R codes of MCnebula2

Contents

1 File: base-generic.R

```
# ==========================================================================
# 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
```

```r
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<-",
```

```r
           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<-"))
```

2 File: class-command.R

```
# ==========================================================================
# 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
```

```r
#' @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))
          })
```

3 File: class-ggset.R

```r
# ==========================================================================
# 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)
          })
```

4 File: class-mcn_dataset.R

```
# ==========================================================================
# 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]]))
          })
```

```r
#' @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)
          })
```

5 File: class-mcnebula.R

```r
# ==========================================================================
# 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.
#'
```

```r
#' @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'
```

```r
#'   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" ]]
          })
```

```r
#' @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" ]]
          })
```

```r
.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)))
          })
```

6 File: class-melody.R

```
# ==========================================================================
# a class to store hex color set.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass melody
#'
#' @aliases melody
#'
#' @title Mutiple color palette in hexadecimal code
#'
#' @description
```

```r
#' 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
  )
```

```r
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod show
#' @aliases show
#' @rdname melody-class
setMethod("show",
          signature = c(object= "melody"),
          function(object){
            .show(object)
          })
```

```r
#' @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)
                 })
```

7 File: class-msframe.R

```
# ==========================================================================
# 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.
```

```r
#' @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}.
```

```r
#' 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)
          })
```

8 File: class-nebula.R

```r
# ==========================================================================
# 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)
                 })
```

```r
#' @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
```

```r
#' @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)
                 })
```

9 File: class-project_api.R

```r
# ==========================================================================
# 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)
                 })
```

10 File: class-project_conformation.R

```
# ==========================================================================
# 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
                                 else FALSE
                               }
                             })
              if (any(!check))
                "the colnames not matched."
              else TRUE
            })
```

```
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod show
#' @aliases show
#' @rdname project_conformation-class
setMethod("show",
          signature = c(object = "project_conformation"),
          function(object){
            .show(object)
          })



#' @exportMethod project_conformation
#' @aliases project_conformation
#' @description \code{project_conformation}, \code{project_conformation<-}: getter and setter
#' for the \code{project_conformation} slot of the object.
#' @rdname project_conformation-class
setMethod("project_conformation",
          signature = "ANY",
          function(x){ x@project_conformation })

#' @exportMethod project_conformation<-
#' @aliases project_conformation<-
#' @param value The value for the slot.
#' @rdname project_conformation-class
setReplaceMethod("project_conformation",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, project_conformation = value)
                 })



#' @exportMethod file_name
#' @aliases file_name
#' @description \code{file_name}, \code{file_name<-}: getter and setter
#' for the \code{file_name} slot of the object.
```

```r
#' @rdname project_conformation-class
setMethod("file_name",
          signature = c(x = "project_conformation"),
          function(x){ x@file_name })

#' @exportMethod file_name<-
#' @aliases file_name<-
#' @param value The value for the slot.
#' @rdname project_conformation-class
setReplaceMethod("file_name",
                 signature = c(x = "project_conformation"),
                 function(x, value){
                   initialize(x, file_name = value)
                 })



#' @exportMethod file_api
#' @aliases file_api
#' @description \code{file_api}, \code{file_api<-}: getter and setter
#' for the \code{file_api} slot of the object.
#' @rdname project_conformation-class
setMethod("file_api",
          signature = c(x = "project_conformation"),
          function(x){ x@file_api })

#' @exportMethod file_api<-
#' @aliases file_api<-
#' @param value The value for the slot.
#' @rdname project_conformation-class
setReplaceMethod("file_api",
                 signature = c(x = "project_conformation"),
                 function(x, value){
                   initialize(x, file_api = value)
                 })



#' @exportMethod attribute_name
#' @aliases attribute_name
#' @description \code{attribute_name}, \code{attribute_name<-}: getter and setter
#' for the \code{attribute_name} slot of the object.
#' @rdname project_conformation-class
```

```
setMethod("attribute_name",
          signature = c(x = "project_conformation"),
          function(x){ x@attribute_name })

#' @exportMethod attribute_name<-
#' @aliases attribute_name<-
#' @param value The value for the slot.
#' @rdname project_conformation-class
setReplaceMethod("attribute_name",
                 signature = c(x = "project_conformation"),
                 function(x, value){
                   initialize(x, attribute_name = value)
                 })
```

11 File: class-project_dataset.R

```
# ==========================================================================
# a class to store dataset extract from raw data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass project_dataset
#'
#' @aliases project_dataset
#'
#' @title Store extracted data
#'
#' @description
#' This is a class object used to store extracted data (raw data).
#' See [project-class] for joint application with other related classes.
#'
#' @family projects
#' @family datasets
#'
#' @slot dataset list. See [dataset-class].
#'
#' @rdname project_dataset-class
#'
.project_dataset <-
  setClass("project_dataset",
           contains = "dataset",
           prototype = NULL
           )
```

```r
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod project_dataset
#' @aliases project_dataset
#' @description \code{project_dataset}, \code{project_dataset<-}: getter and setter
#' for the \code{project_dataset} slot of the object.
#' @rdname project_dataset-class
setMethod("project_dataset",
          signature = c(x = "ANY"),
          function(x){ x@project_dataset })

#' @exportMethod project_dataset<-
#' @aliases project_dataset<-
#' @param value The value for the slot.
#' @rdname project_dataset-class
setReplaceMethod("project_dataset",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, project_dataset = value)
                 })


#' @exportMethod latest
#' @aliases latest
#' @description \code{latest}: get the first data in \code{dataset} slot ('list') and
#' format as 'tbl'. Equals:
#' - \code{latest(object)}
#' - \code{tibble::as_tibble(entity(dataset(object)[[1]]))}
#' @family latests
#' @seealso [tibble::as_tibble()]
#' @rdname project_dataset-class
setMethod("latest",
          signature = c(x = "project_dataset"),
          function(x){
            tibble::as_tibble(entity(dataset(x)[[1]]))
          })


#' @exportMethod extract_rawset
#' @aliases extract_rawset
#' @rdname project_dataset-class
setMethod("extract_rawset",
          signature = c(x = "ANY", subscript = "character"),
          function(x, subscript){
            extract_rawset(x, subscript = subscript,
                             fun_collate = function(...){
                               stop("`subscript` not found in `dataset(project_dataset(x))`")
                             })
          })

#' @exportMethod extract_rawset
#' @aliases extract_rawset
#' @description \code{extract_rawset}: For fast extract data in object which containing
#' \code{project_dataset} slot. Normally not used.
#' @param x an object contain \code{project_dataset} slot.
#' @param subscript character. Specified the data in \code{dataset} slot
#' in \code{project_dataset} slot.
#' See [VIRTUAL_subscript-class].
#' @param fun_collate function. If the specified data not exists in \code{dataset} slot,
#' it will be used to collate data. This parameter is not for normal use.
#' @param ... parameters passed to 'fun_collate'.
#' @rdname project_dataset-class
setMethod("extract_rawset",
          signature = c(x = "ANY",
                        subscript = "character",
                        fun_collate = "function"
                        ),
          function(x, subscript, fun_collate, ...){
            if ( any( subscript == names(dataset(project_dataset(x))) ) )
              msframe <- dataset(project_dataset(x))[[ subscript ]]
            else
              msframe <- fun_collate(x, subscript, ...)
            lst <- list(msframe)
            names(lst) <- subscript
            return(lst)
          })
```

