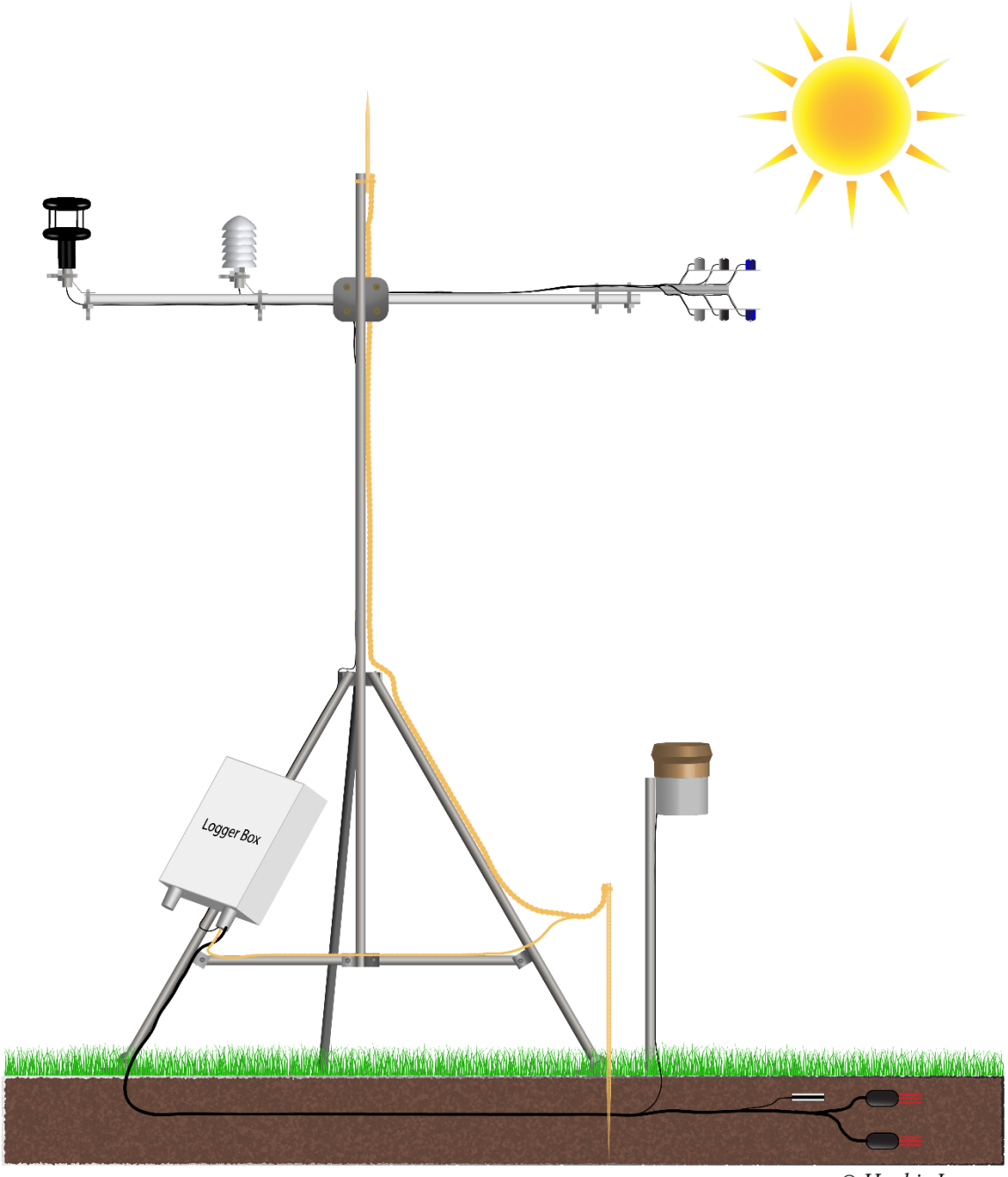
Climate Monitoring Guide

Data management

Author: Dr. Hughie Jones[1,2]
[1] Alexis Nakota Sioux Nation, Treaty 6 Territory
[2] Institute for Resources, Environment and Sustainability, University of British Columbia, Vancouver, British Columbia, Canada
Email: hughie.jones@alumni.ubc.ca



© Hughie Jones

Table of Contents

List of Tables

Table of Figures

Introduction

1 Data management and analysis

Introduction

This section focuses on managing climate data files downloaded from a site datalogger. Overall, this section will describe:

1. where to locate data files downloaded from the site data logger (i.e., data repository)
2. programming script for organizing and visualizing data files in a project database (i.e., hierarchical database).

The example in this section will use Campbell Scientific Inc. data files (.dat) downloaded from the logger shown in Section 3 of the guide (**ICBCM_Climate_Station_Guide_3_installation**) to demonstrate the data management workflow. More specifically, the goals of this section are to introduce:

1. Programming languages (**Table 1**; commonly used for statistics, data management and analysis)
2. Database management system (e.g., data repository and hierarchical database)
3. Programming script for (written for R Project):
    - constructing a hierarchical time-series database
    - fetching and organizing climate data into a hierarchical database
    - graphing the data files in the hierarchical database

Although, there are many open and closed sources statistical programming languages available for project teams, this guide will provide programming script written for R Project.

1.1 Programming languages

Programming languages are a powerful tool in data management and analytics, because they allow for:

1. Repeatability
2. Reliability
3. Consistency
4. Automation
5. Efficiency
6. Accuracy

Table 1. Selection of common open and closed source (i.e., proprietary) programming languages.

