#'@name streamclean

#'@title Cleaning raw streaming Dataflow output

#'@description Cleaning raw streaming Dataflow, C6, Eureka-Manta, and YSI-Exo output

#'@param yearmon numeric designation of survey date formatted as yyyymm

#'@param gps character dataset to use for GPS alignment of other data streams. Choice of "df", "eu", or "exo".

#'@param dfmmin integer optional minimum df measurement frequency (# measurements/min)

#'@param c6mmin integer optional minimum c6 measurement frequency

#'@param eummin integer optional minimum eureka (manta) measurement frequency

#'@param exommin integer optional minimum exo measurement frequency

#'@param tofile logical save cleaned output to DF\_FullDataSets?

#'@param sep character optional predesignation of item seperation character in raw data files

#'@param fdir character file path to local data directory

#'@export

#'@importFrom rgdal readOGR

#'@importFrom zoo zoo na.approx

#'@importFrom sp coordinates CRS spTransform

#'@details Dataflow cleaning drops all minutes that have less measurements than "mmin". C6 data is interpolated to match Dataflow. Automatically compares salinity against conducitivty/temperature recalculated salinity and replaces if slope of fit is not close to 1. Bad DO columns must sometimes be removed manually. TODO - Add check the make sure that the year of the data (not just the filename) matches the year of yearmon

#'@examples \dontrun{

#'#old

#'dt <- streamclean(yearmon = 201505, gps = "df", dfmmin = 7, c6mmin = 10,

#' tofile = FALSE)

#'dt <- streamclean(yearmon = 201513, dfmmin = 7, c6mmin = 12,

#' tofile = FALSE, exommin = 60, eummin = 12)

#'

#'#working

#'dt <- streamclean(yearmon = 201512, gps = "df", c6mmin = 6, dfmmin = 7)

#'dt <- streamclean(yearmon = 201601, gps = "eu", eummin = 12)

#'dt <- streamclean(yearmon = 201603, gps = "exo", exommin = 40, c6mmin = 12)

#'}

streamclean <- function(yearmon, gps, dfmmin = NA, c6mmin = NA, eummin = NA, exommin = NA, tofile = FALSE, sep = ",", fdir = getOption("fdir")){

options(warn = -1)

fdir\_fd <- file.path(fdir, "DF\_FullDataSets", "Raw", "InstrumentOutput")

flist <- list.files(fdir\_fd, include.dirs = T, full.names = T)

flist <- flist[substring(basename(flist), 1, 6) == yearmon]

dflist <- list.files(flist, pattern = c("\*.txt"), include.dirs = T, full.names = T)

if(length(dflist) == 0){

dflist <- list.files(flist, pattern = c("\*.TXT"), include.dirs = T, full.names = T)

}

if(length(dflist) == 0){

dflist <- list.files(flist, pattern = c("\*DF.csv"), include.dirs = T, full.names = T)

}

c6list <- list.files(flist, pattern = c("\*C6.csv"), include.dirs = T, full.names = T)

if(length(c6list) == 0){

c6list <- list.files(flist, pattern = c("\*C6.CSV"), include.dirs = T, full.names = T)

}

eulist <- list.files(flist, pattern = c("\*eu.csv"), include.dirs = T, full.names = T)

if(length(eulist) == 0 ){

eulist <- list.files(flist, pattern = c("\*eu.CSV"), include.dirs = T, full.names = T)

}

exolist <- list.files(flist, pattern = c("\*exo.csv"), include.dirs = T, full.names = T)

if(length(exolist) == 0 ){

exolist <- list.files(flist, pattern = c("\*exo.CSV"), include.dirs = T, full.names = T)

}

# if(length(c6list) != length(dflist)){

# warning("Differing numbers of Dataflow and C6 input files")

# }

survey\_days <- unique(sapply(basename(c(dflist, eulist, exolist, c6list)), function(x) substring(basename(x), 1, 8)))

iterate\_days\_load <- function(survey\_days, file\_listing, reading\_function, cleaning\_function){

reslist <- list()

for(i in 1:length(survey\_days)){

day\_data <- reading\_function(file\_listing[i])

day\_data <- cleaning\_function(day\_data)

reslist[[i]] <- day\_data

}

do.call("rbind", reslist)

}

read\_df <- function(dfpath){

#start with comma sep

sep <- ","

dt <- read.csv(dfpath, skip = 0, header = F, sep = sep, strip.white = TRUE)

#switch to tab sep

if(suppressWarnings(nchar(gsub("\t", "", dt[1,])) < nchar(as.character(dt[1,])))){

sep <- "\t"

dt <- read.csv(dfpath, skip = 0, header = F, sep = sep, stringsAsFactors = FALSE)

