Data for Accountability, Transparency and Impact Monitoring (DATIM)

Validation of SIMS Data Payloads for DATIM Using R Validation Package

September 2020

U.S. Department of State
U.S. Office of Global AIDS Coordinator (OGAC)

Table of Contents

Validation of Data Payloads for DATIM Using R Validation Package

DATIM is based on the DHIS2 software and therefore is capable of importing different types of data, including CSV, JSON, and XML, as well as ADX formats. Users who want to use the data import capabilities of DATIM should familiarize themselves with the various formats that DHIS2 supports and the syntax of each format.

DATIM has strict controls on data imports, including a requirement to adhere to the numerous validation rules of the system. An R package has been created to help data importers prepare their files for submission to DATIM.

What is the SIMS Import Validation R Package?

The SIMS import validation R package is a program library written using R language to validate countries' data against the business logic of DATIM prior to importing the data into the DATIM system.

Its libraries provide an abstraction layer to the various validation routines that are necessary to import data into DATIM. These scripts, to a large extent, emulate the logic of the DHIS2 server.

Basic functions are exposed to allow users to determine whether their data contain invalid metadata (invalid data elements, incorrect disaggregations, inactive mechanisms, invalid organization units), incompatible data types.

Functions that we will be using for SIMS validation are as follows:

- **d2Parser:** This is a general-purpose function to load the different data formats that DATIM accepts and to standardize the format so we can use other validation functions on the data. This function takes the following input parameters:
    - File name: Path and name of the SIMS import file.
    - Type: Type of the import file. It should be "json," "csv," or "xml."
    - Id Scheme: Identification scheme of the file. The easiest way to identify this is to open the file in a text editor and see what id scheme the payload is using. This could be "code" or "id." Note: In addition to idScheme, which applies to all metadata, the function also accepts the optional dataElementIdScheme and orgUnitIdScheme. This allows multiple schemes to be mixed in one file (e.g., using "code" for data elements but "id" for org units). These schemes default to "id" if "code" is not provided.
    - Org Unit ID: UID of the country (operating unit) for which data are being validated. This field is optional and can be left blank. The d2Parser function derives the org unit ID from the user specified in the secrets file and uses the operating unit to which the user has access.
    - invalidData: Whether to exclude records that have either missing or NA entries. Default is FALSE.
    - csv_header: If the import file is a CSV-formatted file, this indicates whether the file includes the header row. Argument is optional, and the default value is TRUE.
- **sims2Parser:** will parse a semi-compliant DHIS2 CSV file and transform it into a standard data frame which can be used in subsequent DATIM validation routines. The difference with d2Parser is that an extra (non-standard) field will be introduced to record the SIMS assessment. This will in turn be used to deduplicate visits which occur at the same site + mechanism + date

