PEPFAR
U.S. President's Emergency Plan for AIDS Relief

Data for Accountability, Transparency and Impact Monitoring (DATIM)

Validation of MER Data Payloads for DATIM Using R Validation Package

January 2019

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 MER Import Validation R Package?

The MER 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, or data that do not meet validation rules.

Functions that we will be using for MER 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 MER 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.
- **getExactDuplicates:** This checks for rows with the same combination of data element, period, org unit, category option combo, and attribute option combo. Whether or not values are different, the same combination is an error.
- **checkDataElementDisaggValidity:** This produces a data frame containing records that have invalid data element/category option combination pairs.

- **checkDataElementOrgunitValidity:** This produces a data frame containing records that contain data elements that are not valid for a given org unit (e.g., community-level indicator specified for a facility organization unit).
- **checkNegativeValues:** This produces a data frame containing records with negative values. DATIM does not accept data values for MER numeric data elements that are negative, unless they are deduplication data, and import of those data is currently not supported.
- **checkValueTypeCompliance:** This produces a data frame containing records that have values that do not meet the value type specifications defined for the data element.
- **checkMechanismValidity:** This returns a data frame containing records that have attribute option combos (funding mechanisms) that are not valid. Mechanism validity issues include mechanism's expiry period being prior to the period of the data record, invalid operating unit, etc.
- **validateData:** This validates data against data validation rules for violations.

Getting Started

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 "datimvalidation." The source code for this library can be found at https://github.com/jason-p-pickering/datim-validation.

Users will need an active Internet connection and an active DATIM user name to use the "datimvalidation" 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 datimvalidation R script to validate MER 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.

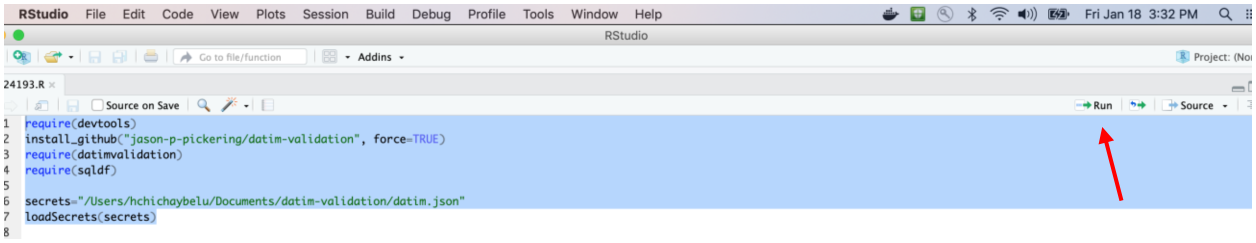
```r
require(devtools)
install_github("jason-p-pickering/datim-validation", force=TRUE)
require(datimvalidation)
require(sqldf)

secrets="/path to secret file/secret.json"
loadSecrets(secrets)
```

Replace "/path to secret file/secret.json" with the location of your secrets file and the name of your file, and then save your .R file.

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

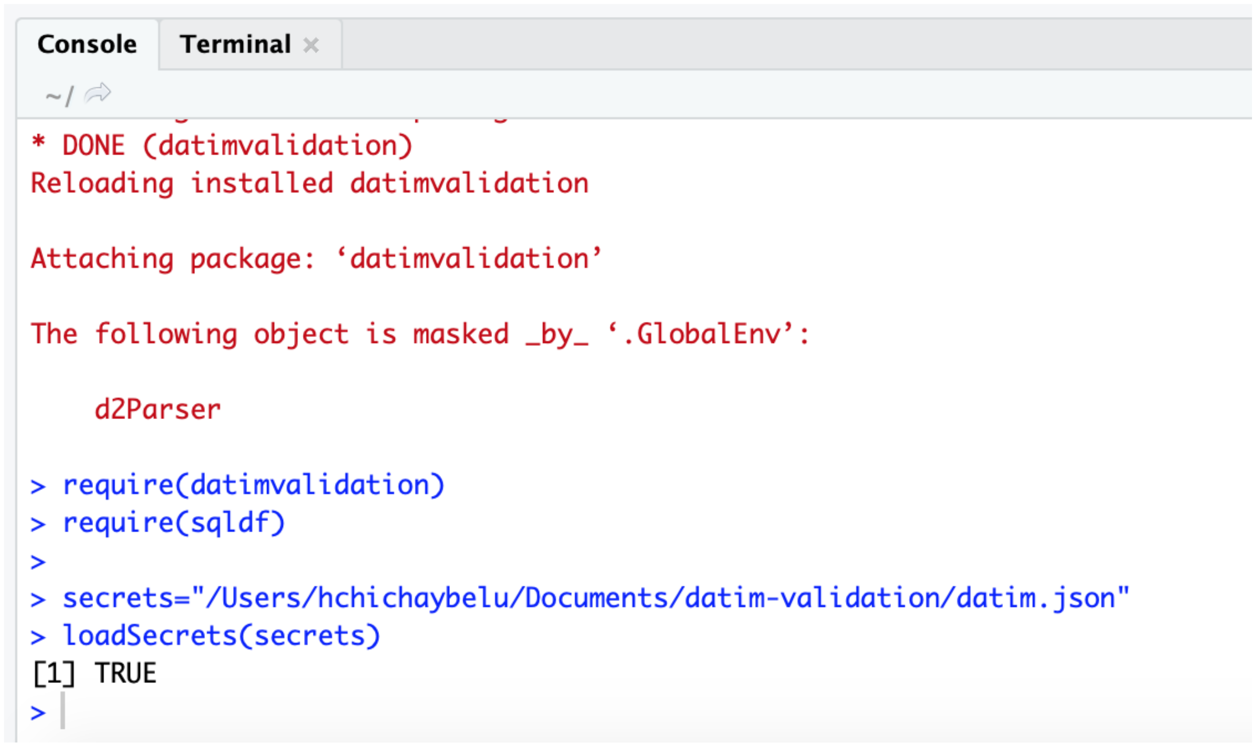
Open your .R file with RStudio.