12 File: class-project_metadata.R

```
# ==========================================================================
# a class to store the metadata of files in project directory, i.e.,
# whether the files exists.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass project_metadata
#'
#' @aliases project_metadata
#'
#' @title Metadata of files
#'
#' @description This is a class object used to store metadata of files.
#' See [project-class] for joint application with other related classes.
#'
#' @note The class is not for normal use of the package.
#'
#' @family projects
#'
#' @slot metadata a list with names of [subscript-class].
#' Each element of the list is a data.frame.
#'
#' @rdname project_metadata-class
#'
.project_metadata <-
  setClass("project_metadata",
           contains = character(),
           representation =
             representation(metadata = "list"
                            ),
           prototype = NULL
           )
```

```
# ==========================================================================
# validity
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
setValidity("project_metadata",
            function(object){
              if ( is.character(names(object@metadata)) )
                TRUE
              else
                FALSE
            })
```

```
# ==========================================================================
# method
```

```r
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod show
#' @aliases show
#' @rdname project_metadata-class
setMethod("show",
          signature = c(object = "project_metadata"),
          function(object){
            .show(object)
          })


#' @exportMethod project_metadata
#' @aliases project_metadata
#' @description \code{project_metadata}, \code{project_metadata<-}: getter and setter
#' for the \code{project_metadata} slot of the object.
#' @rdname project_metadata-class
setMethod("project_metadata",
          signature = c(x = "ANY"),
          function(x){ x@project_metadata })

#' @exportMethod project_metadata<-
#' @aliases project_metadata<-
#' @param value The value for the slot.
#' @rdname project_metadata-class
setReplaceMethod("project_metadata",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, project_metadata = value)
                 })


#' @exportMethod latest
#' @aliases latest
#' @description \code{latest}: get the first data in \code{metadata} slot and
#' format as "tbl".
#' @family latests
#' @rdname project_metadata-class
setMethod("latest",
          signature = c(x = "project_metadata"),
          function(x){
            tibble::as_tibble(metadata(x)[[1]])
          })


#' @exportMethod metadata
#' @aliases metadata
#' @description \code{metadata}, \code{metadata<-}: getter and setter
#' for the \code{metadata} slot of the object.
#' @rdname project_metadata-class
setMethod("metadata",
          signature = c(x = "project_metadata"),
          function(x){ x@metadata })

#' @exportMethod metadata<-
#' @aliases metadata<-
#' @param value The value for the slot.
#' @rdname project_metadata-class
setReplaceMethod("metadata",
                 signature = c(x = "project_metadata"),
                 function(x, value){
                   initialize(x, metadata = value)
                 })


#' @exportMethod add_dataset
#' @aliases add_dataset
#' @description \code{add_dataset}: add the list into slot \code{metadata}.
#' @param list a list (with names) of metadata (data.frame) with names.
#' @rdname project_metadata-class
setMethod("add_dataset",
          signature = c(x = "project_metadata",
                        list = "list"),
          function(x, list){
            metadata <- c(list, metadata(x))
            metadata(x) <- vecter_unique_by_names(metadata)
            return(x)
          })

#' @exportMethod extract_metadata
#' @aliases extract_metadata
#' @description \code{extract_metadata}: use "subscript" to extract metadata from an
#' object with slot \code{project_metadata},
#' and then return it as a new \code{project_metadata}.
#' @param subscript see [subscript-class].
#' @rdname project_metadata-class
setMethod("extract_metadata",
          signature = c(x = "ANY", subscript = "character"),
          function(x, subscript){
            x <- get_metadata(x, subscript = subscript)
            path.set <- metadata(project_metadata(x))[[ subscript ]]
            ## build project_metadata
            path.set <- list(path.set)
            names(path.set) <- subscript
            new("project_metadata", metadata = path.set)
          })

#' @exportMethod get_metadata
#' @aliases get_metadata
#' @description \code{get_metadata}: for an object with slot of \code{project_metadata},
#' get the metadata of files of specified "subscript", then return the object.
#' @param project_metadata [project_metadata-class] object.
#' Used by \code{get_metadata()}. If 'missing', the slot \code{project_metadata} inside
#' the object will be used.
#' @param project_conformation [project_conformation-class] object.
#' Used by \code{get_metadata()}. If 'missing', the slot \code{project_conformation} inside
#' the object will be used.
#' @param path character. The path of the project directory (generally, SIRIUS project).
#' If 'missing', the slot \code{project_path} inside the object will be used.
#' @rdname project_metadata-class
setMethod("get_metadata",
          signature = c(x = "ANY", subscript = "character"),
          function(x, subscript){
            exits_meta <- names( metadata(project_metadata(x)) )
            if (!subscript %in% exits_meta) {
              project_metadata(x) <-
                get_metadata(subscript = subscript,
                             project_metadata = project_metadata(x),
                             project_conformation = project_conformation(x),
                             project_version = project_version(x),
                             path = project_path(x)
                )
            }
            return(x)
          })

#' @exportMethod get_metadata
#' @rdname project_metadata-class
setMethod("get_metadata",
          signature = setMissing("get_metadata",
                                 subscript = "character",
                                 project_metadata = "project_metadata",
                                 project_conformation = "project_conformation",
                                 project_version = "character",
                                 path = "character"),
          function(subscript, project_metadata,
            project_conformation, project_version, path)
          {
            file_name <- file_name(project_conformation)
            file_api <- file_api(project_conformation)
            if (!subscript %in% names(file_api) )
              stop( "`subscript` not descriped in `names(file_api(project_conformation))`" )
            api <- file_api[[ subscript ]]
            api <- strsplit(api, split = "/")[[1]]
            for (i in 1:length(api)) {
              sub <- api[i]
              if ( any(sub == names(metadata(project_metadata))) )
                next
              if ( !sub %in% names(file_name) )
                stop( "`subscript` not descriped in `names(file_name(project_conformation))`" )
              ## get the name of file, or the function name to get file name
              target <- file_name[[sub]]
              ## get the target of filename
              if ( grepl("^FUN_", target) )
                target <- match.fun(target)()
              if ( i == 1 ) {
                fun <- match.fun(paste0("list_files_top.", project_version))
                df <- fun(path, target)
              } else {
                ## get the metadata of upper directory
                df <- metadata(project_metadata)[[ api[i - 1] ]]
                upper <- paste0(apply(df, 1, paste0, collapse = "/"))

                .message_info("project_metadata", "get_metadata",
                          paste0(target, "(", sub, ")"))
                fun <- match.fun(paste0("list_files.", project_version))
                df <- fun(path, upper, target, api[i - 1])
              }
              lst <- list( df )
              names(lst) <- sub
              project_metadata <- add_dataset(project_metadata, lst)
            }
            return(project_metadata)
          })
```

13 File: class-project.R

```
# ==========================================================================
# a class to store information about files in target dir, and as well,
# to read these files and save data in slots.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass project
#'
#' @aliases project
#'
#' @title Collection of Interface for extracting data from raw directory
#'
#' @description
#'
#' This is a class object designed to extract files in the project directory.
#' Its responsibility is to describe the name,
#' path and reading method of the file under the project directory;
#' Use these information to extract and store data.
#'
#' @details
#' It is a collection of classes whose names start with "project_":
#' - [project_conformation-class]: The name, path and attribute name of the file are described.
#' - [project_api-class]: Functions for reading and formatting data are provided.
#' - [project_metadata-class]: Metadata, which records the files stored in the project directory.
#' - [project_dataset-class]: The extracted data is stored here.
#'
#' The above class objects are coordinated into a whole through the "subscript" name
#' (see [subscript-class]).
#' For example, when a command (\code{collate_data(x, ".f3_fingerid")}) requests to
#' extract the files of subscript of ".f3_fingerid", the data extraction module:
#' - from slot of \code{project_conformation},
#' get the file name (pattern string) and path of subscript of ".f3_fingerid";
#' - match the files under the path with the pattern string (i.e., get the metadata of the files),
#' then stored the metadata into slot of \code{project_metadata};
#' - from slot of \code{project_api}, get the functions of subscript of ".f3_fingerid";
#' - use these functions to read and format the data in batches;
#' - store the extracted data into slot of \code{project_dataset}.
#'
#' This class is mainly designed for extracting files under the SIRIUS project directory.
#' These files are: mainly "tables" that can be read through functions such as \code{read.table};
#' numerous and have multiple directories; need to be processed in batches.
#' SIRIUS project may alter the name and path of internal files during version changes,
#' which is in fact deadly for MCnebula2.
#' To make the data extraction module of MCnebula2 free from version issues,
#' this class object is designed to flexibly handle the extraction of internal files.
#' Most contents need to be considered by MCnebula2 developers.
#' The only thing users need to know:
#' slot of [project_dataset-class] object stores the extracted data.
#'
#' @family projects
#'
#' @slot project_version character(1). The target project version. e.g., "sirius.v4".
#' @slot project_path character(1). The target project path.
#' @slot project_conformation [project_conformation-class] object.
#' @slot project_metadata [project_metadata-class] object.
#' @slot project_api [project_api-class] object.
#' @slot project_dataset [project_dataset-class] object.
#'
#' @rdname project-class
#'
.project <-
  setClass("project",
           contains = character(),
           representation =
             representation(project_version = "character",
                            project_path = "character",
                            project_conformation = "project_conformation",
                            project_metadata = "project_metadata",
                            project_api = "project_api",
                            project_dataset = "project_dataset"
                            ),
           prototype = prototype(project_version = character(),
                                project_path = character())
           )
```

```
# =========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod project_version
#' @aliases project_version
#' @description \code{project_version}, \code{project_version<-}: getter and setter
#' for the \code{project_version} slot of the object.
#' @rdname project-class
setMethod("project_version",
          signature = c(x = "ANY"),
          function(x){ x@project_version })

#' @exportMethod project_version<-
#' @aliases project_version<-
#' @param value The value for the slot.
#' @rdname project-class
setReplaceMethod("project_version",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, project_version = value)
                 })

#' @exportMethod project_path
#' @aliases project_path
#' @description \code{project_path}, \code{project_path<-}: getter and setter
#' for the \code{project_path} slot of the object.
#' @rdname project-class
setMethod("project_path",
          signature = c(x = "ANY"),
          function(x){ x@project_path })

#' @exportMethod project_path<-
#' @aliases project_path<-
#' @param value The value for the slot.
#' @rdname project-class
setReplaceMethod("project_path",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, project_path = value)
```

```
                  })



#' @exportMethod file_name
#' @aliases file_name
#' @description  \code{file_name}, \code{file_api}, \code{attribute_name}:
#' fast channel to obtain
#' the downstream slot. e.g., getter
#' for the \code{file_name} slot in sub-object
#' of \code{project_conformation} slot of the object. Equals:
#' - \code{file_name(project_conformation(object))}
#' - \code{file_name(object)}.
#' @rdname project-class
setMethod("file_name",
          signature = c(x = "ANY"),
          function(x){
            file_name(project_conformation(x))
          })

#' @exportMethod file_api
#' @aliases file_api
#' @rdname project-class
setMethod("file_api",
          signature = c(x = "ANY"),
          function(x){
            file_api(project_conformation(x))
          })

#' @exportMethod attribute_name
#' @aliases attribute_name
#' @rdname project-class
setMethod("attribute_name",
          signature = c(x = "ANY"),
          function(x){
            attribute_name(project_conformation(x))
          })



#' @exportMethod project_metadata
#' @aliases project_metadata
#' @description \code{metadata}: fast channel to obtain
```

```r
#' the downstream slot, getter
#' for the \code{metadata} slot in sub-object
#' of \code{project_metadata} slot of the object. Equals:
#' - \code{metadata(project_metadata(object))}
#' - \code{metadata(object)}.
#' @rdname project-class
setMethod("metadata",
          signature = c(x = "ANY"),
          function(x){
            metadata(project_metadata(x))
          })



#' @exportMethod methods_read
#' @aliases methods_read
#' @description \code{methods_read}, \code{methods_format}, \code{methods_match}:
#' fast channel to obtain
#' the downstream slot. e.g., getter
#' for the \code{methods_read} slot in sub-object
#' of \code{project_api} slot of the object. Equals:
#' - \code{methods_read(project_api(object))}
#' - \code{methods_read(object)}.
#' @rdname project-class
setMethod("methods_read",
          signature = c(x = "ANY"),
          function(x){
            methods_read(project_api(x))
          })

#' @exportMethod methods_format
#' @aliases methods_format
#' @rdname project-class
setMethod("methods_format",
          signature = c(x = "ANY"),
          function(x){
            methods_format(project_api(x))
          })

#' @exportMethod methods_match
#' @aliases methods_match
#' @rdname project-class
```

```
setMethod("methods_match",
          signature = c(x = "ANY"),
          function(x){
            methods_match(project_api(x))
          })

#' @exportMethod match.candidates_id
#' @aliases match.candidates_id
#' @description \code{match.candidates_id}, \code{match.features_id}:
#' fast channel to obtain
#' data (mostly 'tbl' or 'data.frame') inside the downstream slot ('list'), getter
#' for the data named \code{match.candidates_id} in
#' \code{methods_match} slot (a 'list') in sub-object
#' of \code{project_api} slot of the object. Equals:
#' - \code{methods_match(project_api(object))$match.candidates_id}
#' - \code{match.candidates_id(object)}.
#' @rdname project-class
setMethod("match.candidates_id",
          signature = c(x = "ANY"),
          function(x){
            methods_match(project_api(x))[[ "match.candidates_id" ]]
          })

#' @exportMethod match.features_id
#' @aliases match.features_id
#' @rdname project-class
setMethod("match.features_id",
          signature = c(x = "ANY"),
          function(x){
            methods_match(project_api(x))[[ "match.features_id" ]]
          })


#' @exportMethod get_upper_dir_subscript
#' @aliases get_upper_dir_subscript
#' @description \code{get_upper_dir_subscript}: Get the "subscript" name of the folder.
#' @param x Maybe object of class inherit [project-class].
#' @param subscript the "subscript" name of file. See [subscript-class].
#' @rdname project-class
setMethod("get_upper_dir_subscript",
          signature = setMissing("get_upper_dir_subscript",
                                x = "ANY",
                                subscript = "character"),
          function(x, subscript){
            stringr::str_extract(file_api(project_conformation(x))[[ subscript ]],
                                paste0("(?<=^|/)[^/]*(?=/", subscript, "|$)"))
          })
```