Programming language	Open or closed source	Link
MATLAB	Closed	https://www.mathworks.com/products/matlab.html
R Project	Open	https://www.r-project.org/
Python	Open	https://www.python.org/
Julia	Open	https://julialang.org/

C	Open	https://www.microsoft.com/en-ca/p/c-c-c-programming/9nblggh08lts?activetab=pivot:overviewtab
Java	Open	https://www.java.com/en/
SAS	Closed	https://www.sas.com/en_ca/home.html
Scala	Open	https://scala-lang.org/
F#	Open	https://fsharp.org/

1.2 Data organization strategy

The function of the data management strategy is to compartmentalize data storage and data analysis. For example, raw data downloaded from a site datalogger should be stored, and remain unaltered, from its original state (in a data repository) and data analysis and graphing should be performed on a separate dataset (a database). The database management system (Fig. 1) in this section of the guide consists of:

1) data repository (data storage)
2) hierarchical database (data analysis and graphing)

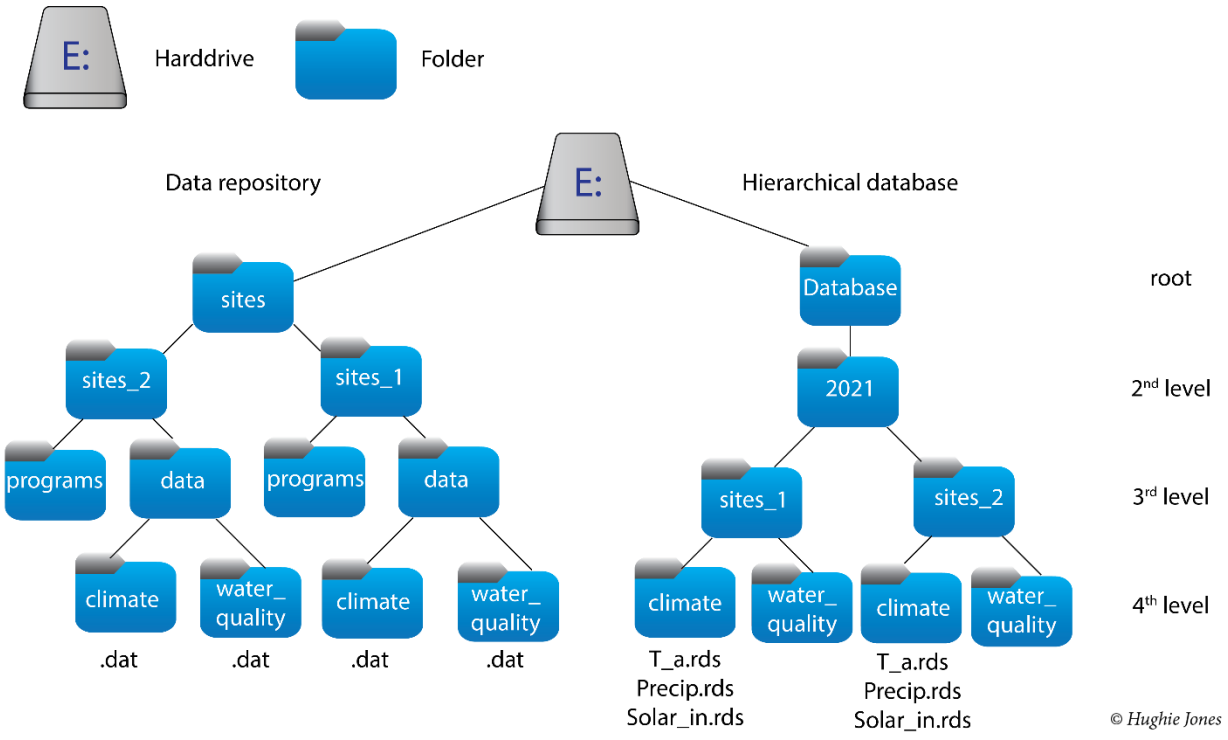


Fig. 1. An illustration of a local hard drive (E:) with a data repository and a hierarchical database folder system (root, 2nd level, 3rd level and 4th level).

1.2.1 Data repository

A data repository is an isolated dataset which will serve as the main archive of raw (i.e., unaltered) data files (i.e., .csv, .dat, .text) collected during a project life-span (**Fig. 2**). The creation of data repository folders can be automated using a programming language (e.g., R Project). In this example they are created manually.

For this example:

1. Create a data repository for "site_1", example: "E:\sites\site_1\data\climate"
2. Copy a data file into the folder, example: "site_1_CR1000X_Climate.dat"

The individual who downloads data from the climate monitoring site ensures that all "site_1" data files are copied into the site data repository, and NO other files are copied there (**Fig. 2**).

