R codes of MCnebula2

# 1 前2500行代码：

# ==========================================================================  
# Generic for base method (get or replace data in slots) of class  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @importFrom methods as formalArgs getGeneric getMethodsForDispatch  
#' @importFrom methods initialize is new selectMethod show showMethods  
## class-melody  
setGeneric("melody",   
 function(x) standardGeneric("melody"))  
setGeneric("melody<-",   
 function(x, value) standardGeneric("melody<-"))  
  
setGeneric("palette\_set",   
 function(x) standardGeneric("palette\_set"))  
setGeneric("palette\_set<-",   
 function(x, value) standardGeneric("palette\_set<-"))  
setGeneric("palette\_gradient",   
 function(x) standardGeneric("palette\_gradient"))  
setGeneric("palette\_gradient<-",   
 function(x, value) standardGeneric("palette\_gradient<-"))  
setGeneric("palette\_stat",   
 function(x) standardGeneric("palette\_stat"))  
setGeneric("palette\_stat<-",   
 function(x, value) standardGeneric("palette\_stat<-"))  
setGeneric("palette\_col",   
 function(x) standardGeneric("palette\_col"))  
setGeneric("palette\_col<-",   
 function(x, value) standardGeneric("palette\_col<-"))  
setGeneric("palette\_label",   
 function(x) standardGeneric("palette\_label"))  
setGeneric("palette\_label<-",   
 function(x, value) standardGeneric("palette\_label<-"))  
  
## class-nebula  
setGeneric("parent\_nebula",   
 function(x) standardGeneric("parent\_nebula"))  
setGeneric("parent\_nebula<-",   
 function(x, value) standardGeneric("parent\_nebula<-"))  
setGeneric("child\_nebulae",   
 function(x) standardGeneric("child\_nebulae"))  
setGeneric("child\_nebulae<-",   
 function(x, value) standardGeneric("child\_nebulae<-"))  
  
setGeneric("igraph",   
 function(x) standardGeneric("igraph"))  
setGeneric("igraph<-",   
 function(x, value) standardGeneric("igraph<-"))  
setGeneric("tbl\_graph",   
 function(x) standardGeneric("tbl\_graph"))  
setGeneric("tbl\_graph<-",   
 function(x, value) standardGeneric("tbl\_graph<-"))  
setGeneric("layout\_ggraph",   
 function(x) standardGeneric("layout\_ggraph"))  
setGeneric("layout\_ggraph<-",   
 function(x, value) standardGeneric("layout\_ggraph<-"))  
setGeneric("grid\_layout",   
 function(x) standardGeneric("grid\_layout"))  
setGeneric("grid\_layout<-",   
 function(x, value) standardGeneric("grid\_layout<-"))  
setGeneric("viewports",   
 function(x) standardGeneric("viewports"))  
setGeneric("viewports<-",   
 function(x, value) standardGeneric("viewports<-"))  
setGeneric("panel\_viewport",   
 function(x) standardGeneric("panel\_viewport"))  
setGeneric("panel\_viewport<-",   
 function(x, value) standardGeneric("panel\_viewport<-"))  
setGeneric("legend\_viewport",   
 function(x) standardGeneric("legend\_viewport"))  
setGeneric("legend\_viewport<-",   
 function(x, value) standardGeneric("legend\_viewport<-"))  
setGeneric("structures\_grob",   
 function(x) standardGeneric("structures\_grob"))  
setGeneric("structures\_grob<-",   
 function(x, value) standardGeneric("structures\_grob<-"))  
setGeneric("nodes\_ggset",   
 function(x) standardGeneric("nodes\_ggset"))  
setGeneric("nodes\_ggset<-",   
 function(x, value) standardGeneric("nodes\_ggset<-"))  
setGeneric("nodes\_grob",   
 function(x) standardGeneric("nodes\_grob"))  
setGeneric("nodes\_grob<-",   
 function(x, value) standardGeneric("nodes\_grob<-"))  
setGeneric("ppcp\_data",   
 function(x) standardGeneric("ppcp\_data"))  
setGeneric("ppcp\_data<-",   
 function(x, value) standardGeneric("ppcp\_data<-"))  
setGeneric("ration\_data",   
 function(x) standardGeneric("ration\_data"))  
setGeneric("ration\_data<-",   
 function(x, value) standardGeneric("ration\_data<-"))  
setGeneric("ggset\_annotate",   
 function(x) standardGeneric("ggset\_annotate"))  
setGeneric("ggset\_annotate<-",   
 function(x, value) standardGeneric("ggset\_annotate<-"))  
  
## class-mcnebula  
setGeneric("creation\_time",   
 function(x) standardGeneric("creation\_time"))  
setGeneric("creation\_time<-",   
 function(x, value) standardGeneric("creation\_time<-"))  
setGeneric("ion\_mode",   
 function(x) standardGeneric("ion\_mode"))  
setGeneric("ion\_mode<-",   
 function(x, value) standardGeneric("ion\_mode<-"))  
setGeneric("match.features\_id",   
 function(x) standardGeneric("match.features\_id"))  
setGeneric("match.candidates\_id",   
 function(x) standardGeneric("match.candidates\_id"))  
setGeneric("specific\_candidate",   
 function(x) standardGeneric("specific\_candidate"))  
setGeneric("classification",   
 function(x) standardGeneric("classification"))  
setGeneric("hierarchy",   
 function(x) standardGeneric("hierarchy"))  
setGeneric("stardust\_classes",   
 function(x) standardGeneric("stardust\_classes"))  
setGeneric("features\_annotation",   
 function(x) standardGeneric("features\_annotation"))  
setGeneric("features\_quantification",   
 function(x) standardGeneric("features\_quantification"))  
setGeneric("features\_quantification<-",   
 function(x, value) standardGeneric("features\_quantification<-"))  
setGeneric("sample\_metadata",   
 function(x) standardGeneric("sample\_metadata"))  
setGeneric("sample\_metadata<-",   
 function(x, value) standardGeneric("sample\_metadata<-"))  
setGeneric("nebula\_index",   
 function(x) standardGeneric("nebula\_index"))  
setGeneric("spectral\_similarity",   
 function(x) standardGeneric("spectral\_similarity"))  
setGeneric("spectral\_similarity<-",   
 function(x, value) standardGeneric("spectral\_similarity<-"))  
  
## class-project  
setGeneric("project\_version",   
 function(x) standardGeneric("project\_version"))  
setGeneric("project\_version<-",   
 function(x, value) standardGeneric("project\_version<-"))  
setGeneric("project\_path",   
 function(x) standardGeneric("project\_path"))  
setGeneric("project\_path<-",   
 function(x, value) standardGeneric("project\_path<-"))  
## class-project\_conformation  
setGeneric("project\_conformation",   
 function(x) standardGeneric("project\_conformation"))  
setGeneric("project\_conformation<-",   
 function(x, value) standardGeneric("project\_conformation<-"))  
  
setGeneric("file\_name",   
 function(x) standardGeneric("file\_name"))  
setGeneric("file\_name<-",   
 function(x, value) standardGeneric("file\_name<-"))  
setGeneric("file\_api",   
 function(x) standardGeneric("file\_api"))  
setGeneric("file\_api<-",   
 function(x, value) standardGeneric("file\_api<-"))  
setGeneric("attribute\_name",   
 function(x) standardGeneric("attribute\_name"))  
setGeneric("attribute\_name<-",   
 function(x, value) standardGeneric("attribute\_name<-"))  
  
## class-project\_metadata  
setGeneric("project\_metadata",   
 function(x) standardGeneric("project\_metadata"))  
setGeneric("project\_metadata<-",   
 function(x, value) standardGeneric("project\_metadata<-"))  
  
setGeneric("metadata",   
 function(x) standardGeneric("metadata"))  
setGeneric("metadata<-",   
 function(x, value) standardGeneric("metadata<-"))  
  
## class-project\_api  
setGeneric("project\_api",   
 function(x) standardGeneric("project\_api"))  
setGeneric("project\_api<-",   
 function(x, value) standardGeneric("project\_api<-"))  
  
setGeneric("methods\_read",   
 function(x) standardGeneric("methods\_read"))  
setGeneric("methods\_read<-",   
 function(x, value) standardGeneric("methods\_read<-"))  
setGeneric("methods\_format",   
 function(x) standardGeneric("methods\_format"))  
setGeneric("methods\_format<-",   
 function(x, value) standardGeneric("methods\_format<-"))  
setGeneric("methods\_match",   
 function(x) standardGeneric("methods\_match"))  
setGeneric("methods\_match<-",   
 function(x, value) standardGeneric("methods\_match<-"))  
  
## class-project\_dataset  
## class-mcn\_dataset  
setGeneric("project\_dataset",   
 function(x) standardGeneric("project\_dataset"))  
setGeneric("project\_dataset<-",   
 function(x, value) standardGeneric("project\_dataset<-"))  
setGeneric("mcn\_dataset",   
 function(x) standardGeneric("mcn\_dataset"))  
setGeneric("mcn\_dataset<-",   
 function(x, value) standardGeneric("mcn\_dataset<-"))  
  
## class-msframe  
setGeneric("msframe",   
 function(x) standardGeneric("msframe"))  
setGeneric("msframe<-",   
 function(x, value) standardGeneric("msframe<-"))  
  
setGeneric("entity",   
 signature = c(msframe = "x"),  
 function(x) standardGeneric("entity"))  
setGeneric("entity<-",   
 signature = c(msframe = "x"),  
 function(x, value) standardGeneric("entity<-"))  
  
## class-command  
setGeneric("command",   
 function(x) standardGeneric("command"))  
setGeneric("command<-",   
 function(x, value) standardGeneric("command<-"))  
  
setGeneric("command\_name",   
 function(x) standardGeneric("command\_name"))  
setGeneric("command\_name<-",   
 function(x, value) standardGeneric("command\_name<-"))  
setGeneric("command\_function",   
 function(x) standardGeneric("command\_function"))  
setGeneric("command\_function<-",   
 function(x, value) standardGeneric("command\_function<-"))  
setGeneric("command\_args",   
 function(x) standardGeneric("command\_args"))  
setGeneric("command\_args<-",   
 function(x, value) standardGeneric("command\_args<-"))  
  
## class-code\_block  
setGeneric("code\_block",   
 function(x) standardGeneric("code\_block"))  
setGeneric("code\_block<-",   
 function(x, value) standardGeneric("code\_block<-"))  
  
setGeneric("codes",   
 function(x) standardGeneric("codes"))  
setGeneric("codes<-",   
 function(x, value) standardGeneric("codes<-"))  
  
## class-ggset  
setGeneric("ggset",   
 function(x) standardGeneric("ggset"))  
setGeneric("ggset<-",   
 function(x, value) standardGeneric("ggset<-"))  
  
setGeneric("layers",   
 function(x) standardGeneric("layers"))  
setGeneric("layers<-",   
 function(x, value) standardGeneric("layers<-"))  
  
## class-section  
setGeneric("section",   
 function(x) standardGeneric("section"))  
setGeneric("section<-",   
 function(x, value) standardGeneric("section<-"))  
setGeneric("heading",   
 function(x) standardGeneric("heading"))  
setGeneric("heading<-",   
 function(x, value) standardGeneric("heading<-"))  
  
setGeneric("level",   
 function(x) standardGeneric("level"))  
setGeneric("level<-",   
 function(x, value) standardGeneric("level<-"))  
setGeneric("paragraph",   
 function(x) standardGeneric("paragraph"))  
setGeneric("paragraph<-",   
 function(x, value) standardGeneric("paragraph<-"))  
  
## class-VIRTUAL  
setGeneric("subscript",   
 function(x) standardGeneric("subscript"))  
setGeneric("subscript<-",   
 function(x, value) standardGeneric("subscript<-"))  
  
setGeneric("dataset",   
 function(x) standardGeneric("dataset"))  
setGeneric("dataset<-",   
 function(x, value) standardGeneric("dataset<-"))  
  
setGeneric("reference",   
 function(x) standardGeneric("reference"))  
setGeneric("reference<-",   
 function(x, value) standardGeneric("reference<-"))  
  
setGeneric("backtrack",   
 function(x) standardGeneric("backtrack"))  
setGeneric("backtrack<-",   
 function(x, value) standardGeneric("backtrack<-"))  
  
setGeneric("export\_name",   
 function(x) standardGeneric("export\_name"))  
setGeneric("export\_name<-",   
 function(x, value) standardGeneric("export\_name<-"))  
setGeneric("export\_path",   
 function(x) standardGeneric("export\_path"))  
setGeneric("export\_path<-",   
 function(x, value) standardGeneric("export\_path<-"))  
  
## class-statistic\_set  
setGeneric("statistic\_set",   
 function(x) standardGeneric("statistic\_set"))  
setGeneric("statistic\_set<-",   
 function(x, value) standardGeneric("statistic\_set<-"))  
  
setGeneric("design\_matrix",   
 function(x) standardGeneric("design\_matrix"))  
setGeneric("design\_matrix<-",   
 function(x, value) standardGeneric("design\_matrix<-"))  
setGeneric("contrast\_matrix",   
 function(x) standardGeneric("contrast\_matrix"))  
setGeneric("contrast\_matrix<-",   
 function(x, value) standardGeneric("contrast\_matrix<-"))  
setGeneric("top\_table",   
 function(x) standardGeneric("top\_table"))  
setGeneric("top\_table<-",   
 function(x, value) standardGeneric("top\_table<-"))  
  
## class-report  
setGeneric("yaml",   
 function(x) standardGeneric("yaml"))  
setGeneric("yaml<-",   
 function(x, value) standardGeneric("yaml<-"))  
# ==========================================================================  
# a class to store function and its name and args  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass command  
#'  
#' @aliases command  
#'  
#' @title Preparation of an instruction to be executed  
#'  
#' @description Packing the funciton and the args inside this class object,  
#' so that it can be performed easily at any time.  
#'  
#' @family call\_commands  
#'  
#' @slot command\_name character(1). Describe the command name.  
#' @slot command\_function function.  
#' @slot command\_args the parameters passed to the function.  
#'  
#' @rdname command-class  
#'  
.command <-   
 setClass("command",   
 contains = character(),  
 representation =   
 representation(command\_name = "character",  
 command\_function = "function",  
 command\_args = "list"  
 ),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod show  
#' @aliases show  
#' @rdname command-class  
setMethod("show",   
 signature = c(object = "command"),  
 function(object){  
 cat(command\_name(object), "\n")  
 args <- vapply(command\_args(object), function(v) class(v)[1], "ch")  
 if (length(args) >= 1) {  
 cat(paste0(paste0(rep(" ", 2), collapse = ""),  
 names(args), ": ", args), sep = "\n")  
 } else {  
 cat(paste0(paste0(rep(" ", 2), collapse = ""),  
 "list()"), "\n")  
 }  
 })  
  
#' @exportMethod command\_name  
#' @aliases command\_name  
#' @description \code{command\_name}, \code{command\_name<-}: getter and setter  
#' for the \code{command\_name} slot of the object.  
#' @rdname command-class  
setMethod("command\_name",   
 signature = c(x = "command"),  
 function(x){ x@command\_name })  
  
#' @exportMethod command\_name<-  
#' @aliases command\_name<-  
#' @param value The value for the slot.  
#' @rdname command-class  
#'  
#' @examples  
#' \dontrun{  
#' ## example 1  
#' com <- new\_command(plot, x = 1:10)  
#' com  
#' call\_command(com)  
#'   
#' ## example 2  
#' com <- new\_command(data.frame, x = 1:10, y = 1:10, z = 1:10)  
#' call\_command(com)  
#'   
#' ## example 3  
#' data <- data.frame(x = 1:10, y = 1:10)  
#' com1 <- new\_command(ggplot, data)  
#' com2 <- new\_command(geom\_point, aes(x = x, y = y))  
#' call\_command(com1) + call\_command(com2)  
#'   
#' ## slots  
#' command\_name(com)  
#' command\_args(com)  
#' command\_function(com)  
#' }  
setReplaceMethod("command\_name",   
 signature = c(x = "command"),  
 function(x, value){  
 initialize(x, command\_name = value)  
 })  
  
#' @exportMethod command\_function  
#' @aliases command\_function  
#' @description \code{command\_function}, \code{command\_function<-}: getter and setter  
#' for the \code{command\_function} slot of the object.  
#' @rdname command-class  
setMethod("command\_function",   
 signature = c(x = "command"),  
 function(x){ x@command\_function })  
  
#' @exportMethod command\_function<-  
#' @aliases command\_function<-  
#' @param value The value for the slot.  
#' @rdname command-class  
setReplaceMethod("command\_function",   
 signature = c(x = "command"),  
 function(x, value){  
 initialize(x, command\_function = value)  
 })  
  
#' @exportMethod command\_args  
#' @aliases command\_args  
#' @description \code{command\_args}, \code{command\_args<-}: getter and setter  
#' for the \code{command\_args} slot of the object.  
#' @rdname command-class  
setMethod("command\_args",   
 signature = c(x = "command"),  
 function(x){ x@command\_args })  
  
#' @exportMethod command\_args<-  
#' @aliases command\_args<-  
#' @param value The value for the slot.  
#' @rdname command-class  
setReplaceMethod("command\_args",   
 signature = c(x = "command"),  
 function(x, value){  
 initialize(x, command\_args = value)  
 })  
  
  
#' @exportMethod new\_command  
#' @aliases new\_command  
#' @description \code{new\_command}: create an object of [command-class].  
#' @param fun function.  
#' @param ... parameters (with names or without names) passed to the function.  
#' @param name character(1). Name to slot \code{command\_name}.  
#' @rdname command-class  
setMethod("new\_command",   
 signature = c(fun = "function",  
 name = "character"),  
 function(fun, ..., name){  
 args <- list(...)  
 if (length(args) != 0) {  
 args\_name <- formalArgs(fun)  
 if (is.null(names(args))) {  
 names(args) <- args\_name[1:length(args)]  
 } else {  
 args\_name <- args\_name[!args\_name %in% names(args)]  
 no\_name\_arg <- which(names(args) == "")  
 names(args)[no\_name\_arg] <- args\_name[1:length(no\_name\_arg)]  
 }  
 }  
 new("command", command\_name = name, command\_function = fun,  
 command\_args = args)  
 })  
  
#' @importFrom rlang as\_label  
#' @exportMethod new\_command  
#' @aliases new\_command  
#' @rdname command-class  
setMethod("new\_command",   
 signature = setMissing("new\_command",  
 fun = "function"),  
 function(fun, ...){  
 name <- rlang::as\_label(substitute(fun))  
 if (length(name) != 1) {  
 name <- paste0(name[2], name[1], name[3])  
 }  
 new\_command(fun, ..., name = name)  
 })  
  
#' @exportMethod call\_command  
#' @aliases call\_command  
#' @description \code{call\_command}: Execute the function (slot \code{command\_function})  
#' with the parameters (slot \code{command\_args}).  
#' @family call\_commands  
#' @rdname command-class  
setMethod("call\_command",   
 signature = c(x = "command"),  
 function(x){  
 do.call(command\_function(x), command\_args(x))  
 })  
# ==========================================================================  
# a class to store a series of 'command' for consisting of a plot of 'ggplot'  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass ggset  
#'  
#' @aliases ggset  
#'  
#' @title Management for 'ggplot' visualzation  
#'  
#' @description  
#' Let each packed "ggplot2" function (packed as [command-class] object)  
#' into layers in sequence, allowing post modifications programmatically  
#' and visualizing as "ggplot2" plot at any time.  
#'  
#' @family layerSets  
#'  
#' @slot layers list with names. Each element of list must be a [command-class] object  
#' packed 'ggplot2' function and its args.  
#'  
#' @rdname ggset-class  
#' @order 1  
#'  
.ggset <-   
 setClass("ggset",   
 contains = c("layerSet"),  
 representation = representation(),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# validity  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
setValidity("ggset",   
 function(object){  
 recepts <- c("command")  
 tip <- paste0("'layer' in 'ggset' must be: ",  
 paste0("'", recepts, "'", collapse = ", "))  
 validate\_class\_in\_list(layers(object), recepts, tip)  
 })  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @importFrom crayon silver  
#' @importFrom crayon yellow  
#' @exportMethod show\_layers  
#' @aliases show\_layers  
#' @description \code{show\_layers}: show functions and parameters in layers  
#' with a pretty and readable form.  
#' @rdname ggset-class  
setMethod("show\_layers",   
 signature = c(x = "ggset"),  
 function(x){  
 layers <- layers(x)  
 cat(crayon::silver("layers of", length(layers), "\n"))  
 mapply(layers, 1:length(layers),  
 FUN = function(com, seq){  
 cat(crayon::silver(" +++ layer", seq, "+++\n"))  
 cat(" ", crayon::yellow(command\_name(com)), "\n",  
 rep(" ", 4), "Args:\n", sep = "")  
 args <- vapply(command\_args(com), function(v) class(v)[1], "ch")  
 if (length(args) >= 1) {  
 cat(paste0(paste0(rep(" ", 6), collapse = ""),  
 names(args), ": ", args), sep = "\n")  
 } else {  
 cat(paste0(paste0(rep(" ", 6), collapse = ""),  
 "list()"), "\n")  
 }  
 cat("\n")  
 })  
 cat("\n")  
 })  
  
