldValue>
    </clinicalComponent>

    <clinicalComponent>
        <loinc_code>11011-4</loinc_code>
        <clinicalComponentName>Hepatitis C virus RNA</clinicalComponentName>

        <redcapFormName>hcv_rna</redcapFormName>

        <redcapFieldNameValue>hcv_lab_result</redcapFieldNameValue>
        <redcapFieldNameUnits>hcv_lab_result_units</redcapFieldNameUnits>
        <redcapFieldNameValueDescriptiveText>HCV RNA results</redcapFieldNameValueDescriptiveText>
    </clinicalComponent>

</rediFieldMap>
```

Clinical Components

Clinical Component is a generic term for test, measurement, or observation. Each Clinical Component is represented by a `clinicalComponent` XML element whose children elements are as follows:

1.7.3 How to Add a Simple REDCap Form to RED-I

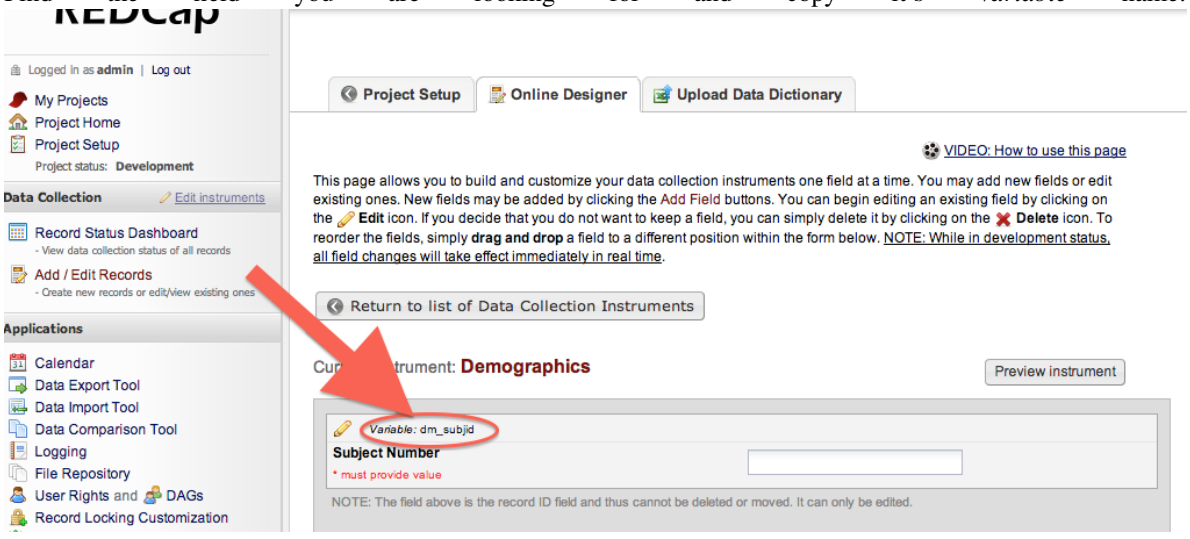
Remember, when "adding a form" you are describing it to RED-I. So, you can open your browser and use the actual REDCap Form to guide you.

1. Edit `formEvents.xml`
2. Copy the contents of the sample data for a `form` element.
3. Replace the text of all XML Elements using the descriptions above.
4. Edit `translationTable.xml`
5. Copy the contents of the sample data for a `clinicalComponent` element
6. Replace the text of all XML Elements using the descriptions above.
7. Repeat for as many clinical components as needed.

Note

You can lookup a field's ID using REDCap's **Data Collection Instruments** editor.

1. Click *Project Setup*
2. Under *Design your data collection instruments*, click *Online Designer*.
3. Find the name of your form (called an Instrument), such as "Demographics".
4. Find the field you are looking for and copy it's *Variable* name.



1.8 Mapping Clinical Components to LOINC Codes

This document explains how to lookup LOINC Codes and map them to a site's local clinical components for RED-I to use.

1.8.1 Lookup LOINC Codes

1. Download the LOINC Table File and open it in your preferred Spreadsheet Viewer.
2. Find the columns **LONG_COMMON_NAME** and **COMMON_TEST_RANK**.
3. Filter the **LONG_COMMON_NAME** by the name of your clinical component.
4. Sort **COMMON_TEST_RANK** descendingly. This column ranks the top 2000 components from most common (ranked 1) to least common (ranked 2000), as described on the LOINC usage site. Unless you are mapping specialized data, it is more likely that the component of interest is ranked more highly, i.e., it is more common.
5. Starting with the highest rank, use LOINC Search to investigate each Component until you've found the one you're looking for.

1.8.2 Map Local ID to LOINC Code

Mapping is done in an XML document; a sample of a mapping file and an explaination of the format is as follows.

- Format: **XML**
- settings.ini configuration:
- Key: **component_to_loinc_code_xml**
- Default: **clinical-component-to-loinc.xml**

Sample

```
<?xml version='1.0' encoding='US-ASCII'?>
<clinical_datum>
    <version>0.1.0</version>
    <Description>
      Mapping of the University Hospital's Lab Component Identifiers to corresponding LOINC codes
    </Description>
    <components>
        <component>
          <description>Leukocytes [#/volume] in Blood</description>
          <source>
            <name>COMPONENT_ID</name>
            <value>8675309</value>
          </source>
          <target>
            <name>loinc_code</name>
            <value>26464-8</value>
          </target>
        </component>
        <component>
          <description>Hepatitis C virus RNA [#/volume]</description>
          <source>
            <name>COMPONENT_ID</name>
            <value>42</value>
          </source>
          <target>
            <name>loinc_code</name>
            <value>26464-8</value>
          </target>
        </component>
```

```
    </components>
</clinical_datum>
```

Clinical Datum

The root element **MUST** be `clinical_datum`.

Required?	Name or Path	Description	Max	Notes
Optional	`version`	Version number of the Clinical Datum XML format.	1	0.1.0 is currently the only version
Optional	`Description`	Description of the XML document	1	Potentially helpful to readers
Required	`components`	The parent element for *Clinical Components Mappings*	1	

Clinical Component Mapping

Represented as `component` elements under `clinical_datum/components`. Each component represents a mapping from a (clinical component) Source to a Target (LOINC code). Source comprises an XML Element name and value; likewise, Target comprises the name and values to use as a replacement.

For every incoming clinical component that will be mapped to a LOINC value, create a new block and complete it according to the below table.

Required?	Name or Path	Description	Max	Notes
Optional	`Description`	Description of the Component	1	Potentially helpful to readers
Required	`source`	Parent element for the Source information	1	
Required	`source/name`	Name of the XML Element	1	
Required	`source/value`	Value of the XML Element	1	
Required	`target`	Parent element for the Target information	1	
Required	`source/name`	Name of the XML Element	1	
Required	`source/value`	Value of the XML Element	1	

Example

Given the sample above, the following input:

```
<COMPONENT_ID>8675309</COMPONENT_ID>
```

would be mapped to:

```
<loinc_code>26464-8</loinc_code>
```

CHAPTER 2

Developer Documentation

2.1 Testing RED-I with a sample REDCap Project

2.1.1 Purpose

The "vagrant" folder was created with the goal of making testing RED-I software as easy as possible. It contains the Vagrantfile which allows you to start a virtual machine capable of running the REDCap software – which means that during virtual machine creation the Apache and MySQL software is installed without any user intervention.

There are a few important things to note before proceeding with running RED-I to import data into a sample REDCap project:

- You have to install the **vagrant** and **virtual box** software
- You have to obtain the closed-source REDCap software from http://project-redcap.org/
- You have to obtain a **Makefile.ini** file in order to be able to execute tasks from the **Makefile**

2.1.2 Steps

1. Install vagrant and virtual box

On a linux machine run:

- sudo apt-get install vagrant
- sudo apt-get install virtualbox

On a mac machine:

- Download and install vagrant from https://www.vagrantup.com/downloads.html
- Download and install the latest virtual box from http://download.virtualbox.org/virtualbox/

For more details about Vagrant software you can go to why-vagrant page.

2. Configure the VM

As mentioned above you have to obtain a copy of the REDCap software from http://project-redcap.org/ and save it as "**redcap.zip**" file in the "**config-example/vagrant-data**" folder. This ensures that in the later steps the bootstrap.sh script can extract the files to the virtual machine path "**/var/www/redcap**".

Now execute the following commands to complete the configuration:

Please verify that the output from "show_config" matches your expectations.

3. Start the VM

To use the vagrant VM you will need to install Vagrant and Virtual Box.

With these packages installed, follow this procedure to use a VM template:

Vagrant will instantiate and provision the new VM. The REDCap web application should be accessible in the browser at

http://localhost:8998/redcap/

If port 8998 is already in use vagrant will choose a different port automatically. Read the log of "vagrant up" and note the port to be used.

4. Verify the VM is running

Verify that the virtual machine is working properly by accessing it using:

5. Import Enrollment Data using RED-I

Import the sample subject list into REDCap by executing:

Note: This step is necessary because in order to associate data with subjects the list of subjects needs to exist in the REDCap database.

6. Import Electronic Health Records using RED-I

Import the sample electronic health records into REDCap by executing:

Verify that the output of this command ends with:

If this step succeded you have verified that RED-I can be used to save time by automating EHR data imports into REDCap.

Congratulations! You can now add your own REDCap project and start using RED-I to move data. Please refer to Add new REDCap Project and API Key document for help.

2.2 Code Review Checklist

When reviewing a pull request there are many quality checks we could perform. This document can serve as a guide for tests we can apply to assess the quality of the work.

- Is the pull request against the right branch? Changes without dependencies should be forked from the head of the ***master*** and merged back to ***develop***. Changes with dependencies should fork from and merge back to a non-production branch, generally ***develop***.
- Does the pull request contain one and only one thing be it a new feature, bug fix, refactor, etc. ?
- Do the commit(s) in the request describe the work?
- Does the pull request comment describe the work?
- Does the pull request pass the Travis tests?

- Do new tests accompany any new code?
- If this is a bug fix, is a test added to test for that bug?
- If this is a new feature, is it documented?
- Does the new code improve or maintain the lint score?
- Does the new code improve or maintain the code coverage score?
- If the new code contains changes that would break existing installations (e.g. database schema, configuration file changes), does it also contain relevant documentation of the change? Does it have a migration procedure? Does it have a migration tool?
- Does the merged code pass an integration test?

2.3 Setting up Travis CI for Open Source REDI project

2.3.1 Steps to setup Travis CI for RED-I

1. Sign in to travis using Github account (ctsit)
2. Give permissions to travis to clone and sync github repositories when asked.
3. Activate a webhook on the repo on your choice

1. Go to repositories in your profile (Click on your profile as shown below)
2. Flip the switch against the repo of your choice to enable webhook for that repo as displayed in the image below.

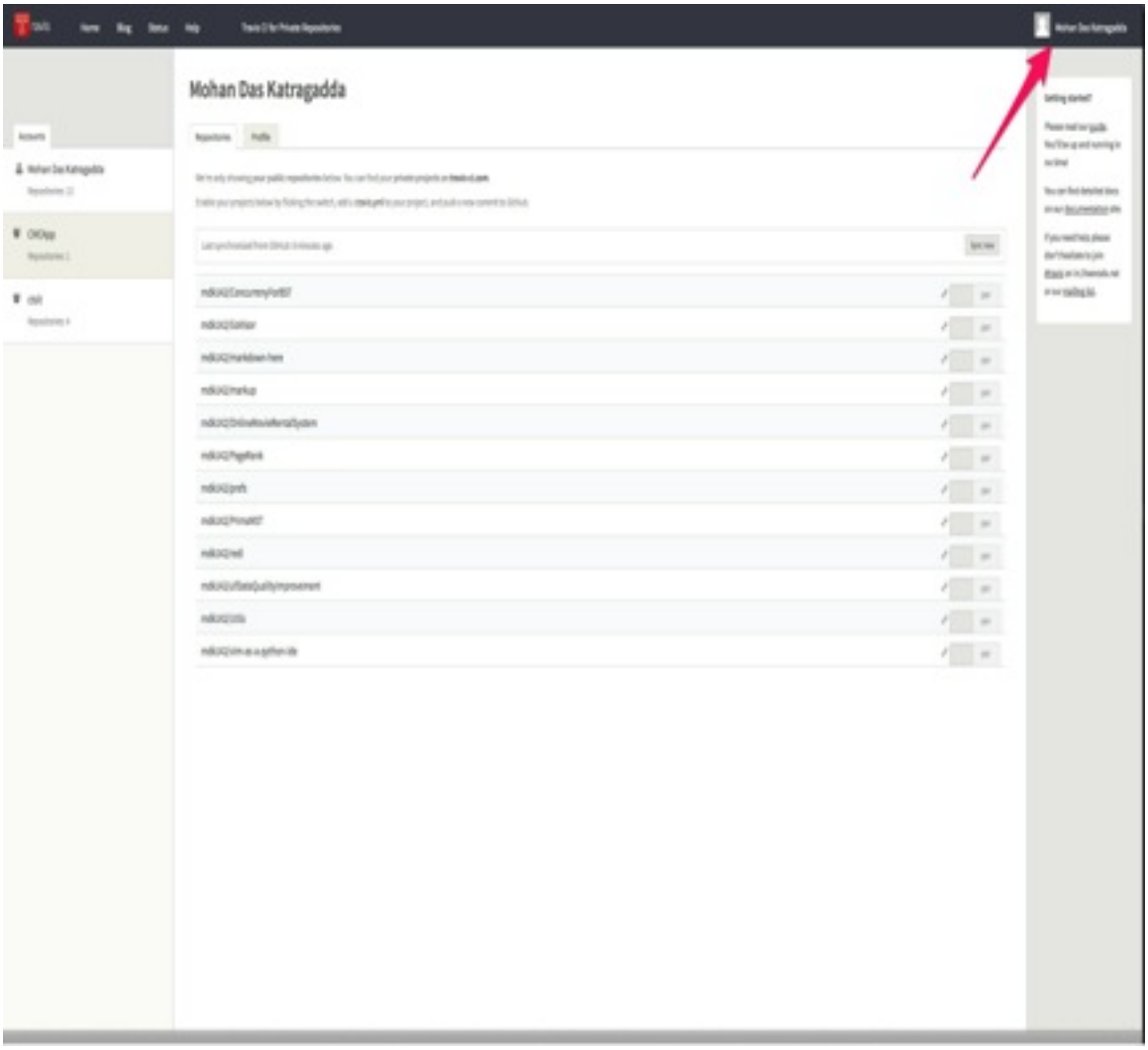
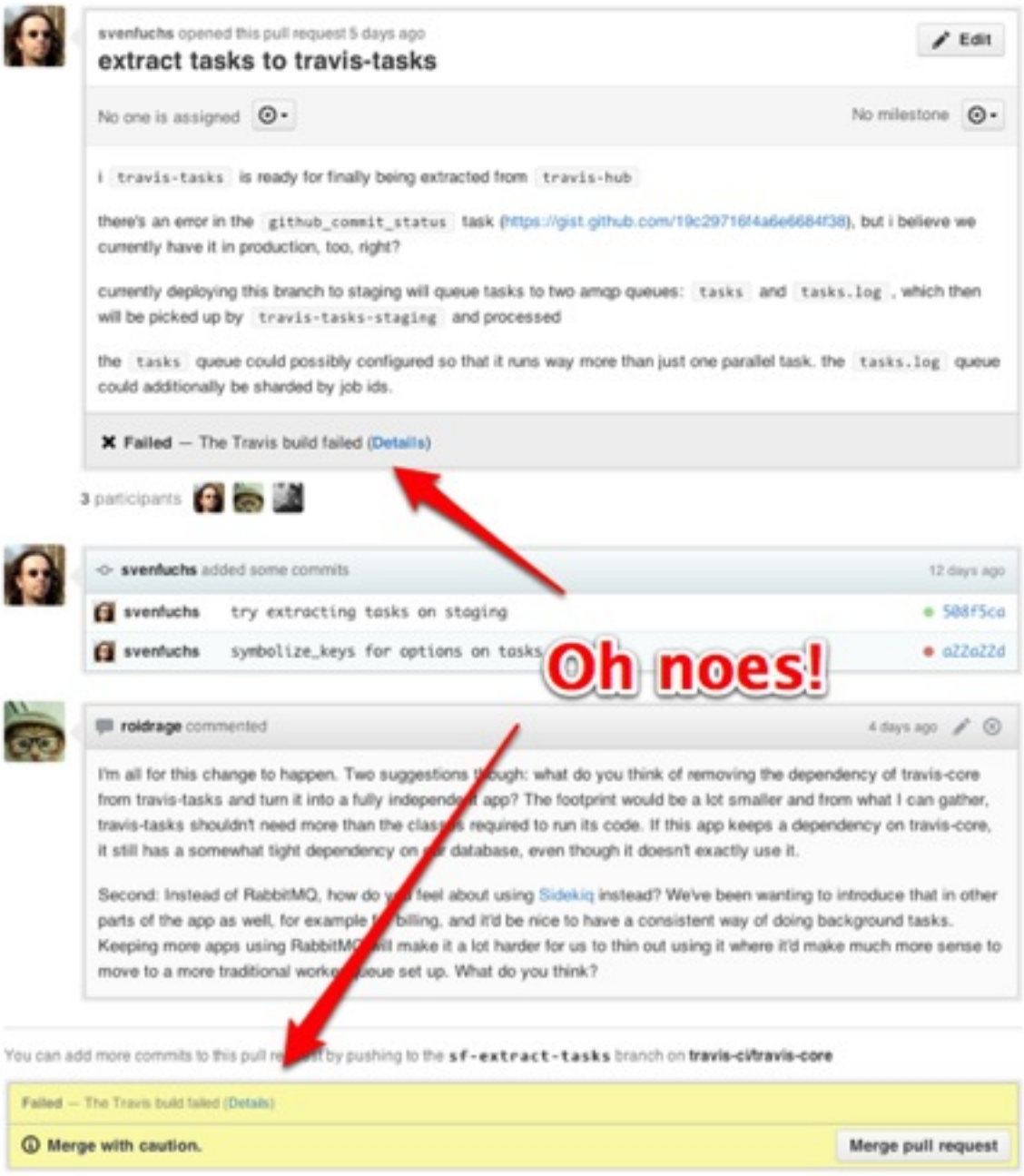


Fig. 2.1: image

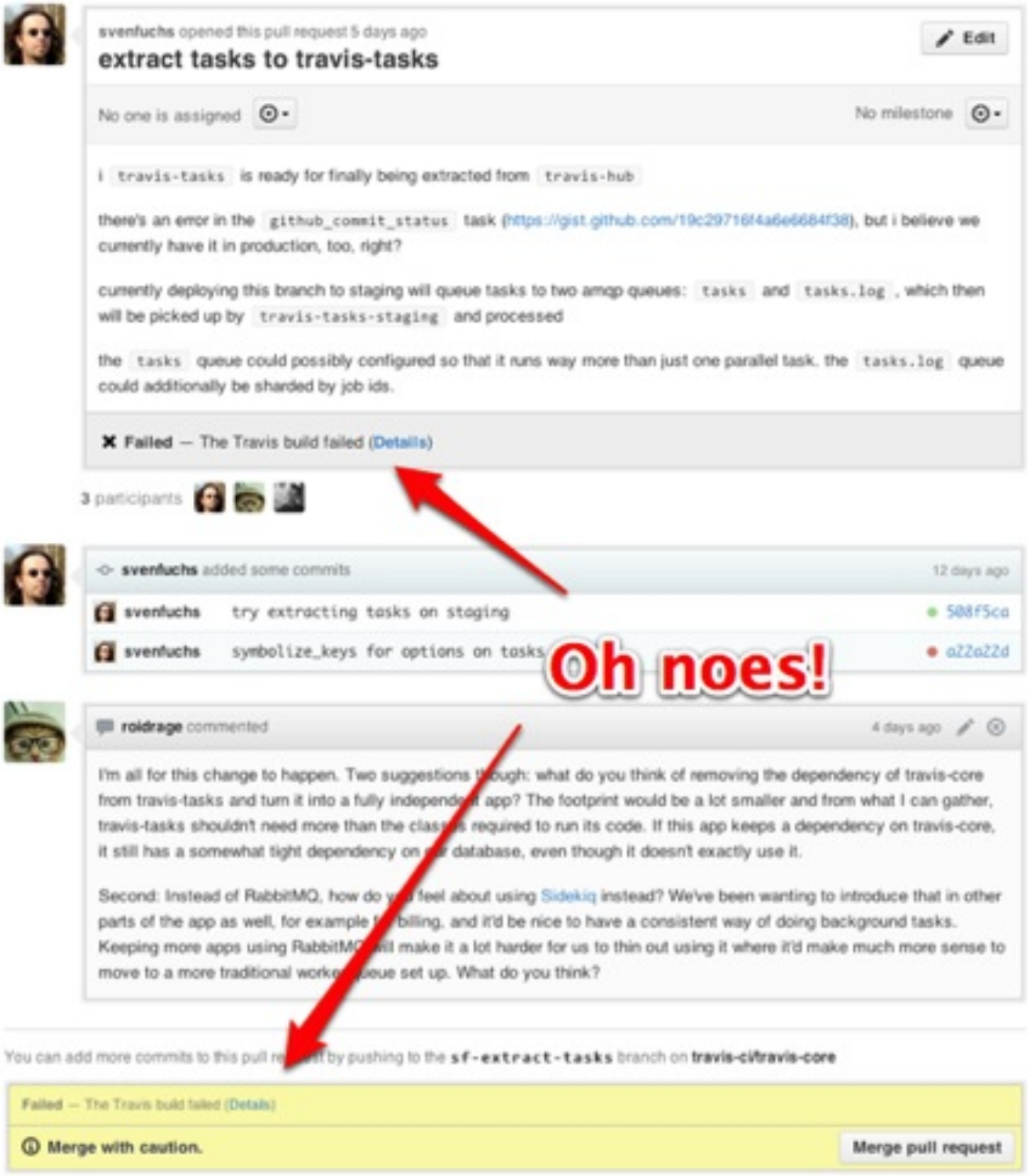


4. Create a .travis.yml file and add it to your repository. I have created a basic .travis.yml, which work for current REDI. Below are its contents