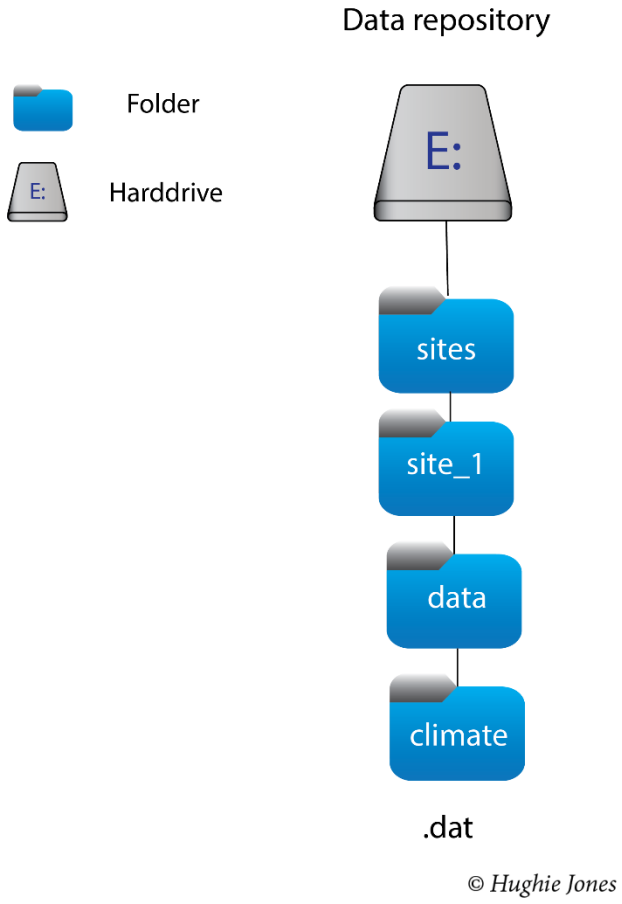


Fig. 2. An illustration of the data repository where all data files will be stored, immediately after they are collected from the climate monitoring site (i.e., site_1). The data files, in the respository are never deleted or altered.

1.2.2 Hierarchical database

A hierarchical database is a data management model which organizes data in tree-like structure (**Fig. 3**). In the database shown in **Fig. 3**, the data files (i.e., .rds) were created from data files stored in the data repository.

Now that the data repository is created and it contains a data file (Section 1.2.1), use the programming scripts

1. db_create.R to create a database
2. db_populate.R to populate the database with data within "site_1_CR1000X_Climate.dat"
3. db_plots.R to plot the data within "site_1_CR1000X_Climate.dat"

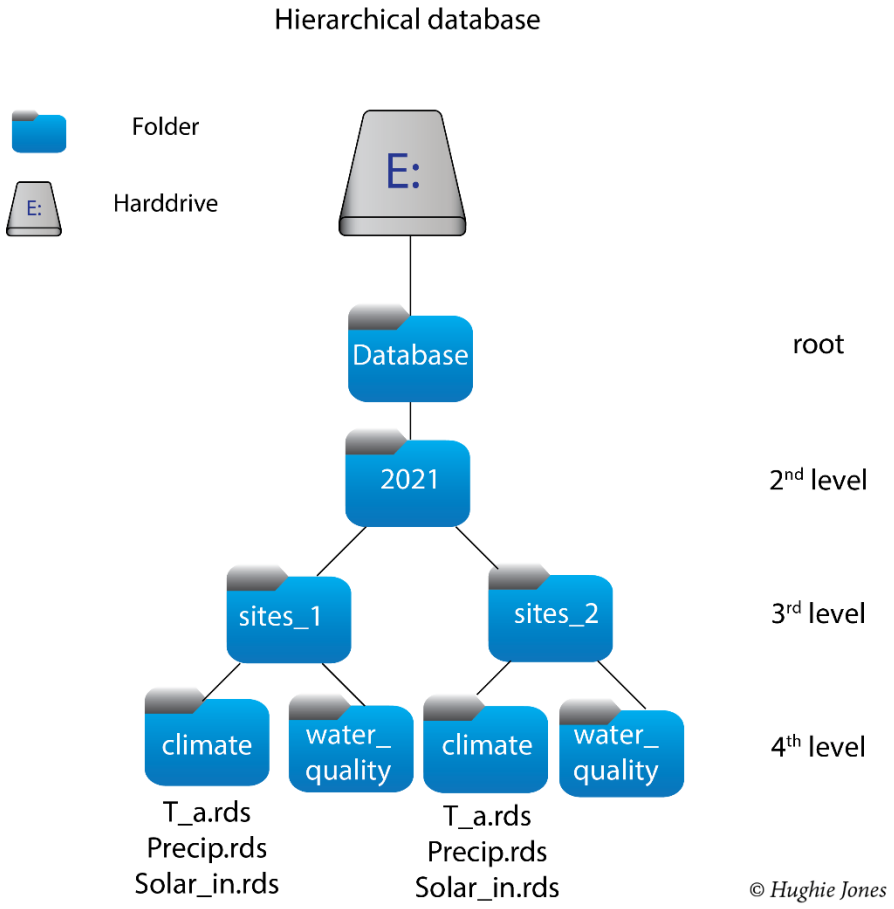


Fig. 3. An illustration of the hierarchical database where all data files will be stored, immediately after they are collected from the climate monitoring site (i.e., site_1).

1.2.2.1 db_create.R

This programming function runs in R Project.

This script will create:

1. a hierarchical database in a location specified by the user
2. a time vector file "clean_tv.rds" which has a constant time interval, specified by the user.

