SESUG 2022 Paper 184

REDCap®: Your SAS® Friend For Electronic Health Record Manual Abstraction

Brooke Ellen Delgoffe, Marshfield Clinic Health System, Marshfield, WI

Steffani Roush, Marshfield Clinic Health System, Marshfield, WI

ABSTRACT

This paper and presentation will demonstrate the utility of REDCap® during abstraction of electronic health record (EHR) data. By utilizing REDCap, we can structure our abstraction forms to minimize entry errors and save time; all in a way that works well with SAS® and allows electronic data transfer using a secure method. REDCap allows SAS to query against it using an API for easy data exchange. This paper will also feature an advanced export macro that automatically downloads the data dictionary and creates a labeled SAS data set (complete with labels and a format library based on the REDCap data dictionary). Special data concerns when using REDCap for housing PHI and methods for interfacing with an EHR will be reviewed with solutions presented.

INTRODUCTION

The healthcare field has long had its challenges with making data available electronically. Even with the advent of patient portals, mobile applications, and web-based care there are still impediments with the utility of the data coming out of these electronic records. As the name suggests, the Electronic Health Record has a primary use in providing medical care. In many cases, the inability to get at information electronically is specifically to protect against access to EHR data for non-patient care reasons or just not a priority to care teams.

While not an EHR's primary use, routine processing of data exists for quality improvement, reporting, and research. In a multi-faceted healthcare system, like Marshfield Clinic Health System, these direct sources of EHR data are often paired with survey data, dental data, and outside data (like registries) before being ready for research. When all this data comes together, you may think that a full picture must be available, but often times it takes a skilled team of abstractors to complete the process. Manual review of EHR records to pull out data not available electronically, perform quality assurance, and reconcile data between sources is often necessary.

Data used for research must also undergo approval by an internal review board (IRB) and meet strict minimum necessary guidelines in regards to viewing and utilizing protected health information. In prospective studies, enrolling participants often includes a consent process and subsequent survey data collection. Generally, specific tools change for each study performed and conclude with an archival process.

REDCap® defines it's self as a "secure web application for building and managing online surveys and databases. While REDCap can be used to collect virtually any type of data in any environment (including compliance with 21 CFR Part 11, FISMA, HIPAA, and GDPR), it is specifically geared to support online and offline data capture for research studies and operations" (Harris, et al., 2009). REDCap is a solution for many research data needs including secure data storage, survey data collection, and secure correspondence between patients and researchers. This paper will also show you how much of a friend it is to SAS users.***The following paper contains screenshots and code that appear to contain patient data. All information presented belongs to test patients. Similarly, readers cannot actually access the REDCap used, since it exists in an offline testing area.***

KEY BENEFITS OF REDCAP

When exploring solutions for collecting any type of data there are a couple priorities that stand out:

1. Affordability & Storage
2. Accessibility & Security
3. Ease of Use
4. Data Cleanliness / Structure

In all of these regards, REDCap becomes a good solution. The REDCap software itself is available at no cost to non-profit organizations, so its cost depends on the size of storage and technical support (which must come from within your organization's own internal IT department) required to support your needs.

REDCap Databases are available via web browser on both an intranet (internal) and internet (external) basis. They can be set up to follow network credentialing practices and come with a variety of different "user rights". The servers they sit on and store data into are not accessible to front-end users and can take on any size or location desired without the need for individual user connection set-ups. By using a credentialed online access point, the data remain both accessible and secure. Exports of the data are monitored and controlled. Changes to the data are tracked and logged for easy debugging and sleuthing.

The structure of REDCap allows for two collection methods: surveys and forms. Forms require users to log in and complete a sequence of fields, allowing for return to records and detailed change tracking. Surveys allow for structured collection of data without the requirement for logging in, offering a method for collecting de-identified data but still with an option for returning later. Whether using the survey mode or forms mode, the point and click user interface (*See Figure 2. Example REDCap* User Interface) supports high usability. Training video links, syntax checkers, and on screen tutorials appear to users whom log in. Form tips and footnotes under fields appear to those entering data (in both form and survey modes).The abundance of reference material at hand makes it easy to learn and use even for novel data guardians.

REDCap has many methods for creating clean, structured, data. Options for field type (radio, checkbox, text box) come with options for defining formatted and unformatted values (0=No, 1=Yes) and/or field validation mechanisms (dates, integers, email, etc.). "Branching Logic" is available to conditionally hide/show fields to avoid erroneous entries and limit viewable fields to only those necessary. On top of that, you can define and apply "data quality rules" (See Figure 3) that can execute in real time (ex. Require Q2 when Q1 is a certain value, Option 4 cannot be selected with Option 5 on Q3) and will present messages to the user when broken.

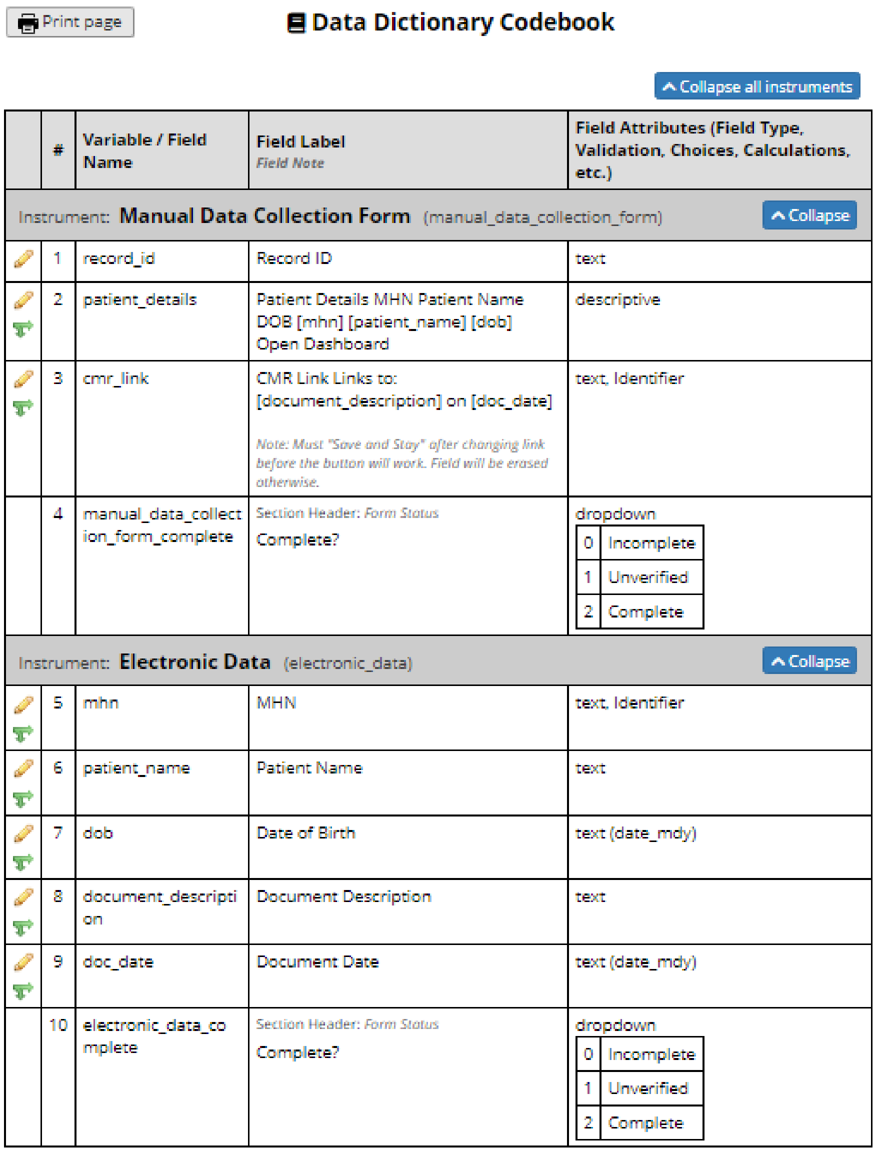
Best of all, this data is very SAS compatible. When using the export functions within REDCap there are a variety of output destinations available. Amongst them is an option to produce a "SAS dataset", in which it actually creates a SAS program that defines a format library, labels, and a DATA step for importing the data. It then gives you an unformatted csv version of the data that is read in by the program. While it may seem like an extra step to getting a dataset, this method actually makes it very easy to make changes to formats or variable names and requires little knowledge of REDCap or SAS. For those whom prefer to stay within SAS, there is an API which will allow you to query against the REDCap database. By using PROC HTTP (See Exporting Data from REDCap using the API) we can get the same CSV export we would get manually exporting from REDCap. Unfortunately, this method does not come with an automatic import to SAS, but the macro provided in this paper will (*See Extra Functionality to Consider).* Whether using the export wizard to write SAS code for importing the csv it creates for you or doing so yourself using the API, REDCap is made to work with SAS.