#' @exportMethod new\_ggset  
#' @aliases new\_ggset  
#' @description \code{new\_ggset}: Simplified creation of [ggset-class] object.  
#' @param ... An arbitrary number of [command-class] object.  
#' @rdname ggset-class  
#'  
#' @examples  
#' \dontrun{  
#' data <- data.frame(x = 1:10, y = 1:10)  
#' layer1 <- new\_command(ggplot, data)  
#' layer2 <- new\_command(geom\_point, aes(x = x, y = y))  
#' layer3 <- new\_command(labs, x = "x label", y = "y label")  
#' layer4 <- new\_command(theme, text = element\_text(family = "Times"))  
#'   
#' ## gather  
#' ggset <- new\_ggset(layer1, layer2, layer3, layer4)  
#' ggset  
#' ## visualize  
#' p <- call\_command(ggset)  
#' p  
#'   
#' ## add layers  
#' layer5 <- new\_command(  
#' geom\_text,  
#' aes(x = x, y = y, label = paste0("label\_", x))  
#' )  
#' layer6 <- new\_command(ggtitle, "this is title")  
#' ggset <- add\_layers(ggset, layer5, layer6)  
#' call\_command(ggset)  
#'   
#' ## delete layers  
#' ggset <- delete\_layers(ggset, 5:6)  
#' call\_command(ggset)  
#'   
#' ## mutate layer  
#' ggset <- mutate\_layer(ggset, "theme",  
#' legend.position = "none",  
#' plot.background = element\_rect(fill = "red")  
#' )  
#' ggset <- mutate\_layer(ggset, "geom\_point",  
#' mapping = aes(x = x, y = y, color = x)  
#' )  
#' call\_command(ggset)  
#' }  
setMethod("new\_ggset",   
 signature = c(... = "ANY"),  
 function(...){  
 args <- list(...)  
 names(args) <- vapply(args, command\_name, "ch")  
 new("ggset", layers = args)  
 })  
  
#' @exportMethod mutate\_layer  
#' @aliases mutate\_layer  
#' @description \code{mutate\_layer}:  
#' Pass new parameters or modify pre-existing parameters to the packed function.  
#' @param x [ggset-class] object  
#' @param layer numeric(1) or character(1). If "character", the name must be unique  
#' in slot \code{layers}.  
#' @param ... parameters passed to the layer.  
#' @rdname ggset-class  
setMethod("mutate\_layer",   
 signature = c(x = "ggset",  
 layer = "numeric"),  
 function(x, layer, ...){  
 args <- list(...)  
 command <- layers(x)[[ layer ]]  
 old <- command\_args(command)  
 if (length(old) > 0) {  
 args <- vecter\_unique\_by\_names(c(args, old))  
 }  
 layers(x)[[ layer ]] <-   
 do.call(new\_command,  
 c(command\_function(command), args,  
 name = command\_name(command)))  
 return(x)  
 })  
  
#' @exportMethod mutate\_layer  
#' @aliases mutate\_layer  
#' @rdname ggset-class  
setMethod("mutate\_layer",   
 signature = c(x = "ggset", layer = "character"),  
 function(x, layer, ...){  
 seq <- which(names(layers(x)) == layer)  
 if (length(seq) == 0) {  
 stop( paste0("'", layer, "' not found") )  
 } else if (length(seq) > 1) {  
 stop(paste0("multiple layers of '", layer, "' were found"))  
 } else {  
 x <- mutate\_layer(x, seq, ...)  
 }  
 return(x)  
 })  
  
#' @exportMethod add\_layers  
#' @aliases add\_layers  
#' @description \code{add\_layers}: add extra [command-class] objects into slot \code{layers}.  
#' @param x object contains slot \code{layers}.  
#' @param ... extra [command-class] objects.  
#' @rdname ggset-class  
setMethod("add\_layers",   
 signature = c(x = "ggset"),  
 function(x, ...){  
 args <- list(...)  
 names(args) <- vapply(args, command\_name, "ch")  
 layers(x) <- c(layers(x), args)  
 return(x)  
 })  
  
#' @exportMethod call\_command  
#' @aliases call\_command  
#' @description \code{call\_command}: plot as 'ggplot' object.  
#' @family call\_commands  
#' @rdname ggset-class  
setMethod("call\_command",   
 signature = c(x = "ggset"),  
 function(x){  
 layers <- layers(x)  
 for (i in 1:length(layers)) {  
 res <- try( call\_command(layers[[i]]), silent = T )  
 if (inherits(res, "try-error")) {  
 stop(paste0("the 'command' named '", command\_name(layers[[i]]),  
 "' (sequence:", i, ") in `layers(x)` caused error."))  
 }  
 if (i == 1) {  
 p <- call\_command(layers[[1]])  
 } else {  
 p <- p + res  
 }  
 }  
 return(p)  
 })  
# ==========================================================================  
# a class to store the filtered dataset from 'project\_dataset'  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass mcn\_dataset  
#'  
#' @aliases mcn\_dataset  
#'  
#' @title Store processed data   
#'  
#' @description  
#' This is a class object used to store filtered data and formated data.  
#' These data would be used for further analysis or visualization.  
#'  
#' @seealso [dataset-class]  
#' @seealso [subscript-class]  
#'  
#' @slot dataset list with names of [subscript-class]. Store preliminary filtered data.  
#' @slot reference list with names of standard names. Store formated data, which is useful  
#' reference for further analysis or visualization.  
#' @slot backtrack list with names. Recovery stations halfway through data processing.  
#'  
#' @rdname mcn\_dataset-class  
#'  
.mcn\_dataset <-   
 setClass("mcn\_dataset",   
 contains = c("dataset", "reference", "backtrack"),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod mcn\_dataset  
#' @aliases mcn\_dataset  
#' @description \code{mcn\_dataset}, \code{mcn\_dataset<-}: getter and setter  
#' for the \code{mcn\_dataset} slot of the object.  
#' @rdname mcn\_dataset-class  
setMethod("mcn\_dataset",   
 signature = c(x = "ANY"),  
 function(x){ x@mcn\_dataset })  
  
#' @exportMethod mcn\_dataset<-  
#' @aliases mcn\_dataset<-  
#' @param value The value for the slot.  
#' @rdname mcn\_dataset-class  
setReplaceMethod("mcn\_dataset",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, mcn\_dataset = value)  
 })  
  
  
#' @exportMethod latest  
#' @aliases latest  
#' @description \code{latest}: get the first data in \code{dataset} slot and  
#' format as "tbl". Equals:  
#' - \code{latest(object)}  
#' - \code{tibble::as\_tibble(entity(dataset(x)[[1]]))}.  
#' @family datasets  
#' @family latests  
#' @rdname mcn\_dataset-class  
setMethod("latest",   
 signature = c(x = "mcn\_dataset"),  
 function(x){  
 tibble::as\_tibble(entity(dataset(x)[[1]]))  
 })  
  
  
#' @exportMethod extract\_mcnset  
#' @aliases extract\_mcnset  
#' @description \code{extract\_mcnset}: For fast extract data in object which containing  
#' \code{mcn\_dataset} slot. Normally not used.  
#' @param subscript See [subscript-class]  
#' @rdname mcn\_dataset-class  
setMethod("extract\_mcnset",   
 signature = c(x = "ANY", subscript = "character"),  
 function(x, subscript){  
 if ( any( subscript == names(dataset(mcn\_dataset(x))) ) )  
 msframe <- dataset(mcn\_dataset(x))[[ subscript ]]  
 else  
 stop("`subscript` not found in `dataset(mcn\_dataset(x))`")  
 lst <- list(msframe)  
 names(lst) <- subscript  
 return(lst)  
 })  
  
  
  
# ==========================================================================  
# MCnebula2 overall object  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @aliases MCnebula2  
#'  
#' @title Overview of MCnebula2  
#'  
#' @description   
#'  
#' MCnebula2 was used for metabonomics data analysis.  
#' It is written in the S4 system of object-oriented programming,  
#' and starts with a "class", namely "mcnebula".  
#' The whole process takes the "mcnebula" as the operating object to obtain visual  
#' results or operating objects.  
#'  
#' Most methods of MCnebula2 are S4 methods and have the characteristics of  
#' parameterized polymorphism, that is, different functions will be used for  
#' processing according to different parameters passed to the same method.  
#'  
#' MCnebula workflow is a complete metabolomics data analysis process,  
#' including initial data preprocessing (data format conversion, feature detection),  
#' compound identification based on MS/MS,  
#' statistical analysis,  
#' compound structure and chemical class focusing,  
#' multi-level data visualization, output report, etc.  
#'  
#' It should be noted that the MCnebula2 R package currently cannot realize  
#' the entire analysis process of MCnebula workflow.  
#' If users want to complete the entire workflow,  
#' other software beyond the R console  
#' (for example, the MSconvert tool of proteowizard is used for data format conversion,  
#' which is a tool widely applicable to metabonomics and proteomics) should be used.  
#' This is a pity, but we will gradually integrate all parts of the workflow into this R package  
#' in the future to achieve one-stop analysis.  
#'  
#' The analysis process in R is integrated into the following methods:  
#'  
#' - [initialize\_mcnebula()]  
#' - [filter\_structure()]  
#' - [create\_reference()]  
#' - [filter\_formula()]  
#' - [create\_stardust\_classes()]  
#' - [create\_features\_annotation()]  
#' - [cross\_filter\_stardust()]  
#' - [create\_nebula\_index()]  
#' - [compute\_spectral\_similarity()]  
#' - [create\_parent\_nebula()]  
#' - [create\_child\_nebulae()]  
#' - [create\_parent\_layout()]  
#' - [create\_child\_layouts()]  
#' - [activate\_nebulae()]  
#' - [visualize()]  
#' - [binary\_comparison()]  
#' - ...  
#'  
#' @details  
#'  
#' \bold{Overall.} We know that the analysis of untargeted LC-MS/MS dataset generally  
#' begin with feature detection.  
#' It detects 'peaks' as features in MS1 (MASS level 1) data.  
#' Each feature may represents a compound, and assigned with MS2 (MASS level 2) spectra.  
#' The MS2 spectra was used to find out the compound identity.  
#' The difficulty lies in annotating these features to discover their compound identity,  
#' mining out meaningful information, so as to serve further biological research.  
#' In addition, the un-targeted LC-MS/MS dataset is general a huge dataset,  
#' which leads to time-consuming analysis of the whole process.  
#' Herein, a classified visualization method, called MCnebula,  
#' was used for addressing these difficulty.  
#'  
#' MCnebula utilizes the state-of-the-art computer prediction technology,  
#' SIRIUS workflow (SIRIUS, ZODIAC, CSI:fingerID, CANOPUS),  
#' for compound formula prediction, structure retrieve and classification prediction  
#' (\url{https://bio.informatik.uni-jena.de/software/sirius/}).  
#' MCnebula integrates an abundance-based classes (ABC) selection algorithm  
#' into features annotation:  
#' depending on the user,  
#' MCnebula focuses chemical classes with more or less features in the dataset  
#' (the abundance of classes), visualizes them, and displays the features they involved;  
#' these classes can be dominant structural classes or sub-structural classes.  
#' With MCnebula, we can switch from untargeted to targeted analysis,  
#' focusing precisely on the compound or chemical class of interest to the researcher.  
#'  
#' \bold{MCnebula2.} The MCnebula2 package itself does not contain any part of   
#' molecular formula prediction, structure prediction and chemical prediction of compounds,  
#' so the accuracy of these parts is not involved.  
#' MCnebula2 performs downstream analysis by extracting the prediction data from SIRIUS project.  
#' The core of MCnebula2 is its chemical filtering algorithm, called ABC selection algorithm.  
#'  
#' \bold{Chemical structure and formula.} To explain the ABC selection algorithm in detail,  
#' we need to start with MS/MS spectral analysis and identification of compounds:  
#' The analysis of MS/MS spectrum is a process of inference and prediction.  
#' For example, we speculate the composition of elements based on the molecular weight of MS1;  
#' combined with the possible fragmentation pattern of MS2 spectrum,  
#' we speculate the potential molecular formula of a compound;  
#' finally, we look for the exact compound from the compound structure database.  
#' Sometimes, this process is full of uncertainty,  
#' because there are too many factors that affect the reliability of MS/MS data  
#' and the correctness of inference.  
#' It can be assumed that there are complex candidates  
#' for the potential chemical molecular formula,  
#' chemical structure and chemical class behind MS/MS spectrum.  
#' Suppose we have these data of candidates now,  
#' MCnebula2 extracted these candidates and obtained the unique  
#' molecular formula and chemical structure for each MS/MS spectrum  
#' based on the highest score of  
#' chemical structure prediction; in this process, as most algorithms do,  
#' we make a choice based on the score,  
#' and only select the result of highest score.  
#'  
#' The chemical formula and structure candidates can obtain by methods:  
#'  
#' - [filter\_formula()]  
#' - [filter\_structure()]  
#'  
#' In order to obtain the best (maybe), corresponding and unique chemical formula  
#' and structure from complex candidates, an important intermediate link:  
#'  
#' - [create\_reference()]  
#'  
#' Above, we talked about chemical molecular formula,  
#' chemical structural formula and chemical classes.  
#' We obtained the unique chemical molecular formula and chemical structure formula  
#' for reference by scoring and ranking.  
#' But for chemical classes, we can't adopt such a simple way to get things done.  
#'   
#' \bold{Chemical classification.} Chemical classification is a complex system.  
#' Here, we only discuss the structure based chemotaxonomy system,  
#' because the MS/MS spectrum is more indicative of the structure of compounds  
#' than biological activity and other information.  
#'  
#' According to the division of the overall structure and local structure of compounds,  
#' we can call the structural characteristics as the dominant structure and substructure.  
#' (\url{https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0174-y}).  
#' Correspondingly, in the chemical classification system,  
#' we can not only classify according to the dominant structure,  
#' but also classify according to the substructure.  
#' The chemical classification based on the dominant structure of compounds is easy to understand,  
#' because we generally define it in this way.  
#' For example, we will classify Taxifolin as "flavones", not "phenols",  
#' although its local structure has a substructure of "phenol".  
#'  
#' We hope to classify a compound by its dominant structure rather than substructure,  
#' because such classify is more concise and contains more information.  
#' However, in the process of MS/MS spectral analysis,  
#' we sometimes can only make chemical classification based on the substructure of compounds,  
#' which may be due to: uncertainty in the process of structural analysis;  
#' it may be an unknown compound; MS/MS spectral fragment information is insufficient.  
#' In this case, it is necessary for us to classify the compounds with the aid of  
#' substructure information, otherwise we have no knowledge of the compounds  
#' for which we cannot obtain dominant structure information.  
#'  
#' Above, we discussed the complex chemical classification  
#' for the substructure and dominant structure of compounds.  
#' We must also be clear about the complexity of another aspect of chemotaxonomy,  
#' i.e., the hierarchy of classification.  
#' This is easy to understand. For example, "Flavones" belongs to its superior, "Flavonoids";  
#' its next higher level, "Phynylpropanoids and polyketides";  
#' the further upward classification is "organic compounds".  
#'  
#' \bold{ABC selection.}  
#' The above section discusses the inferential prediction of individual MS/MS spectrum.  
#' In the un-targeted LC-MS/MS dataset, each feature has a corresponding MS/MS spectrum,  
#' and there are thousands of features in total.  
#' The ABC selection algorithm regards all features as a whole,  
#' examines the number and abundance of features of each chemical classification  
#' (classification at different levels, classification of substructure and dominant structure),  
#' and then selects representative classes  
#' (mainly screening the classes according to the number or abundance range of features)  
#' to serve the subsequent analysis.  
#' The core methods for ABC selection algorithm are:  
#'  
#' - [create\_stardust\_classes()]  
#' - [cross\_filter\_stardust()]  
#' - [create\_nebula\_index()]  
#'  
#' Whether it is all filtered by the algorithm provided by MCnebula2's function  
#' or custom filtered for some chemical classes, we now have a data called 'nebula\_index'.  
#' This data records a number of chemical classes and the 'features' attributed to them.  
#' The subsequent analysis process or visualization will be based on it.  
#' Each chemical class is considered as a 'nebula' and its classified 'features'  
#' are the components of these 'nebulae'. In the visualization, these 'nebulae' will  
#' be visualized as networks. Formally, we call these 'nebulae' formed on the basis  
#' of 'nebula\_index' data as Child-Nebulae. In comparison, when we put all the  
#' 'features' together to form a large network, then this 'nebula' is called Parent-Nebulae.  
#'  
#' @name ABSTRACT-MCnebula2  
NULL  
#> NULL  
  
#' @export mcnebula  
#' @exportClass mcnebula  
#'  
#' @aliases mcnebula  
#'  
#' @title Overall object class of MCnebula2  
#'  
#' @description For analysis of MCnebula2, all data stored in this class object,  
#' all main methods performed with this object.  
#'  
#' @family nebulae  
#'  
#' @slot creation\_time character(1).  
#' @slot ion\_mode character(1).  
#' @slot melody [melody-class] object.  
#' @slot mcn\_dataset [mcn\_dataset-class] object.  
#' @slot statistic\_set [statistic\_set-class] object.  
#' @slot ... Slots inherit from [project-class], [nebula-class], [export-class].  
#'  
#' @rdname mcnebula-class  
#'  
mcnebula <-   
 setClass("mcnebula",   
 contains = c("project", "nebula", "export"),  
 representation =   
 representation(creation\_time = "character",  
 ion\_mode = "character",  
 melody = "melody",  
 mcn\_dataset = "mcn\_dataset",  
 statistic\_set = "statistic\_set"  
 ),  
 prototype = prototype(project\_version = "sirius.v4",  
 project\_path = ".",  
 creation\_time = date(),  
 ion\_mode = "pos")  
 )  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod show  
#' @aliases show  
#' @rdname mcnebula-class  
setMethod("show",   
 signature = c(object = "mcnebula"),  
 function(object){  
 message( "A project of MCnebula2", ": ",  
 format(object.size(object), units = "MB"))  
 })  
  
  
#' @exportMethod latest  
#' @aliases latest  
#' @description \code{latest(x, slot, subscript)}: get the data in slot  
#' (\code{mcn\_dataset(object)} or \code{prject\_dataset(object)})  
#' and format as 'tbl'.  
#' @param x [mcnebula-class] object  
#' @param slot Character. Slot name.  
#' @param subscript numeric or character. The sequence or name for dataset in the 'list'.  
#' @family latests  
#' @family subscripts  
#' @seealso [tibble::as\_tibble()]  
#' @rdname mcnebula-class  
#'  
#' @examples  
#' \dontrun{  
#' test <- mcnebula()  
#' class(test)  
#'   
#' test <- mcn\_5features  
#' ## slots  
#' ion\_mode(test)  
#' project\_version(test)  
#' melody(test)  
#' export\_name(test)  
#' ## ...  
#'   
#' ## 'fast channel'  
#' palette\_label(test)  
#' palette\_stat(test)  
#' sample\_metadata(test)  
#' ## ...  
#' }  
setMethod("latest",   
 signature = c(x = "mcnebula", slot = "character",  
 subscript = "ANY"),  
 function(x, slot, subscript){  
 fun <- match.fun(slot)  
 res <- dataset(fun(x))  
 if (length(res) == 0)  
 return()  
 res <- res[[ subscript ]]  
 if (is.null(res))  
 return()  
 else  
 return(tibble::as\_tibble(entity(res)))  
 })  
  
  
#' @exportMethod latest  
#' @description \code{latest()}: get the default parameters for the method \code{latest}.  
#' @rdname mcnebula-class  
setMethod("latest",   
 signature = setMissing("latest"),  
 function(){  
 list(slot = "mcn\_dataset",  
 subscript = 1)  
 })  
  
#' @exportMethod latest  
#' @description \code{latest(x, ...)}: use the default parameters whatever 'missing'  
#' while performing the method \code{latest}.  
#' @rdname mcnebula-class  
setMethod("latest",   
 signature = c(x = "mcnebula"),  
 function(x, slot, subscript){  
 reCallMethod("latest", .fresh\_param(latest()))  
 })  
  
  
#' @exportMethod creation\_time  
#' @aliases creation\_time  
#' @description \code{creation\_time}, \code{creation\_time<-}: getter and setter  
#' for the \code{creation\_time} slot of the object.  
#' @rdname mcnebula-class  
setMethod("creation\_time",   
 signature = c(x = "mcnebula"),  
 function(x){ x@creation\_time })  
  
#' @exportMethod creation\_time<-  
#' @aliases creation\_time<-  
#' @param value The value for the slot.  
#' @rdname mcnebula-class  
setReplaceMethod("creation\_time",   
 signature = c(x = "mcnebula"),  
 function(x, value){  
 initialize(x, creation\_time = value)  
 })  
  
  
#' @exportMethod ion\_mode  
#' @aliases ion\_mode  
#' @description \code{ion\_mode}, \code{ion\_mode<-}: getter and setter  
#' for the \code{ion\_mode} slot of the object.  
#' @rdname mcnebula-class  
setMethod("ion\_mode",   
 signature = c(x = "mcnebula"),  
 function(x){ x@ion\_mode })  
  
#' @exportMethod ion\_mode<-  
#' @aliases ion\_mode<-  
#' @param value The value for the slot.  
#' @rdname mcnebula-class  
setReplaceMethod("ion\_mode",   
 signature = c(x = "mcnebula"),  
 function(x, value){  
 initialize(x, ion\_mode = value)  
 })  
  
# ==========================================================================  
# get infrustructure object  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod palette\_set  
#' @aliases palette\_set  
#' @description \code{palette\_set}, \code{palette\_gradient}, \code{palette\_stat},  
#' \code{palette\_col}: fast channel to obtain the downstream slot.  
#' For \code{palette\_set}, e.g., getter for the \code{palette\_set} slot in sub-object  
#' of \code{melody} slot of the object. Equals:  
#' - \code{palette\_set(melody(object))}  
#' - \code{palette\_set(object)}.  
#' @rdname mcnebula-class  
setMethod("palette\_set",   
 signature = c(x = "mcnebula"),  
 function(x){  
 palette\_set(melody(x))  
 })  
  
#' @exportMethod palette\_gradient  
#' @aliases palette\_gradient  
#' @rdname mcnebula-class  
setMethod("palette\_gradient",   
 signature = c(x = "mcnebula"),  
 function(x){  
 palette\_gradient(melody(x))  
 })  
  