INPUTS

db_dir – character string. The database root (e.g., "E:/database")

years_in –integer input. The second level of the database is year. A single year can be created (e.g., 2021) or multiple years (e.g., 2000:2021).

sites_in – character string. The third level of the database is site. A single site can be created (e.g., c("site_1")) or multiple sites (e.g., c("site_1", "site_2", "site_3")).

data_cat – character string. The fourth level of the database is data category. A single data category can be created (e.g., c("climate")) or multiple data categories (e.g., c("climate", "water_quality").

time_int – integer. The time interval of the data inside the data repository files, "site_1_CR1000X_Climate.dat" (e.g., 30 min). The time interval must be given in seconds. For example, if the data repository files have a 30-minute interval time_int is equal to 1800 (i.e., time_int = 1800). This value will be used to create a time vector (i.e., clean_tv.rds) in the fourth level of the database.

time_zn – character strong. The time zone your data was collected (e.g., "MST" for Mountain Standard Time)

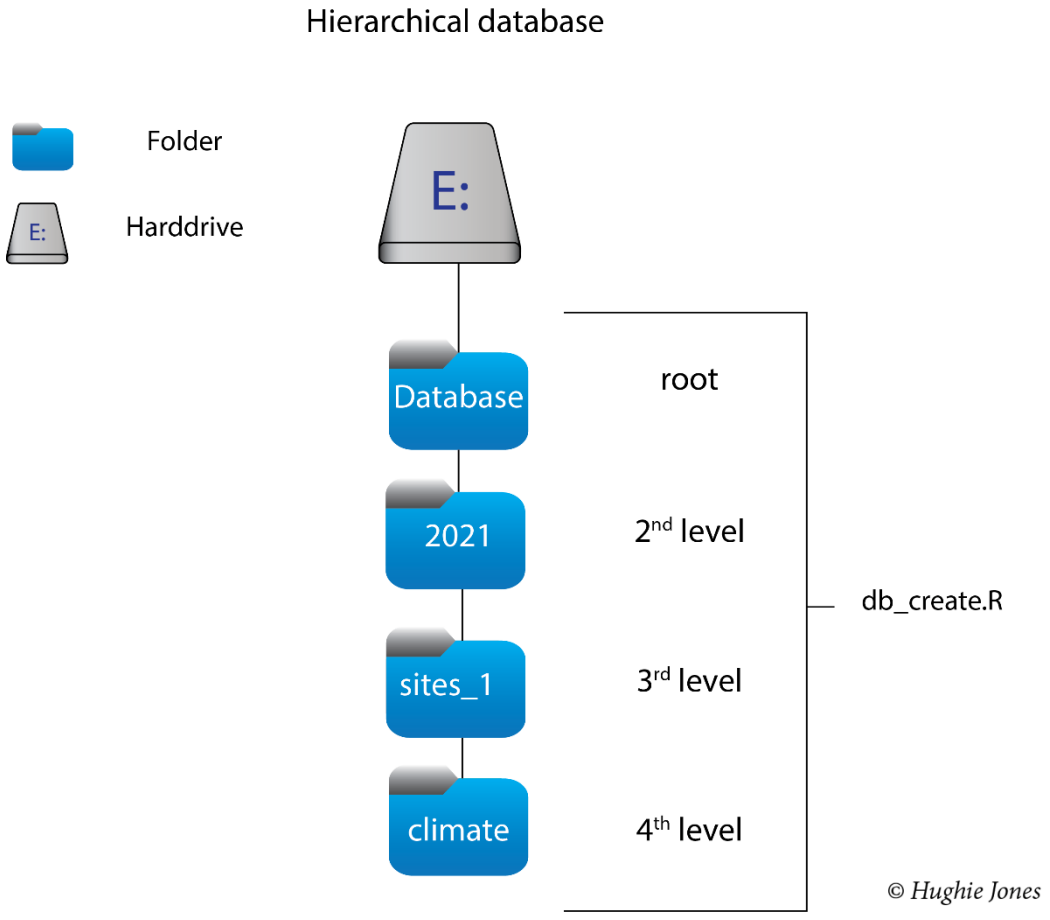


Fig. 4. Illustration of the hierarchical database created using the db_create.R function (run in R Project). The input required to recreate this example is: db_create("E:/database/",2020,c("site_1"),c("climate"),1800,"UTC"). The input required to recreate **Fig. 3** is: db_create("D:/database/",2021,c("site_1","site_2"),c("climate"," water_quality"),1800,"UTC")

```
# Function name: db_create.R
#created in R Project date (yyyy/mm/dd) - 2020/11/09
#author: Hughie Jones
#last changed: 2021/03/31

# Script description:

# This programming script runs in R Project.
# This script will create:
#   1.  a hierarchical database in a location specified by the user
#   2.  a time vector file "clean_tv.rds" which has a constant time interval,
#       specified by the user.

# INPUTS
#****************************************

# db_dir - character string. The database root (e.g., "D:/database")

# years_in - integer input. The second level of the database is year.
#            A single year can be created (e.g., 2021) or multiple years (e.g.,
2000:2021).

# sites_in - character string. The third level of the database is site.
#            A single site can be created (e.g., c("Site_1"))
#            or multiple sites (e.g., c("Site_1", "Site_2", "Site_3")).

# data_cat - character string. The fourth level of the database is data category.
#            A single data category can be created (e.g., c("climate"))
#            or multiple data categories (e.g., c("climate", "water_quality").
#
time_int - integer. The time interval of the data inside the data repository files,
"Site_1_CR1000X_Climate.dat" (e.g., 30 min). The time interval must be given in
seconds. For example, if the data repository files have a 30-minute interval time_int
is equal to 1800 (i.e., time_int = 1800). This value will be used to create a time
vector (i.e., clean_tv.rds) in the fourth level of the database.
# time_zn - character strong. The time zone your data was collected (e.g., "MST" for
Mountain Standard Time)
#****************************************
#created in R Project date (yyyy/mm/dd) - 2020/11/09
#author: Hughie Jones
#last changed: 2020/11/09