14 File: class-report.R

```
# ==========================================================================
# a class for creating documentation for annotating analysis workflow
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass report
#'
#' @aliases report
#'
#' @title Creating a formatted report
#'
#' @description
#' The report module can create output report quickly for
#' and not just for the mcnebula2 workflow.
#' The report system is primarily a class object that manages text and code blocks.
#' Heading or paragraphs or code blocks were treated as individual report units
#' and deposited sequentially in "layers".
#' The report system provides methods to exhibit, modify these layers.
#' Reports can be exported as ".Rmd" text files, and subsequently,
#' users can call [rmarkdown::render()] for output formatted documents.
#'
#' @param x [report-class] object.
#'
#' @family layerSets
#'
#' @slot yaml character. Metadata passed to .Rmd for setting format of documentation.
#' See \url{https://bookdown.org/yihui/rmarkdown/compile.html} for details.
#' @slot layers list. Element in list must be [section-class],
#' [heading-class] or [code_block-class].
#'
#' @rdname report-class
#' @order 1
#'
#' @examples
#' \dontrun{
#'   s1 <- new_heading("Title 1", 1)
#'
#'   s1.5 <- new_section2(
#'     c("..."), NULL
#'   )
#'
#'   s2 <- new_section2(
#'     c("This is sentence 1.",
#'       "This is sentence 2.",
#'       "This is sentence 3."),
#'     rblock({
#'       df <- data.frame(x = 1:10, y = 1:10)
#'       df <- dplyr::mutate(df,
#'         group = rep(paste0("p", 1:3), c(3, 3, 4))
#'       )
#'     })
#'   )
#'
#'   s3 <- new_heading("Title 2", 1)
#'
#'   s4 <- new_section2(
#'     c("This is a paragraph..."),
#'     rblock({
#'       p <- ggplot(df) +
#'         geom_point(aes(x = x, y = y, fill = group))
#'       ggsave(f <- paste0(tempdir(), "/test.pdf"), p)
#'     })
#'   )
#'
#'   s5 <- include_figure(f, "name", "Caption: ...")
#'
#'   s6 <- rblock({
#'     print(1:100)
#'   }, args = list(echo = F, eval = F))
#'
#'   s7 <- c("... paragraph ...")
#'
#'   sections <- gather_sections()
#'   report <- do.call(new_report, sections)
#'   render_report(report, paste0(tempdir(), "/report.Rmd"), T)
```

```
#'
#'   ####################
#'   ##### Another example
#'   ####################
#'
#'   h1 <- new_heading("heading, level 1", 1)
#'
#'   sec1 <- new_section(
#'     "sub-heading", 2,
#'     "This is a description.",
#'     new_code_block(codes = "seq <- lapply(1:10, cat)")
#'   )
#'
#'   h2 <- new_heading("heading 2, level 1", 1)
#'
#'   fig_block <- new_code_block_figure("plot", "this is a caption",
#'     codes = "df <- data.frame(x = 1:10, y = 1:10)
#'       p <- ggplot(df) +
#'         geom_point(aes(x = x, y = y))
#'       p"
#'   )
#'   sec2 <- new_section(
#'     "sub-heading2", 2,
#'     paste0(
#'       "This is a description. ",
#'       "See Figure ", get_ref(fig_block), "."
#'     ),
#'     fig_block
#'   )
#'
#'   a_data <- dplyr::storms[1:15, 1:10]
#'   table_block <- include_table(a_data, "table1", "This is a caption")
#'
#'   sec3 <- new_section(
#'     NULL, ,
#'     paste0("See Table ", get_ref(table_block, "tab"), "."),
#'     NULL
#'   )
#'
#'   tmp_p <- paste0(tempdir(), "/test.pdf")
#'   pdf(tmp_p)
#'   plot(1:10)
#'   dev.off()
#'   fig_block_2 <- include_figure(tmp_p, "plot2", "this is a caption")
#'   sec4 <- history_rblock(, "^tmp_p <- ", "^fig_block_2")
#'   sec4
#'
#'   ## gather
#'   yaml <- "title: 'title'\noutput:\n  bookdown::pdf_document2"
#'   report <- new_report(
#'     h1, sec1, h2, sec2,
#'     table_block, sec3,
#'     fig_block_2, sec4,
#'     yaml = yaml
#'   )
#'   report
#'
#'   ## output
#'   tmp <- paste0(tempdir(), "/tmp_output.Rmd")
#'   render_report(report, tmp)
#'   rmarkdown::render(tmp)
#'   file.exists(sub("Rmd$", "pdf", tmp))
#' }
.report <-
  setClass("report",
           contains = c("layerSet"),
           representation =
             representation(yaml = "character"),
           prototype = prototype(yaml = .yaml_default())
  )
```

```
# ==========================================================================
# validity
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
setValidity("report",
            function(object){
              recepts <- c("section", "heading", "code_block", "character")
              tip <- paste0("'layer' in 'report' must either be: ",
                            paste0("'", recepts, "'", collapse = ", "))
              validate_class_in_list(layers(object), recepts, tip)
            })
```

```r
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @importFrom crayon silver
#' @importFrom crayon blue
#' @importFrom crayon cyan
#' @exportMethod show_layers
#' @aliases show_layers
#' @description \code{show_layers}: show \code{layers} slots in a pretty
#' and readable style.
#' @rdname report-class
setMethod("show_layers",
          signature = c(x = "report"),
          function(x){
            text <- yaml(x)
            textSh(crayon::blue$bold(text[1]), exdent = 0)
            lapply(c(head(as.list(crayon::cyan(text[-1])), n = 4),
                     crayon::silver("...")),
                   function(text){
                     textSh(text, pre_trunc = T, trunc_width = 60,
                            ending = "")
                   })
            cat(crayon::silver("layers of", length(layers(x)), "\n\n"))
            lapply(1:length(layers(x)),
                   function(seq) {
                     cat(crayon::silver("+++ layer", seq, "+++\n"))
                     layer <- layers(x)[[ seq ]]
                     if (is.character(layer) & !is(layer, "heading"))
                       textSh(layer, pre_collapse = T, pre_trunc = T,
                              pre_wrap = T)
                     else
                       show(layer)
                   })
          })

#' @exportMethod yaml
#' @aliases yaml
#' @description \code{yaml}, \code{yaml<-}: getter and setter
#' for the \code{yaml} slot of the object.
#' @rdname report-class
setMethod("yaml",
          signature = c(x = "ANY"),
          function(x){ x@yaml })

#' @exportMethod yaml<-
#' @aliases yaml<-
#' @param value The value for the slot.
#' @rdname report-class
#'
setReplaceMethod("yaml",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, yaml = value)
                 })

#' @exportMethod new_report
#' @aliases new_report
#' @description \code{new_report}: Create a [report-class] object.
#' @param ... An arbitrary number of [heading-class],
#' [section-class] or [code_block-class] in sequence.
#' Specially, \code{NULL} can be passed herein, but would be ignored.
#' @param yaml character. Passed to .Rmd for setting format of documentation.
#' @rdname report-class
setMethod("new_report",
          signature = c(yaml = "character"),
          function(..., yaml){
            layers <- list(...)
            layers <- layers[!vapply(layers, is.null, logical(1))]
            .report(yaml = yaml, layers = layers)
          })

#' @exportMethod new_report
#' @description \code{new_report()}: get the default parameters for the method \code{new_report}.
#' @description \code{new_report(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{new_report}.
#' @rdname report-class
setMethod("new_report",
          signature = setMissing("new_report",
                                 yaml = "missing"),
          function(...){
            args <- list(yaml = .yaml_default())
            if (missing(...))
```

```r
              return(args)
            else
              reCallMethod("new_report", args, ...)
          })

#' @exportMethod call_command
#' @aliases call_command
#' @description \code{call_command}: Format 'report' object as character, which can be output
#' by \code{writeLines()} function as '.Rmd' file and than use \code{rmarkdown::render} output
#' as pdf, html, or other format files.
#' @family call_commands
#' @seealso [writeLines()], [rmarkdown::render()]...
#' @rdname report-class
setMethod("call_command",
          signature = c(x = "report"),
          function(x){
            yaml <- c("---", yaml(x), "---")
            layers <- unlist(lapply(layers(x), call_command))
            c(yaml, "", layers)
          })

setMethod("call_command",
          signature = c(x = "character"),
          function(x){
            if (tail(x, n = 1) != "")
              c(x, "")
            else
              x
          })
```

```r
# ==========================================================================
# function
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @export search_heading
#' @aliases search_heading
#' @description \code{search_heading}: Regex match for [heading-class] object
#' @param pattern character(1). For Regex match. Allowed Perl expression.
#' @param level numeric.
#' in slot \code{layers}.
#' @rdname report-class
search_heading <-
```

```
function(x, pattern, level = NULL) {
  log <- vapply(layers(x), FUN.VALUE = logical(1),
                function(s) {
                  if (is(s, "heading")) {
                    if (!is.null(level)) {
                      if (!any(level(s) == level))
                        return(F)
                    }
                    grepl(pattern, s, perl = T)
                  } else F
                })
  (1:length(layers(x)))[ log ]
}
```

