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

Data for Accountability, Transparency and Impact Monitoring (DATIM)

Validation of SIMS Data Payloads for DATIM Using R Validation Package

May 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 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.

- **getDataElementMap:** Utility function of extraction of data element ids, codes, shortName and names.
- **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).
- **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.

Getting Started

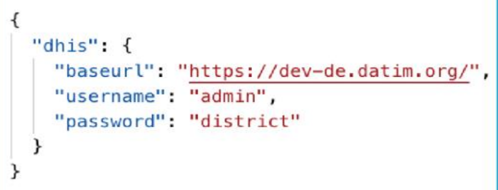
Below is a diagram that depicts steps that need to be followed in order to use the datim validation package to validate import data.

1. Getting started

- Install either RStudio (https://www.rstudio.com) or R console (https://cran.r-project.org). (We prefer RStudio)

2. Create a secrets file

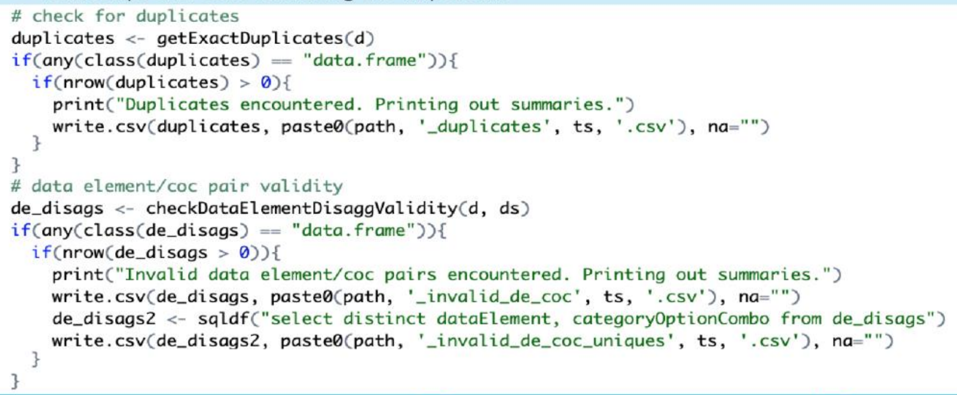
- Create a JSON file for storing DATIM URL and login information
- Specify the URL, user's user name and password and save the file.



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

3. Create the R script that will be calling on datim validation libraries

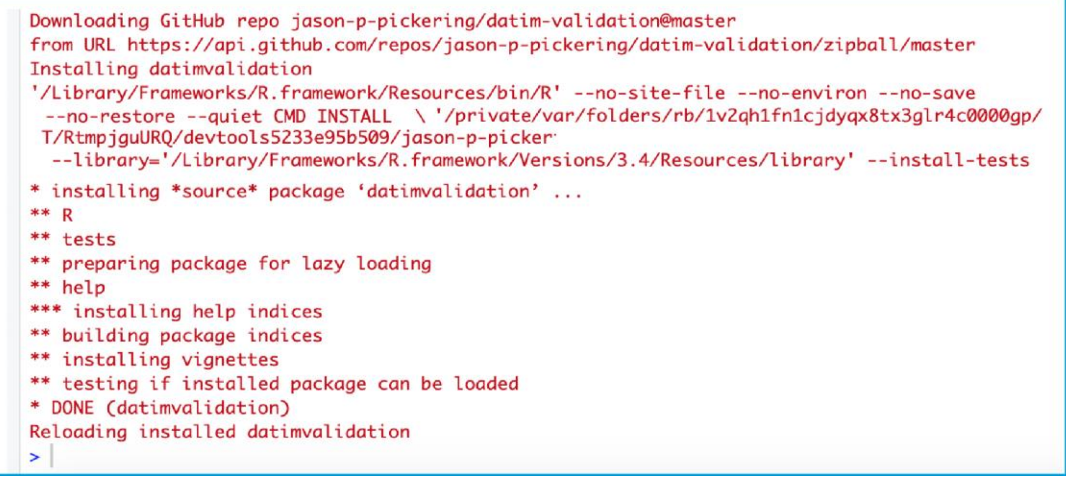
- Create an R script file with the script developed for calling datim validation libraries
- Modify variables according to user's needs
- Run script and start validating the import file



```
# 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="")
  }
}
```

4. Correct validation errors

- Check the RStudio console for the results of validation
- Correct all validation errors (if any) in the import file until there are no more errors



```
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-picker
   --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
>
```

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 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.