#e.g.,
# db_create("D:/database/",2020,c("site_1"),c("climate"),1800,"UTC")

  db_create <- function(db_dir,
                        years_in,
                        sites_in,
```

```
                        data_cat,
                        time_int,
                        time_zn){

#required packages
    library(stringr)
    library(chron)

#create database directory unles it already exists
if(isFALSE(dir.exists(db_dir))) {

  dir.create(db_dir)

  }

# db_year <- str_sub(years_in)
#
# db_sites <- sites_in
#
# db_data_cat <- data_cat

#smallest time interval of database
time_int_cat = "sec"

#time interval of .dat filesa and time vector (clean_tv.rds)
time_int_num = as.character(time_int)

#combine number of seconds with units
time_int <- paste0(time_int_num," ",time_int_cat)

#create time vector label
time_lab <- c("clean_tv")

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

  #get current year
  year_now <- years_in[i]
  #create name of second level of database
  db_dir_year <- paste0(db_dir,year_now,'/')
  #create second level
  dir.create(db_dir_year)
  #create start tv stamp of the current year
  year_start = as.POSIXct(paste0(year_now,"-1-1 00:00:00"), tz = time_zn)
  #create end tv stamp of the current year
  year_end = as.POSIXct(paste0(as.numeric(year_now)+1,"-1-1 00:00:00"), tz = time_zn)-
as.numeric(time_int_num)
  #create time vector
  clean_tv <- seq(year_start, year_end, time_int)
```

```
    for(j in 1:length(sites_in)){
      #create name of third level of database
      db_dir_year_sites <- paste0(db_dir_year,sites_in[j],'/')
      #create third level
      dir.create(db_dir_year_sites)

        for(k in 1:length(data_cat)){
        #create name of fourth level of database
        db_dir_year_sites_cat <- paste0(db_dir_year_sites,data_cat[k],"/")
        #create fourth level
        dir.create(db_dir_year_sites_cat)
        #create final output path (full directory and file name)
        exp_pth = sprintf("%s%s.rds",db_dir_year_sites_cat,time_lab)
        #save the file
        saveRDS(clean_tv, file = exp_pth)

        }
    }
  }
}
```

1.2.2.2 db_populate.R

This programming function runs in R Project.

This script will:

1. Fetch the data files in the data repository, one at a time, and create a data table in R Project Environment which contains the data.
2. For each column in the data table an empty vector is created and populated with the data within each column. If the file exists in the fourth level of the database created using db_create.R the script will fetch the existing file and append that file (i.e., add to the existing file). The vector created will have a length equal to "clean_tv.rds".

INPUTS

input_folder – character string. Data repository folder location (e.g., "E:/ all_sites/site_1/data/climate/")

output_folder – character string. database root directory (e.g., "E:/database/")

site_in - character string. name of site in database (e.g., "site_1")

data_cat - character string. name of data category in site (e.g., "climate")

tv_col_num - integer. column in data table (data_all) where the time stamps are located (e.g., 1)

data_row_start - integer. row in data table (data_all) where the data starts (e.g., 4).

label_row - integer. the row in the data table (data_all) where the data labels are located