15 File: class-section.R

```
# ==========================================================================
# a class contains slots for building a text section with code block,
# figure, and table
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass code_block
#'
#' @aliases code_block
#'
#' @title Sequestrate code and setting run parameters.
#'
#' @description Mainly desiged for R code block.
#' The job of this class object is to record the codes and the running parameters
#' of its source language or program;
#' These information can then be output as formatted code block text (use [call_command()]).
#'
#' @slot codes character. Codes.
#' @slot command_name character(1). Program or language. e.g., "r".
#' @slot command_function function. Used for gather the codes and args as code block.
#' @slot command_args list. Args passed to program.
#'
#' @seealso \code{\link{command-class}}.
#' \url{https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref}.
#' \url{https://bookdown.org/yihui/rmarkdown/compile.html}.
#'
#' @rdname code_block-class
```

```r
#' @order 1
#'
.code_block <-
  setClass("code_block",
           contains = c("command"),
           representation =
             representation(codes = "character"),
           prototype =
             prototype(command_name = "r",
                       command_function = .write_block,
                       command_args = .args_r_block(),
                       codes = "## codes"
           )
  )

#' @exportClass code_block_table
#'
#' @aliases code_block_table
#'
#' @description \code{code_block_table}: class inherit from \code{code_block}, with
#' default values for slot \code{command_args} facilitate showing table in document.
#'
#' @rdname code_block-class
#'
.code_block_table <-
  setClass("code_block_table",
           contains = c("code_block"),
           representation =
             representation(),
           prototype = prototype(command_args = .args_r_block_table())
           )

#' @exportClass code_block_figure
#'
#' @aliases code_block_figure
#'
#' @description \code{code_block_figure}: class inherit from \code{code_block}, with
#' default values for slot \code{command_args} facilitate showing figure in document.
#'
#' @rdname code_block-class
#'
```

```r
.code_block_figure <-
  setClass("code_block_figure",
           contains = c("code_block"),
           representation =
             representation(),
           prototype = prototype(command_args = .args_r_block_figure())
           )

#' @exportClass heading
#'
#' @aliases heading
#'
#' @description This is a class object used to clarify the heading and its hierarchy.
#'
#' @slot .Data character(1). Text of heading.
#' @slot level numeric. Level of heading.
#'
#' @rdname section-class
#'
.heading <-
  setClass("heading",
           contains = "character",
           representation =
             representation(level = "numeric"),
           prototype = prototype(level = 2)
           )

#' @exportClass section
#'
#' @aliases section
#'
#' @title Basic cells in the report
#'
#' @description A class object consist of [heading-class], paragraph (character),
#' and [code_block-class]. These [section-class] belong to basic cells of report.
#'
#' @slot heading [heading-class] object.
#' @slot paragraph character. Text for description.
#' @slot code_block [code_block-class] object.
#'
#' @rdname section-class
#' @order 1
#'
.section <-
  setClass("section",
           contains = character(),
           representation =
             representation(heading = "ANY",
                            paragraph = "character",
                            code_block = "ANY"
                            ),
           prototype = prototype(heading = .heading("An analysis step"),
                                 paragraph = "Description",
                                 code_block = .code_block())
           )

setClassUnion("maybe_code_block", c("code_block", "NULL"))
```

```r
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @importFrom crayon silver
#' @exportMethod show
#' @aliases show
#' @param object [code_block-class] object.
#' @rdname code_block-class
#'
#' @examples
#' \dontrun{
#'   ## general
#'   codes <- "df <- data.frame(x = 1:10)
#'     df<-dplyr::mutate(df,y=x*1.5)%>%
#'     dplyr::filter(x >= 5)
#'     p <- ggplot(df)+
#'     geom_point(aes(x=x,y=y))
#'     p"
#'   block <- new_code_block("r", codes, list(eval = T, echo = T, message = T))
#'   ## see results
#'   block
#'   call_command(block)
#'   writeLines(call_command(block))
#'
#'   ## figure
#'   fig_block <- new_code_block_figure(
#'     "plot1",
#'     "this is a caption",
#'     codes = codes
#'   )
#'   ## see results
#'   fig_block
#'   writeLines(call_command(fig_block))
#'   command_args(fig_block)
#'   cat(get_ref(fig_block), "\n")
#'
#'   ## table
#'   codes <- "df <- data.frame(x = 1:10) %>%
#'     dplyr::mutate(y = x, z = x * y)
#'     knitr::kable(df, format = 'markdown', caption = 'this is a caption') "
#'   tab_block <- new_code_block_table("table1", codes = codes)
#'   ## see results
#'   tab_block
#'   cat(get_ref(tab_block), "\n")
#'
#'   ## default parameters
#'   new_code_block()
#'
#' }
setMethod("show",
          signature = c(object = "code_block"),
          function(object){
            content <- call_command(object)
            content <- lapply(content,
                              function(text){
                                if (grepl("#\\s*#", text))
                                  crayon::silver(text)
                                else
                                  text
                              })
            content[1] <- .text_fold(content[1], width = 60)
            if (length(content) > 4) {
              content <- c(content[1:3], crayon::silver(" + <codes-fold> + "),
                           tail(content, n = 1))
            }
            textSh(content, pre_collapse = T)
          })

#' @exportMethod show
#' @aliases show
#' @rdname code_block-class
setMethod("show",
          signature = c(object = "code_block_table"),
          function(object){
            textSh(crayon::silver("use for cross-referencing:",
                                  get_ref(object, "tab")), ending = NULL)
            selectMethod("show", "code_block")@.Data(object)
            if (!grepl("kable\\([^\\(]*caption", codes(object))) {
              textSh(crayon::silver("make sure the codes contain:",
                                    "`knitr::kable(..., caption = '...')`"))
            }
          })

#' @exportMethod show
#' @aliases show
#' @rdname code_block-class
setMethod("show",
          signature = c(object = "code_block_figure"),
          function(object){
            textSh(crayon::silver("use for cross-referencing:",
                                  get_ref(object, "fig")), ending = NULL)
            selectMethod("show", "code_block")@.Data(object)
          })

#' @importFrom crayon green
#' @exportMethod show
#' @aliases show
#' @rdname code_block-class
setMethod("show",
          signature = c(object = "heading"),
          function(object){
            textSh(crayon::green$bold(call_command(object)), exdent = 0)
          })

#' @exportMethod show
#' @aliases show
#' @rdname code_block-class
```

```r
setMethod("show",
          signature = c(object = "section"),
          function(object){
            nshow(heading(object))
            textSh(paragraph(object),
                   pre_collapse = T, pre_trunc = T, pre_wrap = T)
            nshow(code_block(object))
          })




#' @exportMethod code_block
#' @aliases code_block
#' @description \code{code_block}, \code{code_block<-}: getter and setter
#' for the \code{code_block} slot of the object.
#' @rdname code_block-class
setMethod("code_block",
          signature = c(x = "ANY"),
          function(x){ x@code_block })

#' @exportMethod code_block<-
#' @aliases code_block<-
#' @param value The value for the slot.
#' @rdname code_block-class
#'
setReplaceMethod("code_block",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, code_block = value)
                 })



#' @exportMethod codes
#' @aliases codes
#' @description \code{codes}, \code{codes<-}: getter and setter
#' for the \code{codes} slot of the object.
#' @rdname code_block-class
setMethod("codes",
          signature = c(x = "code_block"),
          function(x){ x@codes })
```

```r
#' @exportMethod codes<-
#' @aliases codes<-
#' @param value The value for the slot.
#' @rdname code_block-class
setReplaceMethod("codes",
                 signature = c(x = "code_block"),
                 function(x, value){
                   initialize(x, codes = value)
                 })


#' @exportMethod new_code_block
#' @aliases new_code_block
#' @description \code{new_code_block}: create a [code_block-class] object.
#' @param language character(1). For slot \code{command_name}.
#' @param codes character. For slot \code{codes}.
#' @param args list. For slot \code{command_args}.
#' @param prettey logical. If ture, use [styler::style_text()] to pretty the codes.
#' @param fun_prettey function. Default is \code{styler::style_text}.
#' @rdname code_block-class
setMethod("new_code_block",
          signature = c(language = "character", codes = "character",
                        args = "list", prettey = "logical",
                        fun_prettey = "function"),
          function(language, codes, args, prettey, fun_prettey){
            if (length(codes) > 1)
              codes <- paste0(codes, collapse = "\n")
            if (prettey) {
              codes <- paste0(fun_prettey(codes), collapse = "\n")
            }
            .code_block(command_name = language, codes = codes,
                        command_args = args)
          })


#' @exportMethod new_code_block
#' @description \code{new_code_block()}: get the default parameters
#' for the method \code{new_code_block}.
#' @rdname code_block-class
setMethod("new_code_block",
          signature = setMissing("new_code_block",
                                 x = "missing"),
          function(){
            list(language = "r",
                 codes = "## codes",
                 args = .args_r_block(),
                 prettey = T,
                 fun_prettey = styler::style_text
            )
          })

#' @exportMethod new_code_block
#' @description \code{new_code_block(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{new_code_block}.
#' @rdname code_block-class
setMethod("new_code_block",
          signature = c(language = "ANY"),
          function(language, codes, args, prettey, fun_prettey){
            reCallMethod("new_code_block", .fresh_param(new_code_block()))
          })


#' @exportMethod new_code_block_figure
#' @aliases new_code_block_figure
#' @description \code{new_code_block_figure}: create [code_block_figure-class] object.
#' This methods simplified parameter settings for displaying figures in documents.
#'
#' @param name character(1). For cross-reference in document.
#' See \url{https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref}.
#' @param caption character(1). Caption of figure display in document.
#' @param ... Other parameters passed to [new_code_block()].
#'
#' @rdname code_block-class
setMethod("new_code_block_figure",
          signature = c(name = "character"),
          function(name, caption, ...){
            args <- .fresh_param(new_code_block(), list(...))
            args$args$fig.cap <- caption
            args$language <- paste0("r ", name)
            as(do.call(new_code_block, args), "code_block_figure")
          })
```

```r
#' @exportMethod new_code_block_table
#' @aliases new_code_block_table
#' @description \code{new_code_block_table}: create [code_block_table-class] object.
#' This methods simplified parameter settings for displaying table in documents.
#' @rdname code_block-class
setMethod("new_code_block_table",
          signature = c(name = "character"),
          function(name, ...){
            args <- .fresh_param(new_code_block(), list(...))
            args$language <- paste0("r ", name)
            as(do.call(new_code_block, args), "code_block_table")
          })


#' @exportMethod call_command
#' @aliases call_command
#' @description \code{call_command}: Format 'code_block' object as character.
#' @family call_commands
#' @rdname code_block-class
setMethod("call_command",
          signature = c(x = "code_block"),
          function(x){
            do.call(command_function(x),
                    c(command_name = command_name(x),
                      command_args(x), codes = codes(x)))
          })



#' @exportMethod heading
#' @aliases heading
#' @description \code{heading}, \code{heading<-}: getter and setter
#' for the \code{heading} slot of the object.
#' @rdname section-class
setMethod("heading",
          signature = c(x = "ANY"),
          function(x){ x@heading })

#' @exportMethod heading<-
#' @aliases heading<-
#' @param value The value for the slot.
#' @rdname section-class
#'
#' @examples
#' \dontrun{
#'   ## ------------------------------------
#'   ## heading
#'   new_heading("this is a heading", 2)
#'
#'   ## ------------------------------------
#'   ## section
#'   ## example 1
#'   para <- "This is a paragraph stating"
#'   section <- new_section("this is a heading", 2, para)
#'   ## see results
#'   section
#'   call_command(section)
#'   writeLines(call_command(section))
#'
#'   ## example 2
#'   para <- "This is a paragraph stating"
#'   section <- new_section(NULL, , para, NULL)
#'
#'   ## example 3
#'   para <- "This is a paragraph stating"
#'   block <- new_code_block(codes = "df <- data.frame(x = 1:10)")
#'   section <- new_section("heading", 2, para, block)
#'   section
#'
#'   ## example 4
#'   codes <- "df <- data.frame(x = 1:10, y = 1:10)
#'     p <- ggplot(df) +
#'       geom_point(aes(x = x, y = y))
#'     p"
#'   fig_block <- new_code_block_figure("plot", "this is caption", codes = codes)
#'   para <- paste0("This is a paragraph describing the picture. ",
#'                  "See Figure ", get_ref(fig_block), ".")
#'   section <- new_section("heading", 2, para, fig_block)
#'   section
#'   ## output
#'   tmp <- paste0(tempdir(), "/tmp_output.Rmd")
#'   writeLines(call_command(section), tmp)
#'   rmarkdown::render(tmp, output_format = "bookdown::pdf_document2")
#'   file.exists(sub("Rmd$", "pdf", tmp))
#'   ## see [report-class] object:
#'   ## A complete output report, including multiple 'section'.
#'
#'   ## defalt parameters
#'   new_section()
#' }
setReplaceMethod("heading",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, heading = value)
                 })

#' @exportMethod level
#' @aliases level
#' @description \code{level}, \code{level<-}: getter and setter
#' for the \code{level} slot of the object.
#' @rdname section-class
setMethod("level",
          signature = c(x = "heading"),
          function(x){ x@level })

#' @exportMethod level<-
#' @aliases level<-
#' @param value The value for the slot.
#' @rdname section-class
setReplaceMethod("level",
                 signature = c(x = "heading"),
                 function(x, value){
                   initialize(x, level = value)
                 })

#' @exportMethod new_heading
#' @aliases new_heading
#' @description \code{new_heading}: create [heading-class] object.
#' @param heading character(1). For slot \code{.Data}.
#' @param level numeric(1). For slot \code{level}.
#' @rdname section-class
setMethod("new_heading",
          signature = c(heading = "character",
                         level = "numeric"),
          function(heading, level){
            .heading(heading, level = level)
          })

#' @exportMethod call_command
#' @aliases call_command
#' @description \code{call_command}: Format 'heading' object as character.
#' @family call_commands
#' @rdname section-class
setMethod("call_command",
          signature = c(x = "heading"),
          function(x){
            paste0(paste0(rep("#", level(x)), collapse = ""),
                   " ", x)
          })



#' @exportMethod paragraph
#' @aliases paragraph
#' @description \code{paragraph}, \code{paragraph<-}: getter and setter
#' for the \code{paragraph} slot of the object.
#' @rdname section-class
setMethod("paragraph",
          signature = c(x = "section"),
          function(x){ x@paragraph })

#' @exportMethod paragraph<-
#' @aliases paragraph<-
#' @param value The value for the slot.
#' @rdname section-class
setReplaceMethod("paragraph",
                 signature = c(x = "section"),
                 function(x, value){
                   initialize(x, paragraph = value)
                 })

#' @exportMethod new_section
#' @description \code{new_section()}: get the default parameters for
#' the method \code{new_section}.
#' @rdname section-class
setMethod("new_section",
          signature = setMissing("new_section",
                                 heading = "missing"),
          function(heading, level, paragraph, code_block){
            list(heading = "heading",
                 level = 2,
                 paragraph = "Description",
                 code_block = .code_block()
            )
          })

#' @exportMethod new_section
#' @description \code{new_section(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{new_section}.
#' @rdname section-class
setMethod("new_section",
          signature = c(heading = "ANY"),
          function(heading, level, paragraph, code_block){
            args <- .fresh_param(new_section())
            reCallMethod("new_section", args)
          })

#' @exportMethod new_section
#' @aliases new_section
#' @description \code{new_section}: create [section-class] object.
#' @param paragraph character. Text for description.
#' @param code_block [code_block-class] object.
#' @rdname section-class
setMethod("new_section",
          signature = c(heading = "character", level = "numeric",
                        paragraph = "character", code_block = "maybe_code_block"),
          function(heading, level, paragraph, code_block){
            .section(heading = .heading(heading, level = level),
                     paragraph = paragraph, code_block = code_block)
          })

#' @exportMethod new_section
#' @aliases new_section
#' @rdname section-class
setMethod("new_section",
          signature = c(heading = "NULL", level = "numeric",
                        paragraph = "character", code_block = "maybe_code_block"),
          function(heading, level, paragraph, code_block){
            .section(heading = NULL, paragraph = paragraph, code_block = code_block)
          })

#' @export new_section2
#' @description \code{new_section2}: Identical to \code{new_section(NULL, , ...)}
#' @rdname section-class
new_section2 <- function(paragraph, code_block) {
  new_section(NULL, 2, paragraph, code_block)
}

#' @exportMethod call_command
#' @aliases call_command
#' @description \code{call_command}: Format 'section' object as character.
#' @rdname section-class
setMethod("call_command",
          signature = c(x = "section"),
          function(x){
            .part(call_command(heading(x)),
                  paragraph(x),
                  call_command(code_block(x))
            )
          })



#' @exportMethod call_command
#' @aliases call_command
#' @rdname section-class
setMethod("call_command",
          signature = c(x = "NULL"),
          function(x){
            return()
          })
```