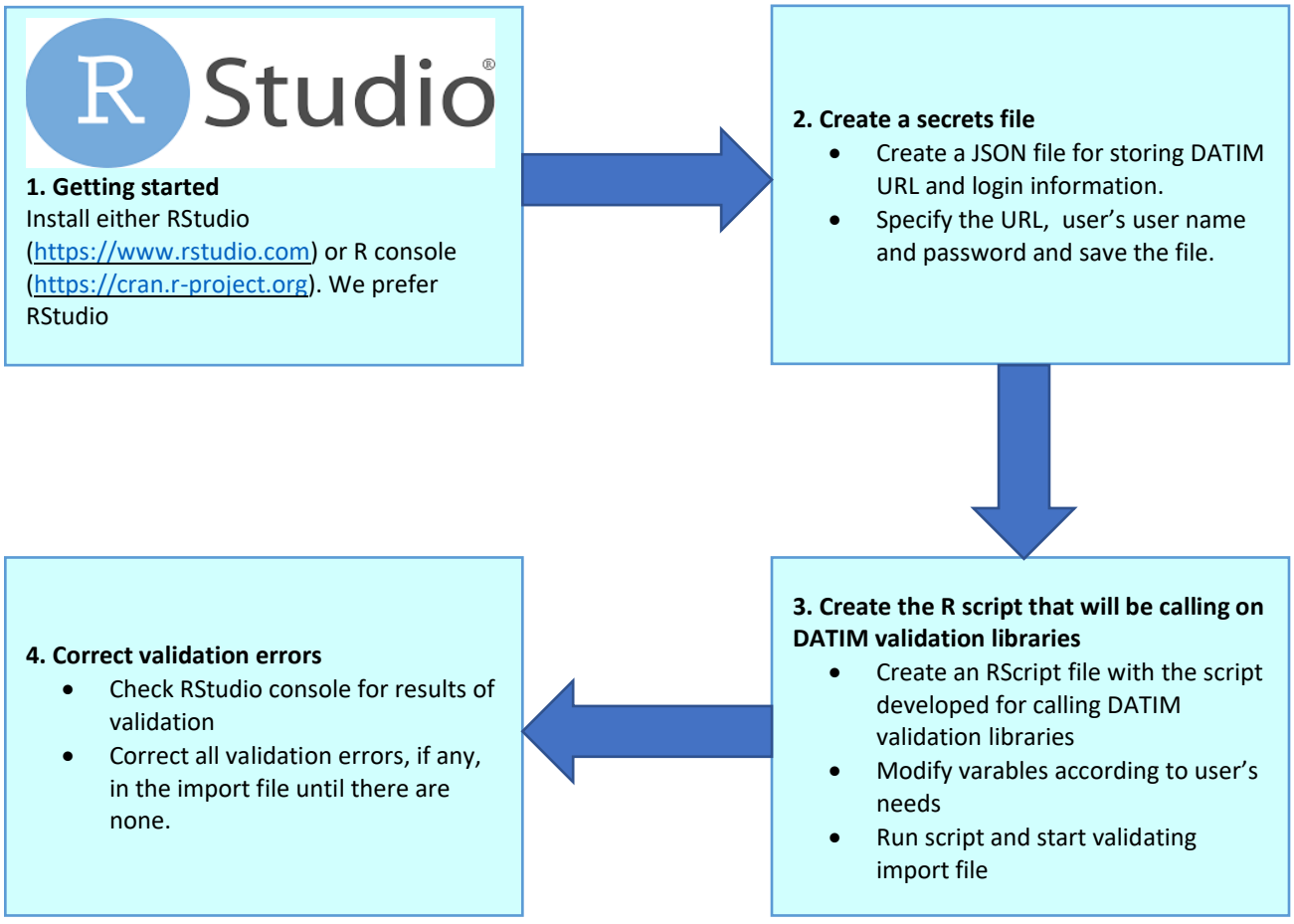
combination. This function will automatically decollide these types of visits, by shifting the period attribute of one of the overlapping assessments to the nearest available date.
    - filename: Location of the payload to be imported. Should be a valid SIMS import file.
    - dataElementIdScheme: Coding scheme of the data elements in the import file, it should be one of either code, name, shortName, or id. The default is "id", which assumes data elements to be defined using their DATIM UIDs.
    - orgUnitIdScheme: Coding scheme of the organization units. It should be one of the following: code, name, shortName, or id. The default is “id”, which assumes that the organization unit identifiers in the import file are DATIM UIDs. Note, that UID is recommended coding scheme for organization units, as other schemes do not guarantee unique identification.

    - idScheme: Mapping scheme to apply to all metadata objects (unless specific metadata objects are overridden by dataElementIdScheme and/or orgUnitIdScheme) to be used for all metadata objects in the import file.
    - invalidData: Specifies whether the resulting data frame should include records that are identified invalid by the parser (records with missing identifiers, or NAs). Default value is FALSE.
    - hasHeader: TRUE by default. Should be set to FALSE if the file does not contain the header row.
    - isoPeriod: period to be used for date shift boundaries. If not provided, no boundaries are set.
- **checkMechanismValidity**: verifies whether mechanisms are valid for the organization unit and period specified in the import data and returns invalid mechanism.
- **checkValueTypeCompliance**: identifies and returns invalid data value types.
- **checkDataElementOrgunitValidity**: verifies whether the provided data element/ organization unit associations are valid and returns organization units with invalid assosiations.
- **checkCoverSheetCompleteness:** checks whether the import file contains all required coversheet data elements including at least one reason data element for comprehensive assessments and all coversheet data elements except reason for follow-up assessments.
- **checkForWrongAssessmentType:** checks if value for assessment type cover sheet is site, all CEE data elements are site and the assessment does not have any data elements that are above site, and check whether reverse is true for above site assessments.
- **checkForCEEValidity**: First checks if the import file contains score and fail the validation if it doesn’t. If it passes the first validation, then checks if all CEEs have valid values and fails the validation if any of them have invalid values.