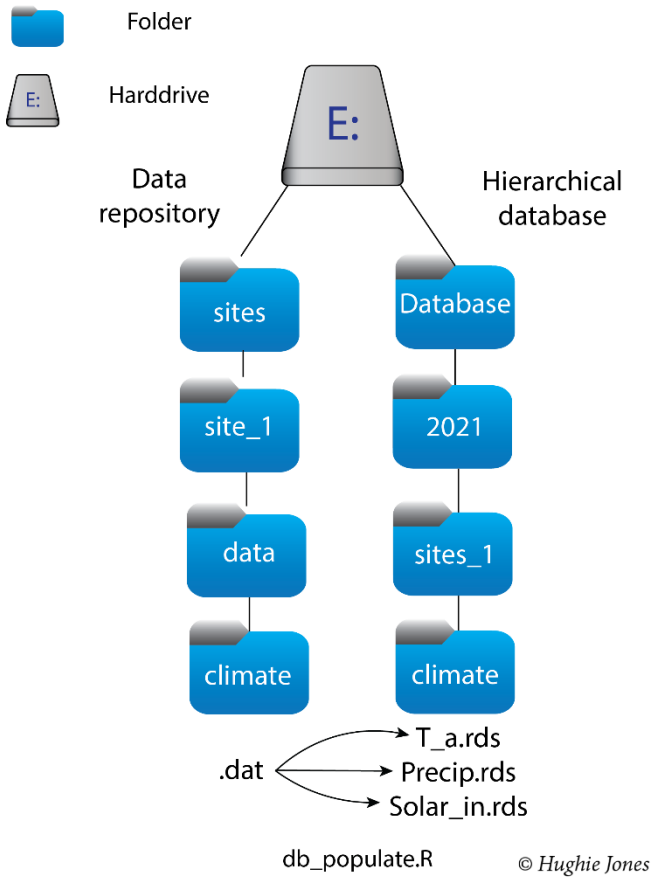


Fig. 5 Illustation of the fetching and reorganization of data in the data repository into the hierarchical database. The input required to recreate this example is db_populate("E:/sites/site_1/data/climate/","E:/database/",c("site_1"),"climate").

```
# Function name: db_populate.R
#created in R Project date (yyyy/mm/dd) - 2020/11/09
#author: Hughie Jones
#last changed: 2021/03/31

# Script description:

# 1.  Fetch the data files in the data repository, one at a time,
#     and create a data table in R Project Environment which contains the data.
# 2.  For each column in the data table an empty vector is created
#     and populated with the data within each column.
#     If the file exists in the fourth level of the database created
#     using db_create.R the script will fetch the existing file
#     and append that file (i.e., add to the existing file).
#     The vector created will have a length equal to "clean_tv.rds".


#   This function will organize raw INPUT data files and OUTPUT data
#   into a hierarchical database which must be created previous to
#   running this function using 'db_create.R'. The user of this function
#   must place all raw data files in an INPUT folder (input_folder)
#   for a specific project data collection site (site) and data category (data_cat).
#   In addition, the user must specify the details regarding the OUTPUT folder
#   including the database directory (db_dir), site name (site) and data category
#   that describes the data (data_cat).

# *NOTE*

#     The function 'db_create.R' must be used create the database folder which
#   will be populated by this function. For the example below,
"C:/database_new/yyyy/site_1/clim/"
#   must be created before running this function. The user is not required to
#   specific the year (i.e., 'yyyy') for the OUTPUT folder because
#   this function will discover 'yyyy' within the raw INPUT data files.

# INPUTS
#****************************************

# input_folder -  Character string. input raw data (i.e., .dat file) directory (e.g.,
"C:/ all_sites/site_1/raw_data/climate/")

#             only files associated with the site and data category used below
#             should be in this folder
```

```
# output_folder -  Character string. output database root directory (e.g.,
"C:/database_new/")

#             this directory should be created using 'db_create.R'
#             previous to running this function.

# site -  Character string. name of site in db_dir (e.g., "site_1")

#             this site name will be appended to db_dir ("C:/database_new/site_1/")
#             and must exist OR be created using 'db_create.R' previous to running
this function.

# data_cat -  Character string. name of data category in site (e.g., "clim")

#             this data category name will be appended to db_dir and
#             site ("C:/database_new/site_1/clim/") and must exist OR
#             be created using 'db_create.R' previous to running this function.

# tv_col_num - integer. column in data table (data_all) where the time stamps are
located (e.g., 1)
#
# data_row_start - integer. row in data table (data_all) where the data starts (e.g.,
4).
#
# label_row - integer. the row in the data table (data_all) where the data labels are
located

#*****************************************

# e.g.,
# db_populate("D:/Sites/site_1/data/Climate/","D:/database/","site_1","climate")
# db_populate("D:/Sites/","D:/database/","site_1","climate")


db_populate <- function(input_folder,
                        output_folder,
                        site_in,
                        data_cat,
                        tv_col_num,
                        data_row_start,
                        label_row) {

  # get current system time
  time_start <- Sys.time()

  # required packages
  library(stringr)
```

```
library(chron)

#loop through sites
for (i in 1:length(site_in)){

  #current site
  site_now = site_in[i]

  #loop through data categories
  for(ii in 1:length(data_cat)){

    #current data category
    data_cat_now = data_cat[ii]

    # create full input path with site and data category
    full_inuput_fold = paste0(input_folder,site_now,"/data/",data_cat_now,"/")

    #get the list of .dat files in the pull input path
    input_file_names = list.files(full_inuput_fold)

    #loop through the input files
    for (j in 1:length(input_file_names)) {

      #get system time
      time_file_start <- Sys.time()

      #current file name
      file_now <- input_file_names[j]

      #create full path for indivdual file
      full_pth <- paste0(full_inuput_fold, file_now)

      #Read logger information in first row of .dat file
      logger_ID <- read.table(full_pth, sep = ",", nrows = 1)

      #Read remaining rows of .dat file (excluding logger information)
      data_all <- read.table(full_pth,
                             sep = ",",
                             header = FALSE,
                             skip = 1)

      #get the dimensions of the .dat file (excluding logger information)
      data_dim <- dim(data_all)

      # get a list of all the time stamps in the current file
      data_tv_list <- as.POSIXct(data_all[data_row_start:data_dim[1], tv_col_num])

      # get a list of all the years in the current file