16 File: class-statistic_set.R

```
# ==========================================================================
# a class for statistic analysis
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportClass statistic_set
#'
#' @aliases statistic_set
#'
#' @title Data used for statistic analysis
#'
#' @description A class object for statistic analysis, associate with package of "limma"
#' for binary comparison.
#'
#' @slot design_matrix matrix. Create by [stats::model.matrix()].
#' @slot contrast_matrix matrix. Create by [limma::makeContrasts()].
#' @slot dataset ANY. Dataset used for [limma::lmFit()], [limma::eBayes()]
#' and other functions.
#' @slot top_table list with names. Each element of list should be "data.frame" or "tbl".
#'
#' @rdname statistic_set-class
#'
.statistic_set <-
  setClass("statistic_set",
           contains = character(),
           representation =
             representation(design_matrix = "matrix",
                            contrast_matrix = "matrix",
                            dataset = "ANY",
                            top_table = "list"
                            ),
           prototype = NULL
           )
```

```
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod statistic_set
#' @aliases statistic_set
#' @description \code{statistic_set}, \code{statistic_set<-}: getter and setter
#' for the \code{statistic_set} slot of the object.
#' @rdname statistic_set-class
setMethod("statistic_set",
          signature = c(x = "ANY"),
          function(x){ x@statistic_set })
```

```r
#' @exportMethod statistic_set<-
#' @aliases statistic_set<-
#' @param value The value for the slot.
#' @rdname statistic_set-class
setReplaceMethod("statistic_set",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, statistic_set = value)
                 })


#' @exportMethod design_matrix
#' @aliases design_matrix
#' @description \code{design_matrix}, \code{design_matrix<-}: getter and setter
#' for the \code{design_matrix} slot of the object.
#' @rdname statistic_set-class
setMethod("design_matrix",
          signature = c(x = "ANY"),
          function(x){ x@design_matrix })

#' @exportMethod design_matrix<-
#' @aliases design_matrix<-
#' @param value The value for the slot.
#' @rdname statistic_set-class
setReplaceMethod("design_matrix",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, design_matrix = value)
                 })


#' @exportMethod contrast_matrix
#' @aliases contrast_matrix
#' @description \code{contrast_matrix}, \code{contrast_matrix<-}: getter and setter
#' for the \code{contrast_matrix} slot of the object.
#' @rdname statistic_set-class
setMethod("contrast_matrix",
          signature = c(x = "ANY"),
          function(x){ x@contrast_matrix })

#' @exportMethod contrast_matrix<-
#' @aliases contrast_matrix<-
#' @param value The value for the slot.
#' @rdname statistic_set-class
setReplaceMethod("contrast_matrix",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, contrast_matrix = value)
                 })


#' @exportMethod top_table
#' @aliases top_table
#' @description \code{top_table}, \code{top_table<-}: getter and setter
#' for the \code{top_table} slot of the object.
#' @rdname statistic_set-class
setMethod("top_table",
          signature = c(x = "ANY"),
          function(x){ x@top_table })

#' @exportMethod top_table<-
#' @aliases top_table<-
#' @param value The value for the slot.
#' @rdname statistic_set-class
setReplaceMethod("top_table",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, top_table = value)
                 })
```