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



```r
require(devtools)
install_github("jason-p-pickering/datim-validation", force=TRUE)
require(datimvalidation)
require(sqldf)

secrets="/Users/hchichaybelu/Documents/datim-validation/datim.json"
loadSecrets(secrets)
```

If the secret file contains the correct credentials and all goes well, you will see the code below in the console window.



```
* DONE (datimvalidation)
Reloading installed datimvalidation

Attaching package: ‘datimvalidation’

The following object is masked _by_ ‘.GlobalEnv’:

    d2Parser

> require(datimvalidation)
> require(sqldf)
>
> secrets="/Users/hchichaybelu/Documents/datim-validation/datim.json"
> loadSecrets(secrets)
[1] TRUE
>
```

As mentioned earlier, cached metadata objects are stored for a week and then invalidated. If users think that the cache contains old code lists, they can force the script to clear the cache after loading the datim validation library by running this code.

```
cleareCache(force=TRUE)
```

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 and not the fiscal one. For example, the period for data for the quarter matching October–December 2018 should be 2018Q4 in the import file.

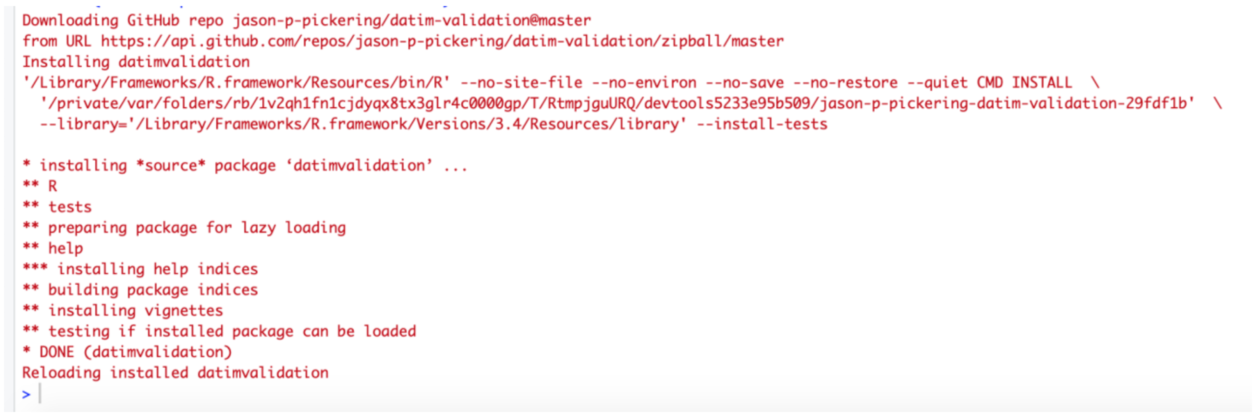
Please refer to the data import reference documentation (code lists) for the list of data set UIDs. The example that follows includes UIDs for all MER results data sets for FY19 Q1.

```
path <- " path to the import file/Test_import.csv"
type <- "csv"
idScheme <- "id"
dataElementIdScheme <- "id"
orgUnitIdScheme <- "id"
expectedPeriod <- "2018Q4"
```

```
ds <-
c("zUoy5hk8r0q","PyD4x9oFwxJ","KWRj80vEfHU","fi9yMqWLWVy","IZ71Y2mEBJF
","ndp6aR3e1X3","pnlFw2gDGHD","gc4KOv8kGlI","FsYxodZiXyH","iJ4d5HdGiqG
")
d <- d2Parser(file=path, type = type, invalidData = TRUE,
idScheme=idScheme,dataElementIdScheme=dataElementIdScheme,orgUnitIdSch
eme=orgUnitIdScheme)


# check for expected period
invalidPeriod <- sqldf(paste0("select * from d where period != '",
expectedPeriod, "'"))
if(nrow(invalidPeriod) != 0){
  write.csv(invalidPeriod, paste0(path, '_invalidPeriod', ts, '.csv'),
na="")
}


# check for duplicates
duplicates <- getExactDuplicates(d)
if(any(class(duplicates) == "data.frame")){
  if(nrow(duplicates) > 0){
    print("Duplicates encountered. Printing out summaries.")
    write.csv(duplicates, paste0(path, '_duplicates', ts, '.csv'),
na="")
  }
}

# data element/coc pair validity
de_disags <- checkDataElementDisaggValidity(d, ds)
if(any(class(de_disags) == "data.frame")){
  if(nrow(de_disags > 0)){
    print("Invalid data element/coc pairs encountered. Printing out
summaries.")
    write.csv(de_disags, paste0(path, '_invalid_de_coc', ts, '.csv'),
na="")
    de_disags2 <- sqldf("select distinct dataElement,
categoryOptionCombo from de_disags")
    write.csv(de_disags2, paste0(path, '_invalid_de_coc_uniques', ts,
'.csv'), na="")
  }
}

# data element/org unit check
de_ou <- checkDataElementOrgunitValidity(data=d, datasets=ds)
if(any(class(de_ou) == "data.frame")){
  if(nrow(de_ou) > 0){
    print("Invalid data element/org unit pairs encountered. Printing
out summaries.")
    write.csv(de_ou, paste0(path, '_invalid_de_ou', ts, '.csv'),
na="")
  }
}

# Negative values
negativeValues <- checkNegativeValues(d)
if(any(class(negativeValues) == "data.frame")){