```

```
        yearin_list <- substr(data_tv_list, 1, 4)

        #get the final occurance of all unique years
        last_year = length(yearin_list) -
          match(unique(yearin_list), rev(yearin_list)) + 1

        # find the number of unique years
        last_year_len = length(last_year)

        #add the number of rows before the first data row
        last_year_len_real = last_year + (data_row_start-1)

        #loop through the rows
        for (jj in data_row_start:data_dim[1]) {

          #get time stamp for jj row
          data_tv <- data_all[jj, tv_col_num]

          #get the year from jj row
          yearin <- substr(data_tv, 1, 4)

          #create the output folder
          output_folder_now <-
             paste0(output_folder, yearin, "/", site_now, "/", data_cat_now, "/")

          #get the clean_tv.rds created using db_create.R
          year_tv_curr <- readRDS(paste0(output_folder_now, "clean_tv.rds"))

          #create calender date with existing clean_tv.rds time zone
          data_tv_new <- as.POSIXct(data_tv,tz = attr(year_tv_curr, "tzone"))

          #find position (index) where clean_tv.rds and the time stamp for jj row math
          ind_time <- which(year_tv_curr == data_tv_new)

          #create empty vector with the dimensions of clean_tv.rds
          year_tv_curr_vec <-
             matrix(0, nrow = length(year_tv_curr), ncol = 1)

          #loop through the columns
          for (k in (tv_col_num+1):data_dim[2]) {

            #get variable name
            vect_name <- paste0(data_all[label_row, k])

            #convert variable name to character
            vect_name_ch <- as.character(vect_name)

            #variable filename
```

```
            vect_name_file <- paste0(data_all[label_row, k], ".rds")

            #variable file folder output location
            output_folder_vector <-
              paste0(output_folder_now, vect_name_file)

            #get individual value with specific row and column
            now_data <- as.vector(data_all[jj, k])

            #convert individual value to numeric
            now_data <- as.numeric(now_data)

            #place the individual value in the location in the empty vector
            #the location matches where clean_tv.rds
            # and the time stamp for jj row math
            year_tv_curr_vec[ind_time] <- now_data


            # find if variable DOES EXIST in the output_folder_now
            # and DOES EXIST in the global environment.
            if (isTRUE(file.exists(output_folder_vector)) &
                isTRUE(exists(vect_name_ch))) {

              temp_str <- paste0(vect_name_ch, "[ind_time] <- now_data")

              eval(parse(text = temp_str))

            }
            # find if variable DOES NOT EXIST in the output_folder_now
            # and DOES EXIST in the global environment.
            else if (isFALSE(file.exists(output_folder_vector)) &
                     isTRUE(exists(vect_name_ch))) {

              temp_str <- paste0(vect_name_ch, "[ind_time] <- now_data")

              eval(parse(text = temp_str))

            }
            # find if variable DOES EXIST in the output_folder_now
            # and DOES NOT EXIST in the global environment.
            else if (isTRUE(file.exists(output_folder_vector)) &
                       isFALSE(exists(vect_name_ch))) {

              ex_file  <- readRDS(output_folder_vector)

              assign(vect_name, ex_file)

              temp_str <- paste0(vect_name_ch, "[ind_time] <- now_data")
```

```
          eval(parse(text = temp_str))

        }
        # find if variable DOES NOT EXIST in the output_folder_now
        # and DOES NOT EXIST in the global environment.
        else if (isFALSE(exists(vect_name_ch)) &
                   isFALSE(file.exists(output_folder_vector))) {

          year_tv_curr_vec <- as.vector(rep(NA, length(year_tv_curr)))

          assign(vect_name, year_tv_curr_vec)

        }

        # find whether jj (row) number has reached ANY of the row values
        # listed in last_year_len_real
        if (isTRUE(any(last_year_len_real == jj))) {

          #create final output path
          pth_exp <- paste0(output_folder_now, vect_name_ch, ".rds")

          #save the file in the output folder location
          saveRDS(eval(parse(text = vect_name_ch)), file = pth_exp)

          #delete the variable from the global environment
          temp_del_var = paste0("rm(",vect_name_ch,")")
          eval(parse(text = temp_del_var))

        }

      }

    }

    # get the time when tbe file is processed
    time_file_end <- Sys.time()

    #calculate the time it took to process the file
    time_file <-
      format(round(time_file_end - time_file_start, 2), nsmall = 2)

    #print the processing time
    message("processed - ", file_now, " (processing time = ", time_file, ")")

  }

  #get the time when all files are processed in the input folder
```

```
      time_end <- Sys.time()

      #calculate the time it took to process all the files in the input folder
      time_file <- format(round(time_end - time_start, 2), nsmall = 2)

      #print the total timeit took to process all the files in the input
      message("total session processing time = (", time_file, ")")

    }
  }
}
```