}

#detect beginning of measurements

fskip <- 1

while(!(!(class(dt[,1]) != "integer") | !(class(dt[,1]) != "numeric"))){

dt <- read.csv(dfpath, skip = fskip, header = F, sep = sep, stringsAsFactors = FALSE)

#print(dt[1,1])

if(!any(!is.na(dt[,1])) | mean(nchar(as.character(dt[,1]))) < 1 | sum(is.na(dt[,1])) > (nrow(dt) / 2) | sum(nchar(gsub("\_", "", as.character(dt[,1]))) - nchar(as.character(dt[,1]))) != 0){

dt <- dt[,-1]

}

fskip <- fskip + 1

if(fskip > 20){

stop(paste("Cannot find beginning of measurements!", dfpath))

}

}

sep <- ","

dt

}

clean\_df <- function(dt){

# print(names(dt))

#remove bad columns

if(class(dt[,3]) == "integer"){

dt <- dt[,-3]

print("removing existing seconds column")

}#remove existing sec column

#remove bad columns of all 0 or NA

if(ncol(dt) > 14){

dtno.na <- dt[complete.cases(dt[,1:12]),]

dt <- dt[,apply(dtno.na, 2, function(x) abs(sum(as.numeric(x), na.rm = T)) != 0)]

}

#temp should never be less than 10, these are likely 'bad' DO columns?

if(mean(as.numeric(dt[,4]), na.rm = T) < 10 & mean(as.numeric(dt[,5]), na.rm = T) < 10){

dt <- dt[,-4:-5]

}

#print(names(dt))

dt <- dt[,apply(dt, 2, function(x) abs(sum(as.numeric(x), na.rm = T)) > 38)]#take out all 0 (22 or 38 is an arbitrary "tolerance" value)

ones <- apply(dt, 2, function(x) sd(as.numeric(x)[as.numeric(x) != 0 & !is.na(as.numeric(x))])) != 0

ones[is.na(ones)] <- TRUE

ones[1:2] <- TRUE

dt <- dt[,ones]

dt[,ncol(dt)] <- trimws(dt[,ncol(dt)])

dt <- dt[,apply(dt, 2, function(x) mean(nchar(x), na.rm = T)) >= 3.0] #(3 is an arbitrary "tolerance" value; accounts for a 3 digit timestamp?)

dt <- dt[,apply(dt[,3:ncol(dt)], 2, function(x) length(unique(x)) != 3)]

names(dt) <- c("date", "time", "chla", "temp", "cond", "sal", "trans", "cdom", "lat\_dd", "lon\_dd")

#MIGHT NEED TO ADD VARIABLES TO THIS, BUT SEEMS LIKE THEY ARE BEING READ ALREADY. FOR EXAMPLE, "bga.pe.rfu", "fdom", AND "ph" ARE NOT LISTED BUT ARE BEING INCORPORATED INTO COMBINED FILE AND INTERPOLATED; MAYBE LIST JUST IN CASE?

#convert factors to numeric

dt <- data.frame(as.matrix(dt))

factorToNumeric <- function(f) as.numeric(levels(f))[f]

#check to make sure that there are any factor class columns

if(any(sapply(dt,class) == "factor")){

dt <- data.frame(sapply(dt, factorToNumeric))

}

#fix lon lat formatting

if(mean(nchar(as.character(round(dt[,"lat\_dd"]))), na.rm = TRUE) != 2){

lat <- dt[,"lat\_dd"]

latdeg <- as.numeric(substr(lat, 0, 2))

latmin <- as.numeric(substr(lat, 3, 8))

dt[,"lat\_dd"] <- latdeg + latmin / 60

lon <- dt[,"lon\_dd"]

londeg <- as.numeric(substr(lon, 0, 2))

lonmin <- as.numeric(substr(lon, 3, 8))

dt[,"lon\_dd"] <- (londeg + lonmin / 60) \* -1

}

dt$time <- as.numeric(dt$time)

dt$date <- as.numeric(dt$date)

#remove rows of all NA values

dt <- dt[as.numeric(rowSums(is.na(dt))) < ncol(dt) - 1,]

dt <- dt[as.numeric(rowSums(is.na(dt[,c("lat\_dd", "lon\_dd")]))) < 2,]

#remove unrealistic coordinates

dt <- dt[abs(dt$lat\_dd) > 24.5 & abs(dt$lat\_dd) < 25.5, ]

dt <- dt[abs(dt$lon\_dd) > 80.1 & abs(dt$lon\_dd) < 82, ]