THE BASICS OF REDCAP

Just like other databases you may have encountered, REDCap databases contain fields, labels, attributes, and relationships. In the background, data generally follows a rectangular structure (rows and columns). There are many customizations available and several common features you might be familiar with: hyperlinking, dashboards, reporting, logging, alerts, and data quality constraints.

In its most basic form, REDCap defines fields that can be collected in a user interface via a form or survey. REDCap calls each grouping of fields an instrument. Consider the example "Codebook" in Figure 1 below.

Figure 1. Example REDCap Code Book



Data Dictionary Codebook

	#	Variable / Field Name	Field Label *Field Note*	Field Attributes (Field Type, Validation, Choices, Calculations, etc.)
Instrument: **Manual Data Collection Form** (manual_data_collection_form)				
	1	record_id	Record ID	text
	2	patient_details	Patient Details MHN Patient Name DOB [mhn] [patient_name] [dob] Open Dashboard	descriptive
	3	cmr_link	CMR Link Links to: [document_description] on [doc_date] *Note: Must "Save and Stay" after changing link before the button will work. Field will be erased otherwise.*	text, Identifier
	4	manual_data_collection_form_complete	Section Header: *Form Status* Complete?	dropdown 0 Incomplete; 1 Unverified; 2 Complete
Instrument: **Electronic Data** (electronic_data)				
	5	mhn	MHN	text, Identifier
	6	patient_name	Patient Name	text
	7	dob	Date of Birth	text (date_mdy)
	8	document_description	Document Description	text
	9	doc_date	Document Date	text (date_mdy)
	10	electronic_data_complete	Section Header: *Form Status* Complete?	dropdown 0 Incomplete; 1 Unverified; 2 Complete

In this test project, there are two instruments: one for manual abstraction and one for electronic abstraction. Each contains defined variables and an automatic variable ending in "_complete" that indicates the status of each instrument.

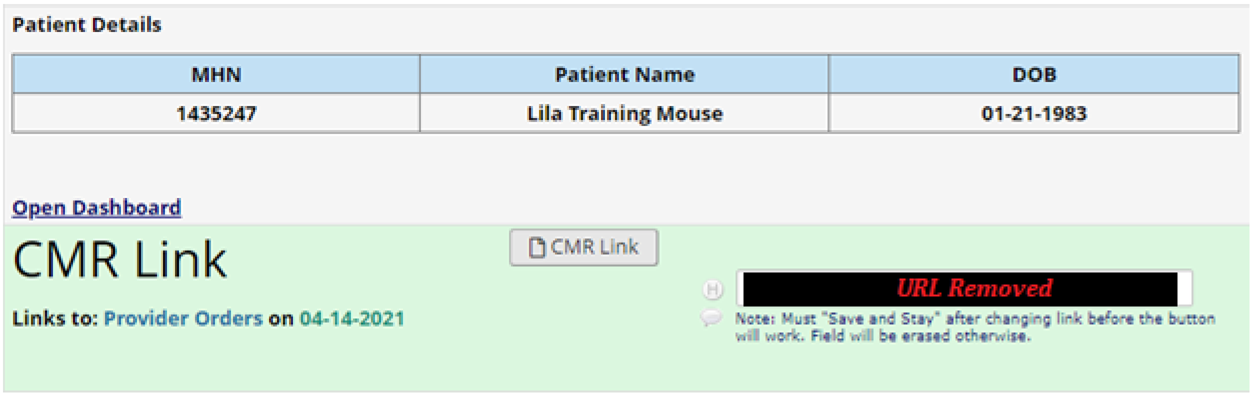
HTML FOR DATA ORGANIZATION

Note that in the field "patient_details" (Figure 1), HTML is used (not noted in the Codebook) to create a table and embed the fields of interest so that they appear as they do in Figure 2 below. This data can be pre-loaded into the project so that it is available to the abstractor when they open each record.

A customized view that organizes important information can be crucial to avoiding patient mismatch errors and provide a location for important directions for how and what should be abstracted.

They can also add comments about the values they entered by clicking on the comment bubble next to the input field (to left of "URL Removed" below).

Figure 2. Example REDCap User Interface



Patient Details

MHN	Patient Name	DOB
1435247	Lila Training Mouse	01-21-1983

Open Dashboard

CMR Link

Links to: Provider Orders on 04-14-2021

Note: Must "Save and Stay" after changing link before the button will work. Field will be erased otherwise.

In this case, we are providing information in patient_details ahead of the fields for manual abstraction by uploading data to REDCap.

Other good things to include in these sections are:

- Warnings or special instructions for the form (underneath fields)
- Links to documentation
- Instructions for where to find information
- Contact information for questions about values

By providing your abstractors a set of clear instructions and a place for comments, you will be able to appropriately document your abstraction and save them time spent emailing or looking up instructions. When one person asks a question, we are populating the answer for everyone to see.

COMBINED MEDICAL RECORD LINK (CMR LINK)

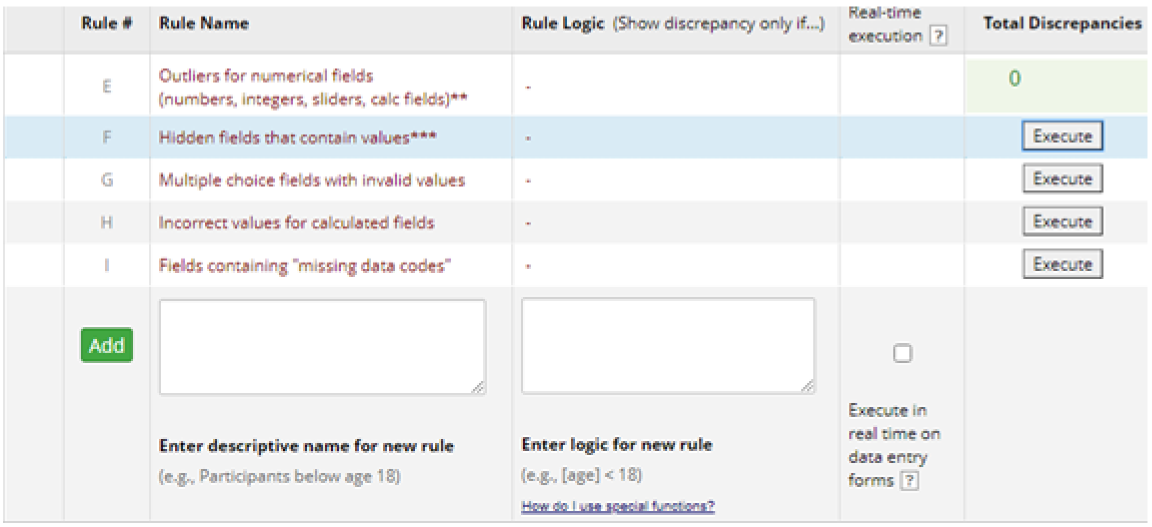
In Figure 2, there is a button marked "CMR Link". This is an "External Module" (custom functionality) developed for our abstractors. It allows them to have our medical record opened to a specific location. Utilizing the existing link structure, we provide the button a URL that points to a specific document. By providing this information ahead of abstraction, we greatly reduce the time spent searching the EHR for the correct document/information. In addition to saving abstractors search time, we are also greatly diminishing the amount of health data that needs to be reviewed; keeping the minimum necessary consideration at the forefront of our minds.

