**fun\_gg\_donut <- function(**

**data1,**

**freq,**

**categ,**

**fill.palette = NULL,**

**fill.color = NULL,**

**hole.size = 0.5,**

**hole.text.size = 12,**

**border.color = "gray50",**

**border.size = 12,**

**title = "",**

**title.text.size = 12,**

**annotation = NULL,**

**annotation.size = 3,**

**annotation.force.pull = 100,**

**legend.show = TRUE,**

**legend.width = 0.5,**

**legend.name = NULL,**

**legend.limit = NULL,**

**add = NULL,**

**return = FALSE,**

**return.ggplot = FALSE,**

**return.gtable = TRUE,**

**plot = TRUE,**

**warn.print = FALSE,**

**lib.path = NULL**

**){**

*# AIM*

*# Plot a ggplot2 donut using contingency data, with frequencies sorted from the hisgest to the lowest, starting at the top and turning clockwise*

*# For ggplot2 specifications, see: https://ggplot2.tidyverse.org/articles/ggplot2-specs.html*

*# WARNINGS*

*# Rows containing NA in data1[, c(freq, categ)] will be removed before processing, with a warning (see below)*

*# Size arguments (hole.text.size, border.size, title.text.size and annotation.size) are in mm. See Hadley comment in https://stackoverflow.com/questions/17311917/ggplot2-the-unit-of-size. See also http://sape.inf.usi.ch/quick-reference/ggplot2/size). Unit object are not accepted, but conversion can be used (e.g., grid::convertUnit(grid::unit(0.2, "inches"), "mm", valueOnly = TRUE))*

*# ARGUMENTS*

*# data1: a dataframe compatible with ggplot2*

*# freq: single character string of the data1 column name of the frequencies*

*# categ: single character string of the data1 column name of categories (qualitative variable)*

*# fill.palette: single character string of a palette name (see ?ggplot2::scale\_fill\_brewer() for the list).Ignored if fill.color is not NULL*

*# fill.color: either (1) NULL, or (2) a vector of character strings or integers of same length as the number of classes in categ. Colors can be color names (see ?colors() in R), hexadecimal color codes, or integers (see ?palette() in R). The order of the elements will be used according to the frequency values, from highest to lowest. An easy way to use this argument is to sort data1 according to the frequencies values, add a color column with the corresponding desired colors and use the content of this column as values of fill.color. If color is NULL and fill.palette is NULL, default colors of ggplot2 are used. If color is not NULL, it overrides fill.palette*

*# hole.size: single positive proportion of donut central hole, 0 meaning no hole and 1 no donut*

*# hole.text.size: single positive numeric value of the title font size in mm*

*# border.color: a single character string or integer. Colors can be color names (see ?colors() in R), hexadecimal color codes, or integers (see ?palette() in R)*

*# border.size: single numeric value of border tickness in mm. Write zero for no dot border*

*# title: single character string of the graph title*

*# title.text.size: single numeric value of the title font size in mm*

*# annotation: single character string of the data1 column name of annotations. Values inside this column will be displayed over the corresponding slices of the donut. Write NULL if not required*

*# annotation.size: single positive numeric value of the annotation font size in mm*

*# annotation.force: single positive numeric value of the force of repulsion between overlapping text labels. See ?ggrepel::geom\_text\_repel() in R*

*# annotation.force.pull: single positive numeric value of the force of attraction between a text label and its corresponding data point. See ?ggrepel::geom\_text\_repel() in R*

*# legend.show: logical (either TRUE or FALSE). Show legend?*

*# legend.width: single proportion (between 0 and 1) indicating the relative width of the legend sector (on the right of the plot) relative to the width of the plot. Value 1 means that the window device width is split in 2, half for the plot and half for the legend. Value 0 means no room for the legend, which will overlay the plot region. Write NULL to inactivate the legend sector. In such case, ggplot2 will manage the room required for the legend display, meaning that the width of the plotting region can vary between graphs, depending on the text in the legend*

*# legend.name: character string of the legend title. If legend.name is NULL then legend.name is the value of the categ argument*

*# legend.limit: single positive proportion of the classes displayed in the legend for which the corresponding proportion is over legend.limit. Write NULL to display all the classes*

*# legend.add.prop: logical (either TRUE or FALSE). add the proportion after the class names in the legend ?*

*# add: character string allowing to add more ggplot2 features (dots, lines, themes, facet, etc.). Ignored if NULL*

*# WARNING: (1) the string must start with "+", (2) the string must finish with ")" and (3) each function must be preceded by "ggplot2::". Example: "+ ggplot2::coord\_flip() + ggplot2::theme\_bw()"*

*# If the character string contains the "ggplot2::theme" string, then the article argument of fun\_gg\_scatter() (see above) is ignored with a warning. In addition, some arguments can be overwritten, like x.angle (check all the arguments)*

*# Handle the add argument with caution since added functions can create conflicts with the preexisting internal ggplot2 functions*

*# WARNING: the call of objects inside the quotes of add can lead to an error if the name of these objects are some of the fun\_gg\_scatter() arguments. Indeed, the function will use the internal argument instead of the global environment object. Example article <- "a" in the working environment and add = '+ ggplot2::ggtitle(article)'. The risk here is to have TRUE as title. To solve this, use add = '+ ggplot2::ggtitle(get("article", envir = .GlobalEnv))'*

*# return: logical (either TRUE or FALSE). Return the graph parameters?*

*# return.ggplot: logical (either TRUE or FALSE). Return the ggplot object in the output list? Ignored if return argument is FALSE. WARNING: always assign the fun\_gg\_scatter() function (e.g., a <- fun\_gg\_scatter()) if return.ggplot argument is TRUE, otherwise, double plotting is performed. See $ggplot in the RETURN section below for more details*

*# return.gtable: logical (either TRUE or FALSE). Return the ggplot object as gtable of grobs in the output list? Ignored if plot argument is FALSE. Indeed, the graph must be plotted to get the grobs dispositions. See $gtable in the RETURN section below for more details*

*# plot: logical (either TRUE or FALSE). Plot the graphic? If FALSE and return argument is TRUE, graphical parameters and associated warnings are provided without plotting*

*# warn.print: logical (either TRUE or FALSE). Print warnings at the end of the execution? ? If FALSE, warning messages are never printed, but can still be recovered in the returned list. Some of the warning messages (those delivered by the internal ggplot2 functions) are not apparent when using the argument plot = FALSE*

*# lib.path: character string indicating the absolute path of the required packages (see below). if NULL, the function will use the R library default folders*

*# RETURN*

*# a donut plot if plot argument is TRUE*

*# a list of the graph info if return argument is TRUE:*

*# $data: the initial data with frequencies converted to proportion and with graphic information added*

*# $removed.row.nb: a list of the removed rows numbers in data frames (because of NA). NULL if no row removed*

*# $removed.rows: a list of the removed rows in data frames (because of NA). NULL if no row removed*

*# $panel: the variable names used for the panels (NULL if no panels). WARNING: NA can be present according to ggplot2 upgrade to v3.3.0*

*# $axes: the x-axis and y-axis info*

*# $warn: the warning messages. Use cat() for proper display. NULL if no warning. WARNING: warning messages delivered by the internal ggplot2 functions are not apparent when using the argument plot = FALSE*

*# $ggplot: ggplot object that can be used for reprint (use print($ggplot) or update (use $ggplot + ggplot2::...). NULL if return.ggplot argument is FALSE. Of note, a non-null $ggplot in the output list is sometimes annoying as the manipulation of this list prints the plot*

*# $gtable: gtable object that can be used for reprint (use gridExtra::grid.arrange(...$ggplot) or with additionnal grobs (see the grob decomposition in the examples). NULL if return.ggplot argument is FALSE. Contrary to $ggplot, a non-NULL $gtable in the output list is not annoying as the manipulation of this list does not print the plot*

*# REQUIRED PACKAGES*

*# ggplot2*

*# gridExtra*

*# grid*

*# lemon (in case of use in the add argument)*

*# scales*

*# ggrepel*

*# REQUIRED FUNCTIONS FROM THE cute PACKAGE*

*# fun\_gg\_empty\_graph()*

*# fun\_gg\_palette()*

*# fun\_pack()*

*# fun\_check()*

*# EXAMPLES*

*# set.seed(1) ; obs1 <- data.frame(Km = c(2, 1, 6, 5, 4, 7), Time = c(2, 1, 6, 5, 4, 7)^2, Car = c("TUUT", "TUUT", "TUUT", "WIIM", "WIIM", "WIIM"), Color1 = rep(c("coral", "lightblue"), each = 3), stringsAsFactors = TRUE) ; fun\_gg\_scatter(data1 = obs1, x = "Km", y = "Time")*

*# DEBUGGING*

*# set.seed(1) ; obs1 <- data.frame(km = rnorm(1000, 10, 3), time = rnorm(1000, 10, 3), group1 = rep(c("A1", "A2"), 500), stringsAsFactors = TRUE) ; obs2 <-data.frame(km = rnorm(1000, 15, 3), time = rnorm(1000, 15, 3), group2 = rep(c("G1", "G2"), 500), stringsAsFactors = TRUE) ; set.seed(NULL) ; obs1$km[2:3] <- NA ; data1 = list(L1 = obs1, L2 = obs2) ; x = list(L1 = "km", L2 = "km") ; y = list(L1 = "time", L2 = "time") ; categ = list(L1 = "group1", L2 = "group2") ; categ = NULL ; categ.class.order = NULL ; color = NULL ; geom = "geom\_point" ; geom.step.dir = "hv" ; geom.stick.base = NULL ; alpha = 0.5 ; dot.size = 2 ; dot.shape = 21 ; dot.border.size = 0.5 ; dot.border.color = NULL ; line.size = 0.5 ; line.type = "solid" ; x.lim = NULL ; x.lab = NULL ; x.log = "no" ; x.tick.nb = NULL ; x.second.tick.nb = NULL ; x.include.zero = FALSE ; x.left.extra.margin = 0.05 ; x.right.extra.margin = 0.05 ; x.text.angle = 0 ; y.lim = NULL ; y.lab = NULL ; y.log = "no" ; y.tick.nb = NULL ; y.second.tick.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0.05 ; y.text.angle = 0 ; raster = FALSE ; raster.ratio = 1 ; raster.threshold = NULL ; text.size = 12 ; title = "" ; title.text.size = 12 ; legend.show = TRUE ; legend.width = 0.5 ; legend.name = NULL ; article = TRUE ; grid = FALSE ; add = NULL ; return = FALSE ; return.ggplot = FALSE ; return.gtable = TRUE ; plot = TRUE ; warn.print = FALSE ; lib.path = NULL*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

**arg.names <- names(formals(fun = sys.function(sys.parent(n = 2))))** *# names of all the arguments*

**arg.user.setting <- as.list(match.call(expand.dots=FALSE))[-1]** *# list of the argument settings (excluding default values not provided by the user)*

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_check",**

**"fun\_gg\_empty\_graph",**

**"fun\_gg\_palette",**

**"fun\_pack"**

**)**

**tempo <- NULL**

**for(i1 in req.function){**

**if(length(find(i1, mode = "function"))== 0L){**

**tempo <- c(tempo, i1)**

**}**

**}**

**if( ! is.null(tempo)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nREQUIRED cute FUNCTION", ifelse(length(tempo) > 1, "S ARE", " IS"), " MISSING IN THE R ENVIRONMENT:\n", paste0(tempo, collapse = "()\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end required function checking*

*# reserved words to avoid bugs (used in this function)*

*# end reserved words to avoid bugs (used in this function)*

*# arg with no default values*

**mandat.args <- c(**

**"data1",**

**"freq",**

**"categ"**

**)**

**tempo <- eval(parse(text = paste0("missing(", paste0(mandat.args, collapse = ") | missing("), ")")))**

**if(any(tempo)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nFOLLOWING ARGUMENT", ifelse(length(mandat.args) > 1, "S HAVE", "HAS"), " NO DEFAULT VALUE AND REQUIRE ONE:\n", paste0(mandat.args, collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end arg with no default values*

*# argument primary checking*

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$object.name))**

**if( ! is.null(fill.palette)){**

**tempo <- fun\_check(data = fill.palette, options = c("BrBG", "PiYG", "PRGn", "PuOr", "RdBu", "RdGy", "RdYlBu", "RdYlGn", "Spectral", "Accent", "Dark2", "Paired", "Pastel1", "Pastel2", "Set1", "Set2", "Set3", "Blues", "BuGn", "BuPu", "GnBu", "Greens", "Greys", "Oranges", "OrRd", "PuBu", "PuBuGn", "PuRd", "Purples", "RdPu", "Reds", "YlGn", "YlGnBu", "YlOrBr", "YlOrRd"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

**if( ! is.null(fill.color)){**

**tempo1 <- fun\_check(data = fill.color, class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name)**

**tempo2 <- fun\_check(data = fill.color, class = "factor", na.contain = TRUE, fun.name = function.name)**

**tempo3 <- fun\_check(data = fill.color, class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE & tempo3$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": fill.color ARGUMENT MUST BE A VECTOR OF CHARACTER STRINGS OR INTEGERS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = hole.size, prop = TRUE, neg.values = FALSE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = hole.text.size, class = "vector", mode = "numeric", neg.values = FALSE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo1 <- fun\_check(data = border.color, class = "vector", mode = "character", na.contain = FALSE, fun.name = function.name)**

**tempo2 <- fun\_check(data = border.color, class = "integer", double.as.integer.allowed = TRUE, neg.values = FALSE, na.contain = FALSE, fun.name = function.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE & tempo3$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": fill.color ARGUMENT MUST BE A VECTOR OF CHARACTER STRINGS OR INTEGERS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo1 <- fun\_check(data = data1, class = "data.frame", na.contain = TRUE, fun.name = function.name)**

**tempo2 <- fun\_check(data = data1, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 ARGUMENT MUST BE A DATA FRAME OR A LIST OF DATA FRAMES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(x)){**

**tempo1 <- fun\_check(data = x, class = "vector", mode = "character", na.contain = TRUE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = x, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x ARGUMENT MUST BE A SINGLE CHARACTER STRING OR A LIST OF CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = x, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(y)){**

**tempo1 <- fun\_check(data = y, class = "vector", mode = "character", na.contain = TRUE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = y, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y ARGUMENT MUST BE A SINGLE CHARACTER STRING OR A LIST OF CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = y, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(categ)){**

**tempo1 <- fun\_check(data = categ, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = categ, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ ARGUMENT MUST BE A SINGLE CHARACTER STRING OR A LIST OF CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = categ, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(categ.class.order)){**

**if(is.null(categ)){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.class.order ARGUMENT IS NOT NULL, BUT categ IS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo1 <- fun\_check(data = categ.class.order, class = "vector", mode = "character", fun.name = function.name)**

**tempo2 <- fun\_check(data = categ.class.order, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.class.order ARGUMENT MUST BE A VECTOR OF CHARACTER STRINGS OR A LIST OF VECTOR OF CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = categ.class.order, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(legend.name)){**

**tempo1 <- fun\_check(data = legend.name, class = "vector", mode = "character", na.contain = TRUE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = legend.name, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": legend.name ARGUMENT MUST BE A SINGLE CHARACTER STRING OR A LIST OF CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = legend.name, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo1 <- fun\_check(data = geom, class = "vector", mode = "character", na.contain = FALSE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = geom, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT MUST BE A SINGLE CHARACTER STRING OR A LIST OF CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo1 <- fun\_check(data = geom.step.dir, options = c("vh", "hv", "mid"), na.contain = FALSE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = geom.step.dir, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom.step.dir ARGUMENT MUST BE A SINGLE CHARACTER STRING (\"vh\" OR \"hv\" OR \"mid\") OR A LIST OF THESE CHARACTER STRINGS")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(geom.stick.base)){**