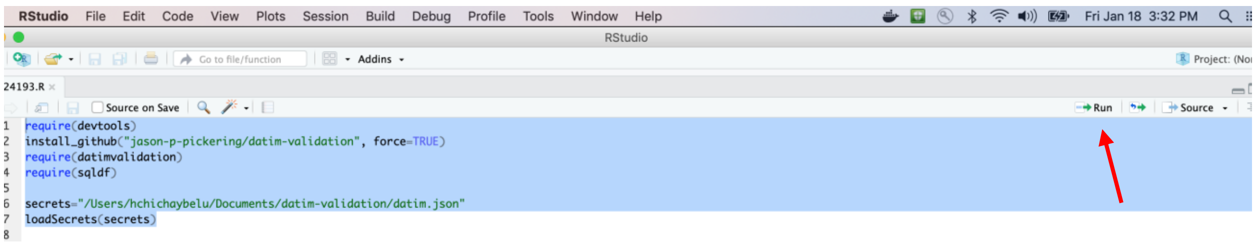
```
require(devtools)
install_github("jason-p-pickering/datim-validation", force=TRUE)
require(datimvalidation)
require(sqldf)

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.

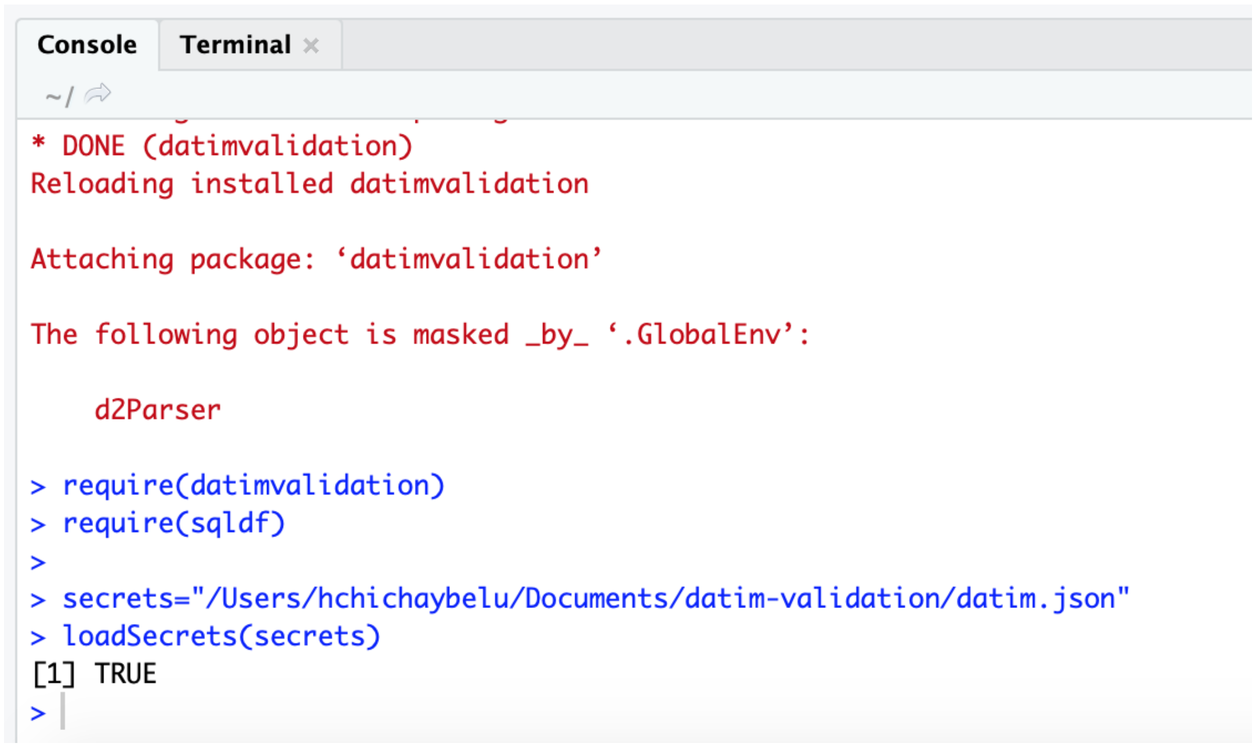
Open your .R file with RStudio.

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



```
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. For SIMS, period is expected to be daily, in YYYYMMDD format. For example, January 27th, 2019 should be coded as 20190127.