  if(nrow(negativeValues) > 0){
    print("Negative values encountered. Printing out summaries.")
    write.csv(negativeValues, paste0(path, '_negativeValues', ts,
'.csv'), na="")
  }
}

# value type compliance
valueTypeCompliance <- checkValueTypeCompliance(d)
if(any(class(valueTypeCompliance) == "data.frame")){
  if(nrow(valueTypeCompliance) > 0){
    print("Value type compliance issues encountered. Printing out
summaries.")
    write.csv(valueTypeCompliance, paste0(path,
'_valueTypeCompliance', ts, '.csv'), na="")
  }
}

# mechanism validity
mechanismValidity <- checkMechanismValidity(d, ou)
if(any(class(mechanismValidity) == "data.frame")){
  if(nrow(mechanismValidity) > 0){
    print("Mechanism validity issues encountered. Printing out
summaries.")
    write.csv(mechanismValidity, paste0(path, '_mechanismValidity',
ts, '.csv'), na="")
  }
}

#validation rules
doMC::registerDoMC(cores=4) # or however many cores you have access to
vr_violations<-validateData(data=d, return_violations_only=TRUE,
parallel=TRUE, datasets=ds)
if(any(class(vr_violations) == "data.frame")){
  if(nrow(vr_violations) > 0){
    print("Validation rule violations encountered. Printing out
summaries.")
    write.csv(vr_violations, paste0(path, '_vr_', ts, '.csv'), na="")
  }
}


# check if there are dedupe mechanisms (warning)
# check if there are zeros (warning)
```

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

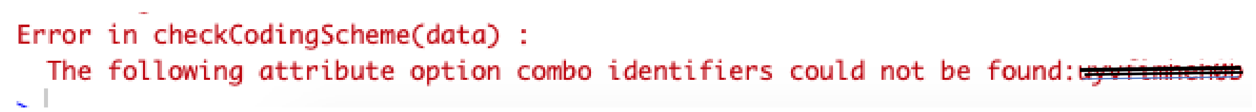
If there are no validation errors in the file, you will see the following message displayed in the console.



```
Downloading GitHub repo jason-p-pickering/datim-validation@master
from URL https://api.github.com/repos/jason-p-pickering/datim-validation/zipball/master
Installing datimvalidation
'/Library/Frameworks/R.framework/Resources/bin/R' --no-site-file --no-environ --no-save --no-restore --quiet CMD INSTALL  \
  '/private/var/folders/rb/1v2qh1fn1cjdyqx8tx3glr4c0000gp/T/RtmpjguURQ/devtools5233e95b509/jason-p-pickering-datim-validation-29fdf1b'  \
  --library='/Library/Frameworks/R.framework/Versions/3.4/Resources/library' --install-tests

* installing *source* package ‘datimvalidation’ ...
** R
** tests
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** installing vignettes
** testing if installed package can be loaded
* DONE (datimvalidation)
Reloading installed datimvalidation
>
```

Validation 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 in checkCodingScheme(data) :
  The following attribute option combo identifiers could not be found:[redacted]
```

This table shows some of the common error messages and solutions.

Error Message	Addressed by	Solution
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.
Invalid data element/coc pairs	checkDataElementDisaggValidity	Make sure that the category option combo in a specified row is paired with the right data element and vice versa. (Refer to code lists for a specific data set.)
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.
Negative values encountered	checkNegativeValues	Make sure that the file does not contain negative values.
Validation rule violations encountered		
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.

Validation rule violations encountered	validateData	Ensure that the data in the import file meet the validation rules identified in the results of the validation.