In addition to the document link, we are also able to provide metadata about that link to the abstractor: "Provider Order on 04-14-2021"; allowing abstractors to confirm they are viewing the correct document. If it is the wrong one, they can update the URL to point to the correct document once they have located it. By doing so, they also reduce the time needed to go back and abstract additional data.

DATA QUALITY RULES

The key to quality research is quality data. REDCap is here to assist you by setting rules for values, which no spreadsheet can compete with. For each project, a user can define a series of data quality rules. Many of the desired checks are pre-built into REDCap. Each can be executed by pressing the execute button; with a resulting value showing in its place at completion. Rules can also be set to execute in real-time.

Figure 3. REDCap Default Data Quality Rules



Rule #	Rule Name	Rule Logic (Show discrepancy only if...)	Real-time execution ?	Total Discrepancies
E	Outliers for numerical fields (numbers, integers, sliders, calc fields)**	-		0
F	Hidden fields that contain values***	-		Execute
G	Multiple choice fields with invalid values	-		Execute
H	Incorrect values for calculated fields	-		Execute
I	Fields containing "missing data codes"	-		Execute
Add	Enter descriptive name for new rule (e.g., Participants below age 18)	Enter logic for new rule (e.g., [age] < 18) How do I use special functions?	Execute in real time on data entry forms ?	

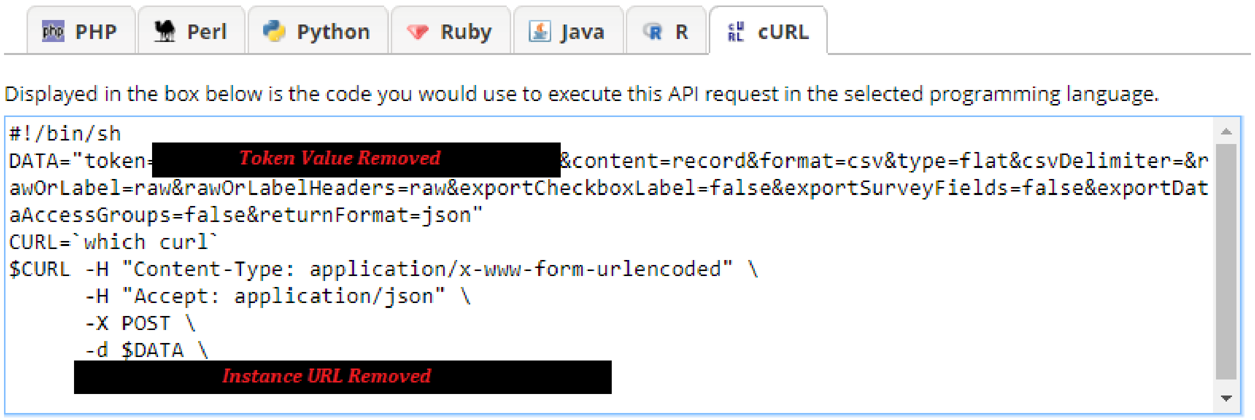
THE API PLAYGROUND

Once you have quality data abstracted into a secure location, the next step is exporting for analysis.

REDCap was built to "talk nice" with a variety of programming languages and conveniently offers a way to experiment with the syntax needed to achieve a variety of import/export data needs: the API Playground.

Found in the left side panel, API and API Playground are your tools for finding the solution to your import/export needs. In this area, you can select which function you are hoping to achieve (for example, "Export Records") and a variety of other options (like .csv format). Once selected, the Playground presents the code needed in a variety of languages, as seen in Figure 4. Unfortunately, it does not give you the exact SAS code needed, but it does give you a usable HTTP string for feeding into PROC HTTP under cURL.

Figure 4. Example of API Playground Auto-Code



Displayed in the box below is the code you would use to execute this API request in the selected programming language.

```
#!/bin/sh
DATA="token=[redacted]&content=record&format=csv&type=flat&csvDelimiter=&r
awOrLabel=raw&rawOrLabelHeaders=raw&exportCheckboxLabel=false&exportSurveyFields=false&exportDat
aAccessGroups=false&returnFormat=json"
CURL=`which curl`
$CURL -H "Content-Type: application/x-www-form-urlencoded" \
      -H "Accept: application/json" \
      -X POST \
      -d $DATA \
     [redacted]
```

Notice there are two items removed from Figure 4: token and instance URL. You will need to request a "token" value in the API area before you can use it. Once obtained, this will be what PROC HTTP uses to gain access to your data. ***Treat this like a username and password; anyone with your token can execute your code and acquire its results.*** Use an external file stored in a secured directory and/or macro variables to house your token securely to avoid misuse (See Appendix 2). The "Instance URL" is the web address to your instance of REDCap that can be hosted internally (accessible by those connected to your intranet) or externally (accessible by anyone with either a code, login, or public survey link).

Once you acquire the API token, the API is now ready for use by SAS.

EXPORTING DATA FROM REDCAP USING THE API

In Appendix 1, we give the full code for exporting data from REDCap as a portable macro definition. A portion of this code originated from SAS Sample 26065 (SAS Institute, 2006) and SAS Sample 24717 (SAS Institute, 2020). In order to use this macro, you will need to insert the URL(s) for your instances of REDCap *(See Required Edit 1and Required Edit 2).* The following sections explain the exporting of the data dictionary in more detail as an example and introduces important options/set-up.

SETTING OPTIONS

The first line of code in the macro sets a few options:

```
options nomprint nosymbolgen nomlogic;
```

These options prevent printing of sensitive information stored in macro variables (like your token value) to the log or from being resolved anywhere that might reveal them. If you are using SAS Enterprise Guide®, the values of these macro variables will still be viewable in the "SAS Macro Variable Viewer" and these will still resolve in put statements. Be careful to remove areas where you resolve these variables before sharing with others.

IDENTIFYING FILES

Before pulling the data down, we want to identify files to house the data temporarily and any files that contain HTTP call information (non-default call strings). We can accomplish this by using filename statements and specifying a network/directory location. The syntax looks like:

```
filename my_dict "&BaseDIR.\Data Dictionary.csv";
```

Notice we are using a standard naming convention for the data dictionary to minimize the amount of information being requesting, but utilizing a macro variable containing the base directory to be sure all files end up in the same location. These statements create the files they name and are ready to be acted on.