```
language: python
python:
  - "2.7"
before_install:
  sudo apt-get install -y python-setuptools libxml2 libxslt1-dev python-dev
install:
  - pip install requests
  - pip install lxml
  - pip install appdirs
```

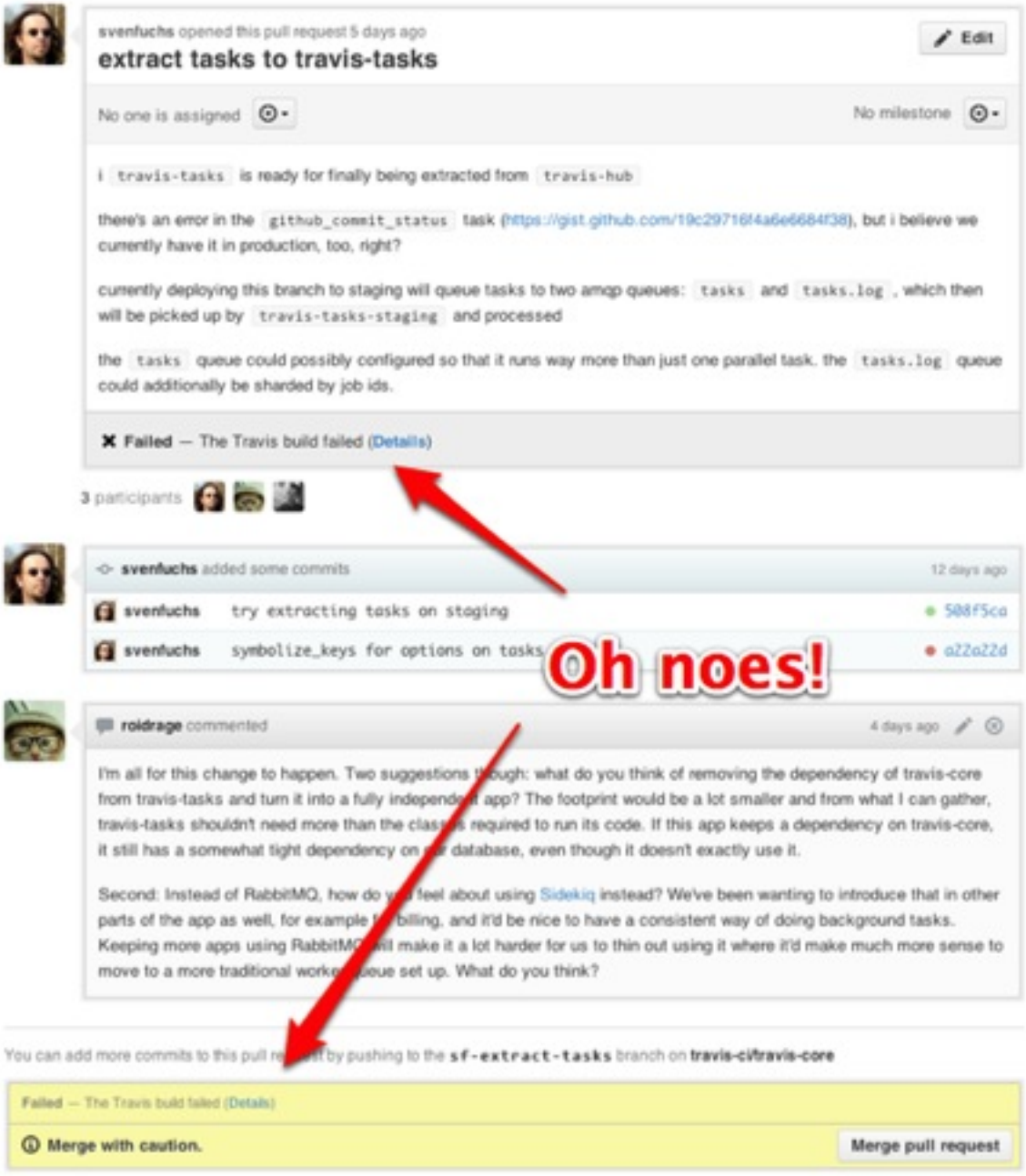
```
script: make test