In addition to the above, the script also identifies overlapping assessments and shifts period to separate them. It then creates a new import file that contains all the data of the original import file with all overlapping assessments shifted. The file will be created in the same location as the original one with “_normalized” added to the end of the original file’s name.

Getting Started

Below is a diagram that depicts steps that need to be followed in order to use the SIMS validation package to validate import data.



To get started, please install either RStudio (https://www.rstudio.com) or R console (https://cran.r-project.org). Both are free to download and use.

To get started with DATIM validation, users will need to have installed the R package "SIMS4Validation." The source code for this library can be found at https://github.com/pepfar-datim/SIMS-Validation/

Users will need an active Internet connection and an active DATIM user name to use the "SIMS4Validation" package. Metadata will be retrieved from the DATIM server using the user's username and password and stored in a local cache. After the objects are cached, the package can be used offline until the cache is invalidated. By default, cached objects are stored for a week and then invalidated.

Please follow these steps to use the SIMS4Validation R script to validate SIMS data.

Step 1: Create a secrets file

To get started, create a "secrets" file, which will contain the authentication information required to access DATIM. You should keep this in a secure place on your computer, because it will require storing the username and password you use to access DATIM as a file on your disk. If you are unable to securely store this file, you can also enter your username and password through a dialog. A secrets file should a single JSON file that looks like this:

```
{
  "dhis": {
    "baseurl": "https://dev-de.datim.org",
    "username": "admin",
    "password": "district"
  }
}
```

Step 2: Create an R file

Install a tool such as RStudio for running R programs.

Create a file with .R extension and add the following as content and provide the location of your secrets file. You will use this file to invoke functions that you will use to validate your data.

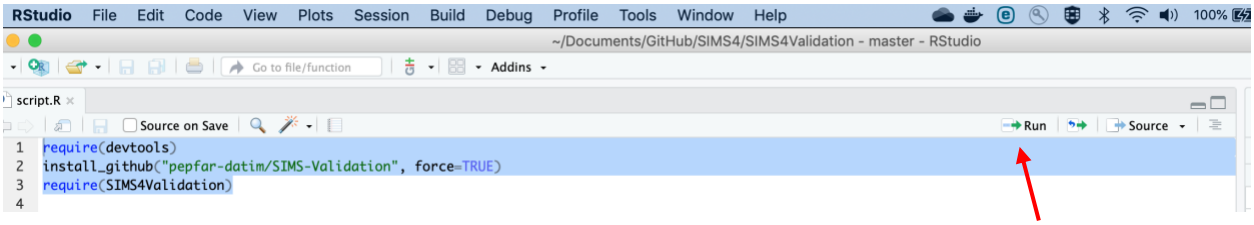
Note: Instead of creating a file, you can also use the script.R file found here and modify the variables as described below.

```
require(devtools)
install_github("pepfar-datim/SIMS-Validation", force=TRUE)
require(SIMS4Validation)
```