#check for incomplete minutes

datelist <- unique(dt$date)

reslist2 <- list()

for(j in 1:length(datelist)){

#j<-1

curdat <- dt[dt$date == datelist[j],]

gdata <- data.frame(table(curdat$time))

fdata <-as.numeric(as.character(gdata[gdata$Freq < dfmmin, 1]))#too few measurements

odata <- as.numeric(as.character(gdata[gdata$Freq > dfmmin, 1]))#too many measurements

if(length(odata) > 0){

for(k in 1:length(odata)){

#k<-1

leng <- nrow(curdat[curdat$time == odata[k],])

remo <- sample(as.numeric(row.names(curdat[curdat$time == odata[k],])), leng - dfmmin)

curdat <- curdat[-match(remo, as.numeric(row.names(curdat))),]

}

}

curdat <- curdat[!curdat$time %in% fdata,]

curdat <- curdat[!is.na(curdat$time),]

curdat$sec <- rep(round(seq(from = 0, to = 60 - 60 / dfmmin, by = 60 / dfmmin), 3), times = nrow(data.frame(table(curdat$time))))

reslist2[[j]] <- curdat

}

dt <- do.call("rbind", reslist2)

#detect when dt is measured at fractional seconds

if(!identical(round(dt$sec), dt$sec)){

dt$sec <- round(dt$sec)

}

#create POSIXct datetime column

yr <- substring(dt$date, nchar(dt$date) - 1, nchar(dt$date))

day <- substring(dt$date, nchar(dt$date) - 3, nchar(dt$date) - 2)

mon <- substring(dt$date, 1, nchar(dt$date) - 4)

hr <- substring(dt$time, 1, nchar(dt$time) - 2)

min <- substring(dt$time, nchar(dt$time) - 1, nchar(dt$time))

if(mean(nchar(mon)) == 1){mon <- paste("0", mon, sep = "")}

dt$datetime <- paste(yr, "-", mon, "-", day, "-", hr, "-", min, "-", dt$sec, sep = "")

rm(min)

dt$datetime <- as.POSIXct(strptime(dt$datetime, format = "%y-%m-%d-%H-%M-%S"))

#clean data frame

#trim beginning and end based on when data is all zeros

trimdt <- function(dt){

j <- 1

for(i in 1:nrow(dt)){

if(dt[i,1:9][order(dt[i,1:9])][2] > 0){

break

}

j <- i + 1

}

k <- nrow(dt)

for(i in nrow(dt):1){

if(!is.na(min(dt[i, 1:9])) > 0){

break

}

k <- i - 1

}

dt[j:k,]

}

dt <- trimdt(dt)

#check for correct cond to salinity calculations

corsal <- DataflowR::cond2sal(dt$cond \* 1000, dt$temp)

if((lm(corsal ~ dt$sal)$coefficients[2] - 1) > 0.02){

dt$sal <- corsal

}

#print(paste(basename(dflist[i]), "processed", sep = " "))

dt

}

read\_c6 <- function(c6path){

read.csv(c6path, skip = 12, header = F)[,1:9]

}

clean\_c6 <- function(c6){

names(c6) <- c("datetime", "brighteners", "phycoe", "phycoc", "c6chla", "c6cdom", "c6turbidity", "depth", "c6temp")

if(!any(!is.na(c6[,"c6temp"]))){

c6 <- c6[,-9]

names(c6)[8] <- "c6temp"

}else{

c6 <- c6[,-8]

}

#NEED TO UPDATE "names(c6)": WE CAN FINALIZE THE LIST ONCE WE CHECK THE ORDER OF PARAMETERS IN THE FILE THAT COMES FROM C6 INSTRUMENT; SHOULD BE SOMETHING LIKE names(c6) <-c("datetime", "c6chlaR", "phycoe", "phycoc", "c6chlaB", "c6turbidity", "depth", "c6temp")); WE CAN UPDATE FOR NOW AND TRY ON EXISTING DATA USING THIS ORDER LISTED HERE

#check for missing seconds information

if(all(is.na(as.POSIXct(strptime(c6$datetime, "%m/%d/%y %H:%M:%S"))))){

c6sec <- unlist(lapply(rle(sapply(c6$datetime, function(x) strftime(strptime(x, format = "%m/%d/%Y %H:%M"), format = "%M")))$lengths, function(x) seq(0, 60 - (60/x), length.out = x)))

c6$datetime <- as.POSIXct(strptime(paste0(c6$datetime, ":", c6sec), "%m/%d/%Y %H:%M:%S"))

}else{

c6$datetime <- as.POSIXct(strptime(c6$datetime, "%m/%d/%y %H:%M:%S"))