**tempo1 <- fun\_check(data = geom.stick.base, class = "vector", mode = "numeric", na.contain = FALSE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = color, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom.stick.base ARGUMENT MUST BE A SINGLE NUMERIC VALUE OR A LIST OF SINGLE NUMERIC VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = geom.stick.base, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo1 <- fun\_check(data = alpha, prop = TRUE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = alpha, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": alpha ARGUMENT MUST BE A SINGLE NUMERIC VALUE BETWEEN 0 AND 1 OR A LIST OF SUCH VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo1 <- fun\_check(data = dot.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name)**

**tempo2 <- fun\_check(data = dot.size, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.size ARGUMENT MUST BE A SINGLE NUMERIC VALUE OR A LIST OF SINGLE NUMERIC VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo1 <- fun\_check(data = dot.shape, class = "vector", length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = dot.shape, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.shape ARGUMENT MUST BE A SINGLE SHAPE VALUE OR A LIST OF SINGLE SHAPE VALUES (SEE** *https://ggplot2.tidyverse.org/articles/ggplot2-specs.html***)")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo1 <- fun\_check(data = dot.border.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name)**

**tempo2 <- fun\_check(data = dot.border.size, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.border.size ARGUMENT MUST BE A SINGLE NUMERIC VALUE OR A LIST OF SINGLE NUMERIC VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(dot.border.color)){**

**tempo1 <- fun\_check(data = dot.border.color, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = dot.border.color, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, length = 1, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

*# integer colors -> gg\_palette*

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.border.color MUST BE A SINGLE CHARACTER STRING OF COLOR OR A SINGLE INTEGER VALUE")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = dot.border.color, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo1 <- fun\_check(data = line.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name)**

**tempo2 <- fun\_check(data = line.size, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo2$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": line.size ARGUMENT MUST BE A SINGLE NUMERIC VALUE OR A LIST OF SINGLE NUMERIC VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo1 <- fun\_check(data = line.type, class = "vector", typeof = "integer", double.as.integer.allowed = FALSE, length = 1, fun.name = function.name)**

**tempo2 <- fun\_check(data = line.type, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**tempo3 <- fun\_check(data = line.type, class = "list", na.contain = TRUE, fun.name = function.name)**

**checked.arg.names <- c(checked.arg.names, tempo3$object.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE & tempo3$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": line.type ARGUMENT MUST BE A SINGLE LINE KIND VALUE OR A LIST OF SINGLE LINE KIND VALUES (SEE** *https://ggplot2.tidyverse.org/articles/ggplot2-specs.html***)")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if( ! is.null(x.lim)){**

**tempo <- fun\_check(data = x.lim, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(x.lim %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.lim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = x.lim, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(x.lab)){**

**if(all(class(x.lab) %in% "expression")){** *# to deal with math symbols*

**tempo <- fun\_check(data = x.lab, class = "expression", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

**tempo <- fun\_check(data = x.lab, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = x.lab, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = x.log, options = c("no", "log2", "log10"), length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(x.tick.nb)){**

**tempo <- fun\_check(data = x.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & x.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.tick.nb ARGUMENT MUST BE A NON-NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = x.tick.nb, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(x.second.tick.nb)){**

**tempo <- fun\_check(data = x.second.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & x.second.tick.nb <= 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x.second.tick.nb ARGUMENT MUST BE A NON-NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = x.second.tick.nb, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = x.include.zero, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = x.left.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = x.right.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = x.text.angle, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, length = 1, neg.values = TRUE, fun.name = function.name) ; eval(ee)**

**if( ! is.null(y.lim)){**

**tempo <- fun\_check(data = y.lim, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(y.lim %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.lim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = y.lim, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(y.lab)){**

**if(all(class(y.lab) %in% "expression")){** *# to deal with math symbols*

**tempo <- fun\_check(data = y.lab, class = "expression", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

**tempo <- fun\_check(data = y.lab, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = y.lab, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = y.log, options = c("no", "log2", "log10"), length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(y.tick.nb)){**

**tempo <- fun\_check(data = y.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & y.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.tick.nb ARGUMENT MUST BE A NON-NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = y.tick.nb, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if( ! is.null(y.second.tick.nb)){**

**tempo <- fun\_check(data = y.second.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & y.second.tick.nb <= 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.second.tick.nb ARGUMENT MUST BE A NON-NULL POSITIVE INTEGER")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = y.second.tick.nb, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = y.include.zero, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = y.top.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = y.bottom.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = y.text.angle, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, length = 1, neg.values = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = raster, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = raster.ratio, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**if( ! is.null(raster.threshold)){**

**tempo <- fun\_check(data = raster.threshold, class = "vector", typeof = "integer", neg.values = FALSE, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = raster.threshold, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title.text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = legend.show, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(legend.width)){**

**tempo <- fun\_check(data = legend.width, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = legend.width, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = article, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = grid, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(add)){**

**tempo <- fun\_check(data = add, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = add, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**tempo <- fun\_check(data = return, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = return.ggplot, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = return.gtable, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = plot, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = warn.print, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}else{**

*# no fun\_check test here, it is just for checked.arg.names*

**tempo <- fun\_check(data = lib.path, class = "vector")**

**checked.arg.names <- c(checked.arg.names, tempo$object.name)**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.7/r\_debugging\_tools-v1.7.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument primary checking*

*# second round of checking and data preparation*

*# management of NA arguments*

**tempo.arg <- names(arg.user.setting)** *# values provided by the user*

**tempo.log <- suppressWarnings(sapply(lapply(lapply(tempo.arg, FUN = get, env = sys.nframe(), inherit = FALSE), FUN = is.na), FUN = any)) & lapply(lapply(tempo.arg, FUN = get, env = sys.nframe(), inherit = FALSE), FUN = length)== 1L** *# no argument provided by the user can be just NA*

**if(any(tempo.log) == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ":\n", ifelse(sum(tempo.log, na.rm = TRUE) > 1, "THESE ARGUMENTS\n", "THIS ARGUMENT\n"), paste0(tempo.arg[tempo.log], collapse = "\n"),"\nCANNOT JUST BE NA")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end management of NA arguments*

*# management of NULL arguments*

**tempo.arg <-c(**

**"data1",**

*# "x", # inactivated because of hline or vline*

*# "y", # inactivated because of hline or vline*

**"geom",**

**"geom.step.dir",**

*# "geom.stick.base", # inactivated because can be null*

**"alpha",**

**"dot.size",**

**"dot.shape",**

**"dot.border.size",**

**"line.size",**

**"line.type",**

**"x.log",**

**"x.include.zero",**

**"x.left.extra.margin",**

**"x.right.extra.margin",**

**"x.text.angle",**

**"y.log",**

**"y.include.zero",**

**"y.top.extra.margin",**

**"y.bottom.extra.margin",**

**"y.text.angle",**

**"raster",**

**"raster.ratio",**

**"text.size",**

**"title",**

**"title.text.size",**

**"legend.show",**

*# "legend.width", # inactivated because can be null*

**"article",**

**"grid",**

**"return",**

**"return.ggplot",**

**"return.gtable",**

**"plot",**

**"warn.print"**

**)**

**tempo.log <- sapply(lapply(tempo.arg, FUN = get, env = sys.nframe(), inherit = FALSE), FUN = is.null)**

**if(any(tempo.log) == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ":\n", ifelse(sum(tempo.log, na.rm = TRUE) > 1, "THESE ARGUMENTS\n", "THIS ARGUMENT\n"), paste0(tempo.arg[tempo.log], collapse = "\n"),"\nCANNOT BE NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}**

*# end management of NULL arguments*

*# code that protects set.seed() in the global environment*

*# end code that protects set.seed() in the global environment*

*# warning initiation*

**ini.warning.length <- options()$warning.length**

**options(warning.length = 8170)**

**warn <- NULL**

**warn.count <- 0**

*# end warning initiation*

*# other checkings*

*# check list lengths (and names of data1 compartments if present)*

**list.color <- NULL**

**list.geom <- NULL**

**list.geom.step.dir <- NULL**

**list.geom.stick.base <- NULL**

**list.alpha <- NULL**

**list.dot.size <- NULL**

**list.dot.shape <- NULL**

**list.dot.border.size <- NULL**

**list.dot.border.color <- NULL**

**list.line.size <- NULL**

**list.line.type <- NULL**

**if(all(class(data1) == "list")){**

**if(length(data1) > 6){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 ARGUMENT MUST BE A LIST OF 6 DATA FRAMES MAXIMUM (6 OVERLAYS MAX)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(is.null(names(data1))){**

**names(data1) <- paste0("L", 1:length(data1))**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL NAME COMPARTMENT OF data1 LIST -> NAMES RESPECTIVELY ATTRIBUTED TO EACH COMPARTMENT:\n", paste(names(data1), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if( ! is.null(x)){**

**if( ! (all(class(x) == "list") & length(data1) == length(x))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}else{**

**x <- vector("list", length(data1))**

**}**

**if( ! is.null(y)){**

**if( ! (all(class(y) == "list") & length(data1) == length(y))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}else{**

**y <- vector("list", length(data1))**

**}**

**if( ! is.null(categ)){**

**if( ! (all(class(categ) == "list") & length(data1) == length(categ))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if( ! is.null(categ.class.order)){**

**if( ! (all(class(categ.class.order) == "list") & length(data1) == length(categ.class.order))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.class.order ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if( ! is.null(color)){**

**if( ! ((all(class(color) == "list") & length(data1) == length(color)) | ((all(mode(color) == "character") | all(mode(color) == "numeric")) & length(color)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": color ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE CHARACTER STRING OR INTEGER")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if((all(mode(color) == "character") | all(mode(color) == "numeric")) & length(color)== 1L){** *# convert the single value into a list of single value*

**list.color <- vector(mode = "list", length = length(data1))**

**list.color[] <- color**

**}**

**}**

**if( ! ((all(class(geom) == "list") & length(data1) == length(geom)) | (all(mode(geom) == "character") & length(geom)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE CHARACTER VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(geom) == "character") & length(geom)== 1L){** *# convert the single value into a list of single value*

**list.geom <- vector(mode = "list", length = length(data1))**

**list.geom[] <- geom**

**}**

**if( ! ((all(class(geom.step.dir) == "list") & length(data1) == length(geom.step.dir)) | (all(mode(geom.step.dir) == "character") & length(geom.step.dir)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom.step.dir ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE CHARACTER VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(geom.step.dir) == "character") & length(geom.step.dir)== 1L){** *# convert the single value into a list of single value*

**list.geom.step.dir <- vector(mode = "list", length = length(data1))**

**list.geom.step.dir[] <- geom.step.dir**

**}**

**if( ! is.null(geom.stick.base)){**

**if( ! ((all(class(geom.stick.base) == "list") & length(data1) == length(geom.stick.base)) | (all(mode(geom.stick.base) == "numeric") & length(geom.stick.base)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom.stick.base ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE NUMERIC VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(geom.stick.base) == "numeric") & length(geom.stick.base)== 1L){** *# convert the single value into a list of single value*

**list.geom.stick.base <- vector(mode = "list", length = length(data1))**

**list.geom.stick.base[] <- geom.stick.base**

**}**

**}**

**if( ! ((all(class(alpha) == "list") & length(data1) == length(alpha)) | (all(mode(alpha) == "numeric") & length(alpha)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": alpha ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE NUMERIC VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(alpha) == "numeric") & length(alpha)== 1L){** *# convert the single value into a list of single value*

**list.alpha <- vector(mode = "list", length = length(data1))**

**list.alpha[] <- alpha**

**}**

**if( ! ((all(class(dot.size) == "list") & length(data1) == length(dot.size)) | (all(mode(dot.size) == "numeric") & length(dot.size)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.size ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE NUMERIC VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(dot.size) == "numeric") & length(dot.size)== 1L){** *# convert the single value into a list of single value*

**list.dot.size <- vector(mode = "list", length = length(data1))**

**list.dot.size[] <- dot.size**

**}**

**if( ! ((all(class(dot.shape) == "list") & length(data1) == length(dot.shape)) | (all(mode(dot.shape) != "list") & length(dot.shape)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.shape ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE SHAPE VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(dot.shape) != "list") & length(dot.shape)== 1L){** *# convert the single value into a list of single value*

**list.dot.shape <- vector(mode = "list", length = length(data1))**

**list.dot.shape[] <- dot.shape**

**}**

**if( ! ((all(class(dot.border.size) == "list") & length(data1) == length(dot.border.size)) | (all(mode(dot.border.size) == "numeric") & length(dot.border.size)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.border.size ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE NUMERIC VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(dot.border.size) == "numeric") & length(dot.border.size)== 1L){** *# convert the single value into a list of single value*

**list.dot.border.size <- vector(mode = "list", length = length(data1))**

**list.dot.border.size[] <- dot.border.size**

**}**

**if( ! is.null(dot.border.color)){**

**if( ! ((all(class(dot.border.color) == "list") & length(data1) == length(dot.border.color)) | ((all(mode(dot.border.color) == "character") | all(mode(dot.border.color) == "numeric")) & length(dot.border.color)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.border.color ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE CHARACTER STRING OR INTEGER")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if((all(mode(dot.border.color) == "character") | all(mode(dot.border.color) == "numeric")) & length(dot.border.color)== 1L){** *# convert the single value into a list of single value*

**list.dot.border.color <- vector(mode = "list", length = length(data1))**

**list.dot.border.color[] <- dot.border.color**

**}**

**}**

**if( ! ((all(class(line.size) == "list") & length(data1) == length(line.size)) | (all(mode(line.size) == "numeric") & length(line.size)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": line.size ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE NUMERIC VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(line.size) == "numeric") & length(line.size)== 1L){** *# convert the single value into a list of single value*

**list.line.size <- vector(mode = "list", length = length(data1))**

**list.line.size[] <- line.size**

**}**

**if( ! ((all(class(line.type) == "list") & length(data1) == length(line.type)) | (all(mode(line.type) != "list") & length(line.type)== 1L))){** *# list of same length as data1 or single value*

**tempo.cat <- paste0("ERROR IN ", function.name, ": line.type ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST, OR A SINGLE LINE KIND VALUE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(all(mode(line.type) != "list") & length(line.type)== 1L){** *# convert the single value into a list of single value*

**list.line.type <- vector(mode = "list", length = length(data1))**

**list.line.type[] <- line.type**

**}**

**if( ! is.null(legend.name)){**

**if( ! (all(class(legend.name) == "list") & length(data1) == length(legend.name))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": legend.name ARGUMENT MUST BE A LIST OF SAME LENGTH AS data1 IF data1 IS A LIST")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**}**

*# end check list lengths (and names of data1 compartments if present)*

*# conversion into lists*

**if(all(is.data.frame(data1))){**

**data1 <- list(L1 = data1)**

**if(all(class(x) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": x ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**x <- list(L1 = x)**

**}**

**if(all(class(y) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**y <- list(L1 = y)**

**}**

**if( ! is.null(categ)){**

**if(all(class(categ) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**categ <- list(L1 = categ)**

**}**

**}**

**if( ! is.null(categ.class.order)){**

**if(all(class(categ.class.order) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.class.order ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**categ.class.order <- list(L1 = categ.class.order)**

**}**

**}**

**if( ! is.null(color)){**

**if(all(class(color) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": color ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**color <- list(L1 = color)**

**}**

**}**

**if(all(class(geom) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**geom <- list(L1 = geom)**

**}**

**if(all(class(geom.step.dir) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom.step.dir ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**geom.step.dir <- list(L1 = geom.step.dir)**