branches:
  only:
    - master
    - develop
```

5. The contents listed above are pretty basic. For any additional features you need to configure refer this link.
6. Travis is now configured to your repository. For each new commit made to the master branch, travis will run new build for your repository and results will be mailed to the email registered in your github account.
7. You can also view results on your Travis dashboard as shown in the below image.

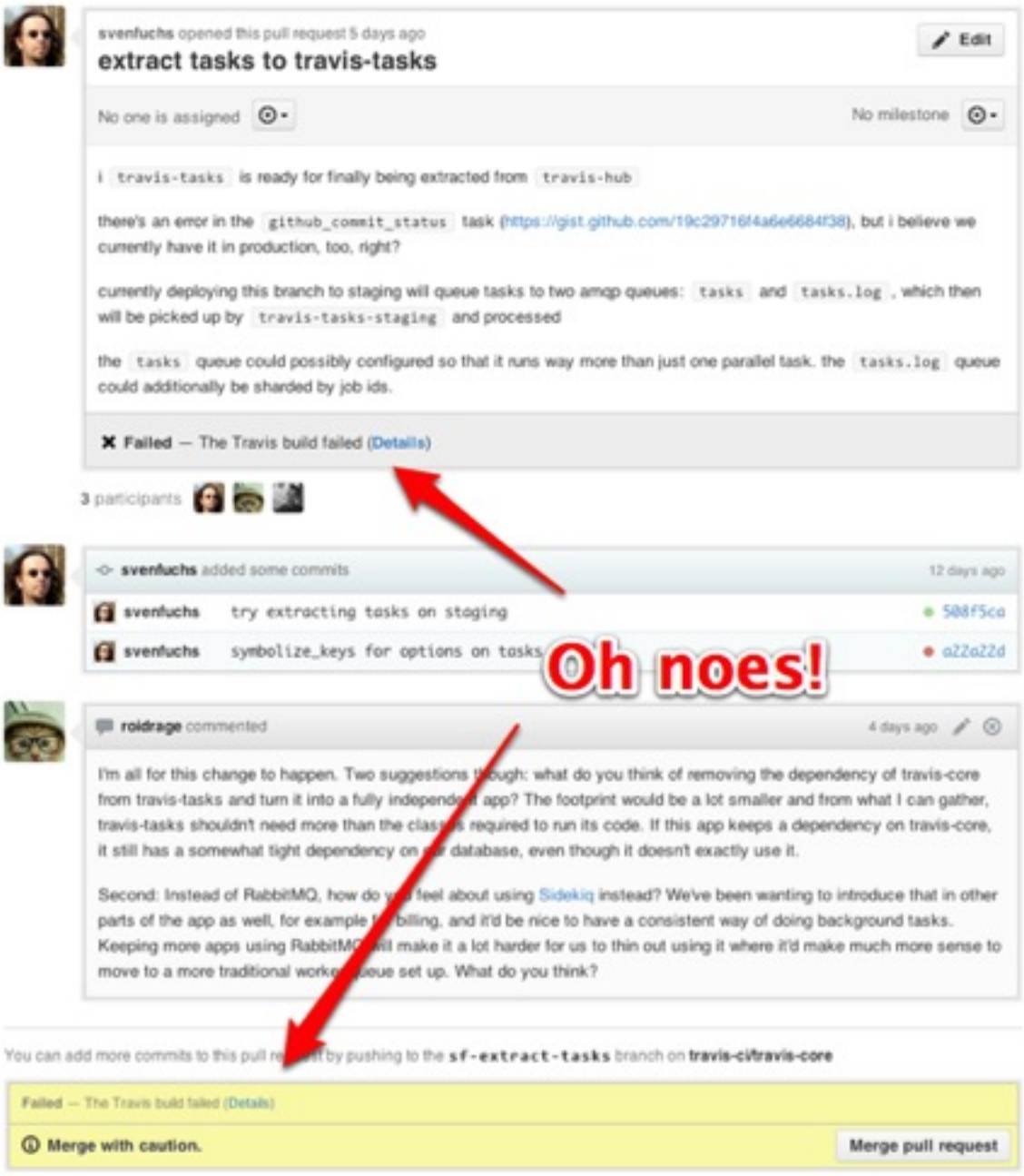


2.3.2 CTS-IT Responsibilities when a pull request is submitted on REDI

1. When REDI gets a pull request, CTS-IT coordinator should open the pull request. This initiates Travis to run tests against pull request and displays the results.
2. Pull requests that passed the tests looks like as shown in the image.



3. Pull requests that failed the tests looks like as shown in the image below.



2.3.3 Responsibilities of the developer submitting a pull request:

1. Fork the REDI repository. You will get a .travis.yml with the fork.
2. Setup a travis account for your github account and give permissions for travis to run build.
3. With this you should receive travis reports for each commit you push to your forked repository.
4. Before submitting a pull request, make sure tests in your travis report are all passing.

2.4 Regression Testing

The intent of the procedure described below is to find new bugs or unexpected side effects in the code after a release lifecycle is completed.

`Goal`: Compare two files obtained from RedCAP after two separate imports are performed using the production and the candidate versions of REDI software.

Note: The login credentials for the RedCAP web application on the testing VM at http://localhost:8998/redcap/index.php are: `admin/password`

2.4.1 Steps

1. Obtain the candidate version (redi_B) of the software

```
$ git clone git@repo_redi redi_new
$ cd redi_new && git checkout develop
```

- Clone the config repository at ./config and checkout the tag that matched the REDI version

  git clone git@repo_config config cd config && git checkout develop

- If needed, revise the settings.ini lines that identify the host and authorize access. For this VM, the lines are

```
"redcap_uri": "http://localhost:8998/redcap/api/",
"token": "121212",

$ sed -i 's/^\s\+"redcap_uri.*/\t"redcap_uri" : "http:\/\/localhost:8998\/redcap\/api\/",/' sett
$ sed -i 's/^\s\+"token.*/\t"token" : "121212",/' settings.ini
```

- Go to the vagrant folder and remove the old virtual machine (VM) instance if necessary:

```
$ cd vagrant && vagrant destroy
```

- Start a fresh VM:

  $ vagrant up

The expected output from this command should look like:

```
redcap.zip content indicates Redcap version: 5.7.4
Setting the connection variables in: /var/www/redcap/database.php
Checking if redcap application is running...
   <b>Welcome to REDCap!</b>
```

- If you need to preserve the database state then you can create a backup of the REDCap database before we insert any data by running the redcapdbm.php script on the VM:

  $ vagrant ssh -c 'cd /vagrant/scripts && php redcapdbm.php -b'

Note: The commands above create a file like `backup-redcap-YYYYmmdd-HHMM.sql` on the VM

- Erase the data in the REDCap instance using redcapdbm.php. First, enumerate the REDCap projects

  $ vagrant ssh -c 'cd /vagrant/scripts && php redcapdbm.php -l'

  The following projects are currently available in the redcap database:

  project_id: 1, project_name: redcap_demo_cda700 project_id: 2, project_name: redcap_demo_f3746b project_id: 3, project_name: redcap_demo_117155 ... project_id: 12, project_name: hcvtarget_20_development ...

Erase the data in the correct project if necessary:

```
$ cd ../vagrant && vagrant ssh -c 'cd /vagrant/scripts && php redcapdbm.php -d 12'

 Deleting data for project: 12, name: hcvtarget_20_development
 Rows deleted from `redcap_surveys_response`:0
 Rows deleted from `redcap_surveys_participants`:0
 Rows deleted from `redcap_surveys_emails`:0
```

- If needed, load a minimal set of data to get research identifiers into REDCap. Use the redcap_records utility to load this data.

```
    $ redcap_records --token=121212 --url=http://localhost:8998/redcap/api/ -i demographic_test_data
```

On success the following text is returned:

```
{u'count':  5}
```

- Upload raw.txt file to REDCap using redi_B

  Note: You may want to increase the number of requests allowed for processing before proceeding with data upload.