#' @exportMethod palette\_stat  
#' @aliases palette\_stat  
#' @rdname mcnebula-class  
setMethod("palette\_stat",   
 signature = c(x = "mcnebula"),  
 function(x){  
 palette\_stat(melody(x))  
 })  
  
#' @exportMethod palette\_col  
#' @aliases palette\_col  
#' @rdname mcnebula-class  
setMethod("palette\_col",   
 signature = c(x = "mcnebula"),  
 function(x){  
 palette\_col(melody(x))  
 })  
  
#' @exportMethod palette\_label  
#' @aliases palette\_label  
#' @rdname mcnebula-class  
setMethod("palette\_label",   
 signature = c(x = "mcnebula"),  
 function(x){  
 palette\_label(melody(x))  
 })  
  
  
#' @exportMethod reference  
#' @aliases reference  
#' @description \code{reference}: fast channel to obtain  
#' the downstream slot, getter  
#' for the \code{reference} slot in sub-object  
#' of \code{mcn\_dataset} slot of the object. Equals:  
#' - \code{reference(mcn\_dataset(object))}   
#' - \code{reference(object)}  
#' @rdname mcnebula-class  
setMethod("reference",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(mcn\_dataset(x))  
 })  
  
#' @exportMethod specific\_candidate  
#' @aliases specific\_candidate  
#' @description \code{specific\_candidate}, \code{hierarchy}, \code{stardust\_classes},  
#' \code{nebula\_index}, \code{spectral\_similarity}, \code{features\_annotation},  
#' \code{features\_quantification}, \code{sample\_metadata}:  
#' fast channel to obtain data (mostly 'tbl' or 'data.frame')  
#' inside the downstream slot ('list'). e.g., getter  
#' for the data named \code{specific\_candidate} in  
#' \code{reference} slot (a 'list') in sub-object  
#' of \code{mcn\_dataset} slot of the object. Equals:  
#' - \code{reference(mcn\_dataset(object))$specific\_candidate}  
#' - \code{specific\_candidate(object)}.  
#' @rdname mcnebula-class  
setMethod("specific\_candidate",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "specific\_candidate" ]]  
 })  
  
#' @exportMethod hierarchy  
#' @aliases hierarchy  
#' @rdname mcnebula-class  
setMethod("hierarchy",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "hierarchy" ]]  
 })  
  
#' @exportMethod stardust\_classes  
#' @aliases stardust\_classes  
#' @rdname mcnebula-class  
setMethod("stardust\_classes",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "stardust\_classes" ]]  
 })  
  
#' @exportMethod nebula\_index  
#' @aliases nebula\_index  
#' @rdname mcnebula-class  
setMethod("nebula\_index",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "nebula\_index" ]]  
 })  
  
#' @exportMethod spectral\_similarity  
#' @aliases spectral\_similarity  
#' @rdname mcnebula-class  
setMethod("spectral\_similarity",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "spectral\_similarity" ]]  
 })  
  
#' @exportMethod spectral\_similarity<-  
#' @aliases spectral\_similarity<-  
#' @description \code{spectral\_similarity<-}, \code{features\_quantification<-},  
#' \code{sample\_metadata<-}: fast channel to replace  
#' data (mostly 'tbl' or 'data.frame') inside the downstream slot ('list'). e.g., setter  
#' for the data named \code{spectral\_similarity} in  
#' \code{reference} slot (a 'list') in sub-object  
#' of \code{mcn\_dataset} slot of the object. Similar:  
#' - \code{reference(mcn\_dataset(object))$spectral\_similarity<-}  
#' - \code{spectral\_similarity(object)<-}.  
#'  
#' But the latter not only replace and also validate.  
#' @rdname mcnebula-class  
setReplaceMethod("spectral\_similarity",   
 signature = c(x = "mcnebula"),  
 function(x, value){  
 .check\_columns(value, list(".features\_id1", ".features\_id2",  
 "similarity"),  
 "spectral\_similarity")  
 reference(mcn\_dataset(x))$spectral\_similarity <- value  
 return(x)  
 })  
  
#' @exportMethod features\_annotation  
#' @aliases features\_annotation  
#' @rdname mcnebula-class  
setMethod("features\_annotation",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "features\_annotation" ]]  
 })  
  
#' @exportMethod features\_quantification  
#' @aliases features\_quantification  
#' @rdname mcnebula-class  
setMethod("features\_quantification",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "features\_quantification" ]]  
 })  
  
.features\_quantification <-   
 function(x){  
 data <- features\_quantification(x)  
 .features\_id <- data$.features\_id  
 data$.features\_id <- NULL  
 data <- as.matrix(data)  
 rownames(data) <- .features\_id  
 data  
 }  
  
#' @importFrom dplyr select  
#' @exportMethod features\_quantification<-  
#' @aliases features\_quantification<-  
#' @rdname mcnebula-class  
setReplaceMethod("features\_quantification",   
 signature = c(x = "mcnebula"),  
 function(x, value){  
 .check\_columns(value, list(".features\_id"),  
 "features\_quantification")  
 .check\_type(dplyr::select(value, -.features\_id),  
 "numeric", "features\_quantification")  
 reference(mcn\_dataset(x))$features\_quantification <- value  
 return(x)  
 })  
  
#' @exportMethod sample\_metadata  
#' @aliases sample\_metadata  
#' @rdname mcnebula-class  
setMethod("sample\_metadata",   
 signature = c(x = "mcnebula"),  
 function(x){  
 reference(x)[[ "sample\_metadata" ]]  
 })  
  
#' @exportMethod sample\_metadata<-  
#' @aliases sample\_metadata<-  
#' @rdname mcnebula-class  
setReplaceMethod("sample\_metadata",   
 signature = c(x = "mcnebula"),  
 function(x, value){  
 .check\_data(x, list(features\_quantification =  
 "features\_quantification"), "(x) <-")  
 .check\_columns(value, list("sample", "group"), "sample\_metadata")  
 if (any(!value$sample %in% colnames(features\_quantification(x))))  
 stop(paste0("the name in 'sample' column in 'sample\_metadata' ",  
 "must all involved in 'features\_quantification'"))  
 reference(mcn\_dataset(x))$sample\_metadata <- value  
 return(x)  
 })  
  
#' @exportMethod classification  
#' @aliases classification  
#' @description \code{classification}: fast channel to obtain  
#' data deeply inside the downstream slot ('list'), getter  
#' for the data named \code{".canopus"} in  
#' \code{dataset} slot (a 'list') in sub-object  
#' of \code{project\_dataset} slot of the object. Equals:  
#' - \code{tibble::as\_tibble(entity(dataset(project\_dataset(object))$.canopus))}  
#' - \code{classification(object)}.  
#' @rdname mcnebula-class  
setMethod("classification",   
 signature = c(x = "mcnebula"),  
 function(x){  
 res <- dataset(project\_dataset(x))[[ ".canopus" ]]  
 if (is.null(res))  
 return()  
 else  
 return(dplyr::as\_tibble(entity(res)))  
 })  
# ==========================================================================  
# a class to store hex color set.  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass melody  
#'  
#' @aliases melody  
#'  
#' @title Mutiple color palette in hexadecimal code  
#'  
#' @description  
#' This is a class object store Hex color used for visualization.  
#' In default (use [initialize\_mcnebula()] to initialize the object),  
#' these these Hex color in each palette were get from package \code{ggsci}.  
#' Most of these palette in this package would passed to [ggplot2::scale\_fill\_manual] for  
#' filling color. So, let these Hex color with names may work well to specify target.  
#'  
#' @seealso [ggsci::pal\_simpsons()], [ggsci::pal\_igv()], [ggsci::pal\_ucscgb()],  
#' [ggsci::pal\_d3()]...  
#'  
#' @slot palette\_set character with names or not. Hex color.  
#' @slot palette\_gradient character with names or not. Hex color.  
#' @slot palette\_stat character with names or not. Hex color.  
#' @slot palette\_col character with names or not. Hex color.  
#' @slot palette\_label character with names or not. Hex color.  
#'  
#' @rdname melody-class  
#'  
.melody <-   
 setClass("melody",   
 contains = character(),  
 representation =   
 representation(palette\_set = "character",  
 palette\_gradient = "character",  
 palette\_stat = "character",  
 palette\_col = "character",  
 palette\_label = "character"  
 ),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod show  
#' @aliases show  
#' @rdname melody-class  
setMethod("show",   
 signature = c(object= "melody"),  
 function(object){  
 .show(object)  
 })  
  
  
#' @exportMethod melody  
#' @aliases melody  
#' @description \code{melody}, \code{melody<-}: getter and setter  
#' for the \code{melody} slot of the object.  
#' @rdname melody-class  
setMethod("melody",   
 signature = c(x = "ANY"),  
 function(x){ x@melody })  
  
#' @exportMethod melody<-  
#' @aliases melody<-  
#' @param value The value for the slot.  
#' @rdname melody-class  
setReplaceMethod("melody",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, melody = value)  
 })  
  
  
#' @exportMethod palette\_set  
#' @aliases palette\_set  
#' @description \code{palette\_set}, \code{palette\_set<-}: getter and setter  
#' for the \code{palette\_set} slot of the object.  
#' @rdname melody-class  
setMethod("palette\_set",   
 signature = c(x = "melody"),  
 function(x){ x@palette\_set })  
  
#' @exportMethod palette\_set<-  
#' @aliases palette\_set<-  
#' @param value The value for the slot.  
#' @rdname melody-class  
setReplaceMethod("palette\_set",   
 signature = c(x = "melody"),  
 function(x, value){  
 initialize(x, palette\_set = value)  
 })  
  
  
#' @exportMethod palette\_gradient  
#' @aliases palette\_gradient  
#' @description \code{palette\_gradient}, \code{palette\_gradient<-}: getter and setter  
#' for the \code{palette\_gradient} slot of the object.  
#' @rdname melody-class  
setMethod("palette\_gradient",   
 signature = c(x = "melody"),  
 function(x){ x@palette\_gradient })  
  
#' @exportMethod palette\_gradient<-  
#' @aliases palette\_gradient<-  
#' @param value The value for the slot.  
#' @rdname melody-class  
setReplaceMethod("palette\_gradient",   
 signature = c(x = "melody"),  
 function(x, value){  
 initialize(x, palette\_gradient = value)  
 })  
  
  
#' @exportMethod palette\_stat  
#' @aliases palette\_stat  
#' @description \code{palette\_stat}, \code{palette\_stat<-}: getter and setter  
#' for the \code{palette\_stat} slot of the object.  
#' @rdname melody-class  
setMethod("palette\_stat",   
 signature = c(x = "melody"),  
 function(x){ x@palette\_stat })  
  
#' @exportMethod palette\_stat<-  
#' @aliases palette\_stat<-  
#' @param value The value for the slot.  
#' @rdname melody-class  
setReplaceMethod("palette\_stat",   
 signature = c(x = "melody"),  
 function(x, value){  
 initialize(x, palette\_stat = value)  
 })  
  
  
#' @exportMethod palette\_col  
#' @aliases palette\_col  
#' @description \code{palette\_col}, \code{palette\_col<-}: getter and setter  
#' for the \code{palette\_col} slot of the object.  
#' @rdname melody-class  
setMethod("palette\_col",   
 signature = c(x = "melody"),  
 function(x){ x@palette\_col })  
  
#' @exportMethod palette\_col<-  
#' @aliases palette\_col<-  
#' @param value The value for the slot.  
#' @rdname melody-class  
setReplaceMethod("palette\_col",   
 signature = c(x = "melody"),  
 function(x, value){  
 initialize(x, palette\_col = value)  
 })  
  
  
#' @exportMethod palette\_label  
#' @aliases palette\_label  
#' @description \code{palette\_label}, \code{palette\_label<-}: getter and setter  
#' for the \code{palette\_label} slot of the object.  
#' @rdname melody-class  
setMethod("palette\_label",   
 signature = c(x = "melody"),  
 function(x){ x@palette\_label })  
  
#' @exportMethod palette\_label<-  
#' @aliases palette\_label<-  
#' @param value The value for the slot.  
#' @rdname melody-class  
setReplaceMethod("palette\_label",   
 signature = c(x = "melody"),  
 function(x, value){  
 initialize(x, palette\_label = value)  
 })  
# ==========================================================================  
# msframe: class based on data.frame  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass msframe  
#'  
#' @aliases msframe  
#'  
#' @title format and filter table data  
#'  
#' @description  
#' Class for table data manipulation inside this package.  
#'  
#' @family subscripts  
#'  
#' @slot entity data.frame.  
#' @slot subscript character(1). See [subscript-class].  
#'  
#' @rdname msframe-class  
#'  
.msframe <-   
 setClass("msframe",   
 contains = "subscript",  
 representation =   
 representation(entity = "data.frame"),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# methods  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod show  
#' @aliases show  
#' @rdname msframe-class  
setMethod("show",   
 signature = c(object = "msframe"),  
 function(object){  
 cat( "A class of \"msframe\" of", subscript(object), "\n")  
 })  
  
  
#' @exportMethod msframe  
#' @aliases msframe  
#' @description \code{msframe}, \code{msframe<-}: getter and setter  
#' for the \code{msframe} slot of the object.  
#' @rdname msframe-class  
setMethod("msframe",   
 signature = c(x = "ANY"),  
 function(x){ x@msframe })  
  
#' @exportMethod msframe<-  
#' @aliases msframe<-  
#' @param value The value for the slot.  
#' @rdname msframe-class  
setReplaceMethod("msframe",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, msframe = value)  
 })  
  
  
#' @exportMethod latest  
#' @aliases latest  
#' @description \code{latest}: get data inside \code{entity(object)} and format as  
#' 'tbl'.  
#' @family latests  
#' @seealso [tibble::as\_tibble()]  
#' @rdname msframe-class  
setMethod("latest",   
 signature = c(x = "msframe"),  
 function(x){  
 tibble::as\_tibble(entity(x))  
 })  
  
  
#' @exportMethod entity  
#' @aliases entity  
#' @description \code{entity}, \code{entity<-}: getter and setter  
#' for the \code{entity} slot of the object.  
#' @rdname msframe-class  
setMethod("entity",   
 signature = c(x = "msframe"),  
 function(x){ x@entity })  
  
#' @exportMethod entity<-  
#' @aliases entity<-  
#' @param value The value for the slot.  
#' @rdname msframe-class  
setReplaceMethod("entity",   
 signature = c(x = "msframe"),  
 function(x, value){  
 initialize(x, entity = value)  
 })  
  
  
#' @exportMethod format\_msframe  
#' @aliases format\_msframe  
#' @rdname msframe-class  
setMethod("format\_msframe",   
 signature = setMissing("format\_msframe",  
 x = "msframe",  
 fun\_format = "function"),  
 function(x, fun\_format){  
 entity(x) <- format\_msframe(entity(x), fun\_format = fun\_format)  
 return(x)  
 })  
  
#' @exportMethod format\_msframe  
#' @aliases format\_msframe  
#' @rdname msframe-class  
setMethod("format\_msframe",   
 signature = setMissing("format\_msframe",  
 x = "data.frame",  
 fun\_format = "function"),  
 function(x, fun\_format){  
 results <- try(fun\_format(x), silent = T)  
 if (!inherits(results, "try-error")) {  
 x[[ ".candidates\_id" ]] <- results  
 }  
 return(x)  
 })  
  
  
#' @exportMethod format\_msframe  
#' @aliases format\_msframe  
#' @rdname msframe-class  
setMethod("format\_msframe",   
 signature = setMissing("format\_msframe",  
 x = "msframe",  
 names = "character",  
 types = "character"),  
 function(x, names, types){  
 if( !is.character(names(names)) )  
 stop( "the `names` is unformat" )  
 if( !is.character(names(types)) )  
 stop( "the `types` is unformat" )  
 .format\_msframe(x, names, types)  
 })  
  
#' @exportMethod format\_msframe  
#' @aliases format\_msframe  
#' @rdname msframe-class  
setMethod("format\_msframe",   
 signature = setMissing("format\_msframe",  
 x = "msframe"),  
 function(x){  
 names <- .get\_attribute\_name\_sirius.v4()  
 types <- .get\_attribute\_type\_sirius.v4()  
 .format\_msframe(x, names, types)  
 })  
  
#' @exportMethod format\_msframe  
#'  
#' @aliases format\_msframe  
#'  
#' @description  
#' \code{format\_msframe}:  
#'  
#' @param x [msframe-class] object.  
#' @param names character with names.  
#' e.g., c(tani.score = "tanimotoSimilarity", mol.formula = "molecularFormula").  
#' @param fun\_names function to get names.  
#' e.g., \code{MCnebula2:::.get\_attribute\_name\_sirius.v4()}  
#' @param types character with names.  
#' e.g., c(tani.score = "numeric", mol.formula = "character").  
#' @param fun\_types function to get types.  
#' e.g., \code{MCnebula2:::.get\_attribute\_type\_sirius.v4()}  
#' @param fun\_format function to format slot \code{entity}.  
#' e.g., \code{MCnebula2:::.format\_msframe()}  
#'  
#' @rdname msframe-class  
#'  
setMethod("format\_msframe",   
 signature = setMissing("format\_msframe",  
 x = "msframe",  
 fun\_names = "function",  
 fun\_types = "function"),  
 function(x, fun\_names, fun\_types){  
 .format\_msframe(x, fun\_names(), fun\_types())  
 })  
  
.format\_msframe <-   
 function(x, names, types){  
 if( any(names(names) == "...sig") ) {  
 rs <- which( names == subscript(x) & names(names) == "...sig")  
 if (length(rs) != 0) {  
 rs <- rs + 1  
 re <- length(names)  
 for( i in rs:length(names) ){  
 if( names(names)[i] == "...sig" ) {  
 re <- i - 1  
 break  
 }  
 }  
 names <- c(names[rs:re], names)  
 }  
 names <- vec\_unique\_by\_value(names)  
 names <- names[names(names) != "...sig"]  
 }  
 x <- .format\_msframe\_names(x, names)  
 names <- names[names(names) %in% colnames(entity(x))]  
 .format\_msframe\_types(x, names, types)  
 }  
  
.format\_msframe\_names <-   
 function(x, names){  
 pattern <- paste0("^", names, "$")  
 colnames(entity(x)) <-  
 mapply\_rename\_col(pattern, names(names), colnames(entity(x)))  
 return(x)  
 }  
  
.format\_msframe\_types <-   
 function(x, names, types){  
 for (i in names(names)) {  
 if (i %in% names(types))  
 target\_type <- types[[i]]  
 else  
 target\_type <- "character"  
 fun <- match.fun(paste0("is.", target\_type))  
 if ( !fun(entity(x)[[i]]) ){  
 fun <- match.fun(paste0("as.", target\_type))  
 entity(x)[[i]] <- fun(entity(x)[[i]])  
 }  
 }  
 return(x)  
 }  
  
  
#' @exportMethod filter\_msframe  
#' @aliases filter\_msframe  
#' @rdname msframe-class  
setMethod("filter\_msframe",   
 signature = setMissing("filter\_msframe",  
 x = "msframe", fun\_filter = "function"),  
 function(x, fun\_filter, ...){  
 filter\_msframe(x, fun\_filter = fun\_filter,  
 f = ~ .features\_id, ...)  
 })  
  
#' @exportMethod filter\_msframe  
#'  
#' @aliases filter\_msframe  
#'  
#' @description \code{filter\_msframe}: filter data in slot \code{entity} (data.frame).  
#' @note The class is not for normal use of the package.  
#'  
#' @param x [msframe-class] object.  
#' @param fun\_filter function used to filter the slot \code{entity} (data.frame).  
#' e.g., \code{dplyr::filter()}, \code{head()}.  
#' @param f formula passed to \code{split()}.  
#' @param ... extra parameter passed to fun\_filter.  
#'  
#' @rdname msframe-class  
#'  
setMethod("filter\_msframe",   
 signature = setMissing("filter\_msframe",  
 x = "msframe", fun\_filter = "function",  
 f = "formula"),  
 function(x, fun\_filter, f, ...){  
 .message\_info("msframe", "filter\_msframe",  
 paste0("group\_by: ", paste0(f, collapse = " "))  
 )  
 entity <- lapply( split(entity(x), f = f), FUN = fun\_filter, ...)  
 entity(x) <- data.table::rbindlist(entity, fill = T)  
 return(x)  
 })  
  
# ==========================================================================  
# a class to store network component  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass parent\_nebula  
#'  
#' @aliases parent\_nebula  
#'  
#' @description \code{parent\_nebula}: Store data for visualization of  
#' Parent-Nebula.  
#'  
#' @rdname nebula-class  
#'  
.parent\_nebula <-   
 setClass("parent\_nebula",   
 contains = character(),  
 representation =   
 representation(igraph = "ANY",  
 tbl\_graph = "ANY",  
 layout\_ggraph = "ANY",  
 ggset = "ggset"  
 ),  
 prototype = NULL  
 )  
  