**}**

**if( ! is.null(geom.stick.base)){**

**if(all(class(geom.stick.base) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom.stick.base ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**geom.stick.base <- list(L1 = geom.stick.base)**

**}**

**}**

**if(all(class(alpha) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": alpha ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**alpha <- list(L1 = alpha)**

**}**

**if(all(class(dot.size) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.size ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**dot.size <- list(L1 = dot.size)**

**}**

**if(all(class(dot.shape) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.shape ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**dot.shape <- list(L1 = dot.shape)**

**}**

**if(all(class(dot.border.size) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.border.size ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**dot.border.size <- list(L1 = dot.border.size)**

**}**

**if( ! is.null(dot.border.color)){**

**if(all(class(dot.border.color) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.border.color ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**dot.border.color <- list(L1 = dot.border.color)**

**}**

**}**

**if(all(class(line.size) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": line.size ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**line.size <- list(L1 = line.size)**

**}**

**if(all(class(line.type) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": line.type ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**line.type <- list(L1 = line.type)**

**}**

**if( ! is.null(legend.name)){**

**if(all(class(legend.name) == "list")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": legend.name ARGUMENT CANNOT BE A LIST IF data1 IS A DATA FRAME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**legend.name <- list(L1 = legend.name)**

**}**

**}**

**}else if( ! all(sapply(data1, FUN = "class") == "data.frame")){** *# if not a data frame, data1 can only be a list, as tested above*

**tempo.cat <- paste0("ERROR IN ", function.name, ": data1 ARGUMENT MUST BE A DATA FRAME OR A LIST OF DATA FRAMES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

*# single value converted into list now reattributed to the argument name*

**if( ! is.null(color)){**

**if( ! is.null(list.color)){**

**color <- list.color**

**}**

**}**

**if( ! is.null(list.geom)){**

**geom <- list.geom**

**}**

**if( ! is.null(list.geom.step.dir)){**

**geom.step.dir <- list.geom.step.dir**

**}**

**if( ! is.null(geom.stick.base)){**

**if( ! is.null(list.geom.stick.base)){**

**geom.stick.base <- list.geom.stick.base**

**}**

**}**

**if( ! is.null(list.alpha)){**

**alpha <- list.alpha**

**}**

**if( ! is.null(list.dot.size)){**

**dot.size <- list.dot.size**

**}**

**if( ! is.null(list.dot.shape)){**

**dot.shape <- list.dot.shape**

**}**

**if( ! is.null(list.dot.border.size)){**

**dot.border.size <- list.dot.border.size**

**}**

**if( ! is.null(dot.border.color)){**

**if( ! is.null(list.dot.border.color)){**

**dot.border.color <- list.dot.border.color**

**}**

**}**

**if( ! is.null(list.line.size)){**

**line.size <- list.line.size**

**}**

**if( ! is.null(list.line.type)){**

**line.type <- list.line.type**

**}**

*# end single value converted into list now reattributed to the argument name*

**# data, x, y, geom, alpha, dot.size, shape, dot.border.size, line.size, line.type, legend.name are list now**

**# if non-null, categ, categ.class.order, legend.name, color, dot.border.color are list now**

*# end conversion into lists*

*# verif of add*

**if( ! is.null(add)){**

**if( ! grepl(pattern = "^\\s\*\\+", add)){** *# check that the add string start by +*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST START WITH \"+\": ", paste(unique(add), collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if( ! grepl(pattern = "(ggplot2|lemon)\\s\*::", add)){** *#*

**tempo.cat <- paste0("ERROR IN ", function.name, ": FOR EASIER FUNCTION DETECTION, add ARGUMENT MUST CONTAIN \"ggplot2::\" OR \"lemon::\" IN FRONT OF EACH GGPLOT2 FUNCTION: ", paste(unique(add), collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if( ! grepl(pattern = ")\\s\*$", add)){** *# check that the add string finished by )*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST FINISH BY \")\": ", paste(unique(add), collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

*# end verif of add*

*# management of add containing facet*

**facet.categ <- NULL**

**if( ! is.null(add)){**

**facet.check <- TRUE**

**tempo <- unlist(strsplit(x = add, split = "\\s\*\\+\\s\*(ggplot2|lemon)\\s\*::\\s\*"))** *#*

**tempo <- sub(x = tempo, pattern = "^facet\_wrap", replacement = "ggplot2::facet\_wrap")**

**tempo <- sub(x = tempo, pattern = "^facet\_grid", replacement = "ggplot2::facet\_grid")**

**tempo <- sub(x = tempo, pattern = "^facet\_rep", replacement = "lemon::facet\_rep")**

**if(length(data1) > 1 & (any(grepl(x = tempo, pattern = "ggplot2::facet\_wrap|lemon::facet\_rep\_wrap")) | grepl(x = add, pattern = "ggplot2::facet\_grid|lemon::facet\_rep\_grid"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nfacet PANELS CANNOT BE USED IF MORE THAN ONE DATA FRAME IN THE data1 ARGUMENT\nPLEASE REWRITE THE add STRING AND RERUN")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**if(any(grepl(x = tempo, pattern = "ggplot2::facet\_wrap|lemon::facet\_rep\_wrap"))){**

**tempo1 <- suppressWarnings(eval(parse(text = tempo[grepl(x = tempo, pattern = "ggplot2::facet\_wrap|lemon::facet\_rep\_wrap")])))**

**facet.categ <- list(names(tempo1$params$facets)) # list of length 1**

**tempo.text <- "facet\_wrap OR facet\_rep\_wrap"**

**facet.check <- FALSE**

**}else if(grepl(x = add, pattern = "ggplot2::facet\_grid|lemon::facet\_rep\_grid")){**

**tempo1 <- suppressWarnings(eval(parse(text = tempo[grepl(x = tempo, pattern = "ggplot2::facet\_grid|lemon::facet\_rep\_grid")])))**

**facet.categ <- list(c(names(tempo1$params$rows), names(tempo1$params$cols))) # list of length 1**

**tempo.text <- "facet\_grid OR facet\_rep\_grid"**

**facet.check <- FALSE**

**}**

**if(facet.check == FALSE & ! all(facet.categ %in% names(data1[[1]]))){** *# WARNING: all(facet.categ %in% names(data1)) is TRUE when facet.categ is NULL*

**tempo.cat <- paste0("ERROR IN ", function.name, "\nDETECTION OF \"", tempo.text, "\" STRING IN THE add ARGUMENT BUT PROBLEM OF VARIABLE DETECTION (COLUMN NAMES OF data1)\nTHE DETECTED VARIABLES ARE:\n", paste(facet.categ, collapse = " "), "\nTHE data1 COLUMN NAMES ARE:\n", paste(names(data1[[1]]), collapse = " "), "\nPLEASE REWRITE THE add STRING AND RERUN")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**}**

**# if facet.categ is not NULL, it is a list of length 1 now**

*# end management of add containing facet*

*# legend name filling*

**if(is.null(legend.name) & ! is.null(categ)){**

**legend.name <- categ**

**}else if(is.null(legend.name) & is.null(categ)){**

**legend.name <- vector("list", length(data1))** *# null list*

**}**

**# legend.name not NULL anymore (list)**

*# end legend name filling*

*# ini categ for legend display*

**fin.lg.disp <- vector("list", 6)** *# will be used at the end to display or not legends*

**fin.lg.disp[] <- FALSE**

**legend.disp <- vector("list", length(data1))**

**if(is.null(categ) | legend.show == FALSE){**

**legend.disp[] <- FALSE**

**}else{**

**for(i2 in 1:length(data1)){**

**if(is.null(categ[[i2]])){**

**legend.disp[[i2]] <- FALSE**

**}else{**

**legend.disp[[i2]] <- TRUE**

**}**

**}**

**}**

*# end ini categ for legend display*

*# integer colors into gg\_palette*

**tempo.check.color <- NULL**

**for(i1 in 1:length(data1)){**

**if(any(is.na(color[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), ": color ARGUMENT CANNOT CONTAIN NA")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**tempo.check.color <- c(tempo.check.color, fun\_check(data = color[[i1]], data.name = ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name)$problem)**

**}**

**tempo.check.color <- ! tempo.check.color** *# invert TRUE and FALSE because if integer, then problem = FALSE*

**if(any(tempo.check.color == TRUE)){** *# convert integers into colors*

**tempo.integer <- unlist(color[tempo.check.color])**

**tempo.color <- fun\_gg\_palette(max(tempo.integer, na.rm = TRUE))**

**for(i1 in 1:length(data1)){**

**if(tempo.check.color[i1] == TRUE){**

**color[[i1]] <-tempo.color[color[[i1]]]**

**}**

**}**

**}**

*# end integer colors into gg\_palette*

*# loop (checking inside list compartment)*

**compart.null.color <- 0** *# will be used to attribute a color when color is non-null but a compartment of color is NULL*

**data1.ini <- data1** *# to report NA removal*

**removed.row.nb <- vector("list", length = length(data1))** *# to report NA removal. Contains NULL*

**removed.rows <- vector("list", length = length(data1))** *# to report NA removal. Contains NULL*

**for(i1 in 1:length(data1)){**

**tempo <- fun\_check(data = data1[[i1]], data.name = ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), class = "data.frame", na.contain = TRUE, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

*# reserved word checking*

*# end reserved word checking*

*# check of geom now because required for y argument*

**tempo <- fun\_check(data = geom[[i1]], data.name = ifelse(length(geom)== 1L, "geom", paste0("geom NUMBER ", i1)), options = c("geom\_point", "geom\_line", "geom\_path", "geom\_step", "geom\_hline", "geom\_vline", "geom\_stick"), length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(geom[[i1]] == "geom\_step" & is.null(geom.step.dir[[i1]])){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(geom.step.dir)== 1L, "geom.step.dir", paste0("ELEMENT ", i1, " OF geom.step.dir ARGUMENT")), ": geom.step.dir ARGUMENT CANNOT BE NULL IF ", ifelse(length(geom)== 1L, "geom", paste0("ELEMENT ", i1, " OF geom")), " ARGUMENT IS \"geom\_step\"\nHERE geom.step.dir ARGUMENT IS: ", paste(geom.step.dir[[i1]], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(geom[[i1]] == "geom\_step" & ! is.null(geom.step.dir[[i1]])){**

**tempo <- fun\_check(data = geom.step.dir[[i1]], data.name = ifelse(length(geom.step.dir)== 1L, "geom.step.dir", paste0("geom.step.dir NUMBER ", i1)), options = c("vh", "hv", "mid"), length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if( ! (is.null(geom.stick.base))){**

**if(geom[[i1]] == "geom\_stick" & ! is.null(geom.stick.base[[i1]])){**

**tempo <- fun\_check(data = geom.stick.base[[i1]], data.name = ifelse(length(geom.stick.base)== 1L, "geom.stick.base", paste0("geom.stick.base NUMBER ", i1)), mode = "numeric", length = 1, na.contain = FALSE, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**}**

*# end check of geom now because required for y argument*

**if(is.null(x[[i1]])){**

**if(all(geom[[i1]] != "geom\_hline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(x)== 1L, "x", paste0("ELEMENT ", i1, " OF x ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ": x ARGUMENT CANNOT BE NULL EXCEPT IF ", ifelse(length(geom)== 1L, "x", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_hline\"\nHERE geom ARGUMENT IS: ", paste(geom[[i1]], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**x[[i1]] <- "fake\_x"**

**data1[[i1]] <- cbind(data1[[i1]], fake\_x = NA, stringsAsFactors = TRUE)**

**data1[[i1]][, "fake\_x"] <- as.numeric(data1[[i1]][, "fake\_x"])**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL ", ifelse(length(x)== 1L, "x", paste0("ELEMENT ", i1, " OF x")), " ARGUMENT ASSOCIATED TO ", ifelse(length(geom)== 1L, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT ", geom[[i1]], " -> FAKE COLUMN ADDED TO DATA FRAME ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", NAMED \"fake\_x\" FOR FINAL DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else{**

**if(all(geom[[i1]] == "geom\_hline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(x)== 1L, "x", paste0("ELEMENT ", i1, " OF x ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ": x ARGUMENT MUST BE NULL IF ", ifelse(length(geom)== 1L, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_hline\"")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**tempo <- fun\_check(data = x[[i1]], data.name = ifelse(length(x)== 1L, "x", paste0("ELEMENT ", i1, " OF x ARGUMENT")), class = "vector", mode = "character", length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if(is.null(y[[i1]])){**

**if(all(geom[[i1]] != "geom\_vline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(y)== 1L, "y", paste0("ELEMENT ", i1, " OF y ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ": y ARGUMENT CANNOT BE NULL EXCEPT IF ", ifelse(length(geom)== 1L, "y", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_vline\"\nHERE geom ARGUMENT IS: ", paste(geom[[i1]], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**y[[i1]] <- "fake\_y"**

**data1[[i1]] <- cbind(data1[[i1]], fake\_y = NA, stringsAsFactors = TRUE)**

**data1[[i1]][, "fake\_y"] <- as.numeric(data1[[i1]][, "fake\_y"])**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL ", ifelse(length(y)== 1L, "y", paste0("ELEMENT ", i1, " OF y")), " ARGUMENT ASSOCIATED TO ", ifelse(length(geom)== 1L, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT ", geom[[i1]], " -> FAKE COLUMN ADDED TO DATA FRAME ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", NAMED \"fake\_y\" FOR FINAL DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else{**

**if(all(geom[[i1]] == "geom\_vline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(y)== 1L, "y", paste0("ELEMENT ", i1, " OF y ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ": y ARGUMENT MUST BE NULL IF ", ifelse(length(geom)== 1L, "geom", paste0("geom NUMBER ", i1)), " ARGUMENT IS \"geom\_vline\"")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**tempo <- fun\_check(data = y[[i1]], data.name = ifelse(length(y)== 1L, "y", paste0("ELEMENT ", i1, " OF y ARGUMENT")), class = "vector", mode = "character", length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**# x[[i1]] and y[[i1]] not NULL anymore**

**if( ! (x[[i1]] %in% names(data1[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(x)== 1L, "x", paste0("ELEMENT ", i1, " OF x")), " ARGUMENT MUST BE A COLUMN NAME OF ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT\nHERE IT IS: ", paste(x[[i1]], collapse = " "))))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if( ! (y[[i1]] %in% names(data1[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(y)== 1L, "y", paste0("ELEMENT ", i1, " OF y")), " ARGUMENT MUST BE A COLUMN NAME OF ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT\nHERE IT IS: ", paste(y[[i1]], collapse = " "))))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**tempo <- fun\_check(data = data1[[i1]][, x[[i1]]], data.name = ifelse(length(x)== 1L, "x ARGUMENT (AS COLUMN NAME OF data1 DATA FRAME)", paste0("ELEMENT ", i1, " OF x ARGUMENT", " (AS COLUMN NAME OF data1 DATA FRAME NUMBER ", i1, ")")), class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**tempo <- fun\_check(data = data1[[i1]][, y[[i1]]], data.name = ifelse(length(y)== 1L, "y ARGUMENT (AS COLUMN NAME OF data1 DATA FRAME)", paste0("ELEMENT ", i1, " OF y ARGUMENT", " (AS COLUMN NAME OF data1 DATA FRAME NUMBER ", i1, ")")), class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(x[[i1]] == "fake\_x" & y[[i1]] == "fake\_y"){** *# because the code cannot accept to be both "fake\_x" and "fake\_y" at the same time*