}

# if(!any(!is.na(c6$datetime))){

# c6 <- read.csv(c6list[c6dfmatch], skip = 12, header = F)[,1:9]

# names(c6) <- c("datetime", "brighteners", "phycoe", "phycoc", "c6chla", "c6cdom", "c6turbidity", "depth", "c6temp")

# if(!any(!is.na(c6[,"c6temp"]))){

# c6 <- c6[,-9]

# names(c6)[8] <- "c6temp"

# }else{

# c6 <- c6[,-8]

# }

#

# if(nchar(strsplit(as.character(c6[,"datetime"]), "/")[[1]][1]) == 1){

# if(rle(as.character(c6[,"datetime"]))$length[1] != c6mmin){#account for less than full minute to start

# c6 <- c6[c6$datetime != rle(as.character(c6[,"datetime"]))$values[1],]

# }

#

# padm\_addsec <- function(x, c6mmin){

# #pad month

# x <- as.character(x)

# x <- paste("0", substring(x, 0, 1), substring(x, 2, nchar(x)), sep = "")

# #add sec

# sseq <- seq(0, 60 - (60 / c6mmin), 60 / c6mmin)

# sseq <- sapply(sseq, function(x) ifelse(nchar(x) == 1, paste("0", x, sep = ""), x))

# paste(x, ":", sseq, sep = "")

# }

# c6[,"datetime"] <- padm\_addsec(c6[,"datetime"], c6mmin = c6mmin)

# }

# c6$datetime <- as.POSIXct(strptime(c6$datetime, "%m/%d/%Y %H:%M:%S"))

# }

c6$sec <- as.numeric(format(c6$datetime, '%S'))

c6freq <- c6$sec[2] - c6$sec[1]

c6$datetime <- as.POSIXct(c6$datetime)

return(c6)

}

read\_eu <- function(eupath){

read.csv(eupath, header = TRUE, stringsAsFactors = FALSE)#[,c(1:13, 17:18)]

}

clean\_eu <- function(eu){

names(eu) <- tolower(make.names(names(eu)))

eu$datetime <- paste(sapply(eu$date, function(x) mdy2mmyyyy(x)), eu$time)

#check for missing seconds information

if(all(is.na(as.POSIXct(strptime(eu$datetime, "%m/%d/%Y %H:%M:%S"))))){

eusec <- unlist(lapply(rle(sapply(eu$date, function(x) strftime(strptime(x, format = "%m/%d/%Y %H:%M"), format = "%M")))$lengths, function(x) seq(0, 60 - (60 / x), length.out = x)))

eu$date <- as.POSIXct(strptime(paste0(eu$date, ":", eusec), "%m/%d/%Y %H:%M:%S"))

}else{

eu$datetime <- as.POSIXct(strptime(eu$datetime,"%m/%d/%Y %H:%M:%S"))

eu$datetime <- as.POSIXct(strptime(eu$datetime, "%Y-%m-%d %H:%M:%S"))

eu <- eu[!is.na(eu$datetime) & nchar(eu$datetime) == 10,]

}

return(eu)

}

read\_exo <- function(exopath){

exo <- read.csv(exopath, header = T, skip = 12, stringsAsFactors = FALSE)

if(substring(names(exo)[1], 1, 4) != "Date"){

exo <- read.csv(exopath, header = T, skip = 24, stringsAsFactors = FALSE)

}

exo

}

clean\_exo <- function(exo){

names(exo) <- tolower(gsub("\\.", "", names(exo)))

exo <- exo[,!(names(exo) %in% c("timefractsec", "sitename","x"))]

exo$datetime <- as.POSIXct(strptime(paste(

exo[,grep("date", names(exo))],

exo[,grep("time", names(exo))]), format = "%m/%d/%Y %H:%M:%S")

)

exo$sec <- strftime(exo$datetime, "%S")

exo <- exo[which(!duplicated(exo$datetime)),]

exo[,"longitudedegrees"] <- exo[,"longitudedegrees"] \* -1

exo <- exo[exo$latitudedegrees > 24,]

exo

}

if(!is.na(dfmmin)){

df <- iterate\_days\_load(survey\_days, dflist, read\_df, clean\_df)

}

if(!is.na(c6mmin)){

c6 <- iterate\_days\_load(survey\_days, c6list, read\_c6, clean\_c6)

}

if(!is.na(eummin)){

eu <- iterate\_days\_load(survey\_days, eulist, read\_eu, clean\_eu)

}

if(!is.na(exommin)){

exo <- iterate\_days\_load(survey\_days, exolist, read\_exo, clean\_exo)

}

#=======================================================================#

streams <- c("df", "c6", "eu", "exo")

streams <- streams[sapply(streams, function(x) exists(x))]

streams <- streams[sapply(streams, function(x) is.data.frame(eval(as.symbol(x))))]

stream\_mmin <- c("dfmmin", "c6mmin", "eummin", "exommin")

target <- eval(as.symbol(gps))

target\_mmin <- eval(as.symbol(stream\_mmin[grep(gps, stream\_mmin)]))

detect\_mmin <- function(dt){

round(60/Mode(diff(as.numeric(format(dt$datetime, '%S')))))