#' @exportClass child\_nebulae  
#'  
#' @aliases child\_nebulae  
#'  
#' @description \code{child\_nebulae}: store data for visualization of  
#' Child-Nebulae.  
#'  
#' @slot igraph "igraph" object or its list. See [igraph::graph\_from\_data\_frame()].  
#' The slot contains edges and nodes data of Child-Nebulae or Parent-Nebula.  
#' The "igraph" object can be output use [igraph::write\_graph()] as ".graphml" file,  
#' which belong to a network data format that can be operated by other software such as  
#' Cytoscape (\url{https://cytoscape.org/}).  
#'  
#' @slot tbl\_graph "tbl\_graph" object or its list. See [tidygraph::as\_tbl\_graph()].  
#' Converted from slot \code{igraph}.  
#'  
#' @slot layout\_ggraph "layout\_ggraph" object or its list. See [ggraph::create\_layout()].  
#' Create from slot \code{tbl\_graph}, passed to [ggraph::ggraph()] for visualization.  
#'  
#' @slot grid\_layout "layout" object. See [grid::grid.layout()].  
#' Grid layout for position of each Child-Nebula to visualize.  
#'  
#' @slot viewports list with names. Each element must be "viewport" object.  
#' See [grid::viewport()]. Position for each Child-Nebula to visualize.  
#'  
#' @slot panel\_viewport "viewport" object. See [grid::viewport()]. For visualization,  
#' the position to place overall Child-Nebulae.  
#'  
#' @slot legend\_viewport "viewport" object. See [grid::viewport()]. For visualization,  
#' the position to place legend.  
#'  
#' @slot ggset [ggset-class] object or its list with names. Each [ggset-class] object  
#' can be visualized directly use [call\_command()].  
#'  
#' @slot structures\_grob list with names. Each element is a "grob" object.  
#' See [grid::grob()]. Use [grid::grid.draw()] to visualize the chemical structure.  
#'  
#' @slot nodes\_ggset list of [ggset-class] object. For drawing each node of 'features'  
#' ('features' means the detected peaks while processing LC-MS data)  
#' with annotation. Use [call\_command()] to visualize the [ggset-class].  
#'  
#' @slot nodes\_grob list of "grob" object. Converted from slot \code{nodes\_ggset} with slot  
#' \code{structures\_grob}. Use [grid::grid.draw()] to visualize the "grob".  
#'  
#' @slot ppcp\_data list with names. Each element is a data.frame. This is an  
#' annotation data of 'features' which would be visualize in nodes border  
#' as a radial bar plot. \code{ppcp\_data}, i.e., posterior probability of  
#' classification prediction. See [filter\_ppcp()].  
#'  
#' @slot ration\_data list with names. Each element is a data.frame. This is an  
#' annotation data of 'features' which would be visualize in nodes nucleus as  
#' ring plot. Generally, \code{ration\_data} is the statistic data for samples.  
#'  
#' @slot ggset\_annotate a list of [ggset-class] object. The annotated Child-Nebulae  
#' gathered from slot \code{ggset} and slot \code{nodes\_grob}.   
#' Use [call\_command()] to visualize the [ggset-class]. Be care, the object  
#' sometimes is too large that need lot of time to loading for visualization.  
#'  
#' @rdname nebula-class  
#'  
.child\_nebulae <-   
 setClass("child\_nebulae",   
 contains = character(),  
 representation =   
 representation(igraph = "list",  
 tbl\_graph = "list",  
 layout\_ggraph = "list",  
 grid\_layout = "ANY",  
 viewports = "list",  
 panel\_viewport = "ANY",  
 legend\_viewport = "ANY",  
 ggset = "list",  
 structures\_grob = "list",  
 nodes\_ggset = "list",  
 nodes\_grob = "list",  
 ppcp\_data = "list",  
 ration\_data = "list",  
 ggset\_annotate = "list"  
 ),  
 prototype = NULL  
 )  
  
#' @exportClass nebula  
#'  
#' @aliases nebula  
#'  
#' @title Visualization component of chemical Nebulae/Nebula  
#'  
#' @description This class store multiple components for visualization.  
#'  
#' @family nebulae  
#'  
#' @slot parent\_nebula [parent\_nebula-class] object.  
#' @slot child\_nebulae [child\_nebulae-class] object.  
#'  
#' @rdname nebula-class  
#' @order 1  
#'  
.nebula <-   
 setClass("nebula",   
 contains = character(),  
 representation =   
 representation(parent\_nebula = "parent\_nebula",  
 child\_nebulae = "child\_nebulae"  
 ),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod parent\_nebula  
#' @aliases parent\_nebula  
#' @rdname nebula-class  
setMethod("show",   
 signature = c(object = "parent\_nebula"),  
 function(object){  
 .show\_nebulae\_data(object)  
 })  
  
#' @exportMethod child\_nebulae  
#' @aliases child\_nebulae  
#' @rdname nebula-class  
setMethod("show",   
 signature = c(object = "child\_nebulae"),  
 function(object){  
 .show\_nebulae\_data(object)  
 })  
  
.show\_nebulae\_data <-   
 function(object){  
 slots\_mapply(object, function(slot, name){  
 if (is(slot, "viewport")) {  
 num <- 1  
 } else if (is.list(slot)) {  
 num <- length(slot)  
 } else {  
 if (is.null(slot))  
 num <- 0  
 else  
 num <- 1  
 }  
 if (num == 0 | is(slot, "name"))   
 return()  
 cat(name, ": ", class(slot)[1], " of ", num,  
 "\n", sep = "")  
 })  
 }  
  
  
#' @exportMethod parent\_nebula  
#' @aliases parent\_nebula  
#' @description \code{parent\_nebula}, \code{parent\_nebula<-}: getter and setter  
#' for the \code{parent\_nebula} slot of the object.  
#' @rdname nebula-class  
setMethod("parent\_nebula",   
 signature = c(x = "ANY"),  
 function(x){ x@parent\_nebula })  
  
#' @exportMethod parent\_nebula<-  
#' @aliases parent\_nebula<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("parent\_nebula",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, parent\_nebula = value)  
 })  
  
#' @exportMethod child\_nebulae  
#' @aliases child\_nebulae  
#' @description \code{child\_nebulae}, \code{child\_nebulae<-}: getter and setter  
#' for the \code{child\_nebulae} slot of the object.  
#' @rdname nebula-class  
setMethod("child\_nebulae",   
 signature = c(x = "ANY"),  
 function(x){ x@child\_nebulae })  
  
#' @exportMethod child\_nebulae<-  
#' @aliases child\_nebulae<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("child\_nebulae",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, child\_nebulae = value)  
 })  
  
  
#' @exportMethod igraph  
#' @aliases igraph  
#' @description \code{igraph}, \code{igraph<-}: getter and setter  
#' for the \code{igraph} slot of the object.  
#' @rdname nebula-class  
setMethod("igraph",   
 signature = c(x = "ANY"),  
 function(x){ x@igraph })  
  
#' @exportMethod igraph<-  
#' @aliases igraph<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("igraph",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, igraph = value)  
 })  
  
  
#' @exportMethod tbl\_graph  
#' @aliases tbl\_graph  
#' @description \code{tbl\_graph}, \code{tbl\_graph<-}: getter and setter  
#' for the \code{tbl\_graph} slot of the object.  
#' @rdname nebula-class  
setMethod("tbl\_graph",   
 signature = c(x = "ANY"),  
 function(x){ x@tbl\_graph })  
  
#' @exportMethod tbl\_graph<-  
#' @aliases tbl\_graph<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("tbl\_graph",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, tbl\_graph = value)  
 })  
  
  
#' @exportMethod layout\_ggraph  
#' @aliases layout\_ggraph  
#' @description \code{layout\_ggraph}, \code{layout\_ggraph<-}: getter and setter  
#' for the \code{layout\_ggraph} slot of the object.  
#' @rdname nebula-class  
setMethod("layout\_ggraph",   
 signature = c(x = "ANY"),  
 function(x){ x@layout\_ggraph })  
  
#' @exportMethod layout\_ggraph<-  
#' @aliases layout\_ggraph<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("layout\_ggraph",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, layout\_ggraph = value)  
 })  
  
  
#' @exportMethod grid\_layout  
#' @aliases grid\_layout  
#' @description \code{grid\_layout}, \code{grid\_layout<-}: getter and setter  
#' for the \code{grid\_layout} slot of the object.  
#' @rdname nebula-class  
setMethod("grid\_layout",   
 signature = c(x = "ANY"),  
 function(x){ x@grid\_layout })  
  
#' @exportMethod grid\_layout<-  
#' @aliases grid\_layout<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("grid\_layout",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, grid\_layout = value)  
 })  
  
  
#' @exportMethod viewports  
#' @aliases viewports  
#' @description \code{viewports}, \code{viewports<-}: getter and setter  
#' for the \code{viewports} slot of the object.  
#' @rdname nebula-class  
setMethod("viewports",   
 signature = c(x = "ANY"),  
 function(x){ x@viewports })  
  
#' @exportMethod viewports<-  
#' @aliases viewports<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("viewports",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, viewports = value)  
 })  
  
  
#' @exportMethod ggset  
#' @aliases ggset  
#' @description \code{ggset}, \code{ggset<-}: getter and setter  
#' for the \code{ggset} slot of the object.  
#' @rdname nebula-class  
setMethod("ggset",   
 signature = c(x = "ANY"),  
 function(x){ x@ggset })  
  
#' @exportMethod ggset<-  
#' @aliases ggset<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("ggset",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, ggset = value)  
 })  
  
  
#' @exportMethod panel\_viewport  
#' @aliases panel\_viewport  
#' @description \code{panel\_viewport}, \code{panel\_viewport<-}: getter and setter  
#' for the \code{panel\_viewport} slot of the object.  
#' @rdname nebula-class  
setMethod("panel\_viewport",   
 signature = c(x = "ANY"),  
 function(x){ x@panel\_viewport })  
  
#' @exportMethod panel\_viewport<-  
#' @aliases panel\_viewport<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("panel\_viewport",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, panel\_viewport = value)  
 })  
  
  
#' @exportMethod legend\_viewport  
#' @aliases legend\_viewport  
#' @description \code{legend\_viewport}, \code{legend\_viewport<-}: getter and setter  
#' for the \code{legend\_viewport} slot of the object.  
#' @rdname nebula-class  
setMethod("legend\_viewport",   
 signature = c(x = "ANY"),  
 function(x){ x@legend\_viewport })  
  
#' @exportMethod legend\_viewport<-  
#' @aliases legend\_viewport<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("legend\_viewport",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, legend\_viewport = value)  
 })  
  
  
#' @exportMethod structures\_grob  
#' @aliases structures\_grob  
#' @description \code{structures\_grob}, \code{structures\_grob<-}: getter and setter  
#' for the \code{structures\_grob} slot of the object.  
#' @rdname nebula-class  
setMethod("structures\_grob",   
 signature = c(x = "ANY"),  
 function(x){ x@structures\_grob })  
  
#' @exportMethod structures\_grob<-  
#' @aliases structures\_grob<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("structures\_grob",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, structures\_grob = value)  
 })  
  
  
#' @exportMethod nodes\_ggset  
#' @aliases nodes\_ggset  
#' @description \code{nodes\_ggset}, \code{nodes\_ggset<-}: getter and setter  
#' for the \code{nodes\_ggset} slot of the object.  
#' @rdname nebula-class  
setMethod("nodes\_ggset",   
 signature = c(x = "ANY"),  
 function(x){ x@nodes\_ggset })  
  
#' @exportMethod nodes\_ggset<-  
#' @aliases nodes\_ggset<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("nodes\_ggset",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, nodes\_ggset = value)  
 })  
  
  
#' @exportMethod nodes\_grob  
#' @aliases nodes\_grob  
#' @description \code{nodes\_grob}, \code{nodes\_grob<-}: getter and setter  
#' for the \code{nodes\_grob} slot of the object.  
#' @rdname nebula-class  
setMethod("nodes\_grob",   
 signature = c(x = "ANY"),  
 function(x){ x@nodes\_grob })  
  
#' @exportMethod nodes\_grob<-  
#' @aliases nodes\_grob<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("nodes\_grob",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, nodes\_grob = value)  
 })  
  
  
#' @exportMethod ppcp\_data  
#' @aliases ppcp\_data  
#' @description \code{ppcp\_data}, \code{ppcp\_data<-}: getter and setter  
#' for the \code{ppcp\_data} slot of the object.  
#' @rdname nebula-class  
setMethod("ppcp\_data",   
 signature = c(x = "ANY"),  
 function(x){ x@ppcp\_data })  
  
#' @exportMethod ppcp\_data<-  
#' @aliases ppcp\_data<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("ppcp\_data",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, ppcp\_data = value)  
 })  
  
  
#' @exportMethod ration\_data  
#' @aliases ration\_data  
#' @description \code{ration\_data}, \code{ration\_data<-}: getter and setter  
#' for the \code{ration\_data} slot of the object.  
#' @rdname nebula-class  
setMethod("ration\_data",   
 signature = c(x = "ANY"),  
 function(x){ x@ration\_data })  
  
#' @exportMethod ration\_data<-  
#' @aliases ration\_data<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("ration\_data",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, ration\_data = value)  
 })  
  
  
#' @exportMethod ggset\_annotate  
#' @aliases ggset\_annotate  
#' @description \code{ggset\_annotate}, \code{ggset\_annotate<-}: getter and setter  
#' for the \code{ggset\_annotate} slot of the object.  
#' @rdname nebula-class  
setMethod("ggset\_annotate",   
 signature = c(x = "ANY"),  
 function(x){ x@ggset\_annotate })  
  
#' @exportMethod ggset\_annotate<-  
#' @aliases ggset\_annotate<-  
#' @param value The value for the slot.  
#' @rdname nebula-class  
setReplaceMethod("ggset\_annotate",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, ggset\_annotate = value)  
 })  
# ==========================================================================  
# a class to store functions of reading or formating the target data  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass project\_api  
#'  
#' @aliases project\_api  
#'  
#' @title Function set for extracting data  
#'  
#' @description   
#' This is a class object used to store various functions for extracting and formatting data.  
#' See [project-class] for joint application with other related classes.  
#' @family projects  
#'  
#' @note The class is not for normal use of the package.  
#'  
#' @slot methods\_read list. Store a list of functions for reading data.  
#' The list with the names: "read" + "subscript". e.g., "read.f3\_fingerid".  
#' @slot methods\_format function. The function is used to format the data  
#' (e.g., rename the column names; convert the columns of character type into numeric).  
#' @slot methods\_match list. Store a list of functions for matching and extracting string.  
#'  
#' @rdname project\_api-class  
#'  
.project\_api <-   
 setClass("project\_api",   
 contains = character(),  
 representation =   
 representation(methods\_read = "list",  
 methods\_format = "function",  
 methods\_match = "list"  
 ),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# method  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod show  
#' @aliases show  
#' @rdname project\_api-class  
setMethod("show",   
 signature = c(object = "project\_api"),  
 function(object){  
 .show(object)  
 })  
  
#' @exportMethod project\_api  
#' @aliases project\_api  
#' @description \code{project\_api}, \code{project\_api<-}: getter and setter  
#' for the \code{project\_api} slot of the object.  
#' @rdname project\_api-class  
setMethod("project\_api",   
 signature = c(x = "ANY"),  
 function(x){ x@project\_api })  
  
#' @exportMethod project\_api<-  
#' @aliases project\_api<-  
#' @param value The value for the slot.  
#' @rdname project\_api-class  
setReplaceMethod("project\_api",   
 signature = c(x = "ANY"),  
 function(x, value){  
 initialize(x, project\_api = value)  
 })  
  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportMethod methods\_read  
#' @aliases methods\_read  
#' @description \code{methods\_read}, \code{methods\_read<-}: getter and setter  
#' for the \code{methods\_read} slot of the object.  
#' @rdname project\_api-class  
setMethod("methods\_read",   
 signature = c(x = "project\_api"),  
 function(x){ x@methods\_read })  
  
#' @exportMethod methods\_read<-  
#' @aliases methods\_read<-  
#' @param value The value for the slot.  
#' @rdname project\_api-class  
setReplaceMethod("methods\_read",   
 signature = c(x = "project\_api"),  
 function(x, value){  
 initialize(x, methods\_read = value)  
 })  
  
  
#' @exportMethod methods\_format  
#' @aliases methods\_format  
#' @description \code{methods\_format}, \code{methods\_format<-}: getter and setter  
#' for the \code{methods\_format} slot of the object.  
#' @rdname project\_api-class  
setMethod("methods\_format",   
 signature = c(x = "project\_api"),  
 function(x){ x@methods\_format })  
  
#' @exportMethod methods\_format<-  
#' @aliases methods\_format<-  
#' @param value The value for the slot.  
#' @rdname project\_api-class  
setReplaceMethod("methods\_format",   
 signature = c(x = "project\_api"),  
 function(x, value){  
 initialize(x, methods\_format = value)  
 })  
  
  
#' @exportMethod methods\_match  
#' @aliases methods\_match  
#' @description \code{methods\_match}, \code{methods\_match<-}: getter and setter  
#' for the \code{methods\_match} slot of the object.  
#' @rdname project\_api-class  
setMethod("methods\_match",   
 signature = c(x = "project\_api"),  
 function(x){ x@methods\_match })  
  
#' @exportMethod methods\_match<-  
#' @aliases methods\_match<-  
#' @param value The value for the slot.  
#' @rdname project\_api-class  
setReplaceMethod("methods\_match",   
 signature = c(x = "project\_api"),  
 function(x, value){  
 initialize(x, methods\_match = value)  
 })  
# ==========================================================================  
# a class to store the characters of files or data in raw project.  
# These generally describe the file name, file path, and attributes name.  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @exportClass project\_conformation  
#'  
#' @aliases project\_conformation  
#'  
#' @title Clarify the name, path and attribute name of files  
#' in the project (directory)  
#'  
#' @description   
#' This is a class object used to record the name, path and attribute name of the file.  
#' These records can be retrieved by "subscript" (see [subscript-class]).  
#' See [project-class] for joint application with other related classes.  
#'  
#' @note The class is not for normal use of the package.  
#'  
#' @family projects  
#' @family subscripts  
#'  
#' @slot file\_name character with names.  
#' Record the filenames or pattern string or function name (begin with "FUN\_")  
#' for each "subscript" (imply file names).  
#' @slot file\_api character with names.  
#' Record the file path for each "subscript" (imply file names).  
#' The path is descriped by "subscript" with "/".  
#' @slot attribute\_name character with names.  
#' Record the attribute name for each "subscript" (imply column names).  
#'  
#' @rdname project\_conformation-class  
#'  
.project\_conformation <-   
 setClass("project\_conformation",   
 contains = character(),  
 representation =   
 representation(file\_name = "character",  
 file\_api = "character",  
 attribute\_name = "character"  
 ),  
 prototype = NULL  
 )  
  
# ==========================================================================  
# validity  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
setValidity("project\_conformation",   
 function(object){  
 check <-   
 slots\_mapply(object,  
 function(slot, name){  
 if (is.character(slot) & length(slot) == 0) {  
 TRUE  
 } else {  
 if ( is.character( names(slot) ))  
 TRUE

# 2 后2500行代码：

#' @param ... Other parameters passed to the function \code{fun\_filter}.  
#' @param by\_reference logical. Use \code{specific\_candidate(object)} data to filter  
#' candidates data. See [create\_reference()].  
#'  
#' @rdname filter\_formula-methods  
#'  
#' @examples  
#' \dontrun{  
#' test <- mcn\_5features  
#'   
#' ## filter chemical formula candidates  
#' ## use default parameters  
#' test1 <- filter\_formula(test)  
#' latest(test1)  
#'   
#' ## the default parameters:  
#' filter\_formula()  
#'   
#' ## customized filtering  
#' ## according to score  
#' test1 <- filter\_formula(test1, dplyr::filter, zodiac.score > 0.5)  
#' latest(test1)  
#'   
#' ## get top rank  
#' test1 <- filter\_formula(test1, dplyr::filter, rank.formula <= 3)  
#' latest(test1)  
#'   
#' ## complex filtering  
#' test1 <- filter\_formula(  
#' test1, dplyr::filter,  
#' ## molecular formula  
#' !grepl("N", mol.formula),  
#' ## mass error  
#' abs(error.mass) < 0.001  
#' )  
#' latest(test1)  
#'   
#' ## select columns  
#' test1 <- filter\_formula(test1, dplyr::select, 1:5)  
#' latest(test1)  
#' }  
setMethod("filter\_formula",   
 signature = setMissing("filter\_formula",  
 x = "mcnebula",  
 fun\_filter = "function",  
 by\_reference = "logical"),  
 function(x, fun\_filter, ..., by\_reference){  
 .message\_info\_formal("MCnebula2", "filter\_formula")  
 subscript <- ".f2\_formula"  
 x <- collate\_data(x, subscript, .collate\_formula.msframe)  
 ## filter  
 msframe.lst <- extract\_rawset(x, subscript)  
 if (by\_reference) {  
 .message\_info("filter\_formula", "by\_reference == T",  
 "\n\tcase formula, ignore `fun\_filter`")  
 .check\_data(x, list(specific\_candidate = "create\_reference"))  
 fun <- methods\_match(project\_api(x))[[ "generate\_candidates\_id" ]]  
 entity(msframe.lst[[1]]) <-   
 merge(specific\_candidate(x),  
 format\_msframe(entity(msframe.lst[[1]]), fun\_format = fun),  
 by = c(".features\_id", ".candidates\_id"))  
 } else {  
 msframe.lst[[1]] <-  
 filter\_msframe(msframe.lst[[1]], fun\_filter = fun\_filter,  
 f = ~.features\_id, ...)  
 }  
 mcn\_dataset(x) <- add\_dataset(mcn\_dataset(x), msframe.lst)  
 return(x)  
 })  
  
.collate\_formula.msframe <-   
 function(x, subscript){  
 msframe <- .collate\_data.msframe(x, subscript)  
 if (!"zodiac.score" %in% colnames(entity(msframe))) {  
 warning("`zodiac.score` not found in `msframe`, fill it with `zodiac.score` = 0")  
 entity(msframe)$zodiac.score <- 0  
 }  
 msframe  
 }  
# ==========================================================================  
# collate ppcp dataset in sirius project and do filtering  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @aliases filter\_ppcp  
#'  
#' @title Collate and filter candidates of chemical classification for each 'feature'  
#'  
#' @description This methods provide an approach to  
#' collate and filter chemical classification candidates data in baches for each  
#' 'feature'.  
#'  
#' @details   
#' Filter for PPCP (posterior probability of classification prediction) data.  
#' See details about classification prediction for compounds:  
#' \url{http://www.nature.com/articles/s41587-020-0740-8}.  
#' See other details in [filter\_formula()].  
#'  
#' @name filter\_ppcp-methods  
#'  
#' @order 1  
NULL  
#> NULL  
  