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 2\nTHE CODE CANNOT ACCEPT x AND y TO BE \"fake\_x\" AND \"fake\_y\" IN THE SAME DATA FRAME ", i1, " ")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(( ! is.null(categ)) & ( ! is.null(categ[[i1]]))){** *# is.null(categ[[i1]]) works even if categ is NULL # is.null(categ[[i1]]) works even if categ is NULL # if categ[[i1]] = NULL, fake\_categ will be created later on*

**tempo <- fun\_check(data = categ[[i1]], data.name = ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")),, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if( ! (categ[[i1]] %in% names(data1[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ")), " ARGUMENT MUST BE A COLUMN NAME OF ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT\nHERE IT IS: ", paste(categ[[i1]], collapse = " "))))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**tempo1 <- fun\_check(data = data1[[i1]][, categ[[i1]]], data.name = ifelse(length(categ)== 1L, "categ OF data1 ARGUMENT", paste0("ELEMENT ", i1, " OF categ ARGUMENT IN DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name)**

**tempo2 <- fun\_check(data = data1[[i1]][, categ[[i1]]], data.name = ifelse(length(categ)== 1L, "categ OF data1 ARGUMENT", paste0("ELEMENT ", i1, " OF categ ARGUMENT IN DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), class = "factor", na.contain = TRUE, fun.name = function.name)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(categ)== 1L, "categ OF data1 ARGUMENT", paste0("ELEMENT ", i1, " OF categ ARGUMENT IN DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " MUST BE A FACTOR OR CHARACTER VECTOR")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(tempo1$problem == FALSE){**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") IN ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", THE CHARACTER COLUMN HAS BEEN CONVERTED TO FACTOR")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if(geom[[i1]] == "geom\_vline" | geom[[i1]] == "geom\_hline"){**

**if(length(unique(data1[[i1]][, categ[[i1]]])) != nrow(data1[[i1]])){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(geom)== 1L, "geom OF data1 ARGUMENT", paste0("geom NUMBER ", i1, " OF DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " ARGUMENT IS ", geom[[i1]], ", MEANING THAT ", ifelse(length(categ)== 1L, "categ OF data1 ARGUMENT", paste0("ELEMENT ", i1, " OF categ ARGUMENT IN DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " MUST HAVE A DIFFERENT CLASS PER LINE OF data1 (ONE x VALUE PER CLASS)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**}else if(( ! is.null(categ)) & is.null(categ[[i1]])){** *# is.null(categ[[i1]]) works even if categ is NULL # if categ[[i1]] = NULL, fake\_categ will be created. WARNING: is.null(categ[[i1]]) means no legend display (see above), because categ has not been precised. This also means a single color for data1[[i1]]*

**if(length(color[[i1]]) > 1){** *# 0 means is.null(color[[i1]]) or is.null(color) and 1 is ok -> single color for data1[[i1]]*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ")), " ARGUMENT BUT CORRESPONDING COLORS IN ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " HAS LENGTH OVER 1\n", paste(color[[i1]], collapse = " "), "\nWHICH IS NOT COMPATIBLE WITH NULL CATEG -> COLOR RESET TO A SINGLE COLOR")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**color[i1] <- list(NULL)** *# will provide a single color below # Warning color[[i1]] <- NULL removes the compartment*

**}**

**categ[[i1]] <- "fake\_categ"**

**data1[[i1]] <- cbind(data1[[i1]], fake\_categ = "", stringsAsFactors = TRUE)**

*# inactivated because give a different color to different "Line\_" categ while a single color for all the data1[[i1]] required. Thus, put back after the color management*

*# if(geom[[i1]] == "geom\_hline" | geom[[i1]] == "geom\_vline"){*

*# data1[[i1]][, "fake\_categ"] <- paste0("Line\_", 1:nrow(data1[[i1]]))*

*# }else{*

**data1[[i1]][, "fake\_categ"] <- data1[[i1]][, "fake\_categ"]** *# as.numeric("") create a vector of NA but class numeric*

*# }*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ")), " ARGUMENT -> FOR DATA FRAME ", ifelse(length(data1)== 1L, "data1 ARGUMENT:", paste0("NUMBER ", i1, " OF data1 ARGUMENT:")), "\n- FAKE \"fake\_categ\" COLUMN ADDED FILLED WITH \"\"(OR WITH \"Line\_...\" FOR LINES)\n- SINGLE COLOR USED FOR PLOTTING\n- NO LEGEND DISPLAYED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**# OK: if categ is not NULL, all the non-null categ columns of data1 are factors from here**

*# management of log scale and Inf removal*

**if(x[[i1]] != "fake\_x"){**

**if(any(( ! is.finite(data1[[i1]][, x[[i1]]])) & ( ! is.na(data1[[i1]][, x[[i1]]])))){** *# is.finite also detects NA: ( ! is.finite(data1[, y])) & ( ! is.na(data1[, y])) detects only Inf*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") PRESENCE OF -Inf OR Inf VALUES IN ", ifelse(length(categ)== 1L, "x", paste0("ELEMENT ", i1, " OF x ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if(y[[i1]] != "fake\_y"){**

**if(any(( ! is.finite(data1[[i1]][, y[[i1]]])) & ( ! is.na(data1[[i1]][, y[[i1]]])))){** *# is.finite also detects NA: ( ! is.finite(data1[, y])) & ( ! is.na(data1[, y])) detects only Inf*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") PRESENCE OF -Inf OR Inf VALUES IN ", ifelse(length(categ)== 1L, "y", paste0("ELEMENT ", i1, " OF y ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# log conversion*

**if(x.log != "no"){**

**tempo1 <- ! is.finite(data1[[i1]][, x[[i1]]]) # where are initial NA and Inf**

**data1[[i1]][, x[[i1]]] <- suppressWarnings(get(x.log)(data1[[i1]][, x[[i1]]]))***# no env = sys.nframe(), inherit = FALSE in get() because look for function in the classical scope*

**if(any( ! (tempo1 | is.finite(data1[[i1]][, x[[i1]]])))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") LOG CONVERSION INTRODUCED -Inf OR Inf OR NaN VALUES IN ", ifelse(length(categ)== 1L, "x", paste0("ELEMENT ", i1, " OF x ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if(y.log != "no"){**

**tempo1 <- ! is.finite(data1[[i1]][, y[[i1]]]) # where are initial NA and Inf**

**data1[[i1]][, y[[i1]]] <- suppressWarnings(get(y.log)(data1[[i1]][, y[[i1]]]))***# no env = sys.nframe(), inherit = FALSE in get() because look for function in the classical scope*

**if(any( ! (tempo1 | is.finite(data1[[i1]][, y[[i1]]])))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") LOG CONVERSION INTRODUCED -Inf OR Inf OR NaN VALUES IN ", ifelse(length(categ)== 1L, "y", paste0("ELEMENT ", i1, " OF y ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# Inf removal*

*# removed.row.nb[[i1]] <- NULL # already NULL and Warning this removes the compartment*

**removed.rows[[i1]] <- data.frame(stringsAsFactors = FALSE)**

**if(any(( ! is.finite(data1[[i1]][, x[[i1]]])) & ( ! is.na(data1[[i1]][, x[[i1]]])))){** *# is.finite also detects NA: ( ! is.finite(data1[[i1]][, x[[i1]]])) & ( ! is.na(data1[[i1]][, x[[i1]]])) detects only Inf*

**removed.row.nb[[i1]] <- c(removed.row.nb[[i1]], which(( ! is.finite(data1[[i1]][, x[[i1]]])) & ( ! is.na(data1[[i1]][, x[[i1]]]))))**

**}**

**if(any(( ! is.finite(data1[[i1]][, y[[i1]]])) & ( ! is.na(data1[[i1]][, y[[i1]]])))){** *# is.finite also detects NA: ( ! is.finite(data1[[i1]][, y[[i1]]])) & ( ! is.na(data1[[i1]][, y[[i1]]])) detects only Inf*

**removed.row.nb[[i1]] <- c(removed.row.nb[[i1]], which(( ! is.finite(data1[[i1]][, y[[i1]]])) & ( ! is.na(data1[[i1]][, y[[i1]]]))))**

**}**

**if( ! is.null(removed.row.nb[[i1]])){**

**removed.row.nb[[i1]] <- unique(removed.row.nb[[i1]])** *# to remove the duplicated positions (NA in both x and y)*

**removed.rows[[i1]] <- rbind(removed.rows[[i1]], data1.ini[[i1]][removed.row.nb[[i1]], ])** *# here data1.ini used to have the y = O rows that will be removed because of Inf creation after log transformation*

**data1[[i1]] <- data1[[i1]][-removed.row.nb[[i1]], ]**

**data1.ini[[i1]] <- data1.ini[[i1]][-removed.row.nb[[i1]], ]** *#*

**}**

**# From here, data1 and data.ini have no more Inf**

*# end Inf removal*

*# x.lim and y.lim dealt later on, after the end f the loop*

*# end management of log scale and Inf removal*

*# na detection and removal*

**column.check <- unique(unlist(c( # unlist because creates a list**

**if(x[[i1]] == "fake\_x"){NULL}else{x[[i1]]},**

**if(y[[i1]] == "fake\_y"){NULL}else{y[[i1]]},**

**if( ! is.null(categ)){if(is.null(categ[[i1]])){NULL}else{categ[[i1]]}},**

**if( ! is.null(facet.categ)){if(is.null(facet.categ[[i1]])){NULL}else{facet.categ[[i1]]}}**

**)))** *# dot.categ because can be a 3rd column of data1*

**if(any(is.na(data1[[i1]][, column.check]))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NA DETECTED IN COLUMNS ", paste(column.check, collapse = " "), " OF ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**for(i3 in 1:length(column.check)){**

**if(any(is.na(data1[[i1]][, column.check[i3]]))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("NA REMOVAL DUE TO COLUMN ", column.check[i3], " OF ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**tempo <- unique(unlist(lapply(lapply(c(data1[[i1]][column.check]), FUN = is.na), FUN = which)))**

**removed.row.nb[[i1]] <- c(removed.row.nb[[i1]], tempo)**

**removed.rows[[i1]] <- rbind(removed.rows[[i1]], data1.ini[[i1]][tempo, ])** *# # tempo used because removed.row.nb is not empty. Here data1.ini used to have the non NA rows that will be removed because of NAN creation after log transformation (neg values for instance)*

**column.check <- column.check[ ! (column.check == x[[i1]] | column.check == y[[i1]])]** *# remove x and y to keep quali columns*

**if(length(tempo) != 0){**

**data1[[i1]] <- data1[[i1]][-tempo, ]** *# WARNING tempo here and not removed.row.nb because the latter contain more numbers thant the former*

**data1.ini[[i1]] <- data1.ini[[i1]][-tempo, ]** *# WARNING tempo here and not removed.row.nb because the latter contain more numbers than the former*

**for(i4 in 1:length(column.check)){**

**if(any( ! unique(removed.rows[[i1]][, column.check[i4]]) %in% unique(data1[[i1]][, column.check[i4]]))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") IN COLUMN ", column.check[i4], " OF ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", THE FOLLOWING CLASSES HAVE DISAPPEARED AFTER NA REMOVAL\n(IF COLUMN USED IN THE PLOT, THIS CLASS WILL NOT BE DISPLAYED):\n", paste(unique(removed.rows[[i1]][, column.check[i4]])[ ! unique(removed.rows[[i1]][, column.check[i4]]) %in% unique(data1[[i1]][, column.check[i4]])], collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**tempo.levels <- levels(data1[[i1]][, column.check[i4]])[levels(data1[[i1]][, column.check[i4]]) %in% unique(as.character(data1[[i1]][, column.check[i4]]))]**

**data1[[i1]][, column.check[i4]] <- factor(as.character(data1[[i1]][, column.check[i4]]), levels = tempo.levels)**

**if(column.check[i4] %in% categ[[i1]] & ! is.null(categ.class.order)){**

**categ.class.order[[i1]] <- levels(data1[[i1]][, column.check[i4]])[levels(data1[[i1]][, column.check[i4]]) %in% unique(data1[[i1]][, column.check[i4]])] # remove the absent class in the categ.class.order vector**

**data1[[i1]][, column.check[i4]] <- factor(as.character(data1[[i1]][, column.check[i4]]), levels = unique(categ.class.order[[i1]]))**

**}**

**}**

**}**

**}**

**}**

*# end na detection and removal*

**# From here, data1 and data.ini have no more NA or NaN in x, y, categ (if categ != NULL) and facet.categ (if categ != NULL)**

**if( ! is.null(categ.class.order)){**

*# the following check will be done several times but I prefer to keep it here, after the creation of categ*

**if(is.null(categ[[i1]]) & ! is.null(categ.class.order[[i1]])){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nCOMPARTMENT ", i1, " OF categ ARGUMENT CANNOT BE NULL IF COMPARTMENT ", i1, " OF categ.class.order ARGUMENT IS NOT NULL: ", paste(categ.class.order, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**if(is.null(categ.class.order[[i1]])){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE categ.class.order COMPARTMENT ", i1, " IS NULL. ALPHABETICAL ORDER WILL BE APPLIED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**data1[[i1]][, categ[[i1]]] <- factor(as.character(data1[[i1]][, categ[[i1]]]))** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.class.order[[i1]] <- levels(data1[[i1]][, categ[[i1]]])** *# character vector that will be used later*

**}else{**

**tempo <- fun\_check(data = categ.class.order[[i1]], data.name = paste0("COMPARTMENT ", i1 , " OF categ.class.order ARGUMENT"), class = "vector", mode = "character", length = length(levels(data1[[i1]][, categ[[i1]]])), fun.name = function.name)** *# length(data1[, categ[i1]) -> if data1[, categ[i1] was initially character vector, then conversion as factor after the NA removal, thus class number ok. If data1[, categ[i1] was initially factor, no modification after the NA removal, thus class number ok*

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if(any(duplicated(categ.class.order[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nCOMPARTMENT ", i1, " OF categ.class.order ARGUMENT CANNOT HAVE DUPLICATED CLASSES: ", paste(categ.class.order[[i1]], collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if( ! (all(categ.class.order[[i1]] %in% unique(data1[[i1]][, categ[[i1]]])) & all(unique(data1[[i1]][, categ[[i1]]]) %in% categ.class.order[[i1]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nCOMPARTMENT ", i1, " OF categ.class.order ARGUMENT MUST BE CLASSES OF COMPARTMENT ", i1, " OF categ ARGUMENT\nHERE IT IS:\n", paste(categ.class.order[[i1]], collapse = " "), "\nFOR COMPARTMENT ", i1, " OF categ.class.order AND IT IS:\n", paste(unique(data1[[i1]][, categ[[i1]]]), collapse = " "), "\nFOR COLUMN ", categ[[i1]], " OF data1 NUMBER ", i1)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]], levels = categ.class.order[[i1]])** *# reorder the factor*

**}**

**names(categ.class.order)[i1] <- categ[[i1]]**

**}**

**}**

**# OK: if categ.class.order is not NULL, all the NULL categ.class.order columns of data1 are character from here**

**if( ! is.null(legend.name[[i1]])){**

**tempo <- fun\_check(data = legend.name[[i1]], data.name = ifelse(length(legend.name)== 1L, "legend.name", paste0("legend.name NUMBER ", i1)),, class = "vector", mode = "character", length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if( ! is.null(color)){** *# if color is NULL, will be filled later on*

*# check the nature of color*

**if(is.null(color[[i1]])){**

**compart.null.color <- compart.null.color + 1**

**color[[i1]] <- grey(compart.null.color / 8)** *# cannot be more than 7 overlays. Thus 7 different greys. 8/8 is excluded because white dots*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL COLOR IN ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " ASSOCIATED TO ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", SINGLE COLOR ", paste(color[[i1]], collapse = " "), " HAS BEEN ATTRIBUTED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**tempo1 <- fun\_check(data = color[[i1]], data.name = ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name) # na.contain = TRUE in case of colum of data1**