17 File: class-VIRTUAL_slots.R

```
# ==========================================================================
# VIRTUAL classes (sharing slots and methods)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases VIRTUAL_dataset dataset
#'
#' @title Share slots and methods for classes inherite from VIRTUAL_dataset
#'
#' @description This VIRTUAL class provides a slot for storing data and methods
#' for accessing data in slot.
#'
```

```r
#' @family datasets
#'
#' @slot dataset list with names (subscript, imply file names).
#'
#' @rdname VIRTUAL_dataset-class
setClass("dataset",
         contains = character(),
         representation =
           representation("VIRTUAL",
                          dataset = "list"
                          ),
         prototype = NULL
         )

#' @aliases VIRTUAL_reference reference
#'
#' @title Share slots and methods for classes inherite from VIRTUAL_reference
#'
#' @description This VIRTUAL class provides a slot for storing processed data.
#'
#' @family references
#'
#' @slot reference list with names (formal name).
#'
#' @rdname VIRTUAL_reference-class
setClass("reference",
         contains = character(),
         representation =
           representation("VIRTUAL",
                          reference = "list"
                          ),
         prototype = NULL
         )

#' @aliases VIRTUAL_backtrack backtrack
#'
#' @title Share slots and methods for classes inherite from VIRTUAL_backtrack
#'
#' @description This VIRTUAL class provides a slot for storing discarded data.
#'
#' @family backtracks
```

```r
#'
#' @slot backtrack list with names.
#'
#' @rdname VIRTUAL_backtrack-class
setClass("backtrack",
         contains = character(),
         representation =
           representation("VIRTUAL",
                          backtrack = "list"
                          ),
         prototype = NULL
         )

#' @aliases VIRTUAL_subscript subscript
#'
#' @title Share slots and methods for classes inherite from VIRTUAL_subscript
#'
#' @description This VIRTUAL class provides a slot for signing the data.
#' The "subscript" like the signature for data, used to distinguish different data
#' or file and retrieve it accurately.
#' The "subscript" is mostly used for [project-class] (as well as its related classes):
#' - imply file names. e.g., for "sirius.v4", ".f3_fingerid" indicate all files in
#' directory of "fingerid" for each features.
#' - imply attribute names. e.g., for "sirius.v4", "tani.score" indicate attribute name
#' of "tanimotoSimilarity".
#'
#' In essence, "subscript" is the alias of a file or data or attribute.
#' In this package, using the "subscript" system means that
#' all external data names are given an alias.
#' In fact, this makes things more complicated. Why did we do this?
#' Because the naming system of external data is not constant,
#' these names may change with the version of the data source.
#' In order to enable this R package to accurately extract and call these data,
#' it is necessary to establish a set of aliases within the package.
#' "Subscript" names are used internally by this package.
#' They correspond to external data and are equivalent to providing an interface
#' to interface with external data.
#'
#' @family subscripts
#'
#' @slot subscript character(1).
```

```r
#'
#' @rdname VIRTUAL_subscript-class
setClass("subscript",
         contains = character(),
         representation =
           representation("VIRTUAL",
                          subscript = "character"
                          ),
         prototype = NULL
         )

#' @aliases VIRTUAL_export export
#'
#' @title Share slots and methods for classes inherite from VIRTUAL_export
#'
#' @description This VIRTUAL class provides slots for recording export path
#' and export name of attributes.
#'
#' @family exports
#'
#' @slot export_path character(1). The export directory path.
#' @slot export_name character with names.
#' While export, the attribute name will be converted to the value.
#'
#' @rdname VIRTUAL_export-class
setClass("export",
         contains = character(),
         representation =
           representation("VIRTUAL",
                          export_path = "character",
                          export_name = "character"
                          ),
         prototype = NULL
         )

#' @aliases VIRTUAL_layerSet layerSet
#'
#' @title Share slots and methods for classes inherite from VIRTUAL_layerSet
#'
#' @description This VIRTUAL class provides: slot \code{layers} for storing
#' hierarchical data; and methods for modify slot \code{layers}.
```

```r
#'
#' @family layerSets
#'
#' @slot layers list with names.
#'
#' @rdname VIRTUAL_layerSet-class
setClass("layerSet",
         contains = character(),
         representation =
           representation("VIRTUAL",
                          layers = "list"),
         prototype = NULL
         )
```

```r
# ==========================================================================
# validate
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
setValidity("reference",
            function(object){
              check <- vapply(reference(object), is.data.frame,
                              T, USE.NAMES = F)
              if (any(!check))
                "the elements in \"reference\" slot must be data.frame."
              else
                TRUE
            })
```

```r
# ==========================================================================
# method
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @exportMethod dataset
#' @aliases dataset
#' @description \code{dataset}, \code{dataset<-}: getter and setter
#' for the \code{dataset} slot of the object.
#' @rdname VIRTUAL_dataset-class
setMethod("dataset", "ANY",
          function(x){ x@dataset })

#' @exportMethod dataset<-
#' @aliases dataset<-
#' @param value The value for the slot.
#' @rdname VIRTUAL_dataset-class
```

```
setReplaceMethod("dataset", "ANY",
                 function(x, value){
                   initialize(x, dataset = value)
                 })

setMethod("add_dataset",
          signature = c(x = "ANY", list = "list"),
          function(x, list){
            dataset <- c(list, dataset(x))
            dataset(x) <- vecter_unique_by_names(dataset)
            return(x)
          })

#' @exportMethod reference
#' @aliases reference
#' @description \code{reference}, \code{reference<-}: getter and setter
#' for the \code{reference} slot of the object.
#' @rdname VIRTUAL_reference-class
setMethod("reference", "ANY",
          function(x){ x@reference })

#' @exportMethod reference<-
#' @aliases reference<-
#' @param value The value for the slot.
#' @param x object inherit class \code{reference}.
#' @rdname VIRTUAL_reference-class
setReplaceMethod("reference", "ANY",
                 function(x, value){
                   initialize(x, reference = value)
                 })

#' @exportMethod backtrack
#' @aliases backtrack
#' @description \code{backtrack}, \code{backtrack<-}: getter and setter
#' for the \code{backtrack} slot of the object.
#' @param x object inherit class \code{backtrack}.
#' @rdname VIRTUAL_backtrack-class
setMethod("backtrack", "ANY",
          function(x){ x@backtrack })

#' @exportMethod backtrack<-
#' @aliases backtrack<-
#' @param value The value for the slot.
#' @rdname VIRTUAL_backtrack-class
setReplaceMethod("backtrack",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, backtrack = value)
                 })

#' @exportMethod subscript
#' @aliases subscript
#' @description \code{subscript}, \code{subscript<-}: getter and setter
#' for the \code{subscript} slot of the object.
#' @rdname VIRTUAL_subscript-class
setMethod("subscript", "ANY",
          function(x){ x@subscript })

#' @exportMethod subscript<-
#' @aliases subscript<-
#' @param value The value for the slot.
#' @param x object inherit class \code{subscript}.
#' @rdname VIRTUAL_subscript-class
setReplaceMethod("subscript", "ANY",
                 function(x, value){
                   initialize(x, subscript = value)
                 })

#' @exportMethod export_name
#' @aliases export_name
#' @description \code{export_name}, \code{export_name<-}: getter and setter
#' for the \code{export_name} slot of the object.
#' @rdname VIRTUAL_export-class
setMethod("export_name",
          signature = c(x = "ANY"),
          function(x){ x@export_name })

#' @exportMethod export_name<-
#' @aliases export_name<-
#' @param value The value for the slot.
#' @param x object inherit class \code{export}.
#' @rdname VIRTUAL_export-class
```

```r
setReplaceMethod("export_name",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, export_name = value)
                 })

#' @exportMethod export_path
#' @aliases export_path
#' @description \code{export_path}, \code{export_path<-}: getter and setter
#' for the \code{export_path} slot of the object.
#' @rdname VIRTUAL_export-class
setMethod("export_path",
          signature = c(x = "ANY"),
          function(x){
            path <- x@export_path
            if (!file.exists(path))
              dir.create(path)
            path
          })

#' @exportMethod export_path<-
#' @aliases export_path<-
#' @param value The value for the slot.
#' @rdname VIRTUAL_export-class
setReplaceMethod("export_path",
                 signature = c(x = "ANY"),
                 function(x, value){
                   initialize(x, export_path = value)
                 })

#' @exportMethod layers
#' @aliases layers
#' @description \code{layers}, \code{layers<-}: getter and setter
#' for the \code{layers} slot of the object.
#' @rdname VIRTUAL_layerSet-class
setMethod("layers",
          signature = c(x = "layerSet"),
          function(x){ x@layers })

#' @exportMethod layers<-
#' @aliases layers<-
#' @param value The value for the slot.
#' @param x object inherit class \code{layerSet}.
#' @rdname VIRTUAL_layerSet-class
setReplaceMethod("layers",
                 signature = c(x = "layerSet"),
                 function(x, value){
                   initialize(x, layers = value)
                 })

#' @exportMethod show
#' @aliases show
#' @rdname VIRTUAL_layerSet-class
setMethod("show",
          signature = c(object = "layerSet"),
          function(object){
            show_layers(object)
          })

#' @exportMethod add_layers
#' @aliases add_layers
#' @description \code{add_layers}: add extra "layer" into slot \code{layers}.
#' @param x object contains slot \code{layers}.
#' @param ... extra "layer".
#' @rdname VIRTUAL_layerSet-class
setMethod("add_layers",
          signature = c(x = "layerSet"),
          function(x, ...){
            args <- list(...)
            layers(x) <- c(layers(x), args)
            return(x)
          })

#' @exportMethod delete_layers
#' @aliases delete_layers
#' @description \code{delete_layers}: delete "layer" in slot \code{layers}.
#' @param layers numeric. The specified "layer" in slot \code{layers}.
#' @rdname VIRTUAL_layerSet-class
setMethod("delete_layers",
          signature = c(x = "layerSet", layers = "numeric"),
          function(x, layers){
            layers(x)[layers] <- NULL
```

```r
            return(x)
          })

#' @exportMethod move_layers
#' @aliases move_layers
#' @description \code{move_layers}: change the order of "layer" in slot \code{layers}.
#' @param from sequence (sequence in list) of "layer" move from.
#' @param to sequence (sequence in list) of "layer" move to.
#' @rdname VIRTUAL_layerSet-class
setMethod("move_layers",
          signature = c(x = "layerSet", from = "numeric", to = "numeric"),
          function(x, from, to){
            layers(x)[c(from, to)] <- layers(x)[c(to, from)]
            return(x)
          })

#' @exportMethod insert_layers
#' @aliases insert_layers
#' @description \code{insert_layers}: Insert "layers" into the specified
#' position (sequence) of slot \code{layers}.
#' @rdname VIRTUAL_layerSet-class
setMethod("insert_layers",
          signature = c(x = "layerSet", to = "numeric"),
          function(x, to, ...){
            before <- length(layers(x))
            x <- add_layers(x, ...)
            now <- length(layers(x))
            x <- move_layers(x, to:before, (before + 1):now)
            return(x)
          })
```