#' @exportMethod filter\_ppcp  
#' @description \code{filter\_ppcp()}: get the default parameters for the method  
#' \code{filter\_ppcp}.  
#' @rdname filter\_ppcp-methods  
setMethod("filter\_ppcp",   
 signature = setMissing("filter\_ppcp",  
 x = "missing"),  
 function(){  
 list(fun\_filter = .filter\_ppcp\_by\_threshold,  
 by\_reference = T  
 )  
 })  
  
#' @exportMethod filter\_ppcp  
#' @description \code{filter\_ppcp(x, ...)}: use the default parameters whatever 'missing'  
#' while performing the method \code{filter\_ppcp}.  
#' @rdname filter\_ppcp-methods  
setMethod("filter\_ppcp",   
 signature = c(x = "mcnebula"),  
 function(x, fun\_filter, ..., by\_reference){  
 reCallMethod("filter\_ppcp",  
 .fresh\_param(filter\_ppcp()), ...)  
 })  
  
#' @exportMethod filter\_ppcp  
#'  
#' @aliases filter\_ppcp  
#'  
#' @inheritParams filter\_formula-methods  
#'  
#' @rdname filter\_ppcp-methods  
#'  
#' @examples  
#' \dontrun{  
#' test <- mcn\_5features  
#'   
#' ## filter chemical class candidates  
#' ## the default parameters:  
#' filter\_ppcp()  
#'   
#' ## if 'by\_reference' set with TRUE, 'create\_reference' should be  
#' ## run previously.  
#' test1 <- filter\_ppcp(test, by\_reference = F)  
#' latest(test1)  
#'   
#' ## customized filtering  
#' ## according to score  
#' test1 <- filter\_ppcp(test1, dplyr::filter, pp.value > 0.5,  
#' by\_reference = F)  
#' latest(test1)  
#'   
#' ## complex filtering  
#' test1 <- filter\_ppcp(  
#' test1, dplyr::filter,  
#' ## PPCP value  
#' pp.value > 0.5,  
#' ## speicifid class  
#' class.name %in% c("Azoles"),  
#' by\_reference = F  
#' )  
#' latest(test1)  
#'   
#' ## select columns  
#' test1 <- filter\_ppcp(test1, dplyr::select, 1:5,  
#' by\_reference = F)  
#' latest(test1)  
#' }  
setMethod("filter\_ppcp",   
 signature = setMissing("filter\_ppcp",  
 x = "mcnebula", fun\_filter = "function",  
 by\_reference = "logical"),  
 function(x, fun\_filter, ..., by\_reference){  
 .message\_info\_formal("MCnebula2", "filter\_ppcp")  
 if (by\_reference) {  
 .message\_info("filter\_ppcp", "by\_reference == T")  
 .check\_data(x, list(specific\_candidate = "create\_reference"))  
 }  
 subscript <- c(".canopus", ".f3\_canopus")  
 if (ion\_mode(x) == "neg")  
 subscript[1] <- c(".canopus\_neg")  
 for (i in subscript) {  
 x <- get\_metadata(x, i)  
 if (by\_reference & i == subscript[2])  
 x <- collate\_data(x, i, reference = specific\_candidate(x))  
 else  
 x <- collate\_data(x, i)  
 }  
 annotation <- entity(dataset(project\_dataset(x))[[ subscript[1] ]])  
 msframe.lst <- extract\_rawset(x, subscript = subscript[2])  
 ## validate  
 if ( !subscript[2] %in% names(dataset(mcn\_dataset(x))) ) {  
 .message\_info("filter\_ppcp", "validate annotation data",  
 paste0(subscript, collapse = " >>> "))  
 validate\_ppcp\_annotation(annotation, msframe.lst)  
 ## add annotation into dataset  
 msframe.lst <- merge\_ppcp\_annotation(annotation, msframe.lst)  
 project\_dataset(x) <- add\_dataset(project\_dataset(x), msframe.lst)  
 }  
 ## filter  
 msframe.lst[[1]] <-  
 filter\_msframe(msframe.lst[[1]], fun\_filter = fun\_filter,  
 f = ~ paste0(.features\_id, "\_", .candidates\_id), ...)  
 mcn\_dataset(x) <- add\_dataset(mcn\_dataset(x), msframe.lst)  
 return(x)  
 })  
  
validate\_ppcp\_annotation <-   
 function(annotation, lst){  
 rows <- nrow(annotation)  
 lst <- split(entity(lst[[1]]), f = ~ paste0(.features\_id, "\_", .candidates\_id))  
 if (!identical( annotation$rel.index, lst[[1]]$rel.index))  
 stop("the annotation not match the classification dataset: 1")  
 lapply(lst, function(df){  
 if (nrow(df) != rows)  
 stop("the annotation not match the classification dataset")  
 })  
 }  
  
merge\_ppcp\_annotation <-   
 function(annotation, msframe.lst){  
 annotation <- dplyr::select(annotation, -.features\_id, -.candidates\_id)  
 col <- colnames(annotation)  
 col <- col[!col %in% colnames(entity(msframe.lst[[1]]))]  
 annotation <- dplyr::select(annotation, rel.index, dplyr::all\_of(col))  
 entity(msframe.lst[[1]]) <-   
 merge(entity(msframe.lst[[1]]), annotation,  
 by = "rel.index", all.x = T, sort = F)  
 return(msframe.lst)  
 }  
# ==========================================================================  
# collate structure dataset in sirius project and do filtering  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @aliases filter\_structure  
#'  
#' @title Collate and filter candidates of chemical structure for each 'feature'  
#'  
#' @description This methods provide an approach to  
#' collate and filter chemical structure candidates data in baches for each  
#' 'feature'.  
#'  
#' @details See details in [filter\_formula()].  
#'  
#' @name filter\_structure-methods  
#'  
#' @order 1  
NULL  
#> NULL  
  
#' @exportMethod filter\_structure  
#' @description \code{filter\_structure()}: get the default parameters for the method  
#' \code{filter\_structure}.  
#' @rdname filter\_structure-methods  
setMethod("filter\_structure",   
 signature = setMissing("filter\_structure",  
 x = "missing"),  
 function(){  
 list(fun\_filter = .rank\_by\_csi.score,  
 by\_reference = F  
 )  
 })  
  
#' @exportMethod filter\_structure  
#' @description \code{filter\_structure(x, ...)}: use the default parameters whatever 'missing'  
#' while performing the method \code{filter\_structure}.  
#' @rdname filter\_structure-methods  
setMethod("filter\_structure",   
 signature = c(x = "mcnebula"),  
 function(x, fun\_filter, ..., by\_reference){  
 reCallMethod("filter\_structure",  
 .fresh\_param(filter\_structure()), ...)  
 })  
  
#' @exportMethod filter\_structure  
#'  
#' @aliases filter\_structure  
#'  
#' @inheritParams filter\_formula-methods  
#'  
#' @rdname filter\_structure-methods  
#'  
#' @examples  
#' \dontrun{  
#' test <- mcn\_5features  
#'   
#' ## filter chemical structure candidates  
#' ## use default parameters  
#' test1 <- filter\_structure(test)  
#' latest(test1)  
#'   
#' ## the default parameters:  
#' filter\_structure()  
#'   
#' ## customized filtering  
#' ## according to score  
#' test1 <- filter\_structure(test1, dplyr::filter, tani.score > 0.4)  
#' latest(test1)  
#'   
#' ## get top rank  
#' test1 <- filter\_structure(test1, dplyr::filter, rank.structure <= 3)  
#' latest(test1)  
#'   
#' ## complex filtering  
#' test1 <- filter\_structure(  
#' test1, dplyr::filter,  
#' ## molecular formula  
#' !grepl("N", mol.formula),  
#' ## Tanimoto similarity  
#' tani.score > 0.4  
#' )  
#' latest(test1)  
#'   
#' ## select columns  
#' test1 <- filter\_structure(test1, dplyr::select, 1:5)  
#' latest(test1)  
#' }  
setMethod("filter\_structure",   
 signature = setMissing("filter\_structure",  
 x = "mcnebula",  
 fun\_filter = "function",  
 by\_reference = "logical"),  
 function(x, fun\_filter, ..., by\_reference){  
 .message\_info\_formal("MCnebula2", "filter\_structure")  
 subscript <- ".f3\_fingerid"  
 x <- collate\_data(x, subscript)  
 ## filter  
 msframe.lst <- extract\_rawset(x, subscript)  
 if (by\_reference) {  
 .message\_info("filter\_structure", "by\_reference == T")  
 .check\_data(x, list(specific\_candidate = "create\_reference"))  
 entity(msframe.lst[[1]]) <-   
 merge(specific\_candidate(x), entity(msframe.lst[[1]]),  
 by = c(".features\_id", ".candidates\_id"))  
 }  
 msframe.lst[[1]] <-  
 filter\_msframe(msframe.lst[[1]], fun\_filter = fun\_filter,  
 f = ~.features\_id, ...)  
 mcn\_dataset(x) <- add\_dataset(mcn\_dataset(x), msframe.lst)  
 return(x)  
 })  
# ==========================================================================  
# set default value for project of MCnebula  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @aliases initialize\_mcnebula  
#'  
#' @title Initialize mcnebula object  
#'  
#' @description  
#' Set SIRIUS project path and its version to initialize [mcnebula-class] object.  
#' In addition, the methods can be used for some related object to given  
#' default value.  
#'  
#' @name initialize\_mcnebula-methods  
#'  
#' @order 1  
NULL  
#> NULL  
  
#' @importFrom methods getFunction  
#' @exportMethod initialize\_mcnebula  
#'  
#' @aliases initialize\_mcnebula  
#'  
#' @param x [mcnebula-class] object, [melody-class] object,  
#' [project\_conformation-class] or [project\_api-class] object.  
#' @param sirius\_version character. e.g., "sirius.v4", "sirius.v5"  
#' @param sirius\_project character. The path of SIRIUS project space.  
#' @param output\_directory character. The path for output.  
#'  
#' @rdname initialize\_mcnebula-methods  
#'  
#' @examples  
#' \dontrun{  
#' ## The raw data used for the example  
#' tmp <- paste0(tempdir(), "/temp\_data")  
#' dir.create(tmp)  
#' eg.path <- system.file("extdata", "raw\_instance.tar.gz",  
#' package = "MCnebula2")  
#'   
#' utils::untar(eg.path, exdir = tmp)  
#'   
#' ## initialize 'mcnebula' object  
#' test <- mcnebula()  
#' test <- initialize\_mcnebula(test, "sirius.v4", tmp)  
#' ## check the setting  
#' export\_path(test)  
#' palette\_set(test)  
#' ion\_mode(test)  
#' project\_version(test)  
#'   
#' ## initialize 'melody' object  
#' test <- new("melody")  
#' test <- initialize\_mcnebula(test)  
#' ## check...  
#' palette\_stat(test)  
#'   
#' ## initialize 'project\_conformation' object  
#' test <- new("project\_conformation")  
#' test <- initialize\_mcnebula(test, "sirius.v4")  
#' ## check  
#' file\_name(test)  
#'   
#' ## initialize 'project\_api' object  
#' test <- new("project\_api")  
#' test <- initialize\_mcnebula(test, "sirius.v4")  
#' ## check  
#' methods\_format(test)  
#'   
#' unlink(tmp, T, T)  
#' }  
setMethod("initialize\_mcnebula",   
 signature = c(x = "mcnebula",  
 sirius\_version = "ANY",  
 sirius\_project = "ANY",  
 output\_directory = "ANY"),  
 function(x, sirius\_version, sirius\_project, output\_directory){  
 if (missing(sirius\_version))  
 sirius\_version <- project\_version(x)  
 else  
 project\_version(x) <- sirius\_version  
 if (missing(sirius\_project))  
 sirius\_project <- project\_path(x)  
 else  
 project\_path(x) <- sirius\_project  
 if (missing(output\_directory)) {  
 if (length(x@export\_path) == 0) {  
 export\_path(x) <- paste0(sirius\_project, "/mcnebula\_results")  
 }  
 } else {  
 export\_path(x) <- output\_directory  
 }  
 getFunction(paste0(".validate\_", sirius\_version),  
 where = parent.env(environment()))(sirius\_project)  
 item <- methods(initialize\_mcnebula)  
 item <- stringr::str\_extract(item, "(?<=,).\*(?=-method)")  
 item <- gsub(",.\*$", "", item)  
 item <- item[item != "mcnebula"]  
 for(i in item){  
 express <- paste0(i, "(x)",  
 "<- initialize\_mcnebula(",  
 ## initialize slot  
 i, "(x)", ", ",  
 ## other args  
 "sirius\_version = sirius\_version,",  
 "sirius\_project = sirius\_project",  
 ")")  
 eval( parse(text = express) )  
 }  
 export\_name(x) <- .get\_export\_name()  
 return(x)  
 })  
  
#' @exportMethod initialize\_mcnebula  
#'  
#' @aliases initialize\_mcnebula  
#'  
#' @seealso [ggsci::pal\_simpsons()], [ggsci::pal\_igv()], [ggsci::pal\_ucscgb()],  
#' [ggsci::pal\_d3()]...  
#'  
#' @rdname initialize\_mcnebula-methods  
#'  
setMethod("initialize\_mcnebula",   
 signature = c(x = "melody"),  
 function(x){  
 ## set color palette  
 palette\_set(x) <- .get\_color\_set()  
 palette\_gradient(x) <- .get\_color\_gradient()  
 palette\_stat(x) <- .get\_color\_stat()  
 palette\_col(x) <- .get\_color\_col()  
 palette\_label(x) <- .get\_label\_color()  
 return(x)  
 })  
  
#' @exportMethod initialize\_mcnebula  
#' @rdname initialize\_mcnebula-methods  
setMethod("initialize\_mcnebula",   
 signature = c(x = "project\_conformation",  
 sirius\_version = "character"),  
 function(x, sirius\_version){  
 slots <- names(attributes(x))  
 slots <- slots[-length(slots)]  
 for (i in slots) {  
 express <-   
 paste0( i, "(x)", "<-", ".get\_", i, "\_", sirius\_version, "()")  
 eval( parse(text = express) )  
 }  
 return(x)  
 })  
  
#' @exportMethod initialize\_mcnebula  
#' @rdname initialize\_mcnebula-methods  
setMethod("initialize\_mcnebula",   
 signature = c(x = "project\_api",  
 sirius\_version = "character"),  
 function(x, sirius\_version){  
 express <- paste0("function(x) format\_msframe(",  
 "x,",  
 "fun\_names = .get\_attribute\_name\_", sirius\_version, ",",  
 "fun\_types = .get\_attribute\_type\_", sirius\_version, "",  
 ")")  
 methods\_format(x) <- eval( parse(text = express) )  
 express <- paste0(".get\_methods\_read\_", sirius\_version, "()")  
 methods\_read(x) <- eval( parse(text = express) )  
 express <- paste0(".get\_methods\_match\_", sirius\_version, "()")  
 methods\_match(x) <- eval( parse(text = express) )  
 return(x)  
 })  
# ==========================================================================  
# extract and visualize 'ggset' in 'mcnebula' object  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @aliases visualize  
#'  
#' @title Visualize Nebulae in R graphic device  
#'  
#' @description  
#' Methods used for visualization.  
#' Show chemical Nebulae (either Parent-Nebula or Child-Nebulae) in R graphic device.  
#' Run after [activate\_nebulae()]  
#'  
#' @name visualize-methods  
#'  
#' @order 1  
NULL  
#> NULL  
  
#' @importFrom tibble tibble  
setClassUnion("numeric\_or\_character", c("numeric", "character"))  
  
#' @exportMethod visualize  
#' @description \code{visualize(x)}: get a 'tbl' about Child-Nebulae candidates  
#' for \code{visualize} methods to visualize.  
#' @rdname visualize-methods  
setMethod("visualize",   
 signature = setMissing("visualize",  
 x = "mcnebula",  
 fun\_modify = "ANY"),  
 function(x, fun\_modify){  
 .message\_info\_formal("MCnebula2", "visualize")  
 cat("\tSpecify item as following to visualize:\n\n")  
 class.name <- names(ggset(child\_nebulae(x)))  
 hierarchy <- vapply(class.name, function(c, h) h[[c]], 1,  
 h = .get\_hierarchy(x))  
 tibble::tibble(seq = 1:length(class.name),  
 hierarchy = hierarchy,  
 class.name = class.name  
 )  
 })  
  
#' @exportMethod visualize  
#' @description \code{visualize()}: get the default parameters for the method  
#' \code{visualize}.  
#' @rdname visualize-methods  
setMethod("visualize",   
 signature = setMissing("visualize"),  
 function(){  
 list(fun\_modify = modify\_set\_labs)  
 })  
  
#' @exportMethod visualize  
#' @description \code{visualize(x, ...)}: use the default parameters whatever 'missing'  
#' while performing the method \code{visualize}.  
#' @rdname visualize-methods  
setMethod("visualize",   
 signature = c(x = "mcnebula"),  
 function(x, item, fun\_modify, annotate){  
 reCallMethod("visualize", .fresh\_param(visualize()))  
 })  
  
#' @exportMethod visualize  
#'  
#' @aliases visualize  
#'  
#' @param item character(1) or numeric(1). If \code{character}, the value should be  
#' a name of chemical class in 'nebula\_index' data. Its Nebulae has been activated  
#' via [activate\_nebulae()]. If \code{numeric}, the value should be the sequence of   
#' Nebulae... Use \code{visualize(object)} to get the optional value.  
#'   
#' @param annotate logical. If \code{TRUE}, visualize the Nebula with the annotation.  
#' Only available [annotate\_nebula()] has been run for the Nebula.  
#'  
#' @rdname visualize-methods  
#'  
#' @examples  
#' \dontrun{  
#' test <- mcn\_5features  
#'   
#' ## the previous steps  
#' test1 <- filter\_structure(test)  
#' test1 <- create\_reference(test1)  
#' test1 <- filter\_formula(test1, by\_reference = T)  
#' test1 <- create\_stardust\_classes(test1)  
#' test1 <- create\_features\_annotation(test1)  
#' test1 <- cross\_filter\_stardust(test1, 2, 1)  
#' test1 <- create\_nebula\_index(test1)  
#' test1 <- compute\_spectral\_similarity(test1)  
#' test1 <- create\_parent\_nebula(test1, 0.01)  
#' test1 <- create\_child\_nebulae(test1, 0.01)  
#' test1 <- create\_parent\_layout(test1)  
#' test1 <- create\_child\_layouts(test1)  
#' test1 <- activate\_nebulae(test1)  
#'   
#' ## optional Child-Nebulae  
#' visualize(test1)  
#'   
#' visualize(test1, "parent")  
#' visualize(test1, 1)  
#' visualize\_all(test1)  
#' ## ...  
#'   
#' ## use 'fun\_modify'  
#' visualize(test1, 1, modify\_default\_child)  
#' visualize(test1, 1, modify\_unify\_scale\_limits)  
#' visualize(test1, 1, modify\_set\_labs)  
#' ## ...  
#' }  
setMethod("visualize",   
 signature = setMissing("visualize",  
 x = "mcnebula",  
 item = "character",  
 fun\_modify = "function"),  
 function(x, item, fun\_modify){  
 .message\_info\_formal("MCnebula2", "visualize")  
 if (item == "parent") {  
 call\_command(fun\_modify(ggset(parent\_nebula(x))))  
 } else {  
 obj <- ggset(child\_nebulae(x))[[ item ]]  
 if (!is.null(obj)) {  
 call\_command(fun\_modify(obj))  
 } else {  
 stop( "the `item` not found in `ggset(child\_nebula(x))`" )  
 }  
 }  
 })  
  
#' @exportMethod visualize  
#' @rdname visualize-methods  
setMethod("visualize",   
 signature = setMissing("visualize",  
 x = "mcnebula",  
 item = "numeric",  
 fun\_modify = "function"),  
 function(x, item, fun\_modify){  
 .message\_info\_formal("MCnebula2", "visualize")  
 call\_command(fun\_modify(ggset(child\_nebulae(x))[[ item ]]))  
 })  
  
#' @exportMethod visualize  
#' @rdname visualize-methods  
setMethod("visualize",   
 signature = setMissing("visualize",  
 x = "mcnebula",  
 item = "numeric\_or\_character",  
 fun\_modify = "function",  
 annotate = "logical"),  
 function(x, item, fun\_modify, annotate){  
 if (annotate) {  
 obj <- ggset\_annotate(child\_nebulae(x))[[ item ]]  
 if (is.null(obj)) {  
 stop( "the `item` not found in `ggset\_annotate(child\_nebula(x))`" )  
 } else {  
 call\_command(fun\_modify(obj))  
 }  
 } else {  
 visualize(x, item)  
 }  
 })  
  
#' @export get\_ggset  
#' @description \code{get\_ggset}: similar to \code{visualize(...)}, but get  
#' [ggset-class] object.  
#' @rdname visualize-methods  
get\_ggset <- function(x, item, fun\_modify, annotate = F) {  
 if (!annotate) {  
 fun\_modify(ggset(child\_nebulae(x))[[ item ]])  
 } else {  
 fun\_modify(ggset\_annotate(child\_nebulae(x))[[ item ]])  
 }  
}  
  
#' @exportMethod visualize\_all  
#' @description \code{visualize\_all()}: get the default parameters for the method  
#' \code{visualize\_all}.  
#' @rdname visualize-methods  
setMethod("visualize\_all",   
 signature = setMissing("visualize\_all",  
 x = "missing"),  
 function(){  
 list(newpage = T,  
 fun\_modify = modify\_default\_child,  
 legend\_hierarchy = T  
 )  
 })  
  