**tempo2 <- fun\_check(data = color[[i1]], data.name = ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), class = "factor", na.contain = TRUE, fun.name = function.name) # idem**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR")** *# integer possible because dealt above*

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if( ! (all(color[[i1]] %in% colors() | grepl(pattern = "^#", color[[i1]])))){** *# check that all strings of low.color start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors() OR A COLUMN NAME OF THE data1 PARAMETER: ", paste(unique(color[[i1]]), collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(any(is.na(color[[i1]]))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") IN ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), ", THE COLORS:\n", paste(unique(color[[i1]]), collapse = " "), "\nCONTAINS NA")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end check the nature of color*

*# check the length of color*

**if(is.null(categ) & length(color[[i1]]) != 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST BE A SINGLE COLOR IF categ IS NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if( ! is.null(categ)){**

*# No problem of NA management by ggplot2 because already removed*

**if(categ[[i1]] == "fake\_categ" & length(color[[i1]]) != 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST BE A SINGLE COLOR IF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IS NULL")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(length(color[[i1]]) == length(unique(data1[[i1]][, categ[[i1]]]))){** *# here length(color) is equal to the different number of categ*

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") IN ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", THE FOLLOWING COLORS:\n", paste(color[[i1]], collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[[i1]][, categ[[i1]]])), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(length(color[[i1]]) == length(data1[[i1]][, categ[[i1]]])){***# here length(color) is equal to nrow(data1[[i1]]) -> Modif to have length(color) equal to the different number of categ (length(color) == length(levels(data1[[i1]][, categ[[i1]]])))*

**data1[[i1]] <- cbind(data1[[i1]], color = color[[i1]], stringsAsFactors = TRUE)**

**tempo.check <- unique(data1[[i1]][ , c(categ[[i1]], "color")])**

**if( ! (nrow(data1[[i1]]) == length(color[[i1]]) & nrow(tempo.check) == length(unique(data1[[i1]][ , categ[[i1]]])))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color")), " ARGUMENT HAS THE LENGTH OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), "\nBUT IS INCORRECTLY ASSOCIATED TO EACH CLASS OF THIS categ:\n", paste(unique(mapply(FUN = "paste", data1[[i1]][ ,categ[[i1]]], data1[[i1]][ ,"color"])), collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**color[[i1]] <- unique(color[[i1]][order(data1[[i1]][, categ[[i1]]])])** *# Modif to have length(color) equal to the different number of categ (length(color) == length(levels(data1[[i1]][, categ[[i1]]])))*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count, ") FROM FUNCTION ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " HAS THE LENGTH OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " COLUMN VALUES\nCOLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ AS:\n", paste(levels(factor(data1[[i1]][, categ[[i1]]])), collapse = " "), "\n", paste(color[[i1]], collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else if(length(color[[i1]])== 1L){**

**data1[[i1]][, categ[[i1]]] <- factor(data1[[i1]][, categ[[i1]]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**color[[i1]] <- rep(color[[i1]], length(levels(data1[[i1]][, categ[[i1]]])))**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") IN ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), ", COLOR HAS LENGTH 1 MEANING THAT ALL THE DIFFERENT CLASSES OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), "\n", paste(levels(factor(data1[[i1]][, categ[[i1]]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(color[[i1]], collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST BE\n(1) LENGTH 1\nOR (2) THE LENGTH OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), " COLUMN VALUES\nOR (3) THE LENGTH OF THE CLASSES IN THIS COLUMN\nHERE IT IS COLOR LENGTH ", length(color[[i1]]), " VERSUS CATEG LENGTH ", length(data1[[i1]][, categ[[i1]]]), " AND CATEG CLASS LENGTH ", length(unique(data1[[i1]][, categ[[i1]]])), "\nPRESENCE OF NA IN THE COLUMN x, y OR categ OF data1 COULD BE THE PROBLEME")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**}**

**if((geom[[i1]] == "geom\_hline" | geom[[i1]] == "geom\_vline") & ! is.null(categ[[i1]])){** *# add here after the color management, to deal with the different lines to plot inside any data[[i1]]*

**if(categ[[i1]] == "fake\_categ"){**

**data1[[i1]][, "fake\_categ"] <- factor(paste0("Line\_", formatC(1:nrow(data1[[i2]]), width = nchar(nrow(data1[[i2]])), flag = "0")))**

**}**

**}**

**tempo <- fun\_check(data = alpha[[i1]], data.name = ifelse(length(alpha)== 1L, "alpha", paste0("alpha NUMBER ", i1)), prop = TRUE, length = 1, fun.name = function.name)**

**if(tempo$problem == TRUE){**

**stop(paste0("\n\n================\n\n", tempo$text, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

*# end loop (checking inside list compartment)*

**if(length(data1) > 1){**

**if(length(unique(unlist(x)[ ! x == "fake\_x"])) > 1){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE x ARGUMENT DOES NOT CONTAIN IDENTICAL COLUMN NAMES:\n", paste(unlist(x), collapse = " "), "\nX-AXIS OVERLAYING DIFFERENT VARIABLES?")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if(length(data1) > 1){**

**if(length(unique(unlist(y)[ ! y == "fake\_y"])) > 1){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE y ARGUMENT DOES NOT CONTAIN IDENTICAL COLUMN NAMES:\n", paste(unlist(y), collapse = " "), "\nY-AXIS OVERLAYING DIFFERENT VARIABLES?")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if(sum(geom %in% "geom\_point") > 3){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT CANNOT HAVE MORE THAN THREE \"geom\_point\" ELEMENTS")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(length(geom) - sum(geom %in% "geom\_point") > 3){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": geom ARGUMENT CANNOT HAVE MORE THAN THREE LINE ELEMENTS")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

*# x.lim management before transfo by x.log*

**if(x.log != "no" & ! is.null(x.lim)){**

**if(any(x.lim <= 0)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nx.lim ARGUMENT CANNOT HAVE ZERO OR NEGATIVE VALUES WITH THE x.log ARGUMENT SET TO ", x.log, ":\n", paste(x.lim, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(any( ! is.finite(if(x.log == "log10"){log10(x.lim)}else{log2(x.lim)}))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nx.lim ARGUMENT RETURNS INF/NA WITH THE x.log ARGUMENT SET TO ", x.log, "\nAS SCALE COMPUTATION IS ", ifelse(x.log == "log10", "log10", "log2"), ":\n", paste(if(x.log == "log10"){log10(x.lim)}else{log2(x.lim)}, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if(x.log != "no" & x.include.zero == TRUE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") x.log ARGUMENT SET TO ", x.log, " AND x.include.zero ARGUMENT SET TO TRUE -> x.include.zero ARGUMENT RESET TO FALSE BECAUSE 0 VALUE CANNOT BE REPRESENTED IN LOG SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**x.include.zero <- FALSE**

**}**

*# end x.lim management before transfo by x.log*

*# y.lim management before transfo by y.log*

**if(y.log != "no" & ! is.null(y.lim)){**

**if(any(y.lim <= 0)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\ny.lim ARGUMENT CANNOT HAVE ZERO OR NEGATIVE VALUES WITH THE y.log ARGUMENT SET TO ", y.log, ":\n", paste(y.lim, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(any( ! is.finite(if(y.log == "log10"){log10(y.lim)}else{log2(y.lim)}))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\ny.lim ARGUMENT RETURNS INF/NA WITH THE y.log ARGUMENT SET TO ", y.log, "\nAS SCALE COMPUTATION IS ", ifelse(y.log == "log10", "log10", "log2"), ":\n", paste(if(y.log == "log10"){log10(y.lim)}else{log2(y.lim)}, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if(y.log != "no" & y.include.zero == TRUE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") y.log ARGUMENT SET TO ", y.log, " AND y.include.zero ARGUMENT SET TO TRUE -> y.include.zero ARGUMENT RESET TO FALSE BECAUSE 0 VALUE CANNOT BE REPRESENTED IN LOG SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**y.include.zero <- FALSE**

**}**

*# end y.lim management before transfo by y.log*

*# end other checkings*

*# reserved word checking*

*#already done above*

*# end reserved word checking*

*# end second round of checking and data preparation*

*# package checking*

**fun\_pack(req.package = c(**

**"gridExtra",**

**"ggplot2",**

**"lemon",**

**"scales"**

**), lib.path = lib.path)**

*# packages Cairo and grid tested by fun\_gg\_point\_rast()*

*# end package checking*

*# main code*

*# axes management*

**if(is.null(x.lim)){**

**if(any(unlist(mapply(FUN = "[[", data1, x, SIMPLIFY = FALSE)) %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE x COLUMN IN data1 CONTAINS -Inf OR Inf VALUES THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**x.lim <- suppressWarnings(range(unlist(mapply(FUN = "[[", data1, x, SIMPLIFY = FALSE)), na.rm = TRUE, finite = TRUE))** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only. y.lim added here. If NULL, ok if y argument has values*

**}else if(x.log != "no"){**

**x.lim <- get(x.log)(x.lim)** *# no env = sys.nframe(), inherit = FALSE in get() because look for function in the classical scope*

**}**

**if(x.log != "no"){**

**if(any( ! is.finite(x.lim))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nx.lim ARGUMENT CANNOT HAVE ZERO OR NEGATIVE VALUES WITH THE x.log ARGUMENT SET TO ", x.log, ":\n", paste(x.lim, collapse = " "), "\nPLEASE, CHECK DATA VALUES (PRESENCE OF ZERO OR INF VALUES)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if(suppressWarnings(all(x.lim %in% c(Inf, -Inf)))){** *# happen when x is only NULL*

**if(all(unlist(geom) %in% c("geom\_vline", "geom\_stick"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " NOT POSSIBLE TO DRAW geom\_vline OR geom\_stick KIND OF LINES ALONE IF x.lim ARGUMENT IS SET TO NULL, SINCE NO X-AXIS DEFINED (", ifelse(length(x)== 1L, "x", paste0("ELEMENT ", i1, " OF x")), " ARGUMENT MUST BE NULL FOR THESE KIND OF LINES)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, " x.lim ARGUMENT MADE OF NA, -Inf OR Inf ONLY: ", paste(x.lim, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**x.lim.order <- order(x.lim)** *# to deal with inverse axis*

**# print(x.lim.order)**

**x.lim <- sort(x.lim)**

**x.lim[1] <- x.lim[1] - abs(x.lim[2] - x.lim[1]) \* ifelse(diff(x.lim.order) > 0, x.right.extra.margin, x.left.extra.margin)** *# diff(x.lim.order) > 0 means not inversed axis*

**x.lim[2] <- x.lim[2] + abs(x.lim[2] - x.lim[1]) \* ifelse(diff(x.lim.order) > 0, x.left.extra.margin, x.right.extra.margin)** *# diff(x.lim.order) > 0 means not inversed axis*

**if(x.include.zero == TRUE){** *# no need to check x.log != "no" because done before*

**x.lim <- range(c(x.lim, 0), na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**x.lim <- x.lim[x.lim.order]**

**if(any(is.na(x.lim))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 3")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(is.null(y.lim)){**

**if(any(unlist(mapply(FUN = "[[", data1, y, SIMPLIFY = FALSE)) %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE y COLUMN IN data1 CONTAINS -Inf OR Inf VALUES THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**y.lim <- suppressWarnings(range(unlist(mapply(FUN = "[[", data1, y, SIMPLIFY = FALSE)), na.rm = TRUE, finite = TRUE))** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only. y.lim added here. If NULL, ok if y argument has values*

**}else if(y.log != "no"){**

**y.lim <- get(y.log)(y.lim)** *# no env = sys.nframe(), inherit = FALSE in get() because look for function in the classical scope*

**}**

**if(y.log != "no"){**

**if(any( ! is.finite(y.lim))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\ny.lim ARGUMENT CANNOT HAVE ZERO OR NEGATIVE VALUES WITH THE y.log ARGUMENT SET TO ", y.log, ":\n", paste(y.lim, collapse = " "), "\nPLEASE, CHECK DATA VALUES (PRESENCE OF ZERO OR INF VALUES)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**if(suppressWarnings(all(y.lim %in% c(Inf, -Inf)))){** *# happen when y is only NULL*

**if(all(unlist(geom) == "geom\_vline")){**

**tempo.cat <- paste0("ERROR IN ", function.name, " NOT POSSIBLE TO DRAW geom\_vline KIND OF LINES ALONE IF y.lim ARGUMENT IS SET TO NULL, SINCE NO Y-AXIS DEFINED (", ifelse(length(y)== 1L, "y", paste0("ELEMENT ", i1, " OF y")), " ARGUMENT MUST BE NULL FOR THESE KIND OF LINES)")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, " y.lim ARGUMENT MADE OF NA, -Inf OR Inf ONLY: ", paste(y.lim, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**y.lim.order <- order(y.lim)** *# to deal with inverse axis*

**y.lim <- sort(y.lim)**

**y.lim[1] <- y.lim[1] - abs(y.lim[2] - y.lim[1]) \* ifelse(diff(y.lim.order) > 0, y.bottom.extra.margin, y.top.extra.margin)** *# diff(y.lim.order) > 0 means not inversed axis*

**y.lim[2] <- y.lim[2] + abs(y.lim[2] - y.lim[1]) \* ifelse(diff(y.lim.order) > 0, y.top.extra.margin, y.bottom.extra.margin)** *# diff(y.lim.order) > 0 means not inversed axis*

**if(y.include.zero == TRUE){** *# no need to check y.log != "no" because done before*

**y.lim <- range(c(y.lim, 0), na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**y.lim <- y.lim[y.lim.order]**

**if(any(is.na(y.lim))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 4")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

*# end axes management*

*# create a fake categ if NULL to deal with legend display*

**if(is.null(categ)){**

**categ <- vector("list", length(data1))**

**categ[] <- "fake\_categ"**

**for(i2 in 1:length(data1)){**

**data1[[i2]] <- cbind(data1[[i2]], fake\_categ = "", stringsAsFactors = TRUE)**

**if(geom[[i2]] == "geom\_hline" | geom[[i2]] == "geom\_vline"){**

**data1[[i2]][, "fake\_categ"] <- factor(paste0("Line\_", 1:nrow(data1[[i2]])))**

**}**

**}**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL categ ARGUMENT -> FAKE \"fake\_categ\" COLUMN ADDED TO EACH DATA FRAME OF data1, AND FILLED WITH \"\"")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**# categ is not NULL anymore**

**if(is.null(categ.class.order)){**

**categ.class.order <- vector("list", length = length(data1))**

**tempo.categ.class.order <- NULL**

**for(i2 in 1:length(categ.class.order)){**

**categ.class.order[[i2]] <- levels(data1[[i2]][, categ[[i2]]])**

**names(categ.class.order)[i2] <- categ[[i2]]**

**tempo.categ.class.order <- c(tempo.categ.class.order, ifelse(i2 != 1, "\n", ""), categ.class.order[[i2]])**

**}**

**if(any(unlist(legend.disp))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") THE categ.class.order SETTING IS NULL. ALPHABETICAL ORDER WILL BE APPLIED FOR CLASS ORDERING:\n", paste(tempo.categ.class.order, collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# end create a fake categ if NULL to deal with legend display*

**# categ.class.order is not NULL anymore**

*# vector of color with length as in levels(categ) of data1*

**if(is.null(color)){**

**color <- vector("list", length(data1))**

**length.categ.list <- lapply(lapply(mapply(FUN = "[[", data1, categ, SIMPLIFY = FALSE), FUN = unique), FUN = function(x){length(x[ ! is.na(x)])})**