}

check\_correct\_mmin <- function(dt, target, target\_mmin){

dt\_mmin <- detect\_mmin(dt)

#create second-wise zoo object

begin\_date <- as.POSIXct(as.Date(unique(strftime(min(dt$datetime), format = "%Y-%m-%d"))[1]))

end\_date <- as.POSIXct(as.Date(unique(strftime(max(dt$datetime), format = "%Y-%m-%d"))[1]) + 1)

target\_dates <- as.Date(seq(range(target$datetime)[1], range(target$datetime)[2], 86400))

if(abs(as.Date(begin\_date) - min(target\_dates)) > 2){

begin\_date <- as.POSIXct(as.Date(min(target\_dates)) - 2)

dt <- dt[dt$datetime > (begin\_date - 86400),]

}

if(abs(as.Date(end\_date) - max(target\_dates)) > 2){

end\_date <- as.POSIXct(as.Date(max(target\_dates)) + 2)

dt <- dt[dt$datetime < (end\_date + 86400),]

}

dt\_zoo <- zoo::zoo(dt, dt$datetime)

dt\_zoo\_full <- data.frame(

seq(

begin\_date,

end\_date, 1

)

)

names(dt\_zoo\_full) <- "datetime"

dt\_zoo\_full <- zoo::zoo(dt\_zoo\_full, dt\_zoo\_full$datetime)

dt\_zoo\_full <- merge(dt\_zoo\_full, dt\_zoo)

dt\_zoo\_full <- dt\_zoo\_full[min(which(!is.na(dt\_zoo\_full[,2]))):max(which(!is.na(dt\_zoo\_full[,2]))),]#trim to the datetime range of non-NA

#select columns to interpolate

is\_interp\_column <- colSums(!is.na(dt\_zoo\_full)) < nrow(dt\_zoo\_full)

is\_interp\_column[c(1:2)] <- FALSE

# is\_interp\_column[c(1:3, (ncol(dt\_zoo\_full) - 1), ncol(dt\_zoo\_full))] <- FALSE #need to make this not dataset-specific

dt\_zoo\_full <- zoo::na.approx(dt\_zoo\_full[,is\_interp\_column])

dt <- data.frame(dt\_zoo\_full, zoo::index(dt\_zoo\_full), row.names = NULL, stringsAsFactors = FALSE)

names(dt)[ncol(dt)] <- "datetime"

dt[,1:(ncol(dt) - 1)] <- apply(dt[,1:(ncol(dt) - 1)], 2, function(x) as.numeric(as.character(x)))

#filter based on target time-stamps

dt\_names <- names(dt)

dt <- merge(target, dt, by = "datetime", all.x = T)

dt[, dt\_names[dt\_names %in% names(dt)]]

}

#loop through contributing streams

if(!is.na(c6mmin) & (gps != "c6")){

c6 <- check\_correct\_mmin(c6, target, target\_mmin)

}

if(!is.na(dfmmin) & (gps != "df")){

df <- check\_correct\_mmin(df, target, target\_mmin)

}

if(!is.na(eummin) & (gps != "eu")){

eu <- check\_correct\_mmin(eu, target, target\_mmin)

}

if(!is.na(exommin) & (gps != "exo")){

exo <- check\_correct\_mmin(exo, target, target\_mmin)

}

#merge contributing streams into target

contributing\_streams <- streams[!(streams %in% gps)]

dt <- target

for(i in contributing\_streams){

dt <- merge(dt, eval(as.symbol(i)), all.x = TRUE)

}

detect\_coord\_names <- function(x){

lat\_name <- names(x)[grep("lat", names(x))]

lon\_name <- names(x)[grep("lon", names(x))]

c(lat\_name, lon\_name)

}

coord\_names <- detect\_coord\_names(eval(as.symbol(gps)))

create\_basin\_labels <- function(dt, coord\_names){

#define projections

projstr <- "+proj=utm +zone=17 +datum=NAD83 +units=m +no\_defs +ellps=GRS80 +towgs84=0,0,0"

latlonproj <- "+proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0"

fathombasins <- rgdal::readOGR(file.path(fdir, "DF\_Basefile/fathom\_basins\_proj.shp"), layer = "fathom\_basins\_proj", verbose = FALSE)

cerpbasins <- rgdal::readOGR(file.path(fdir, "DF\_Basefile/fbfs\_zones.shp"), layer = "fbfs\_zones", verbose = FALSE)

selectiongrid <- rgdal::readOGR(file.path(fdir, "DF\_Basefile/testgrid3.shp"), layer = "testgrid3", verbose = FALSE)