#' @exportMethod visualize\_all  
#' @description \code{visualize\_all(x, ...)}: use the default parameters whatever 'missing'  
#' while performing the method \code{visualize\_all}.  
#' @rdname visualize-methods  
setMethod("visualize\_all",   
 signature = c(x = "mcnebula"),  
 function(x, newpage, fun\_modify, legend\_hierarchy){  
 reCallMethod("visualize\_all",  
 .fresh\_param(visualize\_all()))  
 })  
  
#' @importFrom grid grid.newpage  
#' @importFrom grid viewport  
#' @importFrom grid pushViewport  
#' @importFrom grid upViewport  
#' @importFrom grid grid.draw  
#' @exportMethod visualize\_all  
#'  
#' @description \code{visualize\_all}: visualize overall Child-Nebulae into R graphic device.  
#'  
#' @param x [mcnebula-class] object.  
#' @param newpage logical. If \code{TRUE}, use [grid::grid.newpage()] before visualization.  
#' @param fun\_modify function. Used to post modify the [ggset-class] object before  
#' visualization. See [fun\_modify].  
#' @param legend\_hierarchy logical. If \code{TRUE}, visualize the legend of chemical hierarchy.  
#'  
#' @rdname visualize-methods  
#'  
setMethod("visualize\_all",   
 signature = setMissing("visualize\_all",  
 x = "mcnebula",  
 newpage = "logical",  
 fun\_modify = "function",  
 legend\_hierarchy = "logical"),  
 function(x, newpage, fun\_modify, legend\_hierarchy){  
 .message\_info\_formal("MCnebula2", "visualize\_all")  
 set <- child\_nebulae(x)  
 if (newpage)  
 grid::grid.newpage()  
 .message\_info\_viewport("BEGIN")  
 grid::pushViewport(panel\_viewport(set))  
 layer <- 1  
 .message\_info\_viewport()  
 if (legend\_hierarchy) {  
 .visualize\_legend\_hierarchy(set)  
 layer <- layer + 1  
 }  
 layer <- layer +  
 .visualize\_child\_nebulae(set, fun\_modify)  
 grid::upViewport(layer)  
 .message\_info\_viewport()  
 .visualize\_legend\_nebulae(set, fun\_modify)  
 .message\_info\_viewport("END")  
 })  
  
.visualize\_child\_nebulae <-   
 function(set, fun\_modify = modify\_default\_child, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 if (!is.null(grid\_layout(set))) {  
 grid::pushViewport(grid::viewport(layout = grid\_layout(set)))  
 layer <- 1  
 } else {  
 layer <- 0  
 }  
 lapply(names(ggset(set)),  
 function(name){  
 print(call\_command(fun\_modify(ggset(set)[[ name ]])),  
 vp = viewports(set)[[ name ]],  
 newpage = F)  
 })  
 return(layer)  
 }  
  
.visualize\_legend\_nebulae <-   
 function(set, fun\_modify = modify\_default\_child, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 grid::pushViewport(legend\_viewport(set))  
 .message\_info("visualize", "legend:",  
 paste0("\n\textract legend from ",  
 "`ggset(child\_nebulae(x))[[1]]` ",  
 "(nebula names:", names(ggset(set)[[1]]), ").",  
 "\n\tIn default, legend scales have been unified ",  
 "for all child-nebulae."  
 ))  
 ggset <- fun\_modify(ggset(set)[[1]])  
 if (!is.null(attr(ggset, "modify"))) {  
 ggset <- match.fun(attr(ggset, "modify"))(ggset)  
 }  
 grob <- .get\_legend(call\_command(ggset))  
 grid::grid.draw(grob)  
 }  
  
.visualize\_legend\_hierarchy <-   
 function(set, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 grob <- .legend\_hierarchy(set)  
 pushViewport(viewport(0.5, 0, 1, 0.1,  
 just = c("centre", "bottom"),  
 name = "legend\_hierarchy"))  
 .message\_info\_viewport()  
 grid::grid.draw(grob)  
 upViewport(1)  
 pushViewport(viewport(0.5, 0.1, 1, 0.9,  
 just = c("centre", "bottom"),  
 name = "sub\_panel"))  
 .message\_info\_viewport()  
 }  
  
.legend\_hierarchy <-   
 function(set, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 theme <- layers(ggset(set)[[1]])$theme  
 if (is.null(theme)) {  
 theme <- new\_command(match.fun("theme"), name = "theme")  
 }  
 class.names <- names(ggset(set))  
 .check\_data(x, list("hierarchy" = "create\_hierarchy"))  
 hierarchy <- .get\_hierarchy(x)  
 hierarchy <- vapply(class.names, function(name) hierarchy[[name]], 1)  
 color <- vapply(hierarchy, function(n) palette\_label(x)[[n]], "ch")  
 names(color) <- paste0("Level ", hierarchy)  
 .grob\_legend\_hierarchy\_plot(color, call\_command(theme))  
 }  
  
#' @export visualize\_ids  
#' @aliases visualize\_ids  
#' @description \code{visualize\_ids}: Plot a label map about the location of the 'features'.  
#' @rdname visualize-methods  
visualize\_ids <- function(x, item) {  
 data <- ggset(child\_nebulae(x))[[ item ]]  
 data <- command\_args(layers(data)[[1]])$graph  
 data <- dplyr::select(data, .features\_id = name, x, y)  
 ggplot(data) +  
 geom\_text(aes(x = x, y = y, label = .features\_id), family = .font) +  
 theme(text = element\_text(family = .font))  
}  
# ==========================================================================  
# directory and file names and path in SIRIUS 4 project, and some function  
# for how to read or format these data.  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
.validate\_sirius.v4 <-   
 function(path){  
 sig <- paste0(path, "/.format")  
 content <- "%source\_%name"  
 if (file.exists(sig)) {  
 if (!identical(readLines(sig, warn = F, n = 1), content)) {  
 stop("the content of file \"", sig,  
 "\" is not identical to \"", content, "\"")  
 }  
 }else{  
 stop("file \"", sig, "\" not exists")  
 }  
 }  
  
.get\_file\_name\_sirius.v4 <-   
 function(){  
 set <- c(.id = "FUN\_get\_id\_sirius.v4",  
 .canopus = "^canopus.tsv",  
 .canopus\_summary = "canopus\_summary.tsv",  
 .compound\_identifications = "compound\_identifications.tsv",  
 .formula\_identifications = "formula\_identifications.tsv",  
 .canopus\_neg = "canopus\_neg.tsv",  
 .csi\_fingerid = "csi\_fingerid.tsv",  
 .csi\_fingerid\_neg = "csi\_fingerid\_neg.tsv",  
 .dir\_canopus = "^canopus$",  
 .dir\_fingerid = "^fingerid$",  
 .dir\_scores = "^scores$",  
 .dir\_spectra = "^spectra$",  
 .f2\_ms = "spectrum.ms",  
 .f2\_msms = "spectrum.ms",  
 .f2\_info = "compound.info",  
 .f2\_formula = "formula\_candidates.tsv",  
 .f3\_canopus = "\\.fpt$",  
 .f3\_fingerid = "\\.tsv$",  
 .f3\_scores = "\\.info$",  
 .f3\_spectra = "\\.tsv$"  
 )  
 }  
  
FUN\_get\_id\_sirius.v4 <-   
 function(x){  
 if (missing(x))  
 return("^[0-9](.\*)\_(.\*)\_(.\*)$")  
 stringr::str\_extract(x, "(?<=\_)[^\_|^/]{1,}(?=/|$)")  
 }  
  
.get\_file\_api\_sirius.v4 <-   
 function(){  
 set <- c(.id = ".id",  
 .canopus = ".canopus",  
 .canopus\_summary = ".canopus\_summary",  
 .compound\_identifications = ".compound\_identifications",  
 .formula\_identifications = ".formula\_identifications",  
 .canopus\_neg = ".canopus\_neg",  
 .csi\_fingerid = ".csi\_fingerid",  
 .csi\_fingerid\_neg = ".csi\_fingerid\_neg",  
 .dir\_canopus = ".id/.dir\_canopus",  
 .dir\_fingerid = ".id/.dir\_fingerid",  
 .dir\_scores = ".id/.dir\_scores",  
 .dir\_spectra = ".id/.dir\_spectra",  
 .f2\_ms = ".id/.f2\_ms",  
 .f2\_msms = ".id/.f2\_msms",  
 .f2\_info = ".id/.f2\_info",  
 .f2\_formula = ".id/.f2\_formula",  
 .f3\_canopus = ".id/.dir\_canopus/.f3\_canopus",  
 .f3\_fingerid = ".id/.dir\_fingerid/.f3\_fingerid",  
 .f3\_scores = ".id/.dir\_scores/.f3\_scores",  
 .f3\_spectra = ".id/.dir\_spectra/.f3\_spectra"  
 )  
 }  
  
.get\_attribute\_name\_sirius.v4 <-   
 function(){  
 set <- c(  
 ## .f3\_fingerid  
 ...sig = ".f3\_fingerid",  
 inchikey2d = "inchikey2D",  
 inchi = "inchi",  
 mol.formula = "molecularFormula",  
 rank.structure = "rank",  
 csi.score = "score",  
 synonym = "name",  
 smiles = "smiles",  
 xlogp = "xlogp",  
 pubmed.ids = "PubMedIds",  
 links = "links",  
 tani.score = "tanimotoSimilarity",  
 dbflags = "dbflags",  
 ## .f3\_spectra  
 ...sig = ".f3\_spectra",  
 mz = "mz",  
 int. = "intensity",  
 rel.int. = "rel.intensity",  
 exactmass = "exactmass",  
 formula = "formula",  
 ion. = "ionization",  
 ## .f2\_formula  
 ...sig = ".f2\_formula",  
 adduct = "adduct",  
 pre.formula = "precursorFormula",  
 zodiac.score = "ZodiacScore",  
 sirius.score = "SiriusScore",  
 tree.score = "TreeScore",  
 iso.score = "IsotopeScore",  
 hit.num. = "numExplainedPeaks",  
 hit.int. = "explainedIntensity",  
 error.frag. = "medianMassErrorFragmentPeaks\\(ppm\\)",  
 error.abs.frag. = "medianAbsoluteMassErrorFragmentPeaks\\(ppm\\)",  
 error.mass = "massErrorPrecursor\\(ppm\\)",  
 rank.formula = "rank",  
 ## .f2\_info  
 ...sig = ".f2\_info",  
 rt.secound = "rt",  
 mz = "ionMass",  
 ## .canopus  
 ...sig = ".canopus",  
 rel.index = "relativeIndex",  
 abs.index = "absoluteIndex",  
 chem.ont.id = "id",  
 class.name = "name",  
 parent.chem.ont.id = "parentId",  
 description = "description",  
 ## .canopus\_neg  
 ...sig = ".canopus\_neg",  
 chem.ont.id = "id",  
 class.name = "name",  
 ## .canopus\_summary  
 ...sig = ".canopus\_summary",  
 .id = "name",  
 most.sp.class = "most specific class",  
 level5 = "level 5",  
 subclass = "subclass",  
 class = "class",  
 superclass = "superclass",  
 all.class = "all classifications",  
 ## .compound\_identifications  
 ...sig = ".compound\_identifications",  
 cosmic.score = "ConfidenceScore",  
 .id = "id",  
 ## .f3\_canopus  
 ...sig = ".f3\_canopus",  
 pp.value = "V1",  
 ...sig = "END"  
 )  
 }  
  
.get\_attribute\_type\_sirius.v4 <-   
 function(){  
 set <- c(  
 rank.formula = "integer",  
 rank.structure = "integer",  
 csi.score = "numeric",  
 xlogp = "numeric",  
 tani.score = "numeric",  
 mz = "numeric",  
 rt.secound = "numeric",  
 rt.min = "numeric",  
 int. = "numeric",  
 rel.int. = "numeric",  
 exactmass = "numeric",  
 zodiac.score = "numeric",  
 sirius.score = "numeric",  
 tree.score = "numeric",  
 iso.score = "numeric",  
 hit.num. = "integer",  
 hit.int. = "numeric",  
 error.frag. = "numeric",  
 error.abs.frag. = "numeric",  
 error.mass = "numeric",  
 rel.index = "integer",  
 abs.index = "integer",  
 cosmic.score = "numeric",  
 pp.value = "numeric"  
 )  
 }  
  
.get\_methods\_read\_sirius.v4 <-   
 function(){  
 set <- c(  
 read.canopus = read\_tsv,  
 read.canopus\_summary = read\_tsv,  
 read.compound\_identifications = read\_tsv,  
 read.formula\_identifications = read\_tsv,  
 read.f2\_ms = pbsapply\_read\_tsv,  
 read.f2\_msms = pbsapply\_read\_msms,  
 read.f2\_formula = pbsapply\_read\_tsv,  
 read.f2\_info = pbsapply\_read\_info,  
 read.f3\_fingerid = pbsapply\_read\_tsv,  
 read.f3\_scores = pbsapply\_read\_tsv,  
 read.f3\_spectra = pbsapply\_read\_tsv,  
 read.f3\_canopus = .pbsapply\_read\_fpt  
 )  
 }  
  
list\_files\_top.sirius.v4 <- function(path, pattern){  
 data.frame(files = list.files(path = path, pattern = pattern))  
}  
  
list\_files.sirius.v4 <- function(path, upper, pattern, ...){  
 lst\_file <- pbapply::pbmapply(path, upper, pattern, SIMPLIFY = F,  
 FUN = function(path, upper, pattern){  
 files <- list.files(paste0(path, "/", upper), pattern)  
 if ( length(files) == 0)  
 return( data.frame() )  
 data.frame(upper = upper, files = files)  
 })  
 data.table::rbindlist(lst\_file)  
}  
  
pbsapply\_read\_msms <- function(path){  
 pbapply::pbsapply(path, simplify = F,  
 function(path){  
 lines <- readLines(path)  
 start <- grep("^>ms2peaks", lines) + 1  
 if (length(start) != 0) {  
 lines <- lines[start:length(lines)]  
 data <- data.table::fread(text = lines)  
 colnames(data) <- c("mz", "int.")  
 } else {  
 data <- data.frame(mz = double(0), int. = double(0))  
 }  
 data  
 }  
 )  
}  
  
pbsapply\_read\_info <- function(path){  
 pbapply::pbsapply(path, simplify = F,  
 function(path){  
 lines <- readLines(path)  
 lines <- lines[grepl("^ionMass|^rt", lines)]  
 data.frame(ionMass =  
 stringr::str\_extract(lines[1], "[0-9|.]{1,}"),  
 rt = stringr::str\_extract(lines[2], "[0-9|.]{1,}")  
 )  
 })  
}  
  
.pbsapply\_read\_fpt <- function(path){  
 pbapply::pbsapply(path, simplify = F,  
 function(path){  
 df <- data.table::fread(path, header = F)  
 df$rel.index <- 0:(nrow(df) - 1)  
 df  
 })  
}  
  
.get\_methods\_match\_sirius.v4 <-   
 function(){  
 set <- c(  
 match.features\_id = FUN\_get\_id\_sirius.v4,  
 match.candidates\_id = function(x) stringr::str\_extract(x, "[^/]\*(?=\\.[a-z]\*$)"),  
 generate\_candidates\_id = function(df) {  
 if (is.null(df$pre.formula) | is.null(df$adduct))  
 stop( "columns not found in `df`" )  
 paste0(df$pre.formula, "\_", gsub(" ", "", df$adduct))  
 }  
 )  
 }  
  
# ==========================================================================  
# directory and file names and path in SIRIUS 4 project, and some function  
# for how to read or format these data.  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
  
.validate\_sirius.v5 <-   
 function(path){  
 sig <- paste0(path, "/.format")  
 content <- "%source\_%name"  
 if (file.exists(sig)) {  
 if (!identical(readLines(sig, warn = F, n = 1), content)) {  
 stop("the content of file \"", sig,  
 "\" is not identical to \"", content, "\"")  
 }  
 }else{  
 stop("file \"", sig, "\" not exists")  
 }  
 sig <- paste0(path, "/.compression")  
 if (file.exists(sig)) {  
 lines <- readLines(sig, warn = F)  
 if (lines[1] != "compressionLevels\t1" |  
 lines[2] != "compressionMethod\tDEFLATED")  
 stop("file \"", sig, "\": Inappropriate compression method.")  
 }  
 }  
  
  
.get\_file\_name\_sirius.v5 <-   
 function(){  
 set <- c(.id = "FUN\_get\_id\_sirius.v5",  
 .canopus = "^canopus.tsv",  
 .canopus\_summary = "canopus\_compound\_summary.tsv",  
 .compound\_identifications = "compound\_identifications.tsv",  
 .formula\_identifications = "formula\_identifications.tsv",  
 .canopus\_neg = "canopus\_neg.tsv",  
 .csi\_fingerid = "csi\_fingerid.tsv",  
 .csi\_fingerid\_neg = "csi\_fingerid\_neg.tsv",  
 .zip\_canopus = "^canopus$",  
 .zip\_fingerid = "^fingerid$",  
 .zip\_scores = "^scores$",  
 .zip\_spectra = "^spectra$",  
 .f2\_ms = "spectrum.ms",  
 .f2\_msms = "spectrum.ms",  
 .f2\_info = "compound.info",  
 .f2\_formula = "formula\_candidates.tsv",  
 .f3\_canopus = "\\.fpt$",  
 .f3\_fingerid = "\\.tsv$",  
 .f3\_scores = "\\.info$",  
 .f3\_spectra = "\\.tsv$"  
 )  
 }  
  
FUN\_get\_id\_sirius.v5 <-FUN\_get\_id\_sirius.v4  
  
.get\_file\_api\_sirius.v5 <- function(){  
 set <- c(.id = ".id",  
 .canopus = ".canopus",  
 .canopus\_summary = ".canopus\_summary",  
 .compound\_identifications = ".compound\_identifications",  
 .formula\_identifications = ".formula\_identifications",  
 .canopus\_neg = ".canopus\_neg",  
 .csi\_fingerid = ".csi\_fingerid",  
 .csi\_fingerid\_neg = ".csi\_fingerid\_neg",  
 .zip\_canopus = ".id/.zip\_canopus",  
 .zip\_fingerid = ".id/.zip\_fingerid",  
 .zip\_scores = ".id/.zip\_scores",  
 .zip\_spectra = ".id/.zip\_spectra",  
 .f2\_ms = ".id/.f2\_ms",  
 .f2\_msms = ".id/.f2\_msms",  
 .f2\_info = ".id/.f2\_info",  
 .f2\_formula = ".id/.f2\_formula",  
 .f3\_canopus = ".id/.zip\_canopus/.f3\_canopus",  
 .f3\_fingerid = ".id/.zip\_fingerid/.f3\_fingerid",  
 .f3\_scores = ".id/.zip\_scores/.f3\_scores",  
 .f3\_spectra = ".id/.zip\_spectra/.f3\_spectra"  
 )  
}  
  
.get\_attribute\_name\_sirius.v5 <-   
 function(){  
 set <- c(  
 ## .f3\_fingerid  
 ...sig = ".f3\_fingerid",  
 inchikey2d = "inchikey2D",  
 inchi = "inchi",  
 mol.formula = "molecularFormula",  
 rank.structure = "rank",  
 csi.score = "score",  
 synonym = "name",  
 smiles = "smiles",  
 xlogp = "xlogp",  
 pubmed.ids = "PubMedIds",  
 links = "links",  
 tani.score = "tanimotoSimilarity",  
 dbflags = "dbflags",  
 ## .f3\_spectra  
 ...sig = ".f3\_spectra",  
 mz = "mz",  
 int. = "intensity",  
 rel.int. = "rel.intensity",  
 exactmass = "exactmass",  
 formula = "formula",  
 ion. = "ionization",  
 ## .f2\_formula  
 ...sig = ".f2\_formula",  
 adduct = "adduct",  
 pre.formula = "precursorFormula",  
 zodiac.score = "ZodiacScore",  
 sirius.score = "SiriusScore",  
 tree.score = "TreeScore",  
 iso.score = "IsotopeScore",  
 hit.num. = "numExplainedPeaks",  
 hit.int. = "explainedIntensity",  
 error.frag. = "medianMassErrorFragmentPeaks\\(ppm\\)",  
 error.abs.frag. = "medianAbsoluteMassErrorFragmentPeaks\\(ppm\\)",  
 error.mass = "massErrorPrecursor\\(ppm\\)",  
 rank.formula = "rank",  
 ## .f2\_info  
 ...sig = ".f2\_info",  
 rt.secound = "rt",  
 mz = "ionMass",  
 ## .canopus  
 ...sig = ".canopus",  
 rel.index = "relativeIndex",  
 abs.index = "absoluteIndex",  
 chem.ont.id = "id",  
 class.name = "name",  
 parent.chem.ont.id = "parentId",  
 description = "description",  
 ## .canopus\_neg  
 ...sig = ".canopus\_neg",  
 chem.ont.id = "id",  
 class.name = "name",  
 ## .canopus\_summary  
 ...sig = ".canopus\_summary",  
 .id = "id",  
 npc\_pathway = "NPC#pathway",  
 npc\_pathway\_pp = "NPC#pathway Probability",  
 npc\_superclass = "NPC#superclass",  
 npc\_superclass\_pp = "NPC#superclass Probability",  
 npc\_class = "NPC#class",  
 npc\_class\_pp = "NPC#class Probability",  
 classyfire\_most\_specific\_class = "ClassyFire#most specific class",  
 classyfire\_most\_specific\_class\_pp = "ClassyFire#most specific class Probability",  
 classyfire\_level\_5 = "ClassyFire#level 5",  
 classyfire\_level\_5\_pp = "ClassyFire#level 5 Probability",  
 classyfire\_subclass = "ClassyFire#subclass",  
 classyfire\_subclass\_pp = "ClassyFire#subclass Probability",  
 classyfire\_class = "ClassyFire#class",  
 classyfire\_class\_pp = "ClassyFire#class Probability",  
 classyfire\_superclass = "ClassyFire#superclass",  
 classyfire\_superclass\_pp = "ClassyFire#superclass probability",  
 classyfire\_all\_classifications = "ClassyFire#all classifications",  
 ## .compound\_identifications  
 ...sig = ".compound\_identifications",  
 cosmic.score = "ConfidenceScore",  
 .id = "id",  
 ## .f3\_canopus  
 ...sig = ".f3\_canopus",  
 pp.value = "V1",  
 ...sig = "END"  
 )  
 }  
  