**length.categ.list[sapply(categ, FUN = "==", "fake\_categ")] <- 1** *# when is.null(color), a single color for all the dots or lines of data[[i1]] that contain "fake\_categ" category*

**total.categ.length <- sum(unlist(length.categ.list), na.rm = TRUE)**

**tempo.color <- fun\_gg\_palette(total.categ.length)**

**tempo.count <- 0**

**for(i2 in 1:length(data1)){**

**color[[i2]] <- tempo.color[(1:length.categ.list[[i2]]) + tempo.count]**

**tempo.count <- tempo.count + length.categ.list[[i2]]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") NULL color ARGUMENT -> COLORS RESPECTIVELY ATTRIBUTED TO EACH CLASS OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i2, " OF categ ARGUMENT")), " (", categ[[i2]], ") IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i2, " OF data1 ARGUMENT")), ":\n", paste(color[[i2]], collapse = " "), "\n", paste(if(all(levels(data1[[i2]][, categ[[i2]]]) == "")){'\"\"'}else{levels(data1[[i2]][, categ[[i2]]])}, collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# end vector of color with length as in levels(categ) of data1*

**# color is not NULL anymore**

*# last check*

**for(i1 in 1:length(data1)){**

**if(categ[[i1]] != "fake\_categ" & length(color[[i1]]) != length(unique(data1[[i1]][, categ[[i1]]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " LAST CHECK: ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST HAVE THE LENGTH OF LEVELS OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), "\nHERE IT IS COLOR LENGTH ", length(color[[i1]]), " VERSUS CATEG LEVELS LENGTH ", length(unique(data1[[i1]][, categ[[i1]]])), "\nREMINDER: A SINGLE COLOR PER CLASS OF CATEG AND A SINGLE CLASS OF CATEG PER COLOR MUST BE RESPECTED")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else if(categ[[i1]] == "fake\_categ" & length(color[[i1]]) != 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, " LAST CHECK: ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST HAVE LENGTH 1 WHEN ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IS NULL\nHERE IT IS COLOR LENGTH ", length(color[[i1]]))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

*# end last check*

*# conversion of geom\_hline and geom\_vline*

**for(i1 in 1:length(data1)){**

**if(geom[[i1]] == "geom\_hline" | geom[[i1]] == "geom\_vline"){**

**final.data.frame <- data.frame()**

**for(i3 in 1:nrow(data1[[i1]])){**

**tempo.data.frame <- rbind(data1[[i1]][i3, ], data1[[i1]][i3, ], stringsAsFactors = TRUE)**

**if(geom[[i1]] == "geom\_hline"){**

**tempo.data.frame[, x[[i1]]] <- x.lim**

**}else if(geom[[i1]] == "geom\_vline"){**

**tempo.data.frame[, y[[i1]]] <- y.lim**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": CODE INCONSISTENCY 5")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

*# 3 lines below inactivated because I put that above*

*# if(is.null(categ[[i1]])){*

*# data1[, "fake\_categ"] <- paste0("Line\_", i3)*

*# }*

**final.data.frame <- rbind(final.data.frame, tempo.data.frame, stringsAsFactors = TRUE)**

**}**

**data1[[i1]] <- final.data.frame**

**geom[[i1]] <- "geom\_line"**

**if(length(color[[i1]])== 1L){**

**color[[i1]] <- rep(color[[i1]], length(unique(data1[[i1]][ , categ[[i1]]])))**

**}else if(length(color[[i1]]) != length(unique(data1[[i1]][ , categ[[i1]]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, " geom\_hline AND geom\_vline CONVERSION TO FIT THE XLIM AND YLIM LIMITS OF THE DATA: ", ifelse(length(color)== 1L, "color", paste0("ELEMENT NUMBER ", i1, " OF color ARGUMENT")), " MUST HAVE THE LENGTH OF LEVELS OF ", ifelse(length(categ)== 1L, "categ", paste0("ELEMENT ", i1, " OF categ ARGUMENT")), " IN ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i1, " OF data1 ARGUMENT")), "\nHERE IT IS COLOR LENGTH ", length(color[[i1]]), " VERSUS CATEG LEVELS LENGTH ", length(unique(data1[[i1]][, categ[[i1]]])))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**}**

**}**

*# end conversion of geom\_hline and geom\_vline*

*# kind of geom\_point (vectorial or raster)*

**scatter.kind <- vector("list", length = length(data1))** *# list of same length as data1, that will be used to use either ggplot2::geom\_point() (vectorial dot layer) or fun\_gg\_point\_rast() (raster dot layer)*

**fix.ratio <- FALSE**

**if(is.null(raster.threshold)){**

**if(raster == TRUE){**

**scatter.kind[] <- "fun\_gg\_point\_rast"** *# not important to fill everything: will be only used when geom == "geom\_point"*

**fix.ratio <- TRUE**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") RASTER PLOT GENERATED -> ASPECT RATIO OF THE PLOT REGION SET BY THE raster.ratio ARGUMENT (", fun\_round(raster.ratio, 2), ") TO AVOID A BUG OF ELLIPSOID DOT DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**scatter.kind[] <- "ggplot2::geom\_point"**

**}**

**}else{**

**for(i2 in 1:length(data1)){**

**if(geom[[i2]] == "geom\_point"){**

**if(nrow(data1[[i2]]) <= raster.threshold){**

**scatter.kind[[i2]] <- "ggplot2::geom\_point"**

**}else{**

**scatter.kind[[i2]] <- "fun\_gg\_point\_rast"**

**fix.ratio <- TRUE**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") ", ifelse(length(data1)== 1L, "data1 ARGUMENT", paste0("DATA FRAME NUMBER ", i2, " OF data1 ARGUMENT")), " LAYER AS RASTER (NOT VECTORIAL)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**}**

**if(any(unlist(scatter.kind) == "fun\_gg\_point\_rast")){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") RASTER PLOT GENERATED -> ASPECT RATIO OF THE PLOT REGION SET BY THE raster.ratio ARGUMENT (", fun\_round(raster.ratio, 2), ") TO AVOID A BUG OF ELLIPSOID DOT DRAWING")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

*# end kind of geom\_point (vectorial or raster)*

*# no need loop part*

**coord.names <- NULL**

**tempo.gg.name <- "gg.indiv.plot."**

**tempo.gg.count <- 0**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste0("ggplot2::ggplot()", if(is.null(add)){""}else{add}))))** *# add added here to have the facets*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::xlab(if(is.null(x.lab)){x[[1]]}else{x.lab}))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ylab(if(is.null(y.lab)){y[[1]]}else{y.lab}))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggtitle(title))**

*# text angle management*

**x.tempo.just <- fun\_gg\_just(angle = x.text.angle, pos = "bottom", kind = "axis")**

**y.tempo.just <- fun\_gg\_just(angle = y.text.angle, pos = "left", kind = "axis")**

*# end text angle management*

**add.check <- TRUE**

**if( ! is.null(add)){** *# if add is NULL, then = 0*

**if(grepl(pattern = "ggplot2::theme", add) == TRUE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") \"ggplot2::theme\" STRING DETECTED IN THE add ARGUMENT\n-> INTERNAL GGPLOT2 THEME FUNCTIONS theme() AND theme\_classic() HAVE BEEN INACTIVATED, TO BE USED BY THE USER\n-> article ARGUMENT WILL BE IGNORED\nIT IS RECOMMENDED TO USE \"+ theme(aspect.ratio = raster.ratio)\" IF RASTER MODE IS ACTIVATED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**add.check <- FALSE**

**}**

**}**

**if(add.check == TRUE & article == TRUE){**

*# WARNING: not possible to add several times theme(). NO message but the last one overwrites the others*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::theme\_classic(base\_size = text.size))**

**if(grid == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**legend.key = ggplot2::element\_rect(color = "white", size = 1.5), # size of the frame of the legend**

**line = ggplot2::element\_line(size = 0.5),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),** *# draw lines for the y axis*

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),** *# draw lines for the x axis*

**panel.grid.major.x = ggplot2::element\_line(colour = "grey85", size = 0.75),**

**panel.grid.minor.x = ggplot2::element\_line(colour = "grey90", size = 0.25),**

**panel.grid.major.y = ggplot2::element\_line(colour = "grey85", size = 0.75),**

**panel.grid.minor.y = ggplot2::element\_line(colour = "grey90", size = 0.25),**

**axis.text.x = ggplot2::element\_text(angle = x.tempo.just$angle, hjust = x.tempo.just$hjust, vjust = x.tempo.just$vjust),**

**axis.text.y = ggplot2::element\_text(angle = y.tempo.just$angle, hjust = y.tempo.just$hjust, vjust = y.tempo.just$vjust),**

**aspect.ratio = if(fix.ratio == TRUE){raster.ratio}else{NULL} # for raster**

**))**

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**legend.key = ggplot2::element\_rect(color = "white", size = 1.5), # size of the frame of the legend**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),**

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),**

**axis.text.x = ggplot2::element\_text(angle = x.tempo.just$angle, hjust = x.tempo.just$hjust, vjust = x.tempo.just$vjust),**

**axis.text.y = ggplot2::element\_text(angle = y.tempo.just$angle, hjust = y.tempo.just$hjust, vjust = y.tempo.just$vjust),**

**aspect.ratio = if(fix.ratio == TRUE){raster.ratio}else{NULL} # for raster**

**))**

**}**

**}else if(add.check == TRUE & article == FALSE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**legend.key = ggplot2::element\_rect(color = "white", size = 1.5), # size of the frame of the legend**

**panel.background = ggplot2::element\_rect(fill = "grey95"),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),**

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),**

**panel.grid.major.x = ggplot2::element\_line(colour = "grey85", size = 0.75),**

**panel.grid.minor.x = ggplot2::element\_line(colour = "grey90", size = 0.25),**

**panel.grid.major.y = ggplot2::element\_line(colour = "grey85", size = 0.75),**

**panel.grid.minor.y = ggplot2::element\_line(colour = "grey90", size = 0.25),**

**strip.background = ggplot2::element\_rect(fill = "white", colour = "black"),**

**axis.text.x = ggplot2::element\_text(angle = x.tempo.just$angle, hjust = x.tempo.just$hjust, vjust = x.tempo.just$vjust),**

**axis.text.y = ggplot2::element\_text(angle = y.tempo.just$angle, hjust = y.tempo.just$hjust, vjust = y.tempo.just$vjust),**

**aspect.ratio = if(fix.ratio == TRUE){raster.ratio}else{NULL} # for raster**

*# do not work -> legend.position = "none" # to remove the legend completely: https://www.datanovia.com/en/blog/how-to-remove-legend-from-a-ggplot/*

**))**

**}**

*# end no need loop part*

*# loop part*

**point.count <- 0**

**line.count <- 0**

**lg.order <- vector(mode = "list", length = 6)** *# order of the legend*

**lg.order <- lapply(lg.order, as.numeric)** *# order of the legend*

**lg.color <- vector(mode = "list", length = 6)** *# color of the legend*

**lg.dot.shape <- vector(mode = "list", length = 6)** *# etc.*

**lg.dot.size <- vector(mode = "list", length = 6)** *# etc.*

**lg.dot.size <- lapply(lg.dot.size, as.numeric)** *# etc.*

**lg.dot.border.size <- vector(mode = "list", length = 6)** *# etc.*

**lg.dot.border.size <- lapply(lg.dot.border.size, as.numeric)** *# etc.*

**lg.dot.border.color <- vector(mode = "list", length = 6)** *# etc.*

**lg.line.size <- vector(mode = "list", length = 6)** *# etc.*

**lg.line.size <- lapply(lg.line.size, as.numeric)** *# etc.*

**lg.line.type <- vector(mode = "list", length = 6)** *# etc.*

**lg.alpha <- vector(mode = "list", length = 6)** *# etc.*

**lg.alpha <- lapply(lg.alpha, as.numeric)** *# etc.*

**for(i1 in 1:length(data1)){**

**if(geom[[i1]] == "geom\_point"){**

**point.count <- point.count + 1**

**if(point.count== 1L){**

**fin.lg.disp[[1]] <- legend.disp[[point.count + line.count]]**

**lg.order[[1]] <- point.count + line.count**

**lg.color[[1]] <- color[[i1]]** *# if color == NULL -> NULL*

**lg.dot.shape[[1]] <- dot.shape[[i1]]**

**lg.dot.size[[1]] <- dot.size[[i1]]**

**lg.dot.border.size[[1]] <- dot.border.size[[i1]]**

**lg.dot.border.color[[1]] <- dot.border.color[[i1]]** *# if dot.border.color == NULL -> NULL*

**if(plot == TRUE & fin.lg.disp[[1]] == TRUE & dot.shape[[1]] %in% 0:14 & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list())== 0L & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE DOTS (DOT LAYER NUMBER ", point.count, ") IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**lg.alpha[[1]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**}else{**

**lg.alpha[[1]] <- alpha[[i1]]**

**}**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = scatter.kind[[i1]]))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], fill = categ[[i1]]), shape = dot.shape[[i1]], size = dot.size[[i1]], stroke = dot.border.size[[i1]], color = if(dot.shape[[i1]] %in% 21:24 & ! is.null(dot.border.color)){dot.border.color[[i1]]}else{color[[i1]][i5]}, alpha = alpha[[i1]], show.legend = if(i5== 1L){TRUE}else{FALSE}))** *# WARNING: a single color allowed for color argument outside aesthetic, but here a single color for border --> loop could be inactivated but kept for commodity # legend.show option do not remove the legend, only the aesthetic of the legend (dot, line, etc.). Used here to avoid multiple layers of legend which corrupt transparency*

**coord.names <- c(coord.names, paste0(geom[[i1]], ".", class.categ[i5]))**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_fill\_manual(name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = as.character(color[[i1]]), breaks = class.categ))** *# values are the values of fill, breaks reorder the classes according to class.categ in the legend, order argument of guide\_legend determines the order of the different aesthetics in the legend (not order of classes). See guide\_legend settings of scale\_...\_manual below*

**}**

**if(point.count== 2L){**

**fin.lg.disp[[2]] <- legend.disp[[point.count + line.count]]**

**lg.order[[2]] <- point.count + line.count**

**lg.color[[2]] <- color[[i1]]** *# if color == NULL -> NULL*

**lg.dot.shape[[2]] <- dot.shape[[i1]]**

**lg.dot.size[[2]] <- dot.size[[i1]]**

**lg.dot.border.size[[2]] <- dot.border.size[[i1]]**

**lg.dot.border.color[[2]] <- dot.border.color[[i1]]** *# if dot.border.color == NULL -> NULL*

**if(plot == TRUE & fin.lg.disp[[2]] == TRUE & dot.shape[[2]] %in% 0:14 & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list())== 0L & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE DOTS (DOT LAYER NUMBER ", point.count, ") IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**lg.alpha[[2]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**}else{**

**lg.alpha[[2]] <- alpha[[i1]]**

**}**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = scatter.kind[[i1]]))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], shape = categ[[i1]]), size = dot.size[[i1]], stroke = dot.border.size[[i1]], fill = color[[i1]][i5], color = if(dot.shape[[i1]] %in% 21:24 & ! is.null(dot.border.color)){dot.border.color[[i1]]}else{color[[i1]][i5]}, alpha = alpha[[i1]], show.legend = FALSE))** *# WARNING: a single color allowed for fill argument outside aesthetic, hence the loop # legend.show option do not remove the legend, only the aesthetic of the legend (dot, line, etc.). Used here to avoid multiple layers of legend which corrupt transparency*