PROC HTTP: THE COMMON LANGUAGE

In order for SAS to obtain data from REDCap, they have to speak a common language: HTTP. We will be using PROC HTTP to send and receive data using the following syntax:

```
*** PROC HTTP call for dictionary. ***;
proc http
    in=&dict_string.
    out= my_dict
    url = "&instance_url."
    method="POST"
;
run;
```

In this case, two calls will be required: one for the data dictionary and one for the data itself. In both circumstances, the IN= call will either be a filename reference to an external file containing the HTTP string or an explicit string enclosed in double quotes. The HTTP string is what we saw in the API playground (Figure 4) in double quotes after the DATA=. In the macro, the use of macro variables will control what appears in different parts of the string:

```
token=&mytoken.%NRStr(&content=metadata&type=flat&format=csv&)
```

Masking the HTTP ampersands using %NRSTR() will assure that only the true macro variables are resolved. The value `&mytoken.` should resolve to your token value, but the remaining ampersands need to be passed to the API. After the token, we tell the API what we want from it:

1) content=metadata : Data Dictionary information
2) type=flat : rectangular data (columns/rows)
3) format=csv : file destination is CSV

In PROC HTTP, the next argument is OUT= which will tell SAS where to put the information it receives from the API. Unlike the IN= statement, OUT values can only be a file reference. The file format must also match what is given in the HTTP string (file reference can only be to a .csv if format=csv in HTTP string).

The URL and METHOD values should not change based on which REDCap project you are pulling data from. As mentioned before, you will need to enter the URL that is specific to your instance(s) of REDCap. For more information on other options in PROC HTTP, review the *Base SAS® Procedures Guide*.

EXTRA FUNCTIONALITY TO CONSIDER

In addition to the basics of importing and exporting data from REDCap using PROC HTTP calls, the next few steps are also routine:

1) Import the CSV data to a SAS data set
2) Apply data dictionary attributes to replicate what is seen in REDCap

IMPORTING CSV DATA TO SAS

Unfortunately, PROC HTTP does not provide an option for outputting directly to a SAS data set, which is likely your next goal. During this next step, there are also a few additional considerations. When importing the csv data into SAS, it may not know how to handle certain values it finds in the raw csv files (like carriage returns, HTML, and unpaired quotes). Some variations in processing may be required:

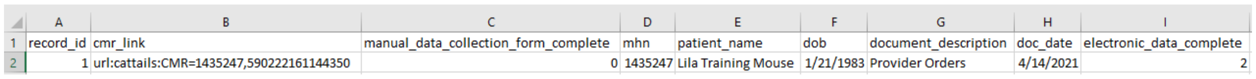
1) Remove Line Feeds and Carriage Returns
2) Do not use labels with HTML code in them
3) Restrict variable names to valid variable names (maximum length of 32, no spaces, no special characters)

In Appendix 1, we can implement each of these fixes by toggling the parameters fed to the macro. For instances where you're looking to use labels that do have HTML code in them, consider modifying the portion of the code that removes line feeds and carriage returns to also remove HTML code. Another option is to strip out the HTML after you read in the data dictionary as SAS Institute does in Sample 24717 (SAS Institute, 2020).

APPLYING SAS ATTRIBUTES

When exporting the data from REDCap it will default to using the REDCap variable names and unformatted values:

Figure 5. Example Raw CSV Data



	A	B	C	D	E	F	G	H	I
1	record_id	cmr_link	manual_data_collection_form_complete	mhn	patient_name	dob	document_description	doc_date	electronic_data_complete
2	1	url:cattails:CMR=1435247,590222161144350	0	1435247	Lila Training Mouse	1/21/1983	Provider Orders	4/14/2021	2

Data exported from REDCap may not be compliant with SAS conventions, but it will also not come with the metadata we have come to expect from SAS datasets (labels, formats, libraries). Here are a few additional measures to take that can make your imported data more user-friendly by leveraging the downloaded data dictionary contents:

1) Use field labels to label SAS variables (CAUTION: Those with HTML may not work as expected).
2) Create formats by parsing the options available in REDCap
3) House formats in an external format library for future use

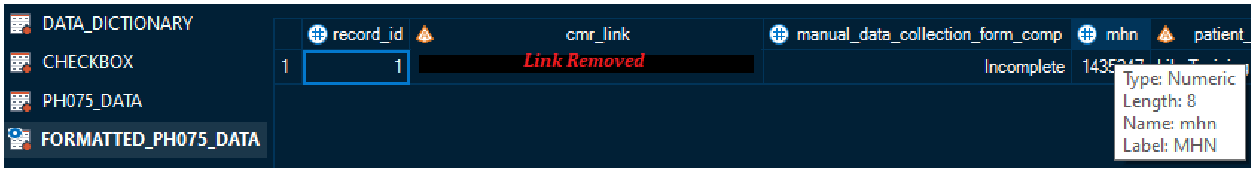
RUNNING THE MACRO: FINAL RESULTS

Using the below example call with fake locations and values:

```
%include 'C:\\REDcap API Labeling Exporter.sas';
libname PH075 'C:\Desktop';
%RedCapAPIExport(External=0
                  ,mytoken=&SECURED_TOKEN.
                          ,BaseDir=%bquote(C:\My Directory with Spaces)
                    ,CSVName=TestData
                          ,ReplaceCRLF=1
                          ,SASDataName=PH075_Data
                          ,format_library=1
                          ,formatlibname=PH075.PH075_Formats
                          ,FormatDataName=Formatted_PH075_Data
                          )
```

;

The results will look like this:



DATA_DICTIONARY
CHECKBOX
PH075_DATA
FORMATTED_PH075_DATA

	record_id	cmr_link	manual_data_collection_form_comp	mhn	patient_
1	1	Link Removed	Incomplete	1435	

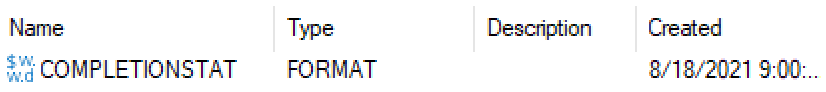
Type: Numeric
Length: 8
Name: mhn
Label: MHN

Figure 6. Final SAS Datasets from REDCap

In the above figure, you will find:

- **DATA_DICTIONARY** (created by default): A dataset containing the REDCap data dictionary for your project.
- **CHECKBOX** (created by default): A dataset containing any check box fields that will be converted to individual indicator fields
- **PH075_DATA**: An unformatted version of the data with or without labels.
- **FORMATTED_PH075_DATA**: A formatted version of the data with or without labels.

The macro will also produce a format library containing all formats available. In this case, we have only the default format for the completion variables, since we did not have any multiple-choice variables.



Name	Type	Description	Created
COMPLETIONSTAT	FORMAT		8/18/2021 9:00:..

Figure 7. Example Format Library

Use the format library to format the unformatted version of the data conditionally when creating reports. It is also helpful when formatting other sources of data in the same manner as your REDCap data.

SECURING YOUR TOKEN AND MINIMIZING DATA DUPLICATION

In the macro call above you see the macro variable *secured_token* is the input for the token value. This is because we don't want you (or those reading the code/log files) to know my token value, because then you'd have access to our data! While placing the token value into the macro call is valid and exports from the project will still appear in the logging, anyone running that code will have access to the data and appear to be the token owner. Yikes!

To avoid unauthorized access, you can store your token value in a file somewhere it's more protected. For example, the C: Drive under your user folder or other folders that only you have access to/require credentials to enter. From there, you can tell SAS to fetch the information only at the time of execution. Is it hack-proof? No. Is it more secure than leaving it in your code? Yes. Storing tokens externally is especially true when storing code in public places or when that code is ran by multiple users. So long as each person has a token value stored in a similar path, the code should use the correct person's token without manual intervention.

In a similar fashion, the creation of CSV files during the import to SAS defeats the security benefits of storing that data in REDCap if they are left there to be unmonitored copies. In medicine, we follow a "minimum necessary" policy for access to protected health information (PHI) and extreme caution should be taken when creating unmonitored PHI. For this reason, it is generally a good idea to delete the CSVs made by the export process.

The macros in Appendix 2 (%REDCAP_TOKEN_SECURE and %REDCAP_TEMP_FILE_CLEAN) accomplish both of these additional security measures. Additionally, %REDCAP_TOKEN_SECURE controls options used in your session to further avoid the printing of that token value during the export process. Before using %REDCAP_TOKEN_SECURE you will need to identify a file path or pattern of file paths that work for you (See Required Edit 3).