1.2.2.3 db_plot_all.R

This program will plot all the vector files (.rds files) contained in the fourth level of the database, created using db_create.R.

This programming function will:

1. fetch data file (.rds) in the fourth level of the database (i.e., data_cat) (created using db_populate.R)

2. create a folder called "figures"

3. create and print graphs of each .rds data file (as .tiff files)

   into the "figures folder"

INPUTS

folder_in - character string.

db_dir - character string. database root, or database entry, directory (e.g., "D:/database").

If the folder already exists a warning is given. If the folder does not exist it will be created

year_in - integer. The second level of the database is the year (years_in) (e.g., 2020:2025).

site_in - character string. The third level of the database is the sites (sites_in) (e.g., c("site_1","site_2","site_3")

data_cat - character string. The fourth level of the database if the data category (data_cat) (e.g., c("climate","water_quality"))

mm_start - integer. the start month

mm_end - integer. the start day

dd_start - integer. the end month

dd_end - integer. the end day

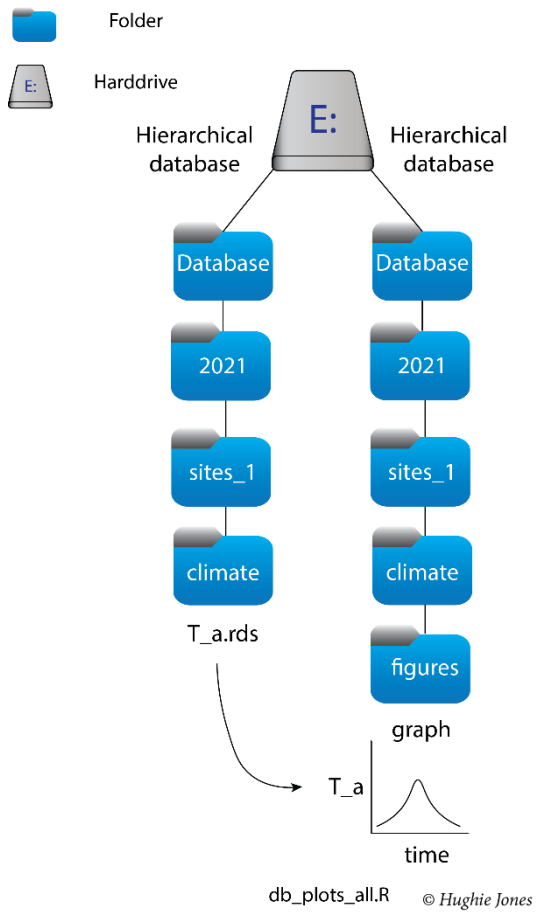


Fig. 6. Illustration of the dataflow/workflow performed by db_plots_all.R**.**

```
# Function name: db_plots_all.R
#created in R Project date (yyyy/mm/dd) - 2020/11/09
#author: Hughie Jones
#last changed: 2021/03/31

# Script description:

#   This function will:
# 1. fetch data file (.rds) in the fourth level of the database (data_cat)
#    (created using db_populate.R)#
# 2. create a folder called "figures"
# 3. create and print graphs of each .rds data file (as .tiff files)
#    into the "figures folder"

# INPUTS
#****************************************

# folder_in - character string.
```

```
# db_dir - character string. database root, or database entry, directory (e.g.,
"D:/database").
#          If the folder already exists a warning is given.
#          If the folder does not exist it will be created

# year_in - integer. The second level of the database is the year (years_in) (e.g.,
2020:2025).

# site_in - character string. The third level of the database is the sites (sites_in)
#           (e.g., c("site_1","site_2","site_3"))

# data_cat - character string. The fourth level of the database if the data category
(data_cat)
#            (e.g., c("climate","water_quality"))

# mm_start - integer. the start month (e.g., 7 for July)

# mm_end - integer. the start day (e.g., 12)

# dd_start - integer. the end month (e.g., 8 for August)

# dd_end - integer. the end day (e.g., 12)
#****************************************

#e.g.,

# db_plots_all("D:/database/",2020,c("ANSN_1"),c("clim"),7,8,12,12)


db_plots_all <- function(db_dir,
                         year_in,
                         site_in,
                         data_cat,
                         mm_start,
                         mm_end,
                         dd_start,
                         dd_end){
  #database directory where .rds files are located
  output_folder_now <- paste0(db_dir, year_in, "/", site_in, "/", data_cat, "/")

  #create a location where figure outputs can be printed to within the
  #database directory
  output_folder_figs <- paste0(output_folder_now,"figures/")

  if(isFALSE(dir.exists(output_folder_figs))) {

    dir.create(output_folder_figs)
```

```
}

#load time vector, clean_tv.rds, created using db_create.R
clean_tv <-readRDS(paste0(output_folder_now,"clean_tv.rds"))

#create the start and end date specified from the inputs
tv_min <- as.POSIXct(paste0(year_in,"-",mm_start,"-",dd_start," 00:00:00"),
                     tz = attr(clean_tv, "tzone"))
tv_max <- as.POSIXct(paste0(year_in,"-",mm_end,"-",dd_end," 00:00:00"),
                     tz = attr(clean_tv, "tzone"))

#find position where clean_tv.rds and the start and end dates match
ind_time_min <- which(clean_tv == tv_min)
ind_time_max <- which(clean_tv == tv_max)

#get list of all .rds files located in the database directory
input_file_names = list.files(output_folder_now,pattern = "rds")

#remove the clean_tv.rds file from the list of files
tv_find <- as.integer(grep("clean_tv.rds", input_file_names))
input_file_names<- input_file_names[-tv_find]

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

  #get current.rds file
  input_file <- input_file_names[i]

  #create full path of vector file
  infold <- paste0(output_folder_now,input_file)

  #read the full path of vector file
  data_now <- readRDS(infold)

  #get name of file without extension (i.e., .rds)
  curr_str = tools::file_path_sans_ext(input_file)

  #create output path
  filename_exp_final = paste0(output_folder_figs,sprintf("%s.tiff",curr_str))

  ###Plotting###
  s_per_day = 86400
  data_max <- round(max(data_now[ind_time_min:ind_time_max],na.rm = TRUE),
                    digits = 1)
  data_min <- round(min(data_now[ind_time_min:ind_time_max],na.rm = TRUE),
                    digits = 1)

  #if there is not data within the specified time period, provide message to user
  if(is.infinite(data_max) || is.infinite(data_min)){
```

```
      print("No values in time period")

      #if there is data within the specified time period, proceed to print the figure
    } else {

      y_offset <- 0.05*(data_max-data_min)
      tiff(file = filename_exp_final)
      par(mar = c(4.1, 4.4, 4.1, 2), xaxs="i", yaxs="i")
      plot(clean_tv, data_now,
           xlab = "Time",
           ylab = input_file, ylim = c(data_min-y_offset, data_max+y_offset),
           xlim = c(clean_tv[ind_time_min],clean_tv[ind_time_max]),type="l",
           xaxt = "n")
      axis(1, at = seq(clean_tv[ind_time_min],clean_tv[ind_time_max],s_per_day),
           labels = seq(clean_tv[ind_time_min],clean_tv[ind_time_max],s_per_day),
           las = 2)

      dev.off()

    }
  }
}
```