**coord.names <- c(coord.names, paste0(geom[[i1]], ".", class.categ[i5]))**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_shape\_manual(name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(dot.shape[[i1]], length(color[[i1]])), breaks = class.categ))** *# values are the values of shape, breaks reorder the classes according to class.categ in the legend. See guide\_legend settings of scale\_...\_manual below*

**}**

**if(point.count== 3L){**

**fin.lg.disp[[3]] <- legend.disp[[point.count + line.count]]**

**lg.order[[3]] <- point.count + line.count**

**lg.color[[3]] <- color[[i1]]** *# if color == NULL -> NULL*

**lg.dot.shape[[3]] <- dot.shape[[i1]]**

**lg.dot.size[[3]] <- dot.size[[i1]]**

**lg.dot.border.size[[3]] <- dot.border.size[[i1]]**

**lg.dot.border.color[[3]] <- dot.border.color[[i1]]** *# if dot.border.color == NULL -> NULL*

**if(plot == TRUE & fin.lg.disp[[3]] == TRUE & dot.shape[[3]] %in% 0:14 & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list())== 0L & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE DOTS (DOT LAYER NUMBER ", point.count, ") IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**lg.alpha[[3]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**}else{**

**lg.alpha[[3]] <- alpha[[i1]]**

**}**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = scatter.kind[[i1]]))(data = tempo.data.frame, mapping = ggplot2::aes\_string(x = x[[i1]], y = y[[i1]], stroke = categ[[i1]]), shape = dot.shape[[i1]], size = dot.size[[i1]], fill = color[[i1]][i5], stroke = dot.border.size[[i1]], color = if(dot.shape[[i1]] %in% 21:24 & ! is.null(dot.border.color)){dot.border.color[[i1]]}else{color[[i1]][i5]}, alpha = alpha[[i1]], show.legend = FALSE))** *# WARNING: a single color allowed for color argument outside aesthetic, hence the loop # legend.show option do not remove the legend, only the aesthetic of the legend (dot, line, etc.). Used here to avoid multiple layers of legend which corrupt transparency*

**coord.names <- c(coord.names, paste0(geom[[i1]], ".", class.categ[i5]))**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "stroke", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(dot.border.size[[i1]], length(color[[i1]])), breaks = class.categ))** *# values are the values of stroke, breaks reorder the classes according to class.categ in the legend. See guide\_legend settings of scale\_...\_manual below*

**}**

**}else{**

**line.count <- line.count + 1**

**if(line.count== 1L){**

**fin.lg.disp[[4]] <- legend.disp[[point.count + line.count]]**

**lg.order[[4]] <- point.count + line.count**

**lg.color[[4]] <- color[[i1]]** *# if color == NULL -> NULL*

**lg.line.size[[4]] <- line.size[[i1]]**

**lg.line.type[[4]] <- line.type[[i1]]**

**if(plot == TRUE & fin.lg.disp[[4]] == TRUE & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list())== 0L & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE LINES (LINE LAYER NUMBER ", line.count, ") IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**lg.alpha[[4]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**}else{**

**lg.alpha[[4]] <- alpha[[i1]]**

**}**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste0("ggplot2::",** *# no CR here te0("ggpl*

**ifelse(geom[[i1]] == 'geom\_stick', 'geom\_segment', geom[[i1]]),** *# geom\_segment because geom\_stick converted to geom\_segment for plotting*

**"(data = tempo.data.frame, mapping = ggplot2::aes(x = ",**

**x[[i1]],**

**ifelse(geom[[i1]] == 'geom\_stick', ", yend = ", ", y = "),**

**y[[i1]],**

**if(geom[[i1]] == 'geom\_stick'){paste0(', xend = ', x[[i1]], ', y = ', ifelse(is.null(geom.stick.base), y.lim[1], geom.stick.base[[i1]]))},**

**", linetype = ",**

**categ[[i1]],**

**"), color = \"",**

**color[[i1]][i5],**

**"\", size = ",**

**line.size[[i1]],**

**ifelse(geom[[i1]] == 'geom\_path', ', lineend = \"round\"', ''),**

**ifelse(geom[[i1]] == 'geom\_step', paste0(', direction = \"', geom.step.dir[[i1]], '\"'), ''),**

**", alpha = ",**

**alpha[[i1]],**

**", show.legend = ",**

**ifelse(i5== 1L, TRUE, FALSE),**

**")"**

**))))** *# WARNING: a single color allowed for color argument outside aesthetic, hence the loop # legend.show option do not remove the legend, only the aesthetic of the legend (dot, line, etc.). Used here to avoid multiple layers of legend which corrupt transparency*

**coord.names <- c(coord.names, paste0(geom[[i1]], ".", class.categ[i5]))**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "linetype", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(line.type[[i1]], length(color[[i1]])), breaks = class.categ))** *# values are the values of linetype. 1 means solid. Regarding the alpha bug, I have tried different things without success: alpha in guide alone, in geom alone, in both, with different values, breaks reorder the classes according to class.categ in the legend*

**}**

**if(line.count== 2L){**

**fin.lg.disp[[5]] <- legend.disp[[point.count + line.count]]**

**lg.order[[5]] <- point.count + line.count**

**lg.color[[5]] <- color[[i1]]** *# if color == NULL -> NULL*

**lg.line.size[[5]] <- line.size[[i1]]**

**lg.line.type[[5]] <- line.type[[i1]]**

**if(plot == TRUE & fin.lg.disp[[5]] == TRUE & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list())== 0L & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE LINES (LINE LAYER NUMBER ", line.count, ") IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**lg.alpha[[5]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**}else{**

**lg.alpha[[5]] <- alpha[[i1]]**

**}**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste0("ggplot2::",** *# no CR here te0("ggpl*

**ifelse(geom[[i1]] == 'geom\_stick', 'geom\_segment', geom[[i1]]),** *# geom\_segment because geom\_stick converted to geom\_segment for plotting*

**"(data = tempo.data.frame, mapping = ggplot2::aes(x = ",**

**x[[i1]],**

**ifelse(geom[[i1]] == 'geom\_stick', ", yend = ", ", y = "),**

**y[[i1]],**

**if(geom[[i1]] == 'geom\_stick'){paste0(', xend = ', x[[i1]], ', y = ', ifelse(is.null(geom.stick.base), y.lim[1], geom.stick.base[[i1]]))},**

**", alpha = ",**

**categ[[i1]],**

**"), color = \"",**

**color[[i1]][i5],**

**"\", size = ",**

**line.size[[i1]],**

**", linetype = ",**

**ifelse(is.numeric(line.type[[i1]]), "", "\""),**

**line.type[[i1]],**

**ifelse(is.numeric(line.type[[i1]]), "", "\""),**

**ifelse(geom[[i1]] == 'geom\_path', ', lineend = \"round\"', ''),**

**ifelse(geom[[i1]] == 'geom\_step', paste0(', direction = \"', geom.step.dir[[i1]], '\"'), ''),**

**", show.legend = FALSE)"**

**))))** *# WARNING: a single color allowed for color argument outside aesthetic, hence the loop # legend.show option do not remove the legend, only the aesthetic of the legend (dot, line, etc.). Used here to avoid multiple layers of legend which corrupt transparency*

**coord.names <- c(coord.names, paste0(geom[[i1]], ".", class.categ[i5]))**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "alpha", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(alpha[[i1]], length(color[[i1]])), breaks = class.categ))** *# values are the values of linetype. 1 means solid. Regarding the alpha bug, I have tried different things without success: alpha in guide alone, in geom alone, in both, with different values, breaks reorder the classes according to class.categ in the legend*

**}**

**if(line.count== 3L){**

**fin.lg.disp[[6]] <- legend.disp[[point.count + line.count]]**

**lg.order[[6]] <- point.count + line.count**

**lg.color[[6]] <- color[[i1]]** *# if color == NULL -> NULL*

**lg.line.size[[6]] <- line.size[[i1]]**

**lg.line.type[[6]] <- line.type[[i1]]**

**if(plot == TRUE & fin.lg.disp[[6]] == TRUE & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list())== 0L & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE LINES (LINE LAYER NUMBER ", line.count, ") IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**lg.alpha[[6]] <- 1** *# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

**}else{**

**lg.alpha[[6]] <- alpha[[i1]]**

**}**

**class.categ <- levels(factor(data1[[i1]][, categ[[i1]]]))**

**for(i5 in 1:length(color[[i1]])){** *# or length(class.categ). It is the same because already checked that lengths are the same*

**tempo.data.frame <- data1[[i1]][data1[[i1]][, categ[[i1]]] == class.categ[i5], ]**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste0("ggplot2::",** *# no CR here te0("ggpl*

**ifelse(geom[[i1]] == 'geom\_stick', 'geom\_segment', geom[[i1]]),** *# geom\_segment because geom\_stick converted to geom\_segment for plotting*

**"(data = tempo.data.frame, mapping = ggplot2::aes(x = ",**

**x[[i1]],**

**ifelse(geom[[i1]] == 'geom\_stick', ", yend = ", ", y = "),**

**y[[i1]],**

**if(geom[[i1]] == 'geom\_stick'){paste0(', xend = ', x[[i1]], ', y = ', ifelse(is.null(geom.stick.base), y.lim[1], geom.stick.base[[i1]]))},**

**", size = ",**

**categ[[i1]],**

**"), color = \"",**

**color[[i1]][i5],**

**"\", linetype = ",**

**ifelse(is.numeric(line.type[[i1]]), "", "\""),**

**line.type[[i1]],**

**ifelse(is.numeric(line.type[[i1]]), "", "\""),**

**ifelse(geom[[i1]] == 'geom\_path', ', lineend = \"round\"', ''),**

**ifelse(geom[[i1]] == 'geom\_step', paste0(', direction = \"', geom.step.dir[[i1]], '\"'), ''),**

**", alpha = ",**

**alpha[[i1]],**

**", show.legend = FALSE)"**

**))))** *# WARNING: a single color allowed for color argument outside aesthetic, hence the loop # legend.show option do not remove the legend, only the aesthetic of the legend (dot, line, etc.). Used here to avoid multiple layers of legend which corrupt transparency*

**coord.names <- c(coord.names, paste0(geom[[i1]], ".", class.categ[i5]))**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "size", name = if(is.null(legend.name)){NULL}else{legend.name[[i1]]}, values = rep(line.size[[i1]], length(color[[i1]])), breaks = class.categ))** *# values are the values of linetype. 1 means solid. Regarding the alpha bug, I have tried different things without success: alpha in guide alone, in geom alone, in both, breaks reorder the classes according to class.categ in the legend*

**}**

**}**

**}**

*# end loop part*

*# legend display*

**tempo.legend.final <- 'ggplot2::guides(**

**fill = if(fin.lg.disp[[1]] == TRUE){**

**ggplot2::guide\_legend(**

**order = lg.order[[1]],**

**override.aes = list(**

**fill = lg.color[[1]],**

**colour = if(lg.dot.shape[[1]] %in% 21:24 & ! is.null(dot.border.color)){lg.dot.border.color[[1]]}else{lg.color[[1]]},** *# lg.dot.shape[[1]] %in% 21:24 are the only one that can be filled*

**shape = lg.dot.shape[[1]],**

**size = lg.dot.size[[1]],**

**stroke = lg.dot.border.size[[1]],**

**alpha = lg.alpha[[1]],**

**linetype = 0**

**)**

**)**

**}else{**

**"none"**

**},**

**shape = if(fin.lg.disp[[2]] == TRUE){**

**ggplot2::guide\_legend(**

**order = lg.order[[2]],**

**override.aes = list(**

**fill = lg.color[[2]],**

**colour = if(lg.dot.shape[[2]] %in% 21:24 & ! is.null(dot.border.color)){lg.dot.border.color[[2]]}else{lg.color[[2]]},** *# lg.dot.shape[[2]] %in% 21:24 are the only one that can be filled*

**shape = lg.dot.shape[[2]],**

**size = lg.dot.size[[2]],**

**stroke = lg.dot.border.size[[2]],**

**alpha = lg.alpha[[2]],**

**linetype = 0**

**)**

**)**

**}else{**

**"none"**

**},**

**stroke = if(fin.lg.disp[[3]] == TRUE){**

**ggplot2::guide\_legend(**

**order = lg.order[[3]],**

**override.aes = list(**

**fill = lg.color[[3]],**

**colour = if(lg.dot.shape[[3]] %in% 21:24 & ! is.null(dot.border.color)){lg.dot.border.color[[3]]}else{lg.color[[3]]},** *# lg.dot.shape[[3]] %in% 21:24 are the only one that can be filled*

**shape = lg.dot.shape[[3]],**

**size = lg.dot.size[[3]],**

**stroke = lg.dot.border.size[[3]],**

**alpha = lg.alpha[[3]],**

**linetype = 0**

**)**

**)**

**}else{**

**"none"**

**},**

**linetype = if(fin.lg.disp[[4]] == TRUE){**

**ggplot2::guide\_legend(**

**order = lg.order[[4]],**

**override.aes = list(**

**color = lg.color[[4]],**

**size = lg.line.size[[4]],**

**linetype = lg.line.type[[4]],**

**alpha = lg.alpha[[4]],**

**shape = NA**

**)**

**)**

**}else{**

**"none"**

**},**

**alpha = if(fin.lg.disp[[5]] == TRUE){**

**ggplot2::guide\_legend(**

**order = lg.order[[5]],**

**override.aes = list(**

**color = lg.color[[5]],**

**size = lg.line.size[[5]],**

**linetype = lg.line.type[[5]],**

**alpha = lg.alpha[[5]],**

**shape = NA**

**)**

**)**

**}else{**

**"none"**

**},**

**size = if(fin.lg.disp[[6]] == TRUE){**

**ggplot2::guide\_legend(**

**order = lg.order[[6]],**

**override.aes = list(**

**color = lg.color[[6]],**

**size = lg.line.size[[6]],**

**linetype = lg.line.type[[6]],**

**alpha = lg.alpha[[6]],**

**shape = NA**

**)**

**)**

**}else{**

**"none"**

**}**

**)'** *# clip = "off" to have secondary ticks outside plot region does not work*

**if( ! is.null(legend.width)){**

**if(any(unlist(legend.disp))){ # means some TRUE**

**tempo.graph.info <- suppressMessages(ggplot2::ggplot\_build(eval(parse(text = paste0(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), ' + ', tempo.legend.final)))))** *# will be recovered later again, when ylim will be considered*

**legend.final <- fun\_gg\_get\_legend(ggplot\_built = tempo.graph.info, fun.name = function.name) # get legend**

**fin.lg.disp[] <- FALSE** *# remove all the legends. Must be done even if fin.lg.disp is not appearing in the code thenafter. Otherwise twice the legend*

**if(is.null(legend.final) & plot == TRUE){** *# even if any(unlist(legend.disp)) is TRUE*

**legend.final <- fun\_gg\_empty\_graph() # empty graph instead of legend**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") LEGEND REQUESTED (NON-NULL categ ARGUMENT OR legend.show ARGUMENT SET TO TRUE)\nBUT IT SEEMS THAT THE PLOT HAS NO LEGEND -> EMPTY LEGEND SPACE CREATED BECAUSE OF THE NON-NULL legend.width ARGUMENT\n")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else if(plot == TRUE){ # means all FALSE**

**legend.final <- ggplot2::ggplot()+ggplot2::theme\_void() # empty graph instead of legend**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") LEGEND REQUESTED (NON-NULL categ ARGUMENT OR legend.show ARGUMENT SET TO TRUE)\nBUT IT SEEMS THAT THE PLOT HAS NO LEGEND -> EMPTY LEGEND SPACE CREATED BECAUSE OF THE NON-NULL legend.width ARGUMENT\n")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**if( ! any(unlist(legend.disp))){**