The calls must "wrap" your other calls/code. For example:

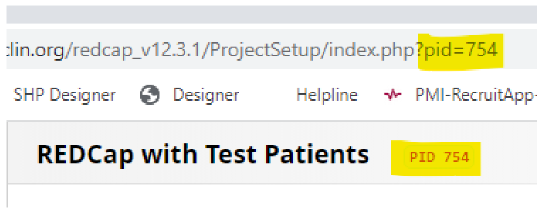
```
%REDCAP_TOKEN_SECURE(PID=001);

[Your Code Here]

%REDCAP_TOKEN_SECURE(CLOAK=OFF);
%REDCAP_TEMP_FILE_CLEAN(LOCATION=%bquote(C:\My Directory with Spaces),
ITEM=ALL);
```

Use extreme caution when providing the value "ALL" (as seen above) to the ITEM parameter. This macro will **delete all files in the directory** under that condition. While this can be extremely helpful, it can also be disastrous if you are storing important items in that directory. To be cautious, we recommend creating a separate REDCap exports directory that contains only delete eligible files.

Note the project id (PID) given is found in REDCap next to the project title (and in the URL) and is used by the macro to identify the correct file containing the token.


:lin.org/redcap_v12.3.1/ProjectSetup/index.php?pid=754

REDCap with Test Patients PID 754

Figure 8. Location of Project ID Values

LIMITED EXPORTS

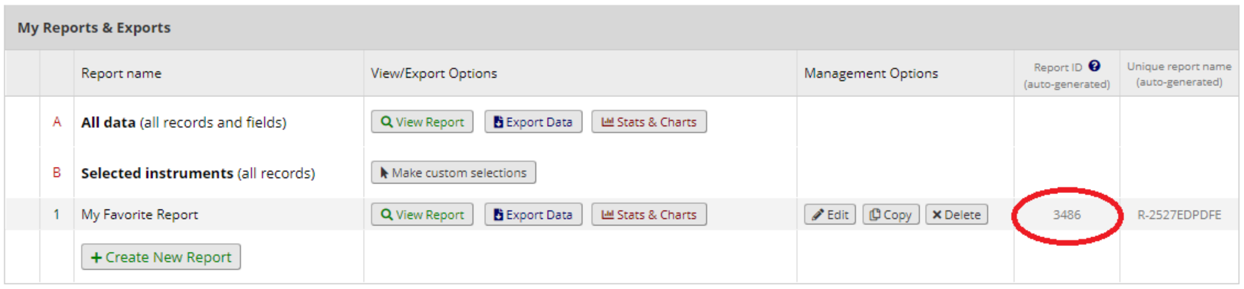
The API playground can help you to create many types of downloads. Nearly all of what exists in REDCap can be exported to SAS using the API method. The macro provided represents a scenario where you are downloading all data that is captured via survey or form instruments. With very little modification to that macro's PROC HTTP call string, you can perform limited exports. Here are two ways to perform a limited export of your data.

DOWNLOADING REPORTS

Within REDCap you can create a bunch of helpful reports that execute in real-time. They can use filter logic, pull in only the fields selected, and offer helpful options (like creating one field with a list of selected checkbox answers, instead of creating an indicator field for each). Reports help you to limit and organize your data while using the online interface and can do the same when you're exporting. Consider the following modification to the PROC HTTP for downloading reports. In this code, the macro variable *report_id* is used to provide the report id given in REDCap (Circled in Figure 9).

```
%let report_id=3486;

proc http
      in="token=&secured_token.%NRSTR(&content=report&format=csv&report
_id=)&report_id.%NRSTR(&csvDelimiter=&rawOrLabel=raw&rawOrLabelHeaders
=raw&exportCheckboxLabel=false&returnFormat=csv)"
      out= my_out
      url = "&instance_url."
      method="POST"
;
debug level=1;
run;
```



My Reports & Exports

	Report name	View/Export Options	Management Options	Report ID (auto-generated)	Unique report name (auto-generated)
A	**All data** (all records and fields)	View Report, Export Data, Stats & Charts			
B	**Selected instruments** (all records)	Make custom selections			
1	My Favorite Report	View Report, Export Data, Stats & Charts	Edit, Copy, Delete	3486	R-2527EDPDFE
	+ Create New Report				

Figure 9. Report Identifiers

SPECIFIC FIELDS EXPORT

In small projects exporting all data from REDCap is not too much of a concern when it comes to processing power and the extra effort it takes to create a specific report is likely not worth the time unless there's a use for it on the front end. However, in larger projects it would not be efficient to download all data every time if you only need a subset of the fields or data. Still, it would not be efficient to create a new report for each set of limiting factors. In this case, the option to place the limiting factors directly in the PROC HTTP string may be the most efficient path.

In this modified PROC HTTP we are limiting to only the fields of interest by providing a comma-delimited list of variable names to macro variable *fieldlist*. Note that you must place the list inside **%bquote()** to avoid commas being seen as additional macro call values.

```
%let FieldList=%bquote(var1,var2);

proc http
      in="token=&mytoken.%NRSTR(&content=record&type=flat&format=csv&fi
elds=)&fieldlist.%NRSTR(&)"
      out= my_out
      url = "&instance_url."
      method="POST"
;
debug level=1;
run;
```

CONCLUSION

REDCap® is a powerful tool in use not only in research, but also in health data manual abstraction as a whole. Whether you are collecting data for a study or creating a repository for patient tracking, the ability to abstract medical data into a secure and electronically accessible location is key. REDCap provides mechanisms for quality data abstraction, documentation, and presentation. Get rid of the Excel spreadsheets and paper records, and get ready for dynamic data ready in real-time for your SAS skills. Together REDCap and SAS are your friends for EHR manual abstraction.

REFERENCES

Harris, P. A., Taylor, R., Minor, B., Elliott, V., Fernandez, M., O'Neal, L., . . . Duda, S. (2019, July). The REDCap consortium: Building an international community of software platform partners. *Journal of Biomedical Informatics, 95*, 103208. doi:10.1016

Harris, P. A., Taylor, R., Thielke, R., Payne, J., Gonzalez, N., & Conde, J. G. (2009, Apr). Research electronic data capture (REDCap) – A metadata-driven methodology and workflow process for providing translational research informatics support. *J Biomed Inform, 42*(2), 377-81. doi:10.1016

SAS Institute. (2006, June 27). *Sample 26065: Remove carriage return and linefeed characters within quoted strings*. Retrieved from SAS Support: https://support.sas.com/kb/26/065.html

SAS Institute. (2020, December 08). *Sample 24717: Remove HTML tags from character strings*. Retrieved from SAS Support: https://support.sas.com/techsup/notes/v8/24/717.html

ACKNOWLEDGMENTS

I would like to thank Brent Olson for providing the initial code, which was modified to make the %REDCAPAPIEXPORT macro and Leila Deering for bringing REDCap® into my life. This paper would not have been possible without you.