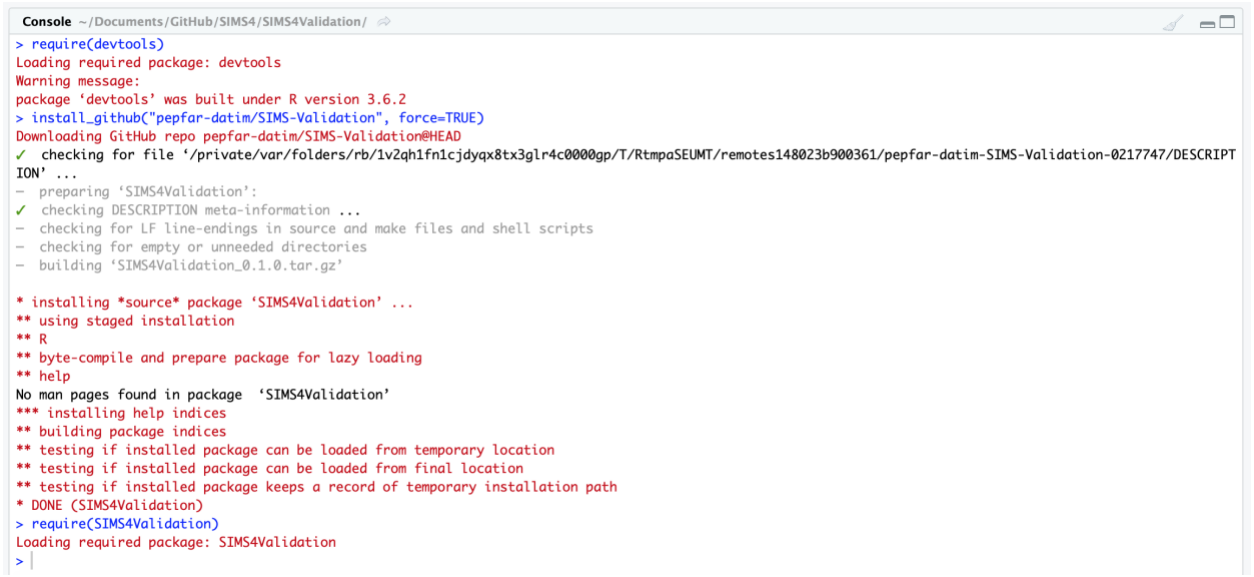
The first three lines of the script load all libraries that are required to run the script.

Open your .R file with RStudio.

Select all of the code in the file and click on the "Run" button.



If all goes well, you will see the code below in the console window.



The following script is an example that uses all available validation procedures from the library, as well as additional checks to verify the integrity of an import file. You can copy and paste the following code to the end of the .R file to run it.

The beginning of the file includes a number of variables that you should modify according to your needs, especially the path and the type. Note that the period expected by DATIM in import files is an ISO calendar period. For SIMS, period is expected to be daily, in YYYYMMDD format. For example, January 27th, 2019 should be coded as 20190127.

Replace `"/path to secret file/secret.json”` with absolute path of the file's location.

Please refer to the data import reference documentation (code lists) for the list of data set UIDs. The example that follows includes UIDs for SIMS 4.0 above site and site data sets for to be used starting FY19 Q2.

```
secrets <- "/path to secret file/secret.json"
```

```
# folder where file is located, and where output files will be written
to

out_dir <- "~/Q1/"


# name of the file to validate

filename <- "q1.csv"


# type of the file (json, xml, or csv)

file_type <- "csv"


# identifier scheme used in the input file

#mechanism identifier: id or code

idScheme <- "code"

#id, code or name

dataElementIdScheme <- "name"

#id or code

orgUnitIdScheme <- "id"


# calendar period (quarter) covered by the input file in YYYYQN
format, e.g. 2019Q3 for July-September 2019

isoPeriod <- "2020Q2"


# whether the input file has the header as the first line

fileHasHeader <- TRUE


SIMS4Validation::SIMSValidationScript(out_dir,filename,file_type,idSch
eme,dataElementIdScheme,orgUnitIdScheme,isoPeriod,fileHasHeader,secret
s)
```

Replace values of the following variables with your values.

- Secrets - location of your secrets file
- dataElementIdScheme – code, name or id
- orgUnitIdScheme - code or id
- idScheme - code or id
- file_type – csv/json/xml/adx/pdf
- out_dir – directory where the file is located
- filename – file name including file extension
- isoPeriod – YYYYQ1/2/3/4
- filesHaveHeader – TRUE/FALSE

Click on the “Source” button (found to the right of the “Run” button) to run the entire validation code.