#NEED TO CHANGE “fathom basins” "fathom\_basins\_proj.shp" to ""fathom\_basins\_proj\_updated.shp" AND “selectiongrid” TO FILE "DF\_BASEFILE/TESTGRID9.SHP" IF CAN BUT IF INTERPOLATION TAKES TOO LONG B/C OF THE LARGE INCREASE IN THE # OF GRID CELLS IN THE EXPANDED AREA, COULD USE "DF\_BASEFILE/TESTGRID9\_OPTIMIZE.SHP"; MAKE SURE TO CHANGE “LAYER=” TO MATCH LAYER NAME

#spatial join

xy <- cbind(dt[,coord\_names[2]], dt[,coord\_names[1]])

xy <- data.frame(xy)

fulldataset <- coordinatize(dt, latname = coord\_names[1], lonname = coord\_names[2])

fulldataset.over <- sp::over(fulldataset, selectiongrid)

fulldataset.over2 <- sp::over(fulldataset, fathombasins[,1:2])

fulldataset.over3 <- sp::over(fulldataset,cerpbasins[,2])

fulldataset.over <- cbind(data.frame(fulldataset), data.frame(fulldataset.over), data.frame(fulldataset.over2), data.frame(fulldataset.over3))

fulldataset.over$lon\_dd <- xy[,1]

fulldataset.over$lat\_dd <- xy[,2]

fulldataset.over[,names(fulldataset.over) != "NA."]

}

dt <- create\_basin\_labels(dt, coord\_names)

names(dt) <- tolower(names(dt))

if(tofile == TRUE){

#add check to verify yearmon before overwriting####

dtname <- file.path(fdir, .Platform$file.sep, "DF\_FullDataSets", .Platform$file.sep, substring(survey\_days[1], 1, 6), "j.csv", fsep = "")

if(!file.exists(dtname)){

write.csv(dt, dtname, row.names = FALSE)

}else{

stop("overwrite file?")

}

}

options(warn = 0)

dt

}

#'@name streamget

#'@title Retrieve previously cleaned full streaming datasets

#'@description Retrieve previously cleaned full streaming datasets

#'@param yearmon numeric date in yyyymm format

#'@param fdir character file path to local data directory

#'@param qa logical strip flagged data?

#'@export

#'@examples \dontrun{

#'yearmon <- 201212

#'dt <- streamget(yearmon)

#'}

streamget <- function(yearmon, qa = TRUE, fdir = getOption("fdir")){

fdir\_fd <- file.path(fdir, "DF\_FullDataSets")

flist <- list.files(fdir\_fd, include.dirs = T, full.names = T)

flist <- flist[substring(basename(flist),1,6) == yearmon]

dt <- read.csv(flist, stringsAsFactors = FALSE)

if(qa == TRUE && file.exists(file.path(fdir, "DF\_FullDataSets", "QA", paste(yearmon, "qa.csv", sep = ""))) && identical(dim(dt), dim(read.csv(file.path(fdir, "DF\_FullDataSets", "QA", paste(yearmon, "qa.csv", sep = "")))))){

qafile <- read.csv(file.path(fdir, "DF\_FullDataSets", "QA", paste(yearmon, "qa.csv", sep = "")))

if(!any(names(qafile) == "chlext") & any(names(dt) == "chlext")){

qafile$chlext <- NA

}

if(!(identical(dim(qafile), dim(dt)))){

warning("QA file dimensions do not match data dimensions")

}

dt[!is.na(qafile)] <- NA

}

dt

}

#'@name streamqa

#'@title Supervised quality control of streaming datasets

#'@description Supervised quality control of streaming datasets

#'@param yearmon numeric date in yyyymm format

#'@param parset character vector of column names to QA

#'@param setthresh logical set parameter thresholds

#'@param trimends logical look to trim ends of data stream? NOT IMPLEMENTED YET

#'@param paired logical examine relationships between paried parameters?

#'@param fdir file.path to data directory

#'@details loop through parameters giving the opportunity to trim measurement ends, set entire variables to NA, remove variables above/below a threshold

#'@return a matrix of the same size/shape of the fulldataset, with entries specifying where to set to NA, saved to DF\_FullDataSets/Raw/IntrumentOutput

#'@export

#'@import graphics

#'@examples \dontrun{

#'dt<-streamqa(yearmon=201410)

#'}

streamqa <- function(yearmon, parset = NA, setthresh = TRUE, trimends = FALSE, paired = TRUE, fdir = getOption("fdir")){

dt <- streamget(yearmon)

dt <- dt[with(dt, order(date, time)),]

if(setthresh == TRUE){

if(file.exists(file.path(fdir, "DF\_FullDataSets", "QA", paste(yearmon, "qa.csv", sep = "")))){

dtqa <- read.csv(file.path(fdir, "DF\_FullDataSets", "QA", paste(yearmon, "qa.csv", sep = "")))