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.

```
clearCache(force=TRUE)


dataElementIdScheme <- "code"

orgUnitIdScheme <- "id"

idScheme <- "code"

file_type <- "csv"


#VALIDATE AGENCY DATA

agencyname <- 'CDC'

dir <- "~/icf/projects/datim/SIMS_FY19Q1/"

filename <- "CDC_FY2019Q1.csv"

isoPeriod <- "2019Q1"

type <- "A"

filesHaveHeader <- FALSE

split <- TRUE

validateAgencyData(dir, filename, isoPeriod, type, filesHaveHeader,
split)


processFile <- function(filename, dir, isoPeriod, toolType, out_dir,
filesHaveHeader){


  file_summary <- c()

  file_summary["file"] <- filename


  path <- paste0(dir, filename)


  excludeInvalidOU <- FALSE

  excludeInvalidMechs <- TRUE
  file_summary <- c()


  file_summary["file"] <- filename


  print(paste("checking ", filename, toolType))

  if(toolType == "A" ){

    dataSets <- c("O392zMXtwar")  # sims 4.0 above site

    file_summary["type"] <- "above site"

  } else if(toolType == "S" ){

    dataSets <- c("rnETToFucnJ9")  # sims 4.0 site

    file_summary["type"] <- "site"

  }


  # parse using regular parser, used to identify period shifts and
overlapping assessments

  d <- d2Parser(file = path, type = file_type, dataElementIdScheme =
dataElementIdScheme, orgUnitIdScheme = orgUnitIdScheme, idScheme =
idScheme, invalidData = TRUE)


  # parse using SIMS parser

  d2 <- sims2Parser(file=path, dataElementIdScheme =
dataElementIdScheme, orgUnitIdScheme = orgUnitIdScheme, idScheme =
idScheme, invalidData=TRUE, hasHeader=filesHaveHeader,
isoPeriod=isoPeriod)


  file_summary["record count"] = length(d2$comment)

  file_summary["assessment count"] = length(unique(d2$comment))


  # identify overlapping assessments

  overlapping_assessment <- sqldf('select period, orgUnit,
attributeOptionCombo, count(distinct(storedby)) as assessment_count
```

```
from d group by period, orgUnit, attributeOptionCombo having
count(distinct(storedby)) > 1')

  if(nrow(overlapping_assessment) != 0) {

    write.csv(overlapping_assessment,file=paste0(out_dir, filename,
"_overlapping_assessment.csv"))



    # list assessments

    overlapping_assessment_list <- sqldf('select distinct d.period,
d.orgUnit, d.attributeOptionCombo, d.storedby from d join
overlapping_assessment o on d.period=o.period and d.orgUnit=o.orgUnit
and d.attributeOptionCombo = o.attributeOptionCombo')

    write.csv(overlapping_assessment_list,file=paste0(out_dir,
filename, "_overlapping_assessment_list.csv"))

  }

  file_summary["overlapping PE/OU/IM count"] =
length(overlapping_assessment$period)



  # identify period shifts

  d_unique = sqldf('select period, storedby from d group by period,
storedby')

  d2_unique = sqldf('select period, comment from d2 group by period,
comment')

  shifts_made = sqldf('select comment as assessment, d_unique.period
as old_period, d2_unique.period as new_period from d_unique join
```

```
d2_unique on d_unique.storedby = d2_unique.comment where
d_unique.period != d2_unique.period order by old_period')

  if(nrow(shifts_made) != 0)
write.csv(shifts_made,file=paste0(out_dir, filename,
"_shifts_made.csv"))

  file_summary["shifted_assessment_count"] = nrow(shifts_made)



  # identify any exact duplicates left (USAID had duplicates with same
assessment id)

  post_shift_duplicates <- getExactDuplicates(d2)



  # used to produce post-shift duplicates with codes

  de_map <- getDataElementMap()



  post_shift_duplicates_w_code <- sqldf('select de_map.code,
post_shift_duplicates.* from  post_shift_duplicates left join de_map
on de_map.id = post_shift_duplicates.dataElement order by dataElement,
period, orgUnit, attributeOptionCombo')

  if(nrow(post_shift_duplicates_w_code) != 0)
write.csv(post_shift_duplicates_w_code,file=paste0(out_dir, filename,
"_post_shift_duplicates.csv"))

  file_summary["post shift duplicate count"] =
length(post_shift_duplicates_w_code$comment)



  # verify mechanism validity

  mechs <- checkMechanismValidity(d2)

  if(any(class(mechs) == "data.frame")){

    if(nrow(mechs) != 0){

      mech2 <- sqldf("select mechs.*, m2.comment as assessment_id from
mechs join (select distinct period, attributeOptionCombo, comment from
d2) m2 on mechs.period = m2.period and mechs.attributeOptionCombo =
m2.attributeOptionCombo")

      write.csv(mech2,file=paste0(out_dir, filename, "_mechs.csv"))

    }

    file_summary["invalid period mechanisms"] =
length(mechs$attributeOptionCombo)

  } else {