**fin.lg.disp[] <- FALSE** *# remove all the legends. Must be done even if fin.lg.disp is not appearing in the code thenafter. Otherwise twice the legend*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = tempo.legend.final)))**

*# end legend display*

*# scale management*

**tempo.coord <- suppressMessages(ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ", ' + ggplot2::scale\_x\_continuous(expand = c(0, 0), limits = sort(x.lim), oob = scales::rescale\_none) + ggplot2::scale\_y\_continuous(expand = c(0, 0), limits = sort(y.lim), oob = scales::rescale\_none)'))))$layout$panel\_params[[1]])** *# here I do not need the x-axis and y-axis orientation, I just need the number of main ticks*

*# x.second.tick.positions # coordinates of secondary ticks (only if x.second.tick.nb argument is non-null or if x.log argument is different from "no")*

**if(x.log != "no"){** *# integer main ticks for log2 and log10*

**tempo.scale <- (as.integer(min(x.lim, na.rm = TRUE)) - 1):(as.integer(max(x.lim, na.rm = TRUE)) + 1)**

**}else{**

**tempo <- if(is.null(attributes(tempo.coord$x$breaks))){tempo.coord$x$breaks}else{unlist(attributes(tempo.coord$x$breaks))}**

**if(all(is.na(tempo))){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nONLY NA IN tempo.coord$x$breaks")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(length(unique(x.lim)) <= 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nIT SEEMS THAT X-AXIS VALUES HAVE A NULL RANGE: ", paste(x.lim, collapse = " "), "\nPLEASE, USE THE x.lim ARGUMENT WITH 2 DIFFERENT VALUES TO SOLVE THIS")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**tempo.scale <- fun\_scale(lim = x.lim, n = ifelse(is.null(x.tick.nb), length(tempo[ ! is.na(tempo)]), x.tick.nb))** *# in ggplot 3.3.0, tempo.coord$x.major\_source replaced by tempo.coord$x$breaks. If fact: n = ifelse(is.null(x.tick.nb), length(tempo[ ! is.na(tempo)]), x.tick.nb)) replaced by n = ifelse(is.null(x.tick.nb), 4, x.tick.nb))*

**}**

**}**

**x.second.tick.values <- NULL**

**x.second.tick.pos <- NULL**

**if(x.log != "no"){**

**tempo <- fun\_inter\_ticks(lim = x.lim, log = x.log)**

**x.second.tick.values <- tempo$values**

**x.second.tick.pos <- tempo$coordinates**

*# if(vertical == TRUE){ # do not remove in case the bug is fixed*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(**

**geom = "segment",**

**x = x.second.tick.pos,**

**xend = x.second.tick.pos,**

**y = if(diff(y.lim) > 0){tempo.coord$y.range[1]}else{tempo.coord$y.range[2]},**

**yend = if(diff(y.lim) > 0){tempo.coord$y.range[1] + abs(diff(tempo.coord$y.range)) / 80}else{tempo.coord$y.range[2] - abs(diff(tempo.coord$y.range)) / 80}**

**))**

*# }else{ # not working because of the ggplot2 bug*

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", y = x.second.tick.pos, yend = x.second.tick.pos, x = tempo.coord$x.range[1], xend = tempo.coord$x.range[1] + diff(tempo.coord$x.range) / 80))*

*# }*

**coord.names <- c(coord.names, "x.second.tick.positions")**

**}else if(( ! is.null(x.second.tick.nb)) & x.log == "no"){**

*# if(x.second.tick.nb > 0){ #inactivated because already checked before*

**if(length(tempo.scale) < 2){**

**tempo.cat1 <- c("x.tick.nb", "x.second.tick.nb")**

**tempo.cat2 <- sapply(list(x.tick.nb, x.second.tick.nb), FUN = paste0, collapse = " ")**

**tempo.sep <- sapply(mapply(" ", max(nchar(tempo.cat1)) - nchar(tempo.cat1) + 3, FUN = rep, SIMPLIFY = FALSE), FUN = paste0, collapse = "")**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nTHE NUMBER OF GENERATED TICKS FOR THE X-AXIS IS NOT CORRECT: ", length(tempo.scale), "\nUSING THESE ARGUMENT SETTINGS (NO DISPLAY MEANS NULL VALUE):\n", paste0(tempo.cat1, tempo.sep, tempo.cat2, collapse = "\n"), "\nPLEASE, TEST OTHER VALUES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**tempo <- fun\_inter\_ticks(lim = x.lim, log = x.log, breaks = tempo.scale, n = x.second.tick.nb)**

**}**

**x.second.tick.values <- tempo$values**

**x.second.tick.pos <- tempo$coordinates**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(**

**geom = "segment",**

**x = x.second.tick.pos,**

**xend = x.second.tick.pos,**

**y = if(diff(y.lim) > 0){tempo.coord$y.range[1]}else{tempo.coord$y.range[2]},**

**yend = if(diff(y.lim) > 0){tempo.coord$y.range[1] + abs(diff(tempo.coord$y.range)) / 80}else{tempo.coord$y.range[2] - abs(diff(tempo.coord$y.range)) / 80}**

**))**

**coord.names <- c(coord.names, "x.second.tick.positions")**

**}**

*# for the ggplot2 bug with x.log, this does not work: eval(parse(text = ifelse(vertical == FALSE & x.log == "log10", "ggplot2::scale\_x\_continuous", "ggplot2::scale\_x\_continuous")))*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_x\_continuous(**

**breaks = tempo.scale,**

**minor\_breaks = x.second.tick.pos,**

**labels = if(x.log == "log10"){scales::trans\_format("identity", scales::math\_format(10^.x))}else if(x.log == "log2"){scales::trans\_format("identity", scales::math\_format(2^.x))}else if(x.log == "no"){ggplot2::waiver()}else{tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 10") ; stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)},**

**expand = c(0, 0),** *# remove space after after axis limits*

**limits = sort(x.lim),** *# NA indicate that limits must correspond to data limits but xlim() already used*

**oob = scales::rescale\_none,**

**trans = ifelse(diff(x.lim) < 0, "reverse", "identity")** *# equivalent to ggplot2::scale\_x\_reverse() but create the problem of x-axis label disappearance with x.lim decreasing. Thus, do not use. Use xlim() below and after this*

**))**

*# end x.second.tick.positions*

*# y.second.tick.positions # coordinates of secondary ticks (only if y.second.tick.nb argument is non-null or if y.log argument is different from "no")*

**if(y.log != "no"){** *# integer main ticks for log2 and log10*

**tempo.scale <- (as.integer(min(y.lim, na.rm = TRUE)) - 1):(as.integer(max(y.lim, na.rm = TRUE)) + 1)**

**}else{**

**tempo <- if(is.null(attributes(tempo.coord$y$breaks))){tempo.coord$y$breaks}else{unlist(attributes(tempo.coord$y$breaks))}**

**if(all(is.na(tempo))){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nONLY NA IN tempo.coord$y$breaks")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}**

**if(length(unique(y.lim)) <= 1){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nIT SEEMS THAT Y-AXIS VALUES HAVE A NULL RANGE: ", paste(y.lim, collapse = " "), "\nPLEASE, USE THE y.lim ARGUMENT WITH 2 DIFFERENT VALUES TO SOLVE THIS")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**tempo.scale <- fun\_scale(lim = y.lim, n = ifelse(is.null(y.tick.nb), length(tempo[ ! is.na(tempo)]), y.tick.nb))** *# in ggplot 3.3.0, tempo.coord$y.major\_source replaced by tempo.coord$y$breaks. If fact: n = ifelse(is.null(y.tick.nb), length(tempo[ ! is.na(tempo)]), y.tick.nb)) replaced by n = ifelse(is.null(y.tick.nb), 4, y.tick.nb))*

**}**

**}**

**y.second.tick.values <- NULL**

**y.second.tick.pos <- NULL**

**if(y.log != "no"){**

**tempo <- fun\_inter\_ticks(lim = y.lim, log = y.log)**

**y.second.tick.values <- tempo$values**

**y.second.tick.pos <- tempo$coordinates**

*# if(vertical == TRUE){ # do not remove in case the bug is fixed*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(**

**geom = "segment",**

**y = y.second.tick.pos,**

**yend = y.second.tick.pos,**

**x = if(diff(x.lim) > 0){tempo.coord$x.range[1]}else{tempo.coord$x.range[2]},**

**xend = if(diff(x.lim) > 0){tempo.coord$x.range[1] + abs(diff(tempo.coord$x.range)) / 80}else{tempo.coord$x.range[2] - abs(diff(tempo.coord$x.range)) / 80}**

**))**

*# }else{ # not working because of the ggplot2 bug*

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", x = y.second.tick.pos, xend = y.second.tick.pos, y = tempo.coord$y.range[1], yend = tempo.coord$y.range[1] + diff(tempo.coord$y.range) / 80))*

*# }*

**coord.names <- c(coord.names, "y.second.tick.positions")**

**}else if(( ! is.null(y.second.tick.nb)) & y.log == "no"){**

*# if(y.second.tick.nb > 0){ #inactivated because already checked before*

**if(length(tempo.scale) < 2){**

**tempo.cat1 <- c("y.tick.nb", "y.second.tick.nb")**

**tempo.cat2 <- sapply(list(y.tick.nb, y.second.tick.nb), FUN = paste0, collapse = " ")**

**tempo.sep <- sapply(mapply(" ", max(nchar(tempo.cat1)) - nchar(tempo.cat1) + 3, FUN = rep, SIMPLIFY = FALSE), FUN = paste0, collapse = "")**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nTHE NUMBER OF GENERATED TICKS FOR THE Y-AXIS IS NOT CORRECT: ", length(tempo.scale), "\nUSING THESE ARGUMENT SETTINGS (NO DISPLAY MEANS NULL VALUE):\n", paste0(tempo.cat1, tempo.sep, tempo.cat2, collapse = "\n"), "\nPLEASE, TEST OTHER VALUES")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)** *# == in stop() to be able to add several messages between ==*

**}else{**

**tempo <- fun\_inter\_ticks(lim = y.lim, log = y.log, breaks = tempo.scale, n = y.second.tick.nb)**

**}**

**y.second.tick.values <- tempo$values**

**y.second.tick.pos <- tempo$coordinates**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(**

**geom = "segment",**

**y = y.second.tick.pos,**

**yend = y.second.tick.pos,**

**x = if(diff(x.lim) > 0){tempo.coord$x.range[1]}else{tempo.coord$x.range[2]},**

**xend = if(diff(x.lim) > 0){tempo.coord$x.range[1] + abs(diff(tempo.coord$x.range)) / 80}else{tempo.coord$x.range[2] - abs(diff(tempo.coord$x.range)) / 80}**

**))**

**coord.names <- c(coord.names, "y.second.tick.positions")**

**}**

*# for the ggplot2 bug with y.log, this does not work: eval(parse(text = ifelse(vertical == FALSE & y.log == "log10", "ggplot2::scale\_x\_continuous", "ggplot2::scale\_y\_continuous")))*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_y\_continuous(**

**breaks = tempo.scale,**

**minor\_breaks = y.second.tick.pos,**

**labels = if(y.log == "log10"){scales::trans\_format("identity", scales::math\_format(10^.x))}else if(y.log == "log2"){scales::trans\_format("identity", scales::math\_format(2^.x))}else if(y.log == "no"){ggplot2::waiver()}else{tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 10") ; stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)},**

**expand = c(0, 0),** *# remove space after axis limits*

**limits = sort(y.lim),** *# NA indicate that limits must correspond to data limits but ylim() already used*

**oob = scales::rescale\_none,**

**trans = ifelse(diff(y.lim) < 0, "reverse", "identity")** *# equivalent to ggplot2::scale\_y\_reverse() but create the problem of y-axis label disappearance with y.lim decreasing. Thus, do not use. Use ylim() below and after this*

**))**

*# end y.second.tick.positions*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(xlim = x.lim, ylim = y.lim))** *# clip = "off" to have secondary ticks outside plot region. The problem is that points out of bounds are also drawn outside the plot region. Thus, I cannot use it # at that stage, x.lim and y.lim not NULL anymore*

*# end scale management*

*# drawing*

**fin.plot <- eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))**

**grob.save <- NULL**

**if(plot == TRUE){**

**if( ! is.null(legend.width)){** *# any(unlist(legend.disp)) == TRUE removed to have empty legend space # not & any(unlist(fin.lg.disp)) == TRUE here because converted to FALSE*

**grob.save <- suppressMessages(suppressWarnings(gridExtra::grid.arrange(fin.plot, legend.final, ncol=2, widths=c(1, legend.width))))**

**}else{**

**grob.save <- suppressMessages(suppressWarnings(print(fin.plot)))**

**}**

**}else{**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") PLOT NOT SHOWN AS REQUESTED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end drawing*

*# output*

**if(warn.print == TRUE & ! is.null(warn)){**

**on.exit(warning(paste0("FROM ", function.name, ":\n\n", warn), call. = FALSE))**

**}**

**on.exit(exp = options(warning.length = ini.warning.length), add = TRUE)**

**if(return == TRUE){**

**output <- suppressMessages(ggplot2::ggplot\_build(fin.plot))**

*# output$data <- output$data[-1] # yes for boxplot but not for scatter # remove the first data because corresponds to the initial empty boxplot*

**if(length(output$data) != length(coord.names)){**

**tempo.cat <- paste0("INTERNAL CODE ERROR IN ", function.name, ": length(output$data) AND length(coord.names) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n", ifelse(is.null(warn), "", paste0("IN ADDITION\nWARNING", ifelse(warn.count > 1, "S", ""), ":\n\n", warn))), call. = FALSE)**

**}else{**

**names(output$data) <- coord.names**

**}**

**if(is.null(unlist(removed.row.nb))){**

**removed.row.nb <- NULL**

**removed.rows <- NULL**

**}else{**

**for(i3 in 1:length(data1)){**

**if( ! is.null(removed.row.nb[[i3]])){**

**removed.row.nb[[i3]] <- sort(removed.row.nb[[i3]])**

**removed.rows[[i3]] <- data1.ini[[i3]][removed.row.nb[[i3]], ]**

**}**

**}**

**}**

**tempo <- output$layout$panel\_params[[1]]**

**output <- list(**

**data = data1,**

**removed.row.nb = removed.row.nb,**

**removed.rows = removed.rows,**

**plot = c(output$data, x.second.tick.values = list(x.second.tick.values), y.second.tick.values = list(y.second.tick.values)),**

**panel = facet.categ,**

**axes = list(**

**x.range = tempo$x.range,**

**x.labels = if(is.null(attributes(tempo$x$breaks))){tempo$x$breaks}else{tempo$x$scale$get\_labels()},** *# is.null(attributes(tempo$x$breaks)) test if it is number (TRUE) or character (FALSE)*

**x.positions = if(is.null(attributes(tempo$x$breaks))){tempo$x$breaks}else{unlist(attributes(tempo$x$breaks))},**

**y.range = tempo$y.range,**

**y.labels = if(is.null(attributes(tempo$y$breaks))){tempo$y$breaks}else{tempo$y$scale$get\_labels()},**

**y.positions = if(is.null(attributes(tempo$y$breaks))){tempo$y$breaks}else{unlist(attributes(tempo$y$breaks))}**

**),**

**warn = paste0("\n", warn, "\n\n"),**

**ggplot = if(return.ggplot == TRUE){fin.plot}else{NULL},** *# fin.plot plots the graph if return == TRUE*

**gtable = if(return.gtable == TRUE){grob.save}else{NULL}** *#*

**)**

**return(output)** *# this plots the graph if return.ggplot is TRUE and if no assignment*

**}**

*# end output*

*# end main code*

**}**