}else{

dtqa <- data.frame(matrix(NA, nrow = nrow(dt), ncol = ncol(dt)))

names(dtqa) <- names(dt)

}

#explore and set parameter threshold limits

par(mfrow = c(1, 1))

if(all(is.na(parset))){

parset <- c("chla", "temp", "cond", "sal", "trans", "cdom", "brighteners", "phycoe", "phycoc", "c6chla", "c6cdom", "c6turbidity", "c6temp")

}

parset <- parset[parset %in% names(dt)]

#UPDATE FACTORS IN PARSET TO REFLECT CORRECT EXO AND C6 NAMES

for(i in parset){

if(any(!is.na(dt[,i]))){

plot(dt[,i], ylab = i)

threshlog <- "c"

thresh <- NA

while(threshlog != "q"){

threshlog <- readline(message("Set threshold? Enter an upper and lower range as c(lower,upper) or press 'q' to move to next QA step: ", appendLF = FALSE))

if(!is.na(threshlog)&threshlog!="q"){

thresh <- threshlog

plot(dt[,i], ylab = i, ylim = eval(parse(text = threshlog)))

}

}

if(!is.na(thresh)){

thresh<-gsub("c\\(","",thresh)

thresh<-gsub(")","",thresh)

thresh<-unlist(lapply(strsplit(thresh,","),as.numeric))

dtqa[,i][dt[,i]<thresh[1]]<-"r"

dt[,i][dt[,i]<thresh[1]]<-NA

dtqa[,i][dt[,i]>thresh[2]]<-"r"

dt[,i][dt[,i]>thresh[2]]<-NA

}

}

}

}

#explore paired parameter relationships

# if(paired==TRUE){

# par(mfrow=c(3,1),mar=c(0,4,0,0))

#

# if(any(!is.na(dt[,"temp"]))&any(!is.na(dt[,"c6temp"]))){

# plot(dt[,"temp"],xaxt="n",xlab="")

# plot(dt[,"c6temp"],xaxt="n",xlab="")

# plot(dt[,"temp"],dt[,"c6temp"])

# abline(a=0,b=1,col="red")

# qalogical<-readline(message("Press 'Enter' to continue, '1' to set top panel to NA, '2' to set middle panel to NA: ",appendLF=FALSE))

# if(qalogical==1|qalogical=='1'){

# dt[,"temp"]<-NA

# dtqa[,"temp"]<-"r"

# }

# if(qalogical==2|qalogical=='2'){

# dt[,"c6temp"]<-NA

# dtqa[,"c6temp"]<-"r"

# }

# }

#

# if(any(!is.na(dt[,"chla"]))&any(!is.na(dt[,"c6chla"]))){

# plot(dt[,"chla"],xaxt="n",xlab="")

# plot(dt[,"c6chla"],xaxt="n",xlab="")

# plot(dt[,"chla"],dt[,"c6chla"])

# abline(lm(dt[,"c6chla"]~dt[,"chla"]),col="red")

# qalogical<-readline(message("Press 'Enter' to continue, '1' to set top panel to NA, '2' to set middle panel to NA: ",appendLF=FALSE))

# if(qalogical==1|qalogical=='1'){

# dt[,"chla"]<-NA

# dtqa[,"chla"]<-"r"

# }

# if(qalogical==2|qalogical=='2'){

# dt[,"c6chla"]<-NA

# dtqa[,"c6chla"]<-"r"

# }

# }

#

# if(any(!is.na(dt[,"cdom"]))&any(!is.na(dt[,"c6cdom"]))){

# plot(dt[,"cdom"],xaxt="n",xlab="")

# plot(dt[,"c6cdom"],xaxt="n",xlab="")

# plot(dt[,"cdom"],dt[,"c6cdom"])

# abline(lm(dt[,"c6cdom"]~dt[,"cdom"]),col="red")

# qalogical<-readline(message("Press 'Enter' to continue, '1' to set top panel to NA, '2' to set middle panel to NA: ",appendLF=FALSE))

# if(qalogical==1|qalogical=='1'){

# dt[,"cdom"]<-NA

# dtqa[,"cdom"]<-"r"

# }

# if(qalogical==2|qalogical=='2'){

# dt[,"c6cdom"]<-NA

# dtqa[,"c6cdom"]<-"r"

# }

# }

#

# }

# if(trimends==TRUE){#NOT IMPLEMENTED YET

# trim<-function(dt){}

# }

message("QA finished. Printing to file...")

message(file.path(fdir,"DF\_FullDataSets","QA",paste(yearmon,"qa",".csv",sep="")))

fdir\_fd<-file.path(fdir,"DF\_FullDataSets","QA")

write.csv(dtqa,file.path(fdir\_fd,paste(yearmon,"qa",".csv",sep="")),row.names = FALSE)

}

#'@name streamparse

#'@title Parse old cleaned streaming files

#'@description Includes checks to ensure that data columns are of type numeric. TODO: check that the fathom basins column is populated

#'@param yearmon numeric yyyymm date

#'@param tofile logical save to file?