.get\_attribute\_type\_sirius.v5 <- .get\_attribute\_type\_sirius.v4  
  
list\_files\_top.sirius.v5 <- list\_files\_top.sirius.v4  
  
#' @importFrom utils unzip  
list\_files.sirius.v5 <- function(path, upper, pattern, info){  
 lst\_file <- pbapply::pbmapply(path, upper, pattern, SIMPLIFY = F,  
 FUN = function(path, upper, pattern){  
 if (grepl("^\\.zip\_", info)) {  
 res <- try(utils::unzip(paste0(path, "/", upper), list = T), silent = T)  
 if (!inherits(res, "try-error")) {  
 files <- res$Name  
 files <- files[ grepl(pattern, files) ]  
 } else {  
 files <- integer(0)  
 }  
 } else {  
 files <- list.files(paste0(path, "/", upper), pattern)  
 }  
 if ( length(files) == 0)  
 return( data.frame() )  
 data.frame(upper = upper, files = files)  
 })  
 data.table::rbindlist(lst\_file)  
}  
  
.get\_methods\_read\_sirius.v5 <-   
 function(){  
 set <- c(  
 read.canopus = read\_tsv,  
 read.canopus\_summary = read\_tsv,  
 read.compound\_identifications = read\_tsv,  
 read.formula\_identifications = read\_tsv,  
 read.f2\_ms = pbsapply\_read\_tsv,  
 read.f2\_msms = pbsapply\_read\_msms,  
 read.f2\_formula = pbsapply\_read\_tsv,  
 read.f2\_info = pbsapply\_read\_info,  
 read.f3\_fingerid = pblapply\_read\_tsv\_fromZip,  
 read.f3\_scores = pblapply\_read\_tsv\_fromZip,  
 read.f3\_spectra = pblapply\_read\_tsv\_fromZip,  
 read.f3\_canopus = .pblapply\_read\_fpt\_fromZip  
 )  
 }  
  
.pblapply\_read\_fpt\_fromZip <- function(path) {  
 pblapply\_read\_tsv\_fromZip(path,  
 function(path) {  
 df <- data.table::fread(path, header = F)  
 df$rel.index <- 0:(nrow(df) - 1)  
 df  
 })  
}  
  
pblapply\_read\_tsv\_fromZip <- function(path, fun = read\_tsv) {  
 zips <- gsub("/[^/]\*$", "", path)  
 files <- stringr::str\_extract(path, "[^/]\*$")  
 lst\_files <- split(files, zips)  
 zips <- unique(zips)  
 lst\_files <- lapply(zips, function(name) lst\_files[[ name ]])  
 zip\_upper <- gsub("/[^/]\*$", "", zips)  
 zip\_name <- stringr::str\_extract(zips, "[^/]\*$")  
 exdir <- paste0(zip\_upper, "/.temp\_", zip\_name)  
 lst <- pbapply::pblapply(1:length(lst\_files),  
 function(n) {  
 utils::unzip(zips[n], exdir = exdir[n])  
 files <- paste0(exdir[n], "/", lst\_files[[ n ]])  
 lst <- lapply(files, fun)  
 unlink(exdir[n], T)  
 return(lst)  
 })  
 lst <- unlist(lst, F)  
 names(lst) <- paste0(zips, "/", unlist(lst\_files))  
 return(lst)  
}  
  
.get\_methods\_match\_sirius.v5 <- .get\_methods\_match\_sirius.v4  
# ==========================================================================  
# Get hexadecimal color with ggsci package  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @importFrom ggsci pal\_simpsons  
#' @importFrom ggsci pal\_igv  
#' @importFrom ggsci pal\_ucscgb  
#' @importFrom ggsci pal\_d3  
.get\_color\_set <-   
 function(){  
 unique(c(rev(ggsci::pal\_d3("category20")(20))[-3],  
 ggsci::pal\_simpsons()(16)[-3],  
 ggsci::pal\_ucscgb()(6)  
 ))  
 }  
  
.get\_color\_col <-   
 function(){  
 unique(c(ggsci::pal\_simpsons()(16),  
 ggsci::pal\_igv("default")(51),  
 ggsci::pal\_ucscgb()(26),  
 ggsci::pal\_d3("category20")(20)  
 ))  
 }  
  
.get\_color\_gradient <-   
 function(){  
 c("#D5E4A2FF", "#FFCD00FF", "#EEA236FF", "#FB6467FF", "#9467BDFF")  
 }  
  
.get\_label\_color <-   
 function(){  
 colorRampPalette(c("#C6DBEFFF", "#3182BDFF", "red"))(10)  
 }  
  
#' @importFrom ggsci pal\_locuszoom  
.get\_color\_stat <-   
 function(){  
 col <- ggsci::pal\_locuszoom()(7)  
 vapply(col, .depigment\_col, "ch", USE.NAMES = F)  
 }  
# ==========================================================================  
# functions to get 'command' of ggplot, grob for visualizing nebulae  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @importFrom ggraph geom\_edge\_fan  
#' @importFrom ggraph geom\_node\_point  
#' @importFrom ggraph scale\_edge\_width  
#' @importFrom stringr str\_wrap  
#' @importFrom stringr str\_wrap  
.command\_parent\_edge <- function(edge\_color = "lightblue"){  
 new\_command(ggraph::geom\_edge\_fan,  
 aes(edge\_width = similarity),  
 color = edge\_color  
 )  
}  
  
.command\_parent\_node <- function(){  
 new\_command(ggraph::geom\_node\_point,  
 aes(size = ifelse(is.na(tani.score), 0.2, tani.score),  
 fill = mz),  
 shape = 21)  
}  
  
.command\_parent\_fill <- function(pal){  
 new\_command(scale\_fill\_gradientn, colours = pal, na.value = "white")  
}  
  
.command\_parent\_fill2 <- function(pal){  
 new\_command(scale\_fill\_manual, values = pal)  
}  
  
.command\_parent\_labs <- function(){  
 new\_command(labs, fill = "m/z", size = "Tanimoto similarity",  
 edge\_width = "Spectral similarity")  
}  
  
.command\_parent\_edge\_width <- function(){  
 new\_command(scale\_edge\_width, range = c(0, 0.7))  
}  
  
.command\_scale\_x <- function(data, factor = 1.05){  
 new\_command(scale\_x\_continuous, limits = zoRange(data$x, factor))  
}  
  
.command\_scale\_y <- function(data, factor = 1.05){  
 new\_command(scale\_y\_continuous, limits = zoRange(data$y, factor))  
}  
  
.command\_parent\_theme <- function(){  
 new\_command(match.fun(theme),  
 text = element\_text(family = .font, face = "bold"),  
 axis.ticks = element\_blank(),  
 axis.text = element\_blank(),  
 axis.title = element\_blank(),  
 panel.grid = element\_blank(),  
 panel.background = element\_rect(fill = "white"),  
 legend.background = element\_rect(fill = "transparent"),  
 name = "theme"  
 )  
}  
  
.command\_child\_title <-  
 function(title){  
 new\_command(ggtitle, stringr::str\_wrap(title, width = 30))  
 }  
  
.command\_child\_theme <-   
 function(fill){  
 command <- .command\_parent\_theme()  
 command\_args(command)[[ "plot.title" ]] <-  
 call\_command(.command\_title\_textbox(fill))  
 command  
 }  
  
.command\_title\_textbox <-   
 function(fill){  
 new\_command(.element\_textbox, fill = fill)  
 }  
  
.command\_node\_nuclear <-   
 function(color){  
 new\_command(geom\_ribbon, fill = color,  
 aes(ymin = -5L, ymax = 0L,  
 x = seq(0, max(seq) + 1, length.out = length(seq)))  
 )  
 }  
  
.command\_node\_border <-   
 function(){  
 new\_command(geom\_ribbon, fill = "black",  
 aes(ymin = 0, ymax = 1.1,  
 x = seq(0, max(seq) + 1, length.out = length(seq)))  
 )  
 }  
  
.command\_node\_radial\_bar <-   
 function(){  
 new\_command(geom\_col, aes(x = seq, y = pp.value,  
 fill = reorder(paste0(rel.index), rel.index)),  
 color = "white", size = 0.25)  
 }  
  
.command\_node\_fill <-   
 function(pal, labels){  
 new\_command(scale\_fill\_manual, values = pal, labels = labels)  
 }  
  
.command\_node\_ylim <-   
 function(){  
 new\_command(ylim, ... = c(-5, 1.3))  
 }  
  
.command\_node\_polar <-   
 function(){  
 new\_command(coord\_polar)  
 }  
  
.command\_node\_theme <-   
 function(){  
 new\_command(match.fun(theme),  
 text = element\_text(family = .font, face = "bold"),  
 name = "theme")  
 }  
  
.command\_node\_ration <-   
 function(df){  
 new\_command(geom\_tile, data = df, size = 0.2, color = "white",  
 aes(y = -2.5, x = x, width = width,  
 height = 2.5, fill = group))  
 }  
  
#' @importFrom ggimage geom\_subview  
.command\_node\_annotate <-   
 function(data, subview){  
 new\_command(ggimage::geom\_subview, data = data,   
 aes(x = x, y = y, width = size, height = size),  
 subview = subview)  
 }  
  
  
.grob\_legend\_hierarchy\_plot <-   
 function(color, theme){  
 df <- data.frame(h = names(color), color = color, y = 1:length(color))  
 p <- ggplot(df) +  
 geom\_tile(aes(x = 1, y = h, fill = h)) +  
 labs(fill = "Class hierarchy") +  
 scale\_fill\_manual(values = color) +  
 guides(fill = guide\_legend(nrow = 1, direction = "horizontal")) +  
 theme  
 .get\_legend(p)  
 }  
  
.grob\_node\_text <-   
 function(label, color = "black"){  
 grid::textGrob(label, y = 0.12,  
 gp = grid::gpar(fontfamily = .font,  
 fontsize = 20, col = color))  
 }  
# ==========================================================================  
# functions to get export setting  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
.get\_export\_name <-   
 function(){  
 set <- c(  
 mz = "m/z",  
 pre.mz = "Precursor m/z",  
 rt.min = "RT (min)",  
 similarity = "Spectral similarity",  
 tani.score = "Tanimoto similarity",  
 rel.index = "Relative index",  
 rel.int. = "Relative intensity",  
 tracer = "Tracer",  
 group = "Group",  
 .features\_id = "ID",  
 mol.formula = "Formula",  
 inchikey2d = "InChIKey planar",  
 error.mass = "Mass error (ppm)",  
 synonym = "Synonym",  
 adduct = "Adduct"  
 )  
 }  
# ==========================================================================  
# algorithmic functions used in methods-\*.R files  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
.rank\_by\_csi.score <-   
 function(df){  
 head( dplyr::arrange(df, desc(csi.score)), n = 1)  
 }  
  
.rank\_by\_default <-   
 function(df){  
 head(df, n = 1)  
 }  
  
.filter\_ppcp\_by\_threshold <-   
 function(df, pp.threshold = 0.5){  
 dplyr::filter(df, pp.value > pp.threshold)  
 }  
  
.decrease\_edges <-  
 function(edges, max\_edge\_number = 5){  
 ## order  
 edges <- edges[order(edges$similarity, decreasing = T), ]  
 edges[[ "...SEQ" ]] <- 1:nrow(edges)  
 freq <- table(c(edges[[ ".features\_id1" ]], edges[[ ".features\_id2" ]]))  
 ## at least loop number  
 while (max(freq) > max\_edge\_number) {  
 target\_id <- names(freq[freq == max(freq)])[1]  
 ## get ...SEQ of the edges which need to be excluded  
 include <- edges[[ ".features\_id1" ]] == target\_id |  
 edges[[ ".features\_id2" ]] == target\_id  
 edges\_include\_target <- edges[include, ]  
 seq\_exclude\_edges <- edges\_include\_target[-(1:max\_edge\_number), ]$...SEQ  
 ## exclude edges  
 edges <- edges[!edges$...SEQ %in% seq\_exclude\_edges, ]  
 freq <- table(c(edges[[ ".features\_id1" ]], edges[[ ".features\_id2" ]]))  
 }  
 edges[[ "...SEQ" ]] <- NULL  
 edges  
 }  
# ==========================================================================  
# functions to modify 'ggset' object  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @aliases fun\_modify  
#'  
#' @title Modify 'ggset' object  
#'  
#' @description  
#' These are multiple functions used for post modification of [ggset-class]  
#' object. These functions provide a convenient, fast, and repeatable way  
#' to make improvements to [ggset-class] object.  
#'  
#' @param ggset [ggset-class] object.  
#' @param x [mcnebula-class] object.  
#'   
#' @seealso [ggset-class]  
#'  
#' @name fun\_modify  
NULL  
#> NULL  
  
#' @export modify\_default\_child  
#' @aliases modify\_default\_child  
#'   
#' @description \code{modify\_default\_child}:  
#' Used for \code{visualize\_all()}.  
#' \code{modify\_rm\_legend} + \code{modify\_set\_labs} + \code{modify\_unify\_scale\_limits}.  
#' In addition, if the 'use\_tracer' is TRUE (see [set\_nodes\_color()]),  
#' \code{modify\_tracer\_node} and \code{modify\_color\_edge} would be performed.  
#'   
#' @rdname fun\_modify  
modify\_default\_child <-   
 function(ggset, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 maps <- .get\_mapping2(ggset)  
 if (maps[[ "fill" ]] == "tracer")  
 ggset <- modify\_color\_edge(modify\_tracer\_node(ggset), "lightblue")  
 modify\_rm\_legend(modify\_set\_labs(modify\_unify\_scale\_limits(ggset)))  
 }  
  
#' @export modify\_stat\_child  
#' @aliases modify\_stat\_child  
#'   
#' @description \code{modify\_stat\_child}:  
#' Repalce [scale\_fill\_gradientn()] with [scale\_fill\_gradient2()] in 'layers';  
#' unify the "aes" scale except for "fill";  
#' perfrom [modify\_set\_labs()];  
#' only keep the legend for 'fill', and adjust its width;  
#' move the position of the legend to the bottom;  
#' remove the title of the legend.  
#'   
#' @rdname fun\_modify  
modify\_stat\_child <-   
 function(ggset, x) {  
 x <- .get\_missing\_x(x, "mcnebula")  
 ## replace  
 seq <- grep("scale\_fill\_gradientn", names(layers(ggset)))  
 if (is.integer(seq) & length(seq) > 0)  
 ggset <- delete\_layers(ggset, seq)  
 args <- list(low = "blue", mid = "grey90", high = "red", na.value = "white")  
 pal <- palette\_gradient(x)  
 pal <- pal[names(pal) %in% names(args)]  
 args <- .fresh\_param(args, as.list(pal))  
 breaks <- function(x) round(seq(floor(min(x)), ceiling(max(x)), length.out = 7), 1)  
 command <- do.call(new\_command, c(fun = scale\_fill\_gradient2,  
 breaks = breaks, args,  
 name = "scale\_fill\_gradient2"))  
 ggset <- add\_layers(ggset, command)  
 ## unify and set labs  
 aes\_name <- names(.get\_mapping2(ggset))  
 ggset <- modify\_unify\_scale\_limits(ggset, aes\_name = aes\_name[aes\_name != "fill"])  
 ggset <- modify\_set\_labs(ggset)  
 ## ...  
 args <- sapply(aes\_name, simplify = F,  
 function(name) {  
 if (name == "fill")  
 guide\_colorbar(title = NULL, barheight = grid::unit(.5, "line"))  
 else "none"  
 })  
 if (any(grepl("^guides|ggplot2::guides", names(layers(ggset)))))  
 ggset <- do.call(mutate\_layer, c(list(x = ggset, layer = "guides"), args))  
 else {  
 command <- do.call(new\_command,  
 c(fun = match.fun("guides"), args, name = "guides"))  
 ggset <- add\_layers(ggset, command)  
 }  
 ggset <- mutate\_layer(ggset, "theme", legend.position = "bottom")  
 attr(ggset, "modify") <- "rev.modify\_stat\_child"  
 ggset  
 }  
  
rev.modify\_stat\_child <-   
 function(ggset){  
 args <- sapply(names(.get\_mapping2(ggset)), simplify = F,  
 function(name) {  
 if (name == "fill") "none" else NULL  
 })  
 ggset <- do.call(mutate\_layer, c(list(x = ggset, layer = "guides"), args))  
 ggset <- mutate\_layer(ggset, "theme", legend.position = "right")  
 ggset  
 }  
  
#' @export modify\_set\_labs\_and\_unify\_scale\_limits  
#' @aliases modify\_set\_labs\_and\_unify\_scale\_limits  
#'   
#' @description \code{modify\_set\_labs\_and\_unify\_scale\_limits}:  
#' \code{modify\_set\_labs} + \code{modify\_unify\_scale\_limits}  
#'   
#' @rdname fun\_modify  
modify\_set\_labs\_and\_unify\_scale\_limits <-   
 function(ggset, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 modify\_set\_labs(modify\_unify\_scale\_limits(ggset))  
 }  
  
#' @export modify\_annotate\_child  
#' @aliases modify\_annotate\_child  
#'   
#' @description \code{modify\_annotate\_child}:  
#' \code{modify\_set\_labs} + ...  
#' (for parameters of \code{panel.grid} and \code{panel.background}  
#' in [ggplot2::theme()]).  
#'   
#' @rdname fun\_modify  
modify\_annotate\_child <-   
 function(ggset, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 mutate\_layer(modify\_set\_labs(ggset), "theme",  
 panel.grid = element\_line("white", inherit.blank = T),  
 panel.background = element\_rect("grey92", color = NA,  
 inherit.blank = T))  
 }  
  
#' @export modify\_rm\_legend  
#' @aliases modify\_rm\_legend  
#'   
#' @description \code{modify\_rm\_legend}: remove the legend.  
#' For parameter of \code{legend.position} in [ggplot2::theme()].  
#'   
#' @rdname fun\_modify  
modify\_rm\_legend <-   
 function(ggset){  
 mutate\_layer(ggset, "theme", legend.position = "none")  
 }  
  
#' @export modify\_tracer\_node  
#' @aliases modify\_tracer\_node  
#' @description \code{modify\_tracer\_node}: Set the stroke for nodes in  
#' Nebulae (network) as 0, and the color as 'transparent';  
#' Override the node color (border color) in legend.  
#' @rdname fun\_modify  
modify\_tracer\_node <-   
 function(ggset){  
 seq <- grep("geom\_node\_point", names(layers(ggset)))  
 ggset <- mutate\_layer(ggset, seq, stroke = 0, color = "transparent")  
 ## override the nodes boder color in legend   
 seq <- grep("^guides|ggplot2::guides", names(layers(ggset)))  
 size\_legend <- guide\_legend(override.aes = list(stroke = .3, color = "black"))  
 fill\_legend <- guide\_legend(override.aes = list(size = 4))  
 if (length(seq) > 0)  
 ggset <- mutate\_layer(ggset, seq, size = size\_legend, fill = fill\_legend)  
 else {  
 command <- new\_command(match.fun("guides"), size = size\_legend,  
 fill = fill\_legend, name = "guides")  
 ggset <- add\_layers(ggset, command)  
 }  
 }  
  
#' @export modify\_color\_edge  
#' @aliases modify\_color\_edge  
#' @description \code{modify\_color\_edge}: Set color for edge.  
#' @param color character(1).  
#' @rdname fun\_modify  
modify\_color\_edge <-   
 function(ggset, color){  
 seq <- grep("geom\_edge\_", names(layers(ggset)))  
 mutate\_layer(ggset, seq, color = color)  
 }  
  
#' @importFrom grid unit  
#' @export modify\_set\_margin  
#' @aliases modify\_set\_margin  
#'   
#' @description \code{modify\_set\_margin}: reduce margin.  
#' For parameter of \code{plot.margin} in [ggplot2::theme()].  
#'   
#' @rdname fun\_modify  
modify\_set\_margin <-   
 function(ggset, margin = grid::unit(rep(-8, 4), "lines")){  
 mutate\_layer(ggset, "theme", plot.margin = margin)  
 }  
  
#' @export modify\_unify\_scale\_limits  
#' @aliases modify\_unify\_scale\_limits  
#'   
#' @description \code{modify\_unify\_scale\_limits}:  
#' Uniform mapping 'scale' for all Child-Nebulae.  
#' Related to \code{ggplot2::scale\_\*} function.  
#' Use \code{MCnebula2:::.LEGEND\_mapping()} to get the possibly mapping.  
#'  
#' @param aes\_name character. Specify which 'aes' to unify scale,  
#' e.g., c("fill", "size", "edge\_width").  
#'   
#' @rdname fun\_modify  
modify\_unify\_scale\_limits <-   
 function(ggset, x, aes\_name = NA){  
 x <- .get\_missing\_x(x, "mcnebula")  
 .check\_data(x, list(features\_annotation = "create\_features\_annotation",  
 spectral\_similarity = "compute\_spectral\_similarity"))  
 layers\_name <- names(layers(ggset))  
 args <- as.list(.get\_mapping2(ggset))  
 if (is.logical(aes\_name))  
 aes\_name <- .LEGEND\_mapping()  
 for (i in aes\_name) {  
 if (is.null(args[[ i ]])) {  
 next  
 }  
 if (i == "edge\_width") {  
 attr <- spectral\_similarity(x)[[ args[[i]] ]]  
 fun <- paste0("scale\_", i)  
 } else {  
 attr <- features\_annotation(x)[[ args[[i]] ]]  
 if (is.null(attr)) {  
 attr <- attr(features\_annotation(x), "extra\_data")[[ args[[i]] ]]  
 if (is.null(attr))  
 stop(paste0("Not found attribute '", args[[i]],  
 "' in `features\_annotation(x)`."))  
 }  
 fun <- paste0("scale\_", i, "\_continuous")  
 }  
 if (!is.numeric(attr)) {  
 next  
 }  
 range <- range(attr, na.rm = T)  
 seq <- grep(paste0("^scale\_", i, "|^ggplot2::scale\_", i), layers\_name)  
 if (length(seq) == 1) {  
 ggset <- mutate\_layer(ggset, seq, limits = range)  
 } else if (length(seq) > 1) {  
 stop(paste0("multiple layers of 'scale\_", i,  
 ".\*", "' were found"))  
 } else {  
 ggset <-  
 add\_layers(ggset,  
 new\_command(match.fun(fun),  
 limits = range,  
 name = fun  
 ))  
 }  
 }  
 ggset  
 }  
  