RECOMMENDED READING

- *Base SAS® Procedures Guide*: HTTP Procedure

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Brooke Ellen Delgoffe
Marshfield Clinic Health System
brooke_delgoffe@hotmail.com
https://www.linkedin.com/in/brookedelgoffe/

APPENDIX 1

```
/**************************************************************
Program          : REDcap API Labeling Exporter.sas
Programmers      : Brooke Ellen Delgoffe & Brent Olson
                   Marshfield Clinic Health System
                   Marshfield, WI 54449
Date       : 05-24-2022

Purpose: This program exports data and its data dictionary
from REDcap and stores it in CSV files.

Optionally, it can also
      --import this data into SAS work or permanent datasets
      --apply labels
      --create a formatted version of the data
      --create a format library

**************************************************************
How to Use this Macro:

1. Place an %include statement calling this program
      in your program.

2. Copy one of the below templates into your code below the include
statement
      and fill in paramters.

**The BaseDir, CSVName, and mytoken fields are required for
all runs. If the data do not read in correctly, changing the
value of ReplaceCRLF may be required. All other options are
optional. To use defaults remove those definitions from the
call altogehter (erase line).

Full Definition:
%RedCapAPIExport(External=[0=Internal, 1=External]
    ,mytoken=[Enter REDcap Token (Request in REDcap prior)]
    ,BaseDir=%bquote([Enter Long UNC Path without Quotes])
    ,CSVName=[Enter Name for CSV]
    ,ReplaceCRLF=[Enter 0 or leave blank to default to 1]
    ,UseProcImport=[Enter 0 or leave blank to default to 1]
    ,SASDataName= [leave blank to name NEW_API_DATA]
    ,NoLabels=[Enter 1 or leave blank to default to 0]
    ,format_library=[Enter 1 or leave blank to default to 0]
    ,formatlibname= [enter format library name if 1 entered above]
    ,FormatDataName=[leave blank to name NEW_API_DATA_F]
    ,dict_string_path=%bquote([Location of HTTPS string for Data
Dictionary])
    ,data_string_path=%str([Location of HTTPS string for Data])
                       )
;
```

```
Minimum Definition:
%RedCapAPIExport(BaseDir=%bquote([Enter Long UNC Path without Quotes])
                 ,CSVName=[Enter Name for CSV]
                 ,mytoken=[Enter REDcap Token (Request in REDcap
prior)]
                       )
;
**************************************************************
Variable Definitions:
*********************
External= Tells SAS which instance of REDcap to use (1=External
0/missing=Internal)
mytoken= API Token Requested in REDcap **required**
BaseDIR= Location you want output data in. Must use %bquote() if
directory has apostrophe in name. **required**
CSVName= Name for Raw Data CSV **required**
SASDataName= Name for SAS Data Set [Default:work.NEW_API_DATA]
NoLabels= Tells SAS not to apply the labels to the SAS dataset if set
to 1.
UseProcImport= 0=Only CSV, 1=CSV+SAS dataset [Default:1]
ReplaceCRLF= 0=Do not remove line feeds 1=remove line feeds [Default:
1]
format_library= 0=no format library, 1=format library created
[Default=0]
formatlibname= name of format library [Default: Missing=creates format
library in work directory]
FormatDataName= Name of formatted SAS dataset [Default:NEW_API_DATA_F]

The following may be used if a different
http string is needed, such as with special field requests.
These will override the defaults given in the macro.

dict_string_path=gives a long path to a file containing the
                      HTTP call string for puling the data dictionary.
data_string_path=gives a long path to a file containing the
                      HTTP call string for pulling data.

**************************************************************
Known Issues:
--If a character field in REDcap is completely null and SAS imports
     the field as numeric, the format area will create an error, but
     becuase there are no values this is not a true issue.
-Cannot be used to create formatted datasets for projects with
     formats containing a vertical bar ('|')
**************************************************************/

/***************************************************/
/***!!!!!! DO NOT CHANGE BELOW DURING USE!!!!!!!!! */
/***************************************************/

%MACRO RedCapAPIExport(external=0 /*Instance of REDcap 1=External
0/null=Internal*/
```

```
           ,mytoken= /*API Token Requested in REDcap*/
           ,BaseDIR= /*Location you want raw data in*/
           ,CSVName= /*Name for Raw Data CSV*/
           ,SASDataName=NEW_API_DATA /*Name for SAS Data Set*/
           ,NoLabels=0 /*Supresses Labels in SAS Data Set*/
           ,UseProcImport=1 /*0=Only CSV, 1=CSV+SAS dataset*/
           ,ReplaceCRLF=1 /*0=Do not remove line feeds 1=remove line
feeds*/
           ,format_library=0 /*0=no format library 1=format library
created*/
           ,formatlibname= /*name of format library*/
           ,FormatDataName=NEW_API_DATA_F /*Name of formatted SAS
dataset*/
           ,dict_string_path=%str() /*Location of HTTPS string for
Dictionary*/
           ,data_string_path=%str() /*Location of HTTPS string for
Data*/
);

options nomprint nosymbolgen nomlogic; *do not unmask variables;

/***********************************************************
Step 1. Establish Name and path of the export file(s).
*******************************************************/
*** .CSV output file to contain the exported data ***;
filename my_dict "&BaseDIR.\Data Dictionary.csv";

*** .CSV output file to contain the exported data ***;
filename my_out "&BaseDIR.\&CSVName..csv";

%if &Dict_string_path. ne %str() %then %do;
     filename d1 "&dict_string_path.";
     filename d2 "&data_string_path.";
     %end;

/***********************************************************
Step 2. Request all observations and data dictionary with one
row per record (TYPE=FLAT).
*******************************************************/

*** Establish URL for Instance of Interest ***;

%if &external.=1 %then %do;
     %let instance_url=[ENTER YOUR EXTERNAL URL];
```
Required Edit 1
```
%end;
%else %do;
     %let instance_url=[ENTER YOUR INTERNAL URL];
```
Required Edit 2
```
%end;
```

```
***Set up File Refs or File Strings**;
%if &dict_string_path. ne %str() %then %do;
            %let dict_string=d1;
      %end;
      %else %do;
            %let
dict_string="token=&mytoken.%NRStr(&content=metadata&type=flat&format=
csv&)";
      %end;

%if &data_string_path. ne %str() %then %do;
      %let data_string=d2;
      %end;
      %else %do;
      %let
data_string="token=&mytoken.%NRStr(&content=record&type=flat&format=cs
v&)";
      %end;

*** PROC HTTP call for dictionary. ***;
proc http
      in=&dict_string.
      out= my_dict
      url = "&instance_url."
      method="POST"
;
run;

*** PROC HTTP call for data.  ***;
proc http
      in=&data_string.
      out= my_out
      url = "&instance_url."
      method="POST";
run;


/*****************************************************************
Step 3. (Optional) Remove carriage returns and linefeeds that can
effect the import to SAS.
*****************************************************************/

%IF &ReplaceCRLF = 1 %Then %Do;
      /************************** CAUTION ***************************/
      /*                                                            */
      /* This program UPDATES IN PLACE. Create a backup copy before */
      /* running.                                                   */
      /*                                                            */
      /************************** CAUTION ***************************/

      /* Replace carriage return and linefeed characters inside     */
      /* double quotes with a specified character.  This sample     */
      /* uses '; ', but any character can be used, including */
      /* spaces.  CR/LFs not in double quotes will not be replaced. */

      %let repA='; ';         /* replacement character */

      data _null_;
        /* RECFM=N reads the file in binary format. The file consists
*/
        /* of a stream of bytes with no record boundaries.
SHAREBUFFERS */
        /* specifies that the FILE statement and the INFILE statement
*/
        /* share the same buffer.
*/
        infile my_out recfm=n sharebuffers;
        file my_out recfm=n;

        /* OPEN is a flag variable used to determine if the CR/LF is
within */
        /* double quotes or not.  Retain this value.
*/
        retain open 0;

        input a $char1.;
        /* If the character is a double quote, set OPEN to its opposite
value. */
        if a = '"' then open = ^(open);

        /* If the CR or LF is after an open double quote, replace the
byte with */
        /* the appropriate value.
*/