    file_summary["invalid period mechanisms"] = 0

  }



  # verify for bad data value

  bad_data_values <- checkValueTypeCompliance(d2)

  if(nrow(bad_data_values) != 0)
write.csv(bad_data_values,file=paste0(out_dir, filename,
"_bad_data_values.csv"))

  file_summary["bad data values"] =
length(bad_data_values$dataElement)





  invalid_orgunits <- checkDataElementOrgunitValidity(data=d2,
datasets=dataSets)

  if(any(class(invalid_orgunits) == "data.frame")){

    if(nrow(invalid_orgunits) > 0){



      invalidOUs <- sqldf('select distinct orgUnit from
invalid_orgunits')

      invalidOUAssessments <- sqldf('select comment as assessment_id,
period, orgUnit from d2 where orgunit in (select orgUnit from
invalidOUs) group by comment, period, orgUnit')

      if(nrow(invalid_orgunits) != 0) {

        write.csv(invalid_orgunits,file=paste0(out_dir, filename,
"_invalid_orgunits.csv"))

        write.csv(invalidOUAssessments,file=paste0(out_dir, filename,
"_invalid_orgunit_list.csv"))
```

```
      }

      file_summary["invalid org units"] = length(invalidOUs$orgUnit)

      file_summary["invalid ou assessments"] = 
length(invalidOUAssessments$orgUnit)

    } else {

      file_summary["invalid org units"] = 0

      file_summary["invalid ou assessments"] = 0

    }

  } else {

    file_summary["invalid org units"] = 0

    file_summary["invalid ou assessments"] = 0

  }


  #filter out invalid mechanisms from output

  if(any(class(mechs) == "data.frame")){

    if(excludeInvalidMechs && nrow(mechs) != 0){

      d2 <- subset(d2,!(attributeOptionCombo %in% 
mechs$attributeOptionCombo))

    }

  }


  # filter out invalid OUs

  d2_wo_invalidOU <- d2

  if(excludeInvalidOU){

    d2_wo_invalidOU <- subset(d2,!(orgUnit %in% invalidOUs$orgUnit))

  }


  file_summary["final record count"] = length(d2_wo_invalidOU$comment)

  file_summary["final assessment count"] = 
length(unique(d2_wo_invalidOU$comment))
```

```
  write.table(as.data.frame(file_summary), file = paste0(out_dir,
filename, "_summary.txt"))



  # write out normalized data

  write.csv(d2_wo_invalidOU[,
c("dataElement","period","orgUnit","categoryOptionCombo","attributeOpt
ionCombo","value", "storedby", "timestamp", "comment")],
paste0(out_dir, filename, "_normalized.csv"), row.names=FALSE, na="")

  #   all_data <<- rbind(all_data, d2_wo_invalidOU)


  return(file_summary)

}


validateAgencyData <- function(dir, filename, isoPeriod, type,
filesHaveHeader, split){


  out_dir <- paste0(dir, "out_", format(Sys.time(), "%y%m%d%H%M%S"),
"/")

  dir.create(file.path(out_dir), showWarnings = FALSE)


  # list of files in csv format. header: path, isoPeriod (calendar
period), facility (TRUE/FALSE)

  file_list_path <- paste0(dir, file_list)

  files <- read.csv(file_list_path)


  df <- NULL

  toolType <- type


    summary <- processFile(filename, dir, isoPeriod, toolType,
out_dir, filesHaveHeader=filesHaveHeader)

    df <<- rbind(summary, df)


    if(split){

      print(paste("splitting"))

      header <- c("de","pe", "ou", "coc", "aoc", "value", "comment")

      types <- c("A", "F", "C");
      f <- paste0(dir, filename)

      d <- read.csv(f, stringsAsFactors = FALSE, header =
filesHaveHeader)

      colnames(d) <- header

      #CS_ASMT_TOOL_TYPE: 2 = C,  3 = A, 1 = F

      assessment_types <- sqldf("select comment, case when value = '1'
then 'F' else case when value = '2' then 'C' else case when value =
'3' then 'A' else '' end end end as type from d where de =
'SIMS.CS_ASMT_TOOL_TYPE'")


      s <- 0

      for(i in 1 : length(types)){

        t <- types[i]

        d2 <- sqldf(paste0("select * from d where comment in (select
distinct(comment) from assessment_types where type = '", t, "')"))

        print(paste(t, nrow(d2)))

        if(nrow(d2) == 0){

          print("skipping")

          next

        }

        f2 <- paste0(filename, "_", t, ".csv")

        f2_path <- paste0(dir, f2)

        write.csv(d2, f2_path, row.names = FALSE)

        s <- s + nrow(d2)

        summary <- processFile(f2, dir, isoPeriod, t, out_dir,
filesHaveHeader=filesHaveHeader)

        file.remove(f2_path) #delete split file after processing

        df <<- rbind(summary, df)

      }

      print(paste("input file:", nrow(d), "sum of splits:", s))

    }

  row.names(df) <- 1:nrow(df)

  df <- as.data.frame(df)

  write.csv(df,file=paste0(out_dir, "_summary.csv"))


  return ()

}
```