If there are no validation errors in the file, you will see the following message displayed in the console.

```
> source('~/Documents/GitHub/SIMS-Validation/script.R')
Downloading GitHub repo pepfar-datim/SIMS-Validation@HEAD
✓  checking for file '/private/var/folders/rb/1v2qh1fn1cjdyqx8tx3glr4c0000gp/T/RtmpaSEUMT/remotes148024b87ec89/pepfar-datim-SIMS-Validation-0217747/DESCRIPT
ION' ...
−  preparing 'SIMS4Validation':
✓  checking DESCRIPTION meta-information ...
−  checking for LF line-endings in source and make files and shell scripts
−  checking for empty or unneeded directories
−  building 'SIMS4Validation_0.1.0.tar.gz'

* installing *source* package 'SIMS4Validation' ...
** using staged installation
** R
** byte-compile and prepare package for lazy loading
** help
No man pages found in package  'SIMS4Validation'
*** installing help indices
** building package indices
** testing if installed package can be loaded from temporary location
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (SIMS4Validation)
Warning messages:
1: In getMechanismsMap() :
  No organisation unit specified. Using orgunit ybg3MO3hcf4
2: In getMechanismsMap() :
  No organisation unit specified. Using orgunit ybg3MO3hcf4
3: In checkDataElementOrgunitValidity(data = d2, datasets = dataSets) :
  Invalid data element/orgunit associations were detected!
>
```

Validation Results

If there is data in the import file that does not meet the validation criteria, a file with name that is descriptive of the type of validation error and with content that shows which part of the import data failed the check is created and stored in the same location as the import file.
The following table shows the different types of validation errors and how they can be resolved.

Validation error	Addressed by	Solution
Invalid data element/org unit pairs	checkDataElementOrgunitValidity	Make sure that a data element in a row is valid for the org unit specified in the same row.
Value type compliance issues encountered	checkValueTypeCompliance	Make sure that values are of the correct data type; for example, integer not character.
Mechanism validity issues encountered	checkMechanismValidity	Make sure that a mechanism in a given row is valid for the period specified in the same row.
Incomplete coversheet	checkCoverSheetCompleteness	Make sure the import data contains all required coversheet data elements
Wrong assessment type	checkForWrongAssessmentType	Make sure all CEEs in the import file are of the right assessment type
Invalid CEEs found	checkForCEEValidity	Make sure the score data element is present and has a valid score and if there are other CEEs, make sure that their values are valid.

Run time Errors

Users may encounter a number of different types of errors when running the validation script. A descriptive error message, such as that shown in the following screen shot, will be displayed in the console whenever the code encounters a validation error.

```
Error: `by` can't contain join column `orgUnit` which is missing from RHS
Call `rlang::last_error()` to see a backtrace
```

Errors stop the script from running at the point where they are encountered. This means the issues need to be addressed before the entire script can be run.

Users might also see warning messages displayed in the R console. Warnings however do not cause the script to stop and can be ignored.

The following table shows some of the common error messages and solutions.

Error Message	Addressed by	Solution
Error: Problem with `summarise()` input `count`. x could not find function "n"	--	**Type library(dplyr) in the console and press enter.**
cannot open file '/path/ImportFile.csv': No such file or directory	d2Parser. Note that you have to address all errors produced by d2Parser before you can use other validation functions.	Ensure that you entered the right import file location for the 'path' argument
Start tag expected, '<' not found	d2Parser	This means you specified the import file type as xml when it is not an xml file. Make sure you specify the correct file type.
Error in parse_con(txt, bigint_as_char) : lexical error: invalid char in json text.	d2Parser	This means you specified the import file type as json when it is not a json file. Make sure you specify the correct file type.
Error in type.convert … invalid multibyte string at …	d2Parser	This means you specified the import file type as csv when it is not a csv file. Make sure you specify the correct file type.
Could not resolve host:	d2Parser	Make sure that the secrets file has the correct DATIM URL specified in the "baseurl" field.
Error in DHISLogin(s) : Could not authenticate you with the server!	d2Parser	Make sure that the user name and password pair specified in the secrets file is correct.
Invalid data elements, org units, category option combos, or attribute option combos	d2Parser	Ensure that metadata in the import file are correct. If using UIDs, make sure that you maintain case sensitivity and do not alter them.
Error during wrapup: missing value where TRUE/FALSE needed	sims2Parser	Make sure that isoPeriod is in the correct format YYYYQ1/2/3/4. Example- 2019Q1