#' @export modify\_set\_labs\_xy  
#' @aliases modify\_set\_labs\_xy  
#'  
#' @description \code{modify\_set\_labs\_xy}:  
#' According to names in slot \code{export\_name} of [mcnebula-class] object  
#' to rename the labs of x and y axis.  
#'   
#' @rdname fun\_modify  
modify\_set\_labs\_xy <-   
 function(ggset, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 .modify\_set\_labs(ggset, x, c("x", "y"))  
 }  
  
#' @export modify\_set\_labs  
#' @aliases modify\_set\_labs  
#'   
#' @description \code{modify\_set\_labs}:  
#' According to names in slot \code{export\_name} of [mcnebula-class] object  
#' to rename the labs of legends.  
#'   
#' @rdname fun\_modify  
modify\_set\_labs <-   
 function(ggset, x){  
 x <- .get\_missing\_x(x, "mcnebula")  
 .modify\_set\_labs(ggset, x)  
 }  
  
.modify\_set\_labs <-   
 function(ggset, x, ...) {  
 export\_name <- as.list(export\_name(x))  
 mapping <- vecter\_unique\_by\_names(.get\_mapping2(ggset, ...))  
 args <- vapply(mapping, FUN.VALUE = "ch",  
 function(attr) {  
 if (is.null(export\_name[[ attr ]]))  
 attr  
 else  
 export\_name[[ attr ]]  
 })  
 seq <- grep("^labs$|^ggplot2::labs$", names(layers(ggset)))  
 if ( length(seq) == 1) {  
 ggset <- do.call(mutate\_layer, c(ggset, seq, args))  
 } else if ( length(seq) > 1 ) {  
 stop( "multiple layers of 'labs' were found" )  
 } else {  
 ggset <- do.call(add\_layers,  
 c(ggset, do.call(new\_command,  
 c(match.fun(labs),  
 args, name = "labs"))))  
 }  
 ggset  
 }  
  
#' @importFrom stringr str\_extract  
.get\_mapping2 <-  
 function(ggset, maps = .LEGEND\_mapping()){  
 args <- .get\_mapping(ggset)  
 pattern <- "[a-z|A-Z|.|\_|0-9]{1,}"  
 args[] <-  
 stringr::str\_extract(args,  
 paste0("(?<=\\()", pattern, "(?=\\),)",  
 "|^", pattern, "$"))  
 args[names(args) %in% maps]  
 }  
  
.LEGEND\_mapping <-   
 function(){  
 c("fill", "color", "colour", "alpha", "size", "edge\_width")  
 }  
  
.get\_mapping <-   
 function(ggset){  
 unlist(lapply(unname(layers(ggset)),  
 function(com){  
 mapping <- command\_args(com)$mapping  
 if (!is.null(mapping)) {  
 vapply(mapping, FUN.VALUE = "ch",  
 function(m) tail(paste0(m), 1))  
 }  
 }))  
 }  
  
# ==========================================================================  
# functions used in 'report' or 'section' class   
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
.write\_block <-   
 function(command\_name, ..., codes){  
 args <- list(...)  
 if (length(args) > 0) {  
 lapply(names(args),  
 function(name) {  
 if (nchar(name) == 0)  
 stop("the args for r block must contain parameter names, ",  
 "e.g., 'eval = FALSE', 'echo = TRUE'")  
 })  
 args <- lapply(args,  
 function(arg) {  
 if (is.character(arg))  
 paste0("'", arg, "'")  
 else  
 arg  
 })  
 args <- paste0(paste0(names(args), " = ", args),  
 collapse = ", ")  
 leader <- paste0("```{", command\_name, ", ", args, "}")  
 } else {  
 leader <- paste0("```{", command\_name, "}")  
 }  
 end <- "```"  
 c(leader, codes, end, "")  
 }  
  
.args\_r\_block <-  
 function(){  
 list(echo = T,  
 eval = T,  
 message = F  
 )  
 }  
  
.args\_r\_block\_table <-  
 function(){  
 list(echo = T,  
 eval = T,  
 message = F  
 )  
 }  
  
.args\_r\_block\_figure <-  
 function(){  
 list(echo = T,  
 eval = T,  
 message = F,  
 fig.cap = "The figure"  
 )  
 }  
  
nshow <- function(object){  
 if (!is.null(object)) {  
 show(object)  
 }  
}  
  
textSh <-   
 function(..., sep = "", exdent = 4, ending = "\n",  
 pre\_collapse = F, collapse = "\n",  
 pre\_trunc = F, trunc\_width = 200,  
 pre\_wrap = F, wrap\_width = 60){  
 text <- list(...)  
 if (pre\_collapse) {  
 text <- vapply(text, paste, "ch", collapse = collapse)  
 }  
 text <- paste(text, sep = sep)  
 if (pre\_trunc) {  
 text <- .text\_fold(text, trunc\_width)  
 }  
 if (pre\_wrap) {  
 text <- paste0(strwrap(text, width = wrap\_width), collapse = "\n")  
 }  
 exdent <- paste0(rep(" ", exdent), collapse = "")  
 writeLines(gsub("(?<=\n)|(?<=^)", exdent, text, perl = T))  
 if (!is.null(ending))  
 cat(ending)  
 }  
  
#' @importFrom stringr str\_trunc  
.text\_fold <-   
 function(text, width = 200, ellipsis = crayon::silver("...(fold)")){  
 stringr::str\_trunc(text, width = width, ellipsis = ellipsis)  
 }  
  
.part <-  
 function(...){  
 args <- list(...)  
 unlist(lapply(args,  
 function(obj) {  
 if (!is.null(obj))  
 c(obj, "")  
 }))  
 }  
  
get\_history <-   
 function(exclude = 0){  
 file1 <- tempfile("Rrawhist")  
 savehistory(file1)  
 rawhist <- readLines(file1)  
 unlink(file1)  
 if (exclude > 0) {  
 exclude <- (length(rawhist) - exclude + 1):length(rawhist)  
 rawhist <- rawhist[-exclude]  
 }  
 rawhist  
 }  
  
#' @importFrom bookdown pdf\_document2  
#' @importFrom BiocStyle pdf\_document  
#' @importFrom BiocStyle html\_document  
default\_pdf <- bookdown::pdf\_document2  
bioc\_pdf <- BiocStyle::pdf\_document  
bioc\_html <- BiocStyle::html\_document  
  
# ==========================================================================  
# get or modify 'yaml' for 'report'  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
.yaml\_default <-   
 function(style = c("default", "BiocStyle", "BiocStyle\_pdf")){  
 style <- match.arg(style)  
 readLines(system.file("extdata", paste0(style, ".yml"),  
 package = "MCnebula2"))  
 }  
# ==========================================================================  
# additional function  
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  
#' @importFrom stats dist hclust model.matrix reorder rnorm  
#' @importFrom utils combn head methods object.size savehistory str tail  
#' @importFrom utils write.table  
setMissing <-   
 function(generic, ..., .SIG = "missing"){  
 args <- list(...)  
 sig <- getGeneric(generic)@signature  
 res <- vapply(sig, FUN.VALUE = "character",  
 function(name){  
 if (is.null(args[[ name ]]))  
 .SIG  
 else  
 args[[ name ]]  
 })  
 names(res) <- sig  
 return(res)  
 }  
  
reCallMethod <-   
 function(funName, args, ...){  
 arg.order <- unname(getGeneric(funName)@signature)  
 args.missing <- !arg.order %in% names(args)  
 if (any(args.missing)) {  
 args.missing <- arg.order[args.missing]  
 args.missing <- sapply(args.missing, simplify = F,  
 function(x) structure(0L, class = "missing"))  
 args <- c(args, args.missing)  
 }  
 args <- lapply(arg.order, function(i) args[[i]])  
 sig <- get\_signature(args)  
 method <- selectMethod(funName, sig)  
 last\_fun <- sys.function(sys.parent())  
 n <- 0  
 while (identical(last\_fun, method@.Data, ignore.environment = T)) {  
 if (n == 0) {  
 mlist <- getMethodsForDispatch(getGeneric(funName))  
 }  
 n <- n + 1  
 rm(list = paste0(method@defined, collapse = "#"), envir = mlist)  
 method <- selectMethod(funName, sig, mlist = mlist)  
 }  
 expr <- paste0("method@.Data(",  
 paste0(paste0(arg.order, " = args[[",  
 1:length(arg.order), "]]"),  
 collapse = ", "),  
 ", ...)")  
 eval(parse(text = expr))  
 }  
  
get\_signature <-   
 function(args){  
 vapply(args, function(arg) class(arg)[1], FUN.VALUE = "ch")  
 }  
  
match\_methods <-   
 function(name, classes){  
 methods <- showMethods(classes = classes, printTo = FALSE)  
 methods <- methods[ grep(paste0("^Function: ", name), methods, perl = T) ]  
 vapply(strsplit(methods, " "), `[`, "character", 2)  
 }  
  
vecter\_unique\_by\_names <-   
 function(lst){  
 unique <- data.frame(names = names(lst),  
 order = 1:length(lst))  
 unique <- unique[!duplicated(unique$names), ]  
 lst[unique$order]  
 }  
  
vec\_unique\_by\_value <-   
 function(vec){  
 unique <- data.frame(value = vec,  
 order = 1:length(vec))  
 unique <- unique[!duplicated(unique$value), ]  
 vec[unique$order]  
 }  
  
  
slots\_mapply <-   
 function(x, fun, ...){  
 slots <- attributes(x)  
 slots <- slots[-length(slots)]  
 res <- mapply(fun, slot = slots, name = names(slots), ...)  
 return(res)  
 }  
  
  
mapply\_rename\_col <-   
 function(  
 mutate\_set,  
 replace\_set,  
 names,  
 fixed = F  
 ){  
 envir <- environment()  
 mapply(mutate\_set, replace\_set,  
 MoreArgs = list(envir = envir, fixed = fixed),  
 FUN = function(mutate, replace, envir,  
 fixed = F, names = get("names", envir = envir)){  
 names <- gsub(mutate, replace, names, perl = ifelse(fixed, F, T), fixed = fixed)  
 assign("names", names, envir = envir)  
 })  
 return(names)  
 }  
  
  
.show <-   
 function(object){  
 cat(class(object), "\n")  
 slots\_mapply(object, function(names, slots){  
 cat(names, ":\n", sep = "")  
 cat(str(slots))  
 cat("\n\n")  
 })  
 }  
  
# # -------------------------------------   
.message\_info <-   
 function(main, sub, arg = NULL, sig = "##"){  
 message(sig, " ", main, ": ", sub, " ", arg)  
 }  
  
.message\_info\_formal <-   
 function(main, sub, arg = NULL, sig = "[INFO]"){  
 message(sig, " ", main, ": ", sub, " ", arg)  
 }  
  
#' @importFrom grid current.viewport  
.message\_info\_viewport <-   
 function(info = "info"){  
 .message\_info(info, "current.viewport:",  
 paste0("\n\t", paste0(grid::current.viewport())))  
 }  
  
.get\_missing\_x <-   
 function(x, class, n = 2, envir = parent.frame(n)){  
 if (missing(x)) {  
 x <- get("x", envir = envir)  
 if (!is(x, class)) {  
 stop( paste0("there must be an `x` of '", class,   
 "' in `parent.frame(", n - 1, ")`" ) )  
 }  
 }  
 return(x)  
 }  
  
  
#' @importFrom rlang as\_label  
.check\_data <-   
 function(object, lst, tip = "(...)"){  
 target <- rlang::as\_label(substitute(object))  
 mapply(lst, names(lst), FUN = function(value, name){  
 obj <- match.fun(name)(object)  
 if (is.null(obj)) {  
 stop(paste0("is.null(", name, "(", target, ")) == T. ",  
 "use `", value, tip, "` previously."))  
 }  
 if (is.list(obj)) {  
 if (length(obj) == 0) {  
 stop(paste0("length(", name, "(", target, ")) == 0. ",  
 "use `", value, tip, "` previously."))  
 }  
 }  
 })  
 }  
  
.check\_names <-   
 function(param, formal, tip1, tip2){  
 if (!is.null(names(param))) {  
 if ( any(!names(formal) %in% names(param)) ) {  
 stop(paste0("the names of `", tip1, "` must contain all names of ",  
 tip2, "; or without names."  
 ))  
 }  
 }  
 }  
  
#' @importFrom rlang as\_label  
.check\_class <-   
 function(object, class = "layout", tip = "grid::grid.layout"){  
 if (!is(object, class)) {  
 stop(paste0("`", rlang::as\_label(substitute(object)),  
 "` should be a '", class, "' object created by ",  
 "`", tip, "`." ))  
 }  
 }  
  
.check\_columns <-   
 function(obj, lst, tip){  
 if (!is.data.frame(obj))  
 stop(paste0("'", tip, "' must be a 'data.frame'."))  
 lapply(lst, function(col){  
 if (is.null(obj[[ col ]]))  
 stop(paste0("'", tip, "' must contains a column of '", col, "'."))  
 })  
 }  
  
.check\_type <-   
 function(obj, type, tip){  
 fun <- match.fun(paste0("is.", type))  
 apply(obj, 2, function(col){  
 if (!fun(col))  
 stop(paste0("data columns in '", tip, "' must all be '", type, "'."))  
 })  
 }  
  
.check\_path <-   
 function(path){  
 if (!file.exists(path)) {  
 dir.create(path, recursive = T)  
 }  
 }  
  
.check\_file <-   
 function(file){  
 if (!file.exists(file)) {  
 stop("file.exists(file) == F, `file` not exists.")  
 }  
 }  
  
validate\_class\_in\_list <-   
 function(lst, recepts, tip){  
 check <-   
 lapply(lst, function(layer) {  
 check <- lapply(recepts, function(class) {  
 if (is(layer, class)) T })  
 if (any(unlist(check))) T else F  
 })  
 if (any(!unlist(check)))  
 stop(tip)  
 else T  
 }  
  
.suggest\_bio\_package <-   
 function(pkg){  
 if (!requireNamespace(pkg, quietly = T))  
 stop("package '", pkg, "' not installed. use folloing to install:\n",  
 '\nif (!require("BiocManager", quietly = TRUE))',  
 '\n\tinstall.packages("BiocManager")',  
 '\nBiocManager::install("', pkg, '")\n\n')  
 }  
  
read\_tsv <- function(path){  
 file <- data.table::fread(input=path, sep="\t", header=T, quote="", check.names=F)  
 return(file)  
}  
  
pbsapply\_read\_tsv <- function(path){  
 data <- pbapply::pbsapply(path, read\_tsv, simplify = F)  
 return(data)  
}  
  
write\_tsv <-  
 function(x, filename, col.names = T, row.names = F){  
 write.table(x, file = filename, sep = "\t",  
 col.names = col.names, row.names = row.names, quote = F)  
 }  
  
  
#' @importFrom grid unit  
#' @importFrom ggtext element\_textbox  
.element\_textbox <-   
 function(family = NULL, face = NULL, size = NULL,  
 colour = "white", fill = "lightblue",  
 box.colour = "white", linetype = 1, linewidth = NULL,  
 hjust = NULL, vjust = NULL,  
 halign = 0.5, valign = NULL, lineheight = NULL,  
 margin = match.fun("margin")(3, 3, 3, 3),  
 padding = match.fun("margin")(2, 0, 1, 0),  
 width = grid::unit(1, "npc"),  
 height = NULL, minwidth = NULL,  
 maxwidth = NULL, minheight = NULL, maxheight = NULL,  
 r = grid::unit(5, "pt"), orientation = NULL,  
 debug = FALSE, inherit.blank = FALSE  
 ){  
 structure(as.list(environment()),  
 class = c("element\_textbox", "element\_text", "element"))  
 }  
  
  
.get\_legend <-   
 function(p){  
 p <- ggplot2:::ggplot\_build.ggplot(p)$plot  
 theme <- ggplot2:::plot\_theme(p)  
 position <- theme$legend.position  
 ggplot2:::build\_guides(p$scales, p$layers, p$mapping,  
 position, theme, p$guides, p$labels)  
 }  
  
.depigment\_col <-   
 function(col, n = 10, level = 5){  
 colorRampPalette(c("white", col))(n)[level]  
 }  
  
  
.simulate\_quant\_set <-   
 function(x){  
 quant <- .simulate\_quant(features\_annotation(x)$.features\_id)  
 meta <- group\_strings(colnames(quant),  
 c(control = "^control", model = "^model",  
 treat = "^treat", pos = "^pos"), "sample")  
 features\_quantification(x) <- quant  
 sample\_metadata(x) <- meta  
 return(x)  
 }  
  
#' @importFrom tibble as\_tibble  
.simulate\_quant <-   
 function(.features\_id, mean = 50, sd = 20, seed = 555,  
 group = c("control", "model", "treat", "pos"), rep = 5){  
 quant <- data.frame(.features\_id = .features\_id)  
 set.seed(seed)  
 lst <- lapply(1:(length(group) \* rep), function(x){  
 rnorm(nrow(quant), mean, sd)  
 })  
 df <- apply(do.call(data.frame, lst), 2, abs)  
 df <- df[, hclust(dist(t(df)))$order]  
 colnames(df) <- unlist(lapply(group, paste0, "\_", 1:rep))  
 tibble::as\_tibble(cbind(quant, df))  
 }  
  
group\_strings <-   
 function(strings, patterns, target = NA){  
 if (is.null(names(patterns)))  
 stop("`patterns` must be characters with names.")  
 lst <- .find\_and\_sort\_strings(strings, patterns)  
 lst <- lapply(names(lst), function(name){  
 data.frame(target = lst[[name]], group = name)  
 })  
 df <- do.call(rbind, lst)  
 if (!is.na(target)) {  
 colnames(df)[1] <- target  
 }  
 tibble::as\_tibble(df)  
 }  
  
.find\_and\_sort\_strings <-   
 function(strings, patterns){  
 lapply(patterns,  
 function(pattern){  
 strings[grepl(pattern, strings, perl = T)]  
 })  
 }  
  
.as\_dic <-   
 function(vec, names, default,  
 fill = T, as.list = T, na.rm = F){  
 if (is.null(names(vec)))  
 names(vec) <- names[1:length(vec)]  
 if (fill) {  
 if (any(!names %in% names(vec))) {  
 ex.names <- names[!names %in% names(vec)]  
 ex <- rep(default, length(ex.names))  
 names(ex) <- ex.names  
 vec <- c(vec, ex)  
 }  
 }  
 if (as.list) {  
 if (!is.list(vec))  
 vec <- as.list(vec)  
 }  
 if (na.rm) {  
 vec <- vec[!is.na(names(vec))]  
 }  
 vec  
 }  
  
.fresh\_param <-   
 function(default, args){  
 if (missing(args))  
 args <- as.list(parent.frame())  
 args <- args[ !vapply(args, is.name, T) ]  
 sapply(unique(c(names(default), names(args))),  
 simplify = F,  
 function(name){  
 if (any(name == names(args)))  
 args[[ name ]]  
 else  
 default[[ name ]]  
 })  
 }  
  
  
#' @importFrom grImport2 readPicture  
#' @importFrom grImport2 grobify  
.cairosvg\_to\_grob <-   
 function(path){  
 grImport2::grobify(grImport2::readPicture(path))  
 }  
  
checkColMerge <- function(x, y, ...){  
 args <- list(...)  
 by <- args$by  
 col <- lapply(list(x, y),  
 function(df){  
 colnames(df)[ !colnames(df) %in% by ]  
 })  
 discard <- col[[2]][col[[2]] %in% col[[1]]]  
 y <- y[, !colnames(y) %in% discard]  
 if (!is.data.frame(y))  
 return(x)  
 args <- c(list(x = x, y = y), args)  
 do.call(merge, args)  
}  
  
zoRange <- function(x, factor) {  
 x <- range(x)  
 ex <- abs(x[2] - x[1]) \* (factor - 1)  
 x[1] <- x[1] - ex  
 x[2] <- x[2] + ex  
 return(x)  
}  
  
  
## default font for visualization  
# @importFrom grDevices pdfFonts  
# .setFont <- function(pattern){  
 # font <- names(pdfFonts())  
 # n <- grep(pattern, font)  
 # if (length(n) >= 1) {  
 # font <- font[n[1]]  
 # } else {  
 # font[1]  
 # }  
# }  
  
# .font <- if (.Platform$OS.type == 'unix') "Times" else "Times New Roman"  
  
#' @export setFont  
#' @title Set font for visualization of MCnebula2  
#' @description \bold{Note that} your R harbours the font you set.  
#' @param font character(1). Such as 'Times'. If you output the  
#' visualization for pdf, use \code{grDevices::pdfFonts()} to checkout  
#' the available fonts; else, you might need help with package \code{extrafont}.  
#' @rdname setFont  
setFont <- function(font = "Times") {  
 assign(".font", font, env = topenv(environment()))  
 options(mcnebulaFont = font)  
}  
.font <- "Times"  
setFont("Times")