        if open then do;
          if a IN ('0D'x,'0A'x) then put &repA;
        end;
      run;

%End;

/*Regardles of Options Selected, replace linefeeds and carriage
returns within cells to spaces in the data dictionary */

      %let repA=' ';
      data _null_;
        /* RECFM=N reads the file in binary format. The file consists
*/
        /* of a stream of bytes with no record boundaries.
SHAREBUFFERS */
        /* specifies that the FILE statement and the INFILE statement
*/
```

```
        /* share the same buffer.
*/
        infile my_dict recfm=n sharebuffers;
        file my_dict recfm=n;

        /* OPEN is a flag variable used to determine if the CR/LF is
within */
        /* double quotes or not.  Retain this value.
*/
        retain open 0;

        input a $char1.;
        /* If the character is a double quote, set OPEN to its opposite
value. */
        if a = '"' then open = ^(open);

        /* If the CR or LF is after an open double quote, replace the
byte with */
        /* the appropriate value.
*/
        if open then do;
          if a IN ('0D'x,'0A'x) then put &repA;
        end;
      run;

/***********************************************************
Step 4. (Optional) Read .CSV data file into SAS and save it.
      Use a standard Proc Import with MAX Guessing Rows.
      Use import of data dictionary to apply labels
      (optional) create format library
      (optional) apply formats creating separate version
***********************************************************/

%IF &UseProcImport = 1 %Then %Do;

      *** prep export filename for SAS ***;
      %let SASname = %sysfunc(TRANSLATE(&CSVName.,'_',' '));
      %IF %sysfunc(ANYDIGIT(&SASname.,1)) = 1 %Then %let SASname =
%sysfunc(CATS(_,&SASname.));
      %let SASname = %sysfunc(COMPRESS(&SASname., ,kn));
      %IF %sysfunc(LENGTH(&SASname.)) > 28 %Then %let SASname =
%sysfunc(SUBSTR(&SASname.,1,28));

      %put ***&SASname.***;

      /* Read in New Version of Data File*/
      PROC IMPORT OUT=RAW_API_DATA
            DATAFILE= my_out
            DBMS=csv REPLACE;
            GETNAMES=YES;
            guessingrows=MAX;
      RUN;
      /* Read in Data Dictionary*/
      PROC IMPORT OUT=Data_Dictionary
            DATAFILE= my_dict
            DBMS=csv REPLACE;
            GETNAMES=YES;
            guessingrows=MAX;
      RUN;

      data Data_Dictionary;
      set Data_Dictionary;

      *remove any html tags;
      retain rx1;
      if _n_=1 then rx1=prxparse("s/<.*?>//");
      call prxchange(rx1,-1,field_label);
      call prxchange(rx1,-1,section_header);
      run;

      /*Create formats*/
      data formats checkbox;
      length format_def $5000. type $9. field_label field_label2 $60.;
      set data_dictionary(keep=field_name field_label field_type
select_choices_or_calculations);
      format field_label field_label2 $60.;

      *Create an indicator variable for each value of a checkbox like
REDCap does;
      if field_type='checkbox' then do;
            do i=1 to countw(select_choices_or_calculations,'|');
*identify number of options by looking for vertical bars;
                        *Format field names to comply with SAS but follow
REDCap structure;

      field_name2=strip(field_name)||"___"||strip(scan(scan(select_choi
ces_or_calculations,i,'|'),1,','));
                        field_name2=substr(field_name2,1,32);
                        field_label2=strip(field_label)||":
"||strip(scan(scan(select_choices_or_calculations,i,'|'),2,','));

      len1=find(scan(select_choices_or_calculations,i,'|'),',')+1;

      len2=length(strip(scan(select_choices_or_calculations,i,'|')));
                  format_def="0='No' 1='Yes'"; *Define a format as
yes/no;
                  type='Numeric';
                  output checkbox;
                  end;
            if field_type='checkbox' then delete; *delete old entry
once individual variables are defined;
            end;
      *For categorical variables, keep original entry;
      else if countw(select_choices_or_calculations,'|') > 0 then do;
      field_name=strip(substr(field_name,1,32)); *SAS compliant names;
      do i=1 to countw(select_choices_or_calculations,'|');*identify
number of options by looking for vertical bars;
      *Assign character if unformatted values contain a letter;
      if
anyalpha(strip(scan(scan(select_choices_or_calculations,i,'|'),1,',')))
) then do;

      val='"'||strip(scan(scan(select_choices_or_calculations,i,'|'),1,
','))||'"="'||strip(substr(scan(select_choices_or_calculations,i,'|'),
find(scan(select_choices_or_calculations,i,'|'),',')+1,length(strip(sc
an(select_choices_or_calculations,i,'|')))))||'"';
            type='Character';

      len1=find(scan(select_choices_or_calculations,i,'|'),',')+1;

      len2=length(strip(scan(select_choices_or_calculations,i,'|')));
            end;
      *Otherwise assume numeric;
      else do;

      val=strip(scan(scan(select_choices_or_calculations,i,'|'),1,','))
||'="'||strip(substr(scan(select_choices_or_calculations,i,'|'),find(s
can(select_choices_or_calculations,i,'|'),',')+1,length(strip(scan(sel
ect_choices_or_calculations,i,'|')))))||'"';
            type='Numeric';
            end;
      format_def=catx(' ',format_def,val); *create a format statement
to go with each variable;
      end;
      drop i val;
      end;
      *For free text no formats are needed;
      else do;
            field_name=strip(substr(field_name,1,32)); *if free text,
then make sure name is SAS compliant;
      end;
      output formats;
      run;

      *keep only format information;
      data formats;
      length field_name $32. field_label $60.;
      set checkbox(drop=field_name field_label
rename=(field_name2=field_name field_label2=field_label)) formats ;
      drop field_name2 field_label2;
      run;

      /*Create label statements*/
```

```
     proc sql noprint;
     select distinct strip(field_name)||'="'||strip(field_label)||'"'
           into :label_statement
           separated by ' '
           from formats;
     quit;

           /*Create format statements*/
     proc sql noprint;
     select distinct
           case when type='Character' then "value
$"||strip(substr(field_name,1,31))||"_  "||strip(format_def)|| ' ;'
                 else "value "||strip(substr(field_name,1,31))||"_
"||strip(format_def)|| ' ;'
                 end
           ,
           case when type='Character' then field_name||"
$"||strip(substr(field_name,1,31))||"_.  "
                 else field_name||"
"||strip(substr(field_name,1,31))||"_.  "
                  end
           into :format_statement
           separated by ' ',
           :applyformats
           separated by ' '
           from formats
           where format_def ne '=""';
     proc sql noprint;
     select distinct field_name
           into :yesno_formats
           separated by ' '
           from data_dictionary
           where field_type='yesno';
     quit;
     *add form complete fields;
     proc sql noprint;
     select distinct substr(strip(form_name)||"_complete",1,32)
           into :completion_fields
           separated by ' '
           from data_dictionary
     quit;


     /*(optional) Create Format Library*/

     %if &format_library.=1 AND %symexist(format_statement) %then %do;
     libname save "&BaseDir.";
     proc format lib=save.&formatlibname.;
     &format_statement.
     value yesno 0='No' 1='Yes';
     value completionstat 0='Incomplete' 1='Unverified' 2='Complete';
     quit;
```

```
	%end;
	%else %if %symexist(format_statement) %then %do;
	proc format;
	&format_statement.
	value yesno 0='No' 1='Yes';
	value completionstat 0='Incomplete' 1='Unverified' 2='Complete';
	quit;
	%end;
	%else %do;
	proc format;
	value completionstat 0='Incomplete' 1='Unverified' 2='Complete';
	quit;
	%end;



	/********************
	Create Final Dataset
		-apply labels
	********************/

	data &SASDataName.;
	set RAW_API_DATA;
		%if &NoLabels.=0 %then %do;
			label &label_statement.;
		%end;
	run;

	%if &FormatDataName. ne %str() %then %do;
	/********************
	(optional) Create Formatted Dataset
		-apply labels and formats
	********************/
		%if &format_library.=1 and %symexist(format_statement)
%then %do;
			options fmtsearch=(save.&formatlibname. WORK);
			%end;

		data &FormatDataName.;
		set RAW_API_DATA;
		%if &NoLabels.=0 %then %do;
			label &label_statement.;
		%end;
		%if %symexist(applyformats) %then %do;
		format &applyformats.
			%if %symexist(yesno_formats)=1 %then %do;
			&yesno_formats. yesno.
			%end;
		;
		%end;
		format &completion_fields. completionstat.;
		run;
```

```
      %end;

%End;

      /*Clear Intermediate Datasets*/
      proc datasets lib=work memtype=data nolist;
      delete raw_api_data formats;
      run;


%MEND RedCapAPIExport;
```