18 File: data.R

```r
#' Example object containing only five 'features'.
#'
#' This is a pre-extracted data from the SIRIUS project of example data using MCnebula2,
#' containing chemical formulae, chemical structure,
#' chemical classification candidates, etc. for five 'features'
#' (It is assumed to be a pre-processed metabolomic dataset).
#' In order to reduce the memory footprint, some of its data columns have been
#' removed, for example, the 'links' data column has been converted to character(1)
#' for the chemical structure data.
#'
#' @details
#' Data extracted via MCnebula2 package from path:
#' - \code{system.file("extdata", "raw_instance.tar.gz", package = "MCnebula2")}.
#'
#' The MS/MS spectra were source from MoNA (MassBank of North America).
#' The 5 MS/MS spectra were randomly extracted from GNPS spectral library of that.
#' The candidates data were predicted via SIRIUS version 4 ...
#'
#' @format ## `mcn_5features`
#' [mcnebula-class] object.
#'
#' @source The related website:
#' - <https://mona.fiehnlab.ucdavis.edu/downloads>.
#' - <https://bio.informatik.uni-jena.de/software/sirius/>
"mcn_5features"

#' Example text for report description.
#'
#' Lazy data used to supplement the presentation of the report.
#' It doesn't make the description of the report outstanding, but it at least
#' makes it decent (maybe).
#'
#' @format ## `reportDoc`
#' A list object.
#'
"reportDoc"
```