```
    Open the address "localhost:8998" using your browser

    Login to REDCap and then click on "Control Center" tab

    Click on the "Security & Authentication Configuration" link on the left menu

    Find and adjust the "Rate Limiter" field to something like 60000


    $ python ../redi/redi.py
```

If the output from the command above produces an exception then check if your IP was not banned due to numerous requests sent (@see related code in Config/init_functions.php: `checkBannedIp() and storeHashedIp()`)

```
select * from redcap_ip_banned where ip = '10.0.2.2';
+----------+---------------------+
| ip       | time_of_ban         |
+----------+---------------------+
| 10.0.2.2 | 2014-06-06 18:59:25 |
+----------+---------------------+
```

To fix this, use these SQL commands:

```
update redcap_config set value = 600000 where field_name = 'page_hit_threshold_per_minute';
delete from redcap_ip_banned;
```

If the token is invalid the following error is returned:

```
Cannot connect to project at http://localhost:8998/redcap/api/ with token
121212
```

- Download relevant forms from REDCap using a command like:

```
    $ redcap_records --token=121212 --url=http://localhost:8998/redcap/api/ -f "demgraphics chemistr
```

If you have a lot of forms, the output comparison is easier if you export the forms separately like this:

```
#!/bin/bash
```

```
batch=$1
forms="demographics chemistry cbc inr hcv_rna_results"
if [ ! -e $batch ]; then
    mkdir $batch
fi

for form in $forms
    do
      redcap_records --token=121212 --url=http://localhost:8998/redcap/api/ --forms=$form > $batch/$f
    done
```

Later do the diff like this:

```
diff -ur a/ b/
```

- At this point we have gathered the output from the release candidate software redi_B. If we there is reference output file available than we can just compare the outputs:

```
    $ diff -u output_A.csv output_B.csv
```

- If there is no reference output file available than we have to get a previous version redi_A and generate it.

  Erase the REDCap data first:

```
    cd ../vagrant && vagrant ssh -c 'cd /vagrant/scripts && php redcapdbm.php -d 12'
```

- Obtain the reference version (redi_A) of redi software.

```
    $ git clone git@repo_redi redi_old
    $ cd redi_old && git checkout TAG_ID_OLD

    $ git clone git@repo_config config
    $ cd ../config && git checkout TAG_ID_OLD_CONFIG
```

- Repeat steps 8-10 with redi_A software with the only difference being that the output file is changed to `output_A.csv` on step 10.
- Compare the files `output_A.csv` and `output_B.csv` to insure there are no differences or expected differences are present:

```
    $ diff -u output_A.csv output_B.csv
```

If no new behavior was introduced the output from the command above should be an empty string.

2.5 redi

2.5.1 redi package

Subpackages

redi.utils package

Submodules

redi.utils.GetEmrData module This module is used to connect to an sftp server and retrieve the raw EMR file to be used as input for RED-I.

class redi.utils.GetEmrData.**EmrFileAccessDetails**(*emr_sftp_project_name*, *emr_download_list*, *emr_host*, *emr_username*, *emr_password*, *emr_port*, *emr_private_key*, *emr_private_key_pass*)

Bases: object

Encapsulate the settings used to retrieve the EMR source file using an SFTP connection @see redi#_run()

redi.utils.GetEmrData.**cleanup**(*file_to_delete*)

redi.utils.GetEmrData.**data_preprocessing**(*input_filename*, *output_filename*)

redi.utils.GetEmrData.**download_files**(*destination*, *access_details*)

Download a file from the sftp server :destination the name of the file which will be downloaded :access_details holds info for accessing the source file over sftp

@see get_emr_data()

redi.utils.GetEmrData.**generate_xml**(*input_filename*, *output_filename*)

redi.utils.GetEmrData.**get_emr_data**(*conf_dir*, *connection_details*)

:conf_dir configuration directory name :connection_details EmrFileAccessDetails object

redi.utils.SimpleConfigParser module SimpleConfigParser

Simple configuration file parser: Python module to parse configuration files without sections. Based on ConfigParser from the standard library.

Author: Philippe Lagadec

Project website: http://www.decalage.info/python/configparser

Inspired from an idea posted by Fredrik Lundh: http://mail.python.org/pipermail/python-dev/2002-November/029987.html

Usage: see end of source code and http://docs.python.org/library/configparser.html

exception redi.utils.SimpleConfigParser.**ConfigurationError**

Bases: exceptions.Exception

class redi.utils.SimpleConfigParser.**SimpleConfigParser**(*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*)

Bases: ConfigParser.RawConfigParser

Simple configuration file parser: based on ConfigParser from the standard library, slightly modified to parse configuration files without sections.

Inspired from an idea posted by Fredrik Lundh: http://mail.python.org/pipermail/python-dev/2002-November/029987.html

check_parameters()

handle required and default optional_parameters_dict

getoption(*option*)

get the value of an option

getoptionslist()

get a list of available options

hasoption(*option*)
return True if an option is available, False otherwise. (NOTE: do not confuse with the original has_option)

read(*filename*)

set_attributes()

redi.utils.SimpleConfigParser.**to_bool**(*value*)
Helper function for translating strings into booleans @see test/TestReadConfig.py

redi.utils.csv2xml module

class redi.utils.csv2xml.**Writer**(*ofile*, *args*)

write(*text*)

write_field(*field*, *index*)

write_file(*data*)

write_record(*record*)

redi.utils.csv2xml.**cleanup_callback**(*option*, *opt*, *value*, *parser*)

redi.utils.csv2xml.**field_subst_factory**(*newline*)

redi.utils.csv2xml.**openio**(*filename*, *mode*, *encoding*, *newline=None*)

redi.utils.csv2xml.**parse_cmdline**()

redi.utils.csv2xml.**replace**(*text*, *s*, *r*)

redi.utils.rawxml module

class redi.utils.rawxml.**RawXml**(*project*, *path*)
Bases: object

This class is used to store details about the input file @see redi.batch.check_input_file()

get_creation_time()
Get the OS creation time

get_info()
Return a string containing all details available about the xml file

get_last_modified_time()
Get the OS modification time

get_project()

redi.utils.redcapClient module

class redi.utils.redcapClient.**RedcapClient**(*redcap_uri*, *token*, *verify_ssl=True*)
Bases: object

Client for a REDCap server.

Parameters

- **redcap_uri** – URI for to REDCap server's API
- **token** – API Token for a REDCap project.
- **verify_ssl** – verify the SSL certificate? (default: True)

Raises

- **RedcapError** – if we failed to get the project's metadata
- **RequestException** – if some other network-related failure occurs

get_data_from_redcap(*records_to_fetch=None*, *events_to_fetch=None*, *fields_to_fetch=None*, *forms_to_fetch=None*, *return_format='xml'*)

Exports REDCap records.

Parameters

- **records_to_fetch** (*list or None*) – if specified, only includes records in this list. Otherwise, includes all records.
- **events_to_fetch** (*list or None*) – if specified, only includes events in this list. Otherwise, includes all events.
- **fields_to_fetch** (*list or None*) – if specified, only includes fields in this list. Otherwise, includes all fields
- **forms_to_fetch** (*list or None*) – if specified, only includes forms in this list. Otherwise, includes all forms.
- **return_format** – specifies the format of the REDCap response (default: xml)

Returns response

send_data_to_redcap(*data*, *max_retry_count*, *overwrite=False*, *retry_count=0*)

Sends records to REDCap.

Parameters

- **of dict objects data** (*list*) – records to send.
- **overwrite** (*bool*) – treat blank values as intentional? (default: False) When sending a record, if a field is blank, by default REDCap will not overwrite any existing value with a blank.

Returns response

Raises RedcapError if failed to send records for any reason.

:If MaxRetryError is caught, the function will try resending the same data for a maximum of max_retry_count times before exitting. For each attempt the wait time before sending is (the_attempt_no * 6)

redi.utils.redi_email module

redi.utils.redi_email.**add_attachment**(*msg*, *body*)

Add the html report as attachment

msg [MIMEMultipart] The object to which we attach the body content

body [string] The html content to be attached

redi.utils.redi_email.**send_email**(*host*, *port*, *sender*, *to_addr_list*, *cc_addr_list*, *subject*, *msg_body*)

The email deliverer. Return True if the email was sent

to_addr_list [list] The recipients of the email

redi.utils.redi_email.**send_email_data_import_completed**(*email_settings*, *body=''*)

Email the html report after redi completed the data transfer Returns a dictionary, with one entry for each recipient that was refused

email_settings [dict] Email params produced by redi.get_email_settings()

body [string] The html content produced by transforming the xsl generated by redi.create_summary_report()

redi.utils.redi_email.**send_email_input_data_unchanged**(*email_settings*, *raw_xml*)
Send a warning email to the *redcap_support_receiver_email* if the input file did not change for more than *batch_warning_days* Return True if the email was sent

email_settings [dictionary] The email delivery parameters

raw_xml [RawXml instance] The object storing details about the input file

redi.utils.redi_email.**send_email_redcap_connection_error**(*email_settings*, *subject=''*, *msg=''*)
Return True if the email was sent. Notify the designated *REDCap support* person about problems with reaching REDCap

email_settings [dict] The dictionary with smtp server parameters

subject [str] The email subject

msg [str] The content to be emailed

redi.utils.throttle module Utility module for throttling calls to a function

class redi.utils.throttle.**Throttle**(*function*, *max_calls*, *interval_in_seconds=60*)
Bases: object

Limits the number of calls to a function to a given rate.

The rate limit is equal to the max_calls over the interval_in_seconds.

Parameters

- **function** – function to call after throttling
- **max_calls** – maximum number of calls allowed
- **interval_in_seconds** – size of the sliding window

Module contents

Submodules

redi.batch module

Functions related to the RediBatch database

redi.batch.**BATCH_STATUS_COMPLETED = 'Completed'**
@see #check_input_file()

The first time we run the app there is no SQLite file where to store the md5 sums of the input file. This function creates an empty RediBatch in the SQLite file specified as *db_path*

@return True if the database file was properly created with an empty table

redi.batch.**add_batch_entry**(*db_path*, *md5*)
Inserts a row into RediBatch table @see #check_input_file() Parameters ————- db_path : string

The SQLite database file name

md5 [string] The md5 sum to be inserted

create_time [string] The batch start time

redi.batch.**check_input_file**(*batch_warning_days*, *db_path*, *email_settings*, *raw_xml_file*, *project*, *start_time*)

redi.batch.**create_empty_md5_database**(*db_path*)

redi.batch.**create_empty_table**(*db_path*)

redi.batch.**dict_factory**(*cursor*, *row*)

redi.batch.**get_batch_by_id**(*db_path*, *batch_id*)

redi.batch.**get_days_since_today**(*date_string*)
@return the number of days passed since the specified date

redi.batch.**get_db_friendly_date**()
@return string in format: 2014-06-24

redi.batch.**get_db_friendly_date_time**()
@return string in format: "2014-06-24 01:23:24"

redi.batch.**get_last_batch**(*db_path*)

redi.batch.**get_md5_input_file**(*input_file*)
@see #check_input_file() @see https://docs.python.org/2/library/hashlib.html @see https://docs.python.org/2/library/sqlite3.html#sqlite3.Connection.row_factory

Returns the md5 sum for the redi input file

redi.batch.**printxml**(*tree*)
Helper function for debugging xml content

redi.batch.**update_batch_entry**(*db_path*, *id*, *status*, *start_time*, *end_time*)
Update the status and the start/end time of a specified batch entry Return True if update succeeded, False otherwise

db_path : string id : integer status : string start_time : datetime string end_time : datetime string

redi.form module

class redi.form.**Event**(*etree_node*)
Bases: object

field(*name*)

fields()

form_name

is_empty()

name

study_id

class redi.form.**Field**(*etree_node*)
Bases: object

clear_value()

name

value

class redi.form.**Form**(*data*)
Bases: object

events()

redi.redi module

redi.py - Converter from raw clinical data in XML format to REDCap API data

Usage: redi.py -h | –help redi.py [-v] [-V] [-k] [-e] [-d] [-r] [-c=<path>] [-D=<datadir>] [-s] [-b]

Options: -h –help Show this help message and exit -v –verbose Increase output verbosity [default:False] -V –version Show version number [default:False] -k –keep Use this option to preserve the files

generated during execution [default:False]

-e –emrdata Use this option to get EMR data [default:False] -d –dryrun To execute redi.py in dry run state. This

is to be able to test each release by doing a dry run, where the data is fetched and processed but not transferred to the production REDCap. Email is also not sent. The processed data is stored as output files under the "out" folder under project root [default:False].

-r –resume WARNING!!! Resumes the last run. This switch is for a specific case. Check the documentation before using it. [default:False]

-c –config-path=<path> Specify the path to the configuration directory -D –datadir=<datadir> Specify the path to the directory containing

project specific input and output data which will help in running multiple simultaneous instances of redi for different projects

-s –skip-blanks Skip blank events when sending data to REDCap [default:False]

-b –bulk-send-blanks Send blank events in bulk instead of individually [default:False]

class redi.redi.**PersonFormEventsRepository**(*filename*, *logger=None*)
Bases: object

Wrapper for the person-form-events XML file

delete()

fetch()

store(*pfe_tree*)

class redi.redi.**SentEvents**(*filename*, *writer=None*, *reader=None*)
Bases: object