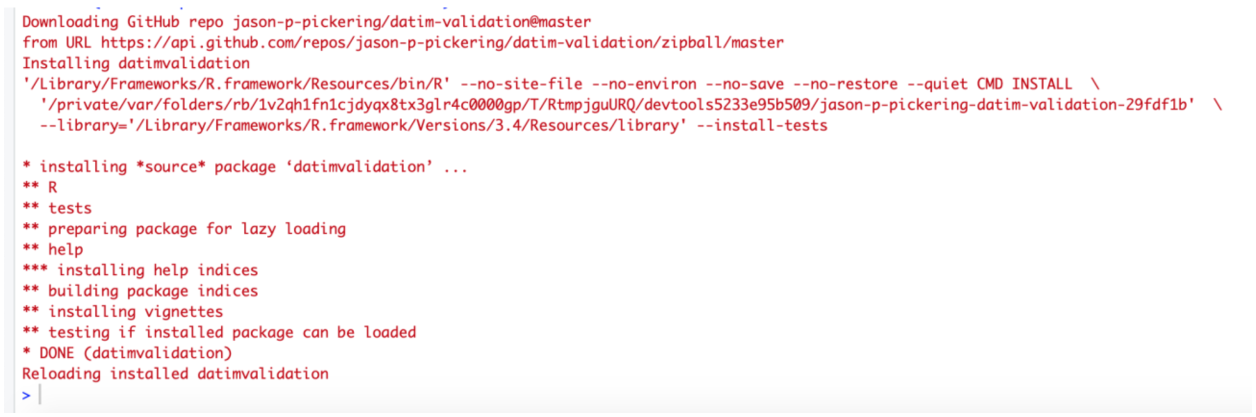
```
writeoutChunked <- function(data, outputFolder, recordCount){
  #chunk data into 200K partitions
  chunk <- recordCount
  n <- nrow(data)
  r  <- rep(1:ceiling(n/chunk),each=chunk)[1:n]
  d <- split(data,r)

  #spit out partitions as csv files
  for (name in names(d)) {
    print(paste(name, nrow(d[[name]])))
    file <- paste0(outputFolder, "partitition_", name, ".csv")
    write.csv(d[[name]], file, row.names=FALSE, na="")
  }
  return ()
}


processAgency <- function(data, mechanisms, agencyname, path){
  print(paste("processing ", agencyname))
  #get agency mechs
  agency_mechs <- subset(mechanisms,(agency == agencyname))

  print(paste(nrow(agency_mechs), "agency mechanisms"))

  #get data by mechanism
  d <- sqldf("select a.* from data a join agency_mechs m on
a.attributeoptioncombo = m.uid")
  print(paste(nrow(d), "records"))


  dir.create(file.path(path, "TO_DELETE"))
  writeoutChunked(d, paste0(path, "/TO_DELETE/") , 200000)
  return()
}
```

Replace values of the following variables with your values.

- dataElementIdScheme – code or id
- orgUnitIdScheme - code or id
- idScheme - code or id
- file_type – csv/json/xml/adx/pdf
- agencyname – DOD/CDC/USAID…
- dir – directory where the file is located
- filename – file name including file extension
- isoPeriod – YYYYQ1/2/3/4
- type – A/S/F/C/(A/C). A- Above site, S- Site, F- Facility, C- community, A/C- Above site/community
- filesHaveHeader – TRUE/FALSE

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

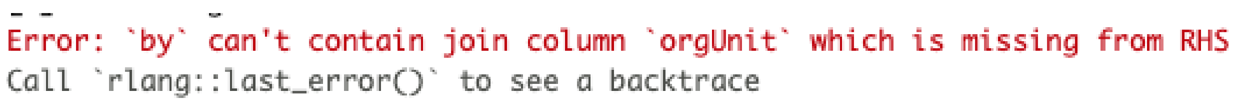
If there are no validation errors in the file, you will see the following message displayed in the console.



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

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.
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
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.