APPENDIX 2

```
/**********************************************
PROGRAM                 : RA_REDCAP_TOKEN_CLOAK.sas
PROGRAMMER              : Steffani Roush & Brooke Ellen Delgoffe
DATE                    : 06-04-2021

PURPOSE: This program defines macros which
"cloak" REDCap token values, delete unnecessary
import/export CSVs and are applied as wrapper code
around existing code used for exporting/importing
data from REDCap.

!!! Before Using !!!
--------------------
--> Make sure you have your [INSTANCE]_[PID]_config.txt
      file and connection strings configured correctly.
--> Use the macro variable &SECURED_TOKEN. anywhere the
      value of the token should appear in your code.
--> Remove options statements for mprint, mlogic, and symbolgen from
code
      going between the macro calls. The value of these options will be
restored
      by the bottom wrapper.
--> Remove put statements that may print strings that include your
token value.
      (EX: Http call strings).

Minimum Macro Call Templates:
-----------------------------
*goes at top of program;
      %REDCAP_TOKEN_SECURE(PID=[REDCap Project ID]);
*go at bottom of program;
      %REDCAP_TOKEN_SECURE(CLOAK=OFF);
*optional at bottom of program;
      %REDCAP_TEMP_FILE_CLEAN(LOCATION=%bquote([export/import
location]), ITEM=[ALL or specific names]);

Call Definitions and Parameters:
```

```
--------------------------------
REDCAP_TOKEN_SECURE: tells SAS to start or stop using
     token information located outside SAS and controls
     options that prevent the values from being displayed
     in the log.
     This macro references the &DWID\[INSTANCE]_[PID]_config.txt
     file stored under [Enter your path to files]
```

Required Edit 3. Long Path to Token Files

```
     Arguments:
     PID: REDCap project ID
          Values: No constraints.
     CLOAK: tells SAS whether you're putting the cloak
          on or taking it off.
          Values:[ON/OFF/DB] DEFAULT=ON (!! Use DB only for debugging
!!)
     INSTANCE(Optional):  Tells SAS the location of your REDCap
Project.
          Values:[INT/EXT] DEFAULT=INTERNAL


REDCAP_TEMP_FILE_CLEAN: Removes temporary files created
     during the import/export of REDCap data. Used to delete
     all in a directory or one file at a time.

     Arguments:
     LOCATION: directory location of temporary files enclosed
          in bquote to avoid issues with spaces in long paths.
     ITEM: The name of files to be deleted or ALL for a full directory
delete.
          Do not include the file extention here. Names are not case-
sensitive.
     EXT(Optional): Specifies the file extension when not a csv. Do
not
          include a period in the value.
          DEFAULT=CSV
************************************************/

%macro REDCAP_TOKEN_SECURE(PID=0 /*REDCap Project ID*/
                                   ,INSTANCE=INT /*Specify REDCap
Instance*/
                                   ,CLOAK=ON /*Turn Cloak on or off*/
                                   );;
/* Save debugging options */
%let oldopts =
     %sysfunc( getoption( mprint))
     %sysfunc( getoption( Symbolgen))
     %sysfunc( getoption( mLogic));

%if &CLOAK.=OFF %then %do;
          /* Turn old debugging options back on */
               option &oldopts. ;
```

```
           %put NOTE: Cloaking behavior was turned off. Options have
been restored to their previous values.;;

%end; /*CLOAK OFF*/
%else %if &CLOAK.=ON %then %do;
      %if &PID.=0 %then %do;
      %put ERROR: REDCap Project ID has not been entered. PID is
indicated after PID= in REDCap URL.;
      %end; /*PID Check*/
      %else %do;
                /*Locate Token Information*/
                options mprint symbolgen mlogic; *turn on to evaluate
path variables;
                filename token "[Enter your path to
files]\&dwid.\&instance._&pid._config.txt";

                /* Prevent macros from being printed to the log
temporarily */
                option nomprint nosymbolgen nomlogic;

                /*Import Token Information*/
                %GLOBAL SECURED_TOKEN;
                data _null_;
                   length token_txt $32767;
                   retain token_txt '';
                   infile token flowover dlmstr='//' end=last;
                   input;
                   token_txt=cats(token_txt,_infile_);
                   if last then call
symput('SECURED_TOKEN',strip(token_txt));
                run;

                %put NOTE: The value of SECURED_TOKEN has been
assigned and cloaking behavior turned on.;

      %end;/*PID specified*/
%end; /*CLOAK ON*/
%else %do;
           /* Allow Macro values to Print to log */
           option mprint symbolgen mlogic;

           %put WARNING: Your token information is vulvernable! Use
this setting only for debugging, specify ON or OFF otherwise.;
                /*Locate and Import Token Information*/
           %global SECURED_TOKEN;
                filename token "[Enter your path to
files]\&dwid.\&instance._&pid._config.txt";
                data _null_;
                   length token_txt $32767;
                   retain token_txt '';
                   infile token flowover dlmstr='//' end=last;
                   input;
```

```
                    token_txt=cats(token_txt,_infile_);
                    if last then call
symput('SECURED_TOKEN',strip(token_txt));
                 run;

           %put NOTE: The value of macro variable SECURED_TOKEN has
been assigned;

%end; /*CLOAK ERROR*/
%mend REDCAP_TOKEN_SECURE;


%macro REDCAP_TEMP_FILE_CLEAN (LOCATION=%str() /*filepath to check*/
                                              ,ITEM=%str()   /*Filename
to delete or ALL*/
                                              ,EXT=csv /*Extension of
file(s) to delete*/
                                              );;
%if &ITEM.=ALL %then %do;
      filename filelist "&location.";
       data _null_;
          dir_id = dopen('filelist');
          total_members = dnum(dir_id);
          do i = 1 to total_members;
             member_name = dread(dir_id,i);
             if scan(lowcase(member_name),2,'.')="&ext." then do;
              file_id = mopen(dir_id,member_name,'i',0);
              if file_id > 0 then do;
                freadrc = fread(file_id);
                rc = fclose(file_id);
                rc = filename('delete',member_name,,,'filelist');
                rc = fdelete('delete');
             end;
             rc = fclose(file_id);
          end;
          end;
          rc = dclose(dir_id);
       run;

      %put NOTE: All .&ext. files in &LOCATION. have been deleted.;

%end;
%else %do;
      /* Delete files one at a time*/
      %if %sysfunc(fileexist("&LOCATION.\&ITEM.%str(.)&ext.")) ge 1
%then %do;
         %let
rc=%sysfunc(filename(temp,"&LOCATION.\&ITEM.%str(.)&ext."));
         %let rc=%sysfunc(fdelete(&temp));
      %end;
      %else %do;
```

```
            %put The file "&LOCATION.\&ITEM.%str(.)&ext." does not 
exist;
            %end;
%end;

%mend REDCAP_TEMP_FILE_CLEAN;
```