List of form events that have been sent to REDCap

Parameters

- **filename** – file location
- **writer** – delegate called after an event has been marked sent
- **reader** – function to read previously sent events from disk

mark_sent(*study_id_key*, *form_name*, *event_name*)

was_sent(*study_id_key*, *form_name*, *event_name*)

redi.redi.**add_elements_to_tree**(*data*)
Add blank elements to fill out in ElementTree.

Add element to data ElementTree for timestamp, redcap form name, eventName, formDateField, and formCompletedFieldName.

Parameters **data** – the input ElementTree from the parsed raw XML file.

redi.redi.**compress_data_using_study_form_date**(*data*)
This function is removing duplicate results which were recorded on same date but different times. Warnings:

- we assume that the passed ElementTree is sorted
- **we skip all "Canceled" results but we want to keep at least one** so when all results are canceled we keep the first one
- the passed object is altered

@see #get_key_date() @see #get_key_timestamp() @see #sort_element_tree()

data: the ElementTree object that needs to be *compressed* return: none

redi.redi.**configure_logging**(*data_folder*, *verbose=False*, *when='D'*, *interval=1*, *backup_count=31*)
Configures the Logger

redi.redi.**connect_to_redcap**(*email_settings*, *redcap_settings*, *dry_run=False*)

redi.redi.**convert_component_id_to_loinc_code**(*data*, *component_to_loinc_code_xml_tree*)
This function converts COMPONENT_ID in raw data to loinc_code based on the mapping provided in the xml file

Parameters

- **data** – Raw data xml tree
- **component_to_loinc_code_xml_tree** – COMPONENT_ID to loinc_code mapping xml file tree.

redi.redi.**convert_none_type_object_to_empty_string**(*my_object*)
replace noneType objects with an empty string. Else return the object.

redi.redi.**copy_data_to_person_form_event_tree**(*raw_data_tree*, *person_form_event_tree*, *form_events_tree*)
This function copies data from the raw_data_tree to the person_form_event_tree

Parameters

- **raw_data_tree** – This parameter holds raw data tree
- **person_form_event_tree** – This parameter holds person form event tree
- **form_events_tree** – This parameter holds form events tree

redi.redi.**create_empty_event_tree_for_study**(*raw_data_tree*, *all_form_events_tree*)
This function uses raw_data_tree and all_form_events_tree and creates a person_form_event_tree for study

Parameters

- **raw_data_tree** – This parameter holds raw data tree
- **all_form_events_tree** – This parameter holds all form events tree

redi.redi.**create_empty_events_for_one_subject**(*form_events_tree*, *translation_table_tree*)

redi.redi.**create_empty_events_for_one_subject_helper**(*form_events_file*, *translation_table_file*)
This function creates new copies of the form_events_tree and translation_table_tree and calls create_empty_events_for_one_subject :param form_events_file: This parameter holds the path of form_events file :param translation_table_file: This parameter holds the path of translation_table file

redi.redi.**get_db_path**(*batch_info_database*, *database_path*)

redi.redi.**get_email_settings**(*settings*)
Helper function for grouping email-related properties

redi.redi.**get_key_date**(*ele*)
Helper function for #compress_data_using_study_form_date()

elem: lxml.etree._Element object for which we build a key returns the corresponding quadruple (study_id, form_name, loinc_code, date)

redi.redi.**get_key_timestamp**(*ele*)
Helper function for #sort_element_tree() @see #compress_data_using_study_form_date()

elem: lxml.etree._Element object for which we build a key returns the corresponding quadruple (study_id, form_name, timestamp)

redi.redi.**get_redcap_settings**(*settings*)
Helper function for grouping redcap connection properties

redi.redi.**load_preproc**(*preprocessors*, *root='./'*)
Copied and modified version of load_rules function. TODO: fix load_rules and load_prerules for better parallelism

redi.redi.**load_rules**(*rules*, *root='./'*)
Load custom post-processing rules.

Rules should be added to the configuration file under a property called "rules", which has key-value pairs mapping a unique rule name to a Python file. Each Python file intended to be used as a rules file should have a run_rules() function which takes one argument.

Example config.json: { "rules": { "my_rules": "rules/my_rules.py" } }

Example rules file:

def run_rules(data): pass

redi.redi.**main**()
Data processing steps:

- parse raw XML to ElementTree: "data"
- call read-in function to load xml into ElementTree
- parse formEvents.xml to ElementTree
- call read-in function to load xml into ElementTree
- parse translationTable.xml to ElementTree
- call read-in function to load xml into ElementTree
- **add element to data ElementTree for timestamp, redcap form name,** eventName, formDateField, and formCompletedFieldName
- write out ElementTree as an XML file
- call read-in function to load xml into ElementTree
- update timestamp using collection_date and collection_time
- write redcapForm name to data ElementTree by a lookup of component ID in translationTable.xml
- sort data by: study_id, form name, then timestamp, ascending order
- write formDateField to data ElementTree via lookup of formName in formEvents.xml
- write formCompletedFieldName to data ElementTree via lookup of formName in formEvents.xml

•write eventName to data ElementTree via lookup of formName in formEvents.xml

Example: <formName value="chemistry">

<event name="1_arm_1" />

</formName>

•write the Final ElementTree to EAV

redi.redi.**parse_form_events**(*form_events_file*)
Parse the form_events file into an ElementTree

Parameters **form_events_file** – the name of the input file (from the json configuration)

Returns ElementTree

redi.redi.**parse_raw_xml**(*raw_xml_file*)
Generate an ElementTree from a raw XML file.

Parameters **raw_xml_file** – the input file.

Returns parsed XML data

redi.redi.**parse_translation_table**(*translation_table_file*)
Parse the translationTable.xml into an ElementTree

Parameters **translation_table_file** – the name of the input file

Returns ElementTree

redi.redi.**read_config**(*config_file*, *configuration_directory*, *file_list*)
Check if files mentioned in configuration files exist

redi.redi.**replace_fields_in_raw_xml**(*data*, *fields_to_replace_xml*)
replace_fields_in_raw_xml: This function renames all fields which need renaming.Fields which need renaming are read from the xml file. Parameters:

data: Raw data xml tree fields_to_replace_xml: Path to xml file which has list of fields which need renaming.

redi.redi.**research_id_to_redcap_id_converter**(*data*, *redcap_client*, *research_id_to_redcap_id*, *configuration_directory*)

This function converts the research_id to redcap_id

1. prepare a dictionary with [key, value] –> [study_id, redcap_id]
2. replace the element tree study_id with the new redcap_id's for each bad id, log it as warn.

Example of xml fragment produced:

<subject lab_id="999-0001"> <NAME>HEMOGLOBIN</NAME> <loinc_code>1534435</loinc_code> <RESULT>1234</RESULT>

... <STUDY_ID>1</STUDY_ID> <!– originally this was "999-0001" –>

</subject>

Note: The next function which reads the "data" tree is #create_empty_event_tree_for_study()

redi.redi.**run_preproc**(*preprocessors*, *settings*)

redi.redi.**run_rules**(*rules*, *person_form_event_tree_with_data*)

redi.redi.**setStat**(*event*, *translation_table_dict*, *translation_table_status_field_text_list*)
Ruchi Vivek Desai, May 13 2014 to assist the updateStatusFieldValueInPersonFormEventTree function

redi.redi.**set_status_for**(*field_name*, *event*, *translation_table_dict*)
Ruchi

redi.redi.**sort_element_tree**(*data*, *data_folder*)
Sort element tree based on three given indices. @see #update_time_stamp()

Keyword argument: data sorting is based on study_id, form name, then timestamp, ascending order

redi.redi.**updateStatusFieldValueInPersonFormEventTree**(*person_form_event_tree*, *translational_table_tree*)
Ruchi Vivek Desai, May 13 2014 This function updates the status field value with either NOT_DONE (value in the translation table) or empty string based on certain conditions

redi.redi.**update_data_from_lookup**(*data*, *element_to_set_in_data*, *index_element_in_data*, *lookup_data*, *element_to_find_in_lookup_data*, *index_element_in_lookup_data*, *value_in_lookup_data*, *undefined*)

Update a single field in an element tree based on a lookup in another element tree

Parameters

- **data** – an element tree with a field that needs to be set
- **element_to_set_in_data** – element that will be set
- **index_element_in_data** – element in data that wil be looked up in lookup table where value of element to be set wil be found
- **lookup_data** – an element tree that contains, the lookup data
- **element_to_find_in_lookup_data** – parameter for the initial findall in the lookup data
- **index_element_in_lookup_data** – the element in the lookup data that will be the key in the lookup table
- **value_in_lookup_data** – element in the lookup data that provides the value in the lookup table
- **undefined** – a string to be returned for all failed lookups in the lookup table

redi.redi.**update_event_name**(*data*, *lookup_data*, *undefined*)
function to update eventName to data ElementTree via lookup of formName in formEvents ElementTree

redi.redi.**update_form_imported_field**(*data*, *lookup_data*, *undefined*)
Update the formImportedFieldName value for all subjects

redi.redi.**update_formcompletedfieldname**(*data*, *lookup_data*, *undefined*)
function to update formCompletedFieldName in data ElementTree via lookup of formName in formEvents ElementTree

redi.redi.**update_formdatefield**(*data*, *form_events_tree*)
Write formDateField to data ElementTree via lookup of formName in form_events_tree ElementTree

redi.redi.**update_recap_form_status**(*data*, *lookup_data*, *undefined*)
Update the redcapStatusFieldName value to all subjects

redi.redi.**update_redcap_field_name_value_and_units**(*data*, *lookup_data*, *undefined*)
function to update redcapFieldNameValue and redcapFieldNameUnits in data ElementTree via lookup of redcapFieldNameValue and redcapFieldNameUnits in translation table tree

redi.redi.**update_redcap_form**(*data*, *lookup_data*, *undefined*)
Lookup component ID in translationTable to get the redcapFormName. Write the redcapForm name to data If component lookup fails, sets formName to undefinedForm

redi.redi.**update_time_stamp**(*data*, *input_date_format*, *output_date_format*)
Update timestamp using input and output data formats. Warnings:

- we modify the data ElementTree
- we affect the sorting order of data elements @see #sort_element_tree()

redi.redi.**validate_xml_file_and_extract_data**(*xmlfilename*, *xsdfilename*)
This function is responsible for validating xml file against an xsd and to extract data from xml if validation succeeds

Parameters

- **xmlfilename** – This parameter holds the path to the xml file
- **xsdfilename** – This parameter holds the path to the xsd file

redi.redi.**verify_and_correct_collection_date**(*data*, *input_date_format*)

redi.redi.**write_element_tree_to_file**(*element_tree*, *file_name*)
Write an ElementTree to a file whose name is provided as an argument

redi.redi_lib module

redi.report module

class redi.report.**ReportCourier**
Bases: object

deliver(*report*)

class redi.report.**ReportCreator**(*report_file_path*, *project_name*, *redcap_uri*, *sort_by_lab_id*, *writer*)
Bases: object