#'@param fdir character file path to local data directory

#'@export

#'@examples \dontrun{dt<-streamparse(yearmon=201002)}

streamparse<-function(yearmon,tofile=FALSE,fdir=getOption("fdir")){

#yearmon<-201109

fdir\_fd<-file.path(fdir,"DF\_FullDataSets","Raw")

flist<-list.files(fdir\_fd,include.dirs=T,full.names=T)

flist<-flist[substring(basename(flist),1,6)==yearmon]

dt<-read.csv(flist)

names(dt)<-tolower(names(dt))

namestemp<-tolower(names(streamget(201505)))#[-1])

#remove bad coord columns

coordnames<-c("lat\_dd","long\_dd","lon\_dd")

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

#i<-1

cname<-which(!is.na(match(names(dt),coordnames[i])))

if(length(cname)!=0){

if(abs(mean(dt[,coordnames[i]]))>100){

dt<-dt[,-cname]

}

}

}

#remove unrealistic coordinates

dt <- dt[abs(dt$lat\_dd) > 24.5 & abs(dt$lat\_dd) < 25.5, ]

dt <- dt[abs(dt$lon\_dd) > 80.1 & abs(dt$lon\_dd) < 82, ]

#create translation key

namesalias<-read.table(text="sec,sec.x

cnd,cond

light,trans

fluor,chla",sep=",")

for(n in 1:length(names(dt))){

#n<-1

if(any(names(dt)[n]==as.character(namesalias[,1]))){

names(dt)[n]<-as.character(namesalias[which(names(dt)[n]==namesalias[,1]),2])

}

}

#remove non-matching columns

dt <- dt[,-which(!is.na(match(names(dt), names(dt)[is.na(match(names(dt), namestemp))])))]

#create extra columns if necessary

dt[,namestemp[is.na(match(namestemp,names(dt)))]]<-NA

#calculate datetime stamp

#create POSIXct datetime column

if(mean(nchar(as.character(dt$date)))>6){

hr<-substring(dt$time,1,nchar(dt$time)-2)

min<-substring(dt$time,nchar(dt$time)-1,nchar(dt$time))

dt$datetime<-as.POSIXct(strptime(paste(as.character(dt$date)," ",hr,":",min,":",dt$sec.x,sep=""),format="%m/%d/%Y %H:%M:%S"))

}else{

yr<-substring(dt$date,nchar(dt$date)-1,nchar(dt$date))

day<-substring(dt$date,nchar(dt$date)-3,nchar(dt$date)-2)

mon<-substring(dt$date,1,nchar(dt$date)-4)

hr<-substring(dt$time,1,nchar(dt$time)-2)

min<-substring(dt$time,nchar(dt$time)-1,nchar(dt$time))

if(mean(nchar(mon))==1){mon<-paste("0",mon,sep="")}

if(!any(!is.na(dt$sec.x))){

mmin<-Mode(rle(dt$time)$lengths)

mminseq<-seq(from=0,to=60-60/mmin,by=60/mmin)

mmin1<-rle(dt$time)$lengths[1]

mmin1seq<-mminseq[(length(mminseq)-mmin1+1):length(mminseq)]

dt$sec.x<-c(mmin1seq,rep\_len(mminseq,length.out=nrow(dt)-mmin1))

}

dt$datetime<-paste(yr,"-",mon,"-",day,"-",hr,"-",min,"-",dt$sec.x,sep="")

#rm(min)

dt$datetime<-as.POSIXct(strptime(dt$datetime,format="%y-%m-%d-%H-%M-%S"))

}

#sort columns to match namestemp

dt<-dt[,match(namestemp,names(dt))]

#ensure that data columns are numeric

#UPDATE FACTORS IN PARSET LINE TO BELOW REFLECT CORRECT EXO AND C6 NAMES

parset<-c("chla","temp","cond","sal","trans","cdom","brighteners","phycoe","phycoc","c6chla","c6cdom","c6turbidity","c6temp")

dt[,parset]<-suppressWarnings(apply(dt[,parset],2,function(x) as.numeric(x)))

if(tofile==TRUE){

#add check to verify yearmon before overwriting

dtname<-file.path(fdir,.Platform$file.sep,"DF\_FullDataSets",.Platform$file.sep,substring(basename(flist[1]),1,6),"j.csv",fsep="")

if(!file.exists(dtname)){

write.csv(dt,dtname,row.names = FALSE)

}else{

stop("overwrite file?")

}

}else{

dt

}

}