create_report(*report_data*, *alert_summary*, *collection_date_summary_dict*, *duration_dict*)

format_seconds_as_string(*seconds*)

Convert seconds to a friendly strings 3662 ==> '01:01:02' 89662 ==> '1 day, 0:54:22'

seconds [integer] The number of seconds to be converted

get_time_diff(*end*, *start*)
Get time difference in seconds from the two dates Parameters ———- end : string

The end timestamp

start [string] The start timestamp

class redi.report.**ReportEmailSender**(*settings*, *logger*)
Bases: *redi.report.ReportCourier*

deliver(*report*)
Deliver summary report as an email

:email_settings dictinary with email parameters :html the actual report content

class redi.report.**ReportFileWriter**(*output_file*, *logger*)
Bases: *redi.report.ReportCourier*

deliver(*report*)
Deliver the summary report by writing it to a file or logging it to the console if writing the file fails

:html_report_path the path where the report will be stored :html the actual report content

redi.report.**gen_ele**(*ele_name*, *ele_text*)
Create an xml element with given name and content

redi.report.**gen_subele**(*parent*, *subele_name*, *subele_text*)

redi.report.**updateReportAlerts**(*root*, *alert_summary*)

redi.report.**updateReportErrors**(*root*, *errors*)

redi.report.**updateReportHeader**(*root*, *report_parameters*)
Update the passed *root* element tree with date, project name and url

redi.report.**updateReportSummary**(*root*, *report_data*)

redi.report.**updateSubjectDetails**(*root*, *subject_details*)
Helper method for #create_summary_report() Adds subject information to the xml tree which is later formated by *redi/utils/report.xsl* into the html *table#subject_details"*

redi.report.**updateSummaryOfSpecimenTakenTimes**(*root*, *collection_date_summary_dict*)

redi.upload module

Functions related to uploading data to REDCap

redi.upload.**create_import_data_json**(*import_data_dict*, *event_tree*)
Convert data from event_tree to json format.

@TODO: evaluate performance @see the caller {@link #redi.upload.generate_output()}

Param import_data_dict: holds the event tree data

Param event_tree: holds the event tree data

Return type dict

:return the json version of the xml data

redi.upload.**create_redcap_records**(*import_data*)
Creates REDCap records from RED-I's form data, AKA import data.

REDCap API only accepts records for importing. Records are differentiated by their unique record ID, unless the REDCap Project is a Longitudinal study. In that case, they are differentiated by a combination of record ID and an event.

Since RED-I views the world in terms of forms, we have to project our form-centric view into REDCap's record-centric world. This is done by combining all form data with the same Subject ID and Event Name into the same record.

Parameters **import_data** – iterable of 4-tuples: (study_id_key, form_name, event_name, json_data_dict)

Returns iterable of REDCap records ready for upload

redi.upload.**generate_output**(*person_tree*, *redcap_client*, *rate_limit*, *sent_events*, *max_retry_count*, *skip_blanks=False*, *bulk_send_blanks=False*)
Note: This function communicates with the redcap application. Steps:

•loop for each person/form/event element

•generate a csv fragment *using create_eav_output*

•send csv fragment to REDCap using *send_eav_data_to_redcap*

@see the caller {@link #redi.redi._run()}

Return type dictionary

Returns the report_data which is passed to the report rendering function

redi.upload.**handle_errors_in_redcap_xml_response**(*study_id*, *redcap_err*, *report_data*)

Checks for any errors in the redcap response and update report data if there are any errors.

redcap_err: RedcapError object report_data: dictionary to which we store error details

Module contents

r

redi, 42
redi.batch, 33
redi.form, 34
redi.redi, 35
redi.report, 40
redi.upload, 41
redi.utils, 33
redi.utils.csv2xml, 31
redi.utils.GetEmrData, 30
redi.utils.rawxml, 31
redi.utils.redcapClient, 31
redi.utils.redi_email, 32
redi.utils.SimpleConfigParser, 30
redi.utils.throttle, 33

A

add_attachment() (in module redi.utils.redi_email), 32
add_batch_entry() (in module redi.batch), 33
add_elements_to_tree() (in module redi.redi), 35

B

BATCH_STATUS_COMPLETED (in module redi.batch), 33

C

check_input_file() (in module redi.batch), 34
check_parameters() (redi.utils.SimpleConfigParser.SimpleConfigParser method), 30
cleanup() (in module redi.utils.GetEmrData), 30
cleanup_callback() (in module redi.utils.csv2xml), 31
clear_value() (redi.form.Field method), 34
compress_data_using_study_form_date() (in module redi.redi), 36
ConfigurationError, 30
configure_logging() (in module redi.redi), 36
connect_to_redcap() (in module redi.redi), 36
convert_component_id_to_loinc_code() (in module redi.redi), 36
convert_none_type_object_to_empty_string() (in module redi.redi), 36
copy_data_to_person_form_event_tree() (in module redi.redi), 36
create_empty_event_tree_for_study() (in module redi.redi), 36
create_empty_events_for_one_subject() (in module redi.redi), 36
create_empty_events_for_one_subject_helper() (in module redi.redi), 36
create_empty_md5_database() (in module redi.batch), 34
create_empty_table() (in module redi.batch), 34
create_import_data_json() (in module redi.upload), 41
create_redcap_records() (in module redi.upload), 41
create_report() (redi.report.ReportCreator method), 40

D

data_preprocessing() (in module redi.utils.GetEmrData), 30
delete() (redi.redi.PersonFormEventsRepository method), 35
deliver() (redi.report.ReportCourier method), 40
deliver() (redi.report.ReportEmailSender method), 40
deliver() (redi.report.ReportFileWriter method), 41
dict_factory() (in module redi.batch), 34
download_files() (in module redi.utils.GetEmrData), 30

E

EmrFileAccessDetails (class in redi.utils.GetEmrData), 30
Event (class in redi.form), 34
events() (redi.form.Form method), 34

F

fetch() (redi.redi.PersonFormEventsRepository method), 35
Field (class in redi.form), 34
field() (redi.form.Event method), 34
field_subst_factory() (in module redi.utils.csv2xml), 31
fields() (redi.form.Event method), 34
Form (class in redi.form), 34
form_name (redi.form.Event attribute), 34
format_seconds_as_string() (redi.report.ReportCreator method), 40

G

gen_ele() (in module redi.report), 41
gen_subele() (in module redi.report), 41
generate_output() (in module redi.upload), 41
generate_xml() (in module redi.utils.GetEmrData), 30
get_batch_by_id() (in module redi.batch), 34
get_creation_time() (redi.utils.rawxml.RawXml method), 31
get_data_from_redcap() (redi.utils.redcapClient.RedcapClient method), 32
get_days_since_today() (in module redi.batch), 34
get_db_friendly_date() (in module redi.batch), 34
get_db_friendly_date_time() (in module redi.batch), 34

get_db_path() (in module redi.redi), 36
get_email_settings() (in module redi.redi), 36
get_emr_data() (in module redi.utils.GetEmrData), 30
get_info() (redi.utils.rawxml.RawXml method), 31
get_key_date() (in module redi.redi), 37
get_key_timestamp() (in module redi.redi), 37
get_last_batch() (in module redi.batch), 34
get_last_modified_time() (redi.utils.rawxml.RawXml method), 31
get_md5_input_file() (in module redi.batch), 34
get_project() (redi.utils.rawxml.RawXml method), 31
get_redcap_settings() (in module redi.redi), 37
get_time_diff() (redi.report.ReportCreator method), 40
getoption() (redi.utils.SimpleConfigParser.SimpleConfigParser method), 30
getoptionslist() (redi.utils.SimpleConfigParser.SimpleConfigParser method), 30

H

handle_errors_in_redcap_xml_response() (in module redi.upload), 42
hasoption() (redi.utils.SimpleConfigParser.SimpleConfigParser method), 30

I

is_empty() (redi.form.Event method), 34

L

load_preproc() (in module redi.redi), 37
load_rules() (in module redi.redi), 37

M

main() (in module redi.redi), 37
mark_sent() (redi.redi.SentEvents method), 35

N

name (redi.form.Event attribute), 34
name (redi.form.Field attribute), 34

O

openio() (in module redi.utils.csv2xml), 31

P

parse_cmdline() (in module redi.utils.csv2xml), 31
parse_form_events() (in module redi.redi), 38
parse_raw_xml() (in module redi.redi), 38
parse_translation_table() (in module redi.redi), 38
PersonFormEventsRepository (class in redi.redi), 35
printxml() (in module redi.batch), 34

R

RawXml (class in redi.utils.rawxml), 31
read() (redi.utils.SimpleConfigParser.SimpleConfigParser method), 31
read_config() (in module redi.redi), 38
RedcapClient (class in redi.utils.redcapClient), 31
redi (module), 42
redi.batch (module), 33
redi.form (module), 34
redi.redi (module), 35
redi.report (module), 40
redi.upload (module), 41
redi.utils (module), 33
redi.utils.csv2xml (module), 31
redi.utils.GetEmrData (module), 30
redi.utils.rawxml (module), 31
redi.utils.redcapClient (module), 31
redi.utils.redi_email (module), 32
redi.utils.SimpleConfigParser (module), 30
redi.utils.throttle (module), 33
replace() (in module redi.utils.csv2xml), 31
replace_fields_in_raw_xml() (in module redi.redi), 38
ReportCourier (class in redi.report), 40
ReportCreator (class in redi.report), 40
ReportEmailSender (class in redi.report), 40
ReportFileWriter (class in redi.report), 40
research_id_to_redcap_id_converter() (in module redi.redi), 38
run_preproc() (in module redi.redi), 38
run_rules() (in module redi.redi), 38

S

send_data_to_redcap() (redi.utils.redcapClient.RedcapClient method), 32
send_email() (in module redi.utils.redi_email), 32
send_email_data_import_completed() (in module redi.utils.redi_email), 32
send_email_input_data_unchanged() (in module redi.utils.redi_email), 33
send_email_redcap_connection_error() (in module redi.utils.redi_email), 33
SentEvents (class in redi.redi), 35
set_attributes() (redi.utils.SimpleConfigParser.SimpleConfigParser method), 31
set_status_for() (in module redi.redi), 39
setStat() (in module redi.redi), 38
SimpleConfigParser (class in redi.utils.SimpleConfigParser), 30
sort_element_tree() (in module redi.redi), 39
store() (redi.redi.PersonFormEventsRepository method), 35
study_id (redi.form.Event attribute), 34

T

Throttle (class in redi.utils.throttle), 33
to_bool() (in module redi.utils.SimpleConfigParser), 31

U

update_batch_entry() (in module redi.batch), 34
update_data_from_lookup() (in module redi.redi), 39
update_event_name() (in module redi.redi), 39
update_form_imported_field() (in module redi.redi), 39
update_formcompletedfieldname() (in module redi.redi), 39
update_formdatefield() (in module redi.redi), 39
update_recap_form_status() (in module redi.redi), 39
update_redcap_field_name_value_and_units() (in module redi.redi), 39
update_redcap_form() (in module redi.redi), 39
update_time_stamp() (in module redi.redi), 40
updateReportAlerts() (in module redi.report), 41
updateReportErrors() (in module redi.report), 41
updateReportHeader() (in module redi.report), 41
updateReportSummary() (in module redi.report), 41
updateStatusFieldValueInPersonFormEventTree() (in module redi.redi), 39
updateSubjectDetails() (in module redi.report), 41
updateSummaryOfSpecimenTakenTimes() (in module redi.report), 41

V

validate_xml_file_and_extract_data() (in module redi.redi), 40
value (redi.form.Field attribute), 34
verify_and_correct_collection_date() (in module redi.redi), 40

W

was_sent() (redi.redi.SentEvents method), 35
write() (redi.utils.csv2xml.Writer method), 31
write_element_tree_to_file() (in module redi.redi), 40
write_field() (redi.utils.csv2xml.Writer method), 31
write_file() (redi.utils.csv2xml.Writer method), 31
write_record() (redi.utils.csv2xml.Writer method), 31
Writer (class in redi.utils.csv2xml), 31