19 File: extra-generic.R

```
# ==========================================================================
# generic for class methods
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
setGeneric("collate_data",
           signature = c(ANY = "x", "character" = "subscript",
                         "function" = "fun_collate"),
           function(x, subscript, fun_collate, ...)
             standardGeneric("collate_data"))
setGeneric("read_data",
           signature = c(ANY = "x",
                         project_metadata = "project_metadata",
                         character = "subscript",
                         character = "path",
                         character = ".features_id",
                         character = ".candidates_id",
                         "function" = "fun_read",
                         "function" = "fun_format"
                         ),
           function(x, project_metadata, subscript,
                    path, .features_id, .candidates_id,
                    fun_read, fun_format) standardGeneric("read_data"))
setGeneric("draw_structures",
           signature = c(ANY = "x", "character" = "nebula_name",
                         "character" = ".features_id",
                         "data.frame" = "data"),
           function(x, nebula_name, .features_id, data, ...)
             standardGeneric("draw_structures"))
setGeneric("show_structure",
           signature = c("ANY" = "x", "character" = ".features_id"),
           function(x, .features_id)
             standardGeneric("show_structure"))
setGeneric("draw_nodes",
           signature = c(ANY = "x", "character" = "nebula_name",
                         "character" = "nodes_color",
                         "logical" = "add_id_text",
                         "logical" = "add_structure", "logical" = "add_ppcp",
                         "logical" = "add_ration"),
           function(x, nebula_name, nodes_color, add_id_text,
                    add_structure, add_ppcp, add_ration)
             standardGeneric("draw_nodes"))
setGeneric("show_node",
           signature = c(ANY = "x", "character" = ".features_id",
                         ANY = "panel_viewport", ANY = "legend_viewport"),
           function(x, .features_id, panel_viewport, legend_viewport)
             standardGeneric("show_node"))
setGeneric("set_ppcp_data",
           signature = c(ANY = "x", "character" = "classes"),
           function(x, classes) standardGeneric("set_ppcp_data"))
setGeneric("set_ration_data",
           signature = c(ANY = "x", "logical" = "mean"),
           function(x, mean) standardGeneric("set_ration_data"))
```

```
setGeneric("set_nodes_color",
           function(x, attribute, extra_data, use_tracer)
             standardGeneric("set_nodes_color"))
setGeneric("set_tracer",
           function(x, .features_id, colors, rest)
             standardGeneric("set_tracer"))

setGeneric("binary_comparison",
           signature = c(ANY = "x", "formula" = "formula",
                         "function" = "fun_norm", "ANY" = "top_coef",
                         "ANY" = "contrasts"),
           function(x, ..., formula, fun_norm, top_coef, contrasts)
             standardGeneric("binary_comparison"))

setGeneric("get_metadata",
           signature = c(ANY = "x",
                         "character" = "subscript",
                         project_metadata = "project_metadata",
                         project_conformation = "project_conformation",
                         "character" = "project_version",
                         "character" = "path"
                         ),
           function(x, subscript, project_metadata, project_conformation,
             project_version, path)
             standardGeneric("get_metadata"))
setGeneric("extract_metadata",
           signature = c(ANY = "x", "character" = "subscript"),
           function(x, subscript) standardGeneric("extract_metadata"))

setGeneric("add_dataset",
           signature = c("ANY" = "x", "list" = "list"),
           function(x, list) standardGeneric("add_dataset"))
setGeneric("extract_rawset",
           signature = c("ANY" = "x", character = "subscript",
                         "function" = "fun_collate"),
           function(x, subscript, fun_collate, ...) standardGeneric("extract_rawset"))
setGeneric("extract_mcnset",
           signature = c("ANY" = "x", character = "subscript"),
           function(x, subscript) standardGeneric("extract_mcnset"))

setGeneric("get_upper_dir_subscript",
```

```r
           signature = c(ANY = "x",
                         character = "subscript",
                         project_conformation = "project_conformation"
                         ),
           function(x, subscript, project_conformation)
             standardGeneric("get_upper_dir_subscript"))

setGeneric("latest",
           function(x, slot, subscript) standardGeneric("latest"))

## rename the colnames and check the values type (character or interger, etc.)
setGeneric("format_msframe",
           signature = c("ANY" = "x",
                         character = "names", "function" = "fun_names",
                         character = "types", "function" = "fun_types",
                         "function" = "fun_format"
                         ),
           function(x, names, fun_names, types, fun_types, fun_format)
             standardGeneric("format_msframe"))
setGeneric("filter_msframe",
           signature = c(msframe = "x", "function" = "fun_filter",
                         "formula" = "f"),
           function(x, fun_filter, f, ...) standardGeneric("filter_msframe"))
```

```r
# ==========================================================================
# for report and ggset
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

setGeneric("new_command",
           signature = c("function" = "fun",
                         "character" = "name"),
           function(fun, ..., name)
             standardGeneric("new_command"))
setGeneric("call_command",
           function(x) standardGeneric("call_command"))

setGeneric("new_code_block",
           signature = c(character = "language", "character" = "codes",
                         "list" = "args", "logical" = "prettey",
                         "function" = "fun_prettey"),
           function(language, codes, args, prettey, fun_prettey)
             standardGeneric("new_code_block"))
```

```r
setGeneric("new_code_block_table",
           signature = c(character = "name"),
           function(name, ...)
             standardGeneric("new_code_block_table"))
setGeneric("new_code_block_figure",
           signature = c(character = "name"),
           function(name, caption, ...)
             standardGeneric("new_code_block_figure"))
setGeneric("include_table",
           function(data, name, caption)
             standardGeneric("include_table"))
setGeneric("include_figure",
           function(file, name, caption)
             standardGeneric("include_figure"))
setGeneric("history_rblock",
           function(nrow, pattern_start, pattern_end, exclude)
             standardGeneric("history_rblock"))

setGeneric("new_heading",
           function(heading, level)
             standardGeneric("new_heading"))
setGeneric("new_section",
           signature = c(character = "heading", "numeric" = "level",
                         "character" = "paragraph", "ANY" = "code_block"),
           function(heading, level, paragraph, code_block)
             standardGeneric("new_section"))
setGeneric("new_report",
           function(..., yaml)
             standardGeneric("new_report"))

setGeneric("new_ggset",
           function(...) standardGeneric("new_ggset"))
setGeneric("show_layers",
           function(x) standardGeneric("show_layers"))
setGeneric("add_layers",
           signature = c(ANY = "x"),
           function(x, ...) standardGeneric("add_layers"))
setGeneric("delete_layers",
           signature = c(ANY = "x", "numeric" = "layers"),
           function(x, layers) standardGeneric("delete_layers"))
setGeneric("move_layers",
           signature = c(ANY = "x", "numeric" = "from", "numeric" = "to"),
           function(x, from, to)
             standardGeneric("move_layers"))
setGeneric("insert_layers",
           signature = c(ANY = "x", "numeric" = "to"),
           function(x, to, ...) standardGeneric("insert_layers"))
setGeneric("mutate_layer",
           signature = c(ANY = "x", "ANY" = "layer"),
           function(x, layer, ...)
             standardGeneric("mutate_layer"))

setGeneric("workflow",
  function(sections, mode, envir, sirius_version, sirius_project, ion_mode, ...)
    standardGeneric("workflow"))
```

20 File: extraMethods-binary_comparison.R

```r
# ==========================================================================
# use 'limma' package to conduct binary comparison between sample group
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases binary_comparison
#'
#' @title Binary comparison for 'features' quantification data
#'
#' @description
#' Use the functions in the 'limma' package for simple binary statistical analysis.
#'
#' @name binary_comparison-methods
#'
#' @seealso [stats::model.matrix()], [limma::makeContrasts()], [limma::lmFit()],
#' [limma::eBayes()], [limma::contrasts.fit()], [limma::topTable()]...
#'
#' @order 1
NULL
#> NULL

#' @exportMethod binary_comparison
#' @description \code{binary_comparison()}:
#' get the default parameters for the method
#' \code{binary_comparison}.
#' @rdname binary_comparison-methods
```

```r
setMethod("binary_comparison",
          signature = setMissing("binary_comparison",
                                 x = "missing"),
          function(){
            list(formula = ~ 0 + group,
                 fun_norm = function(x) {
                   scale(log2(x + 1), center = T, scale = F)
                 },
                 top_coef = "all"
            )
          })

#' @exportMethod binary_comparison
#' @description \code{binary_comparison(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{binary_comparison}.
#' @rdname binary_comparison-methods
setMethod("binary_comparison",
          signature = c(x = "ANY"),
          function(x, ..., formula, fun_norm, top_coef, contrasts){
            if (missing(...) & missing(contrasts)) {
              stop("`...` (group contrast) should be specified, ",
                   "e.g., model - control")
            } else if (missing(contrasts)) {
              contrasts <- as.character(substitute(list(...)))[-1]
            }
            reCallMethod("binary_comparison",
                         .fresh_param(binary_comparison()))
          })

#' @exportMethod binary_comparison
#'
#' @aliases binary_comparison
#'
#' @param x [mcnebula-class] object.
#' @param ... expressions, or character strings which can be parsed to
#' expressions, specifying contrasts. See parameter of \code{...} in
#' [limma::makeContrasts()].
#'
#' @param formula formula. Passed to [model.matrix()].
#' @param fun_norm function. For normalization of 'features' quantification
#' data.
#'
#' @param top_coef list, NULL or character(1). Specified the parameter of
#' \code{coef} in [limma::topTable()]. If \code{"all"}, all coefficient in
#' contrast matrix would be used one by one.
#'
#' @param contrasts character vector specifying contrasts.
#' See parameter \code{contrasts} in [limma::makeContrasts()].
#'
#' @rdname binary_comparison-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_features_annotation(test1)
#'
#'   ## set up a simulated quantification data.
#'   test1 <- .simulate_quant_set(test1)
#'   ## the simulated data
#'   features_quantification(test1)
#'   sample_metadata(test1)
#'
#'   test1 <- binary_comparison(
#'     test1, control - model,
#'     model - control, 2 * model - control
#'   )
#'   ## see results
#'   top_table(statistic_set(test1))
#'
#'   ## the default parameters
#'   binary_comparison()
#' }
setMethod("binary_comparison",
          signature = c(x = "ANY", formula = "formula",
                        fun_norm = "function", top_coef = "ANY",
                        contrasts = "character"),
function(x, formula, fun_norm, top_coef, contrasts){
  .message_info_formal("MCnebula2", "binary_comparison")
  .suggest_bio_package("limma")
  .check_data(x, list(features_quantification = "features_quantification",
                      sample_metadata = "sample_metadata"), "(x) <-")
  ## design matrix
  design <- model.matrix(formula, sample_metadata(x))
  colnames(design) <- gsub("group", "", colnames(design))
  design_matrix(statistic_set(x)) <- design
  ## contrast matrix
  contrast <-
    limma::makeContrasts(contrasts = contrasts, levels = design)
  contrast_matrix(statistic_set(x)) <- contrast
  ## format data
  features_quantification(x) <- dplyr::select(
    features_quantification(x), .data$.features_id,
    dplyr::all_of(sample_metadata(x)$sample)
  )
  data <- fun_norm(.features_quantification(x))
  col <- colnames(data)
  col <- col %in% sample_metadata(x)$sample
  data <- data[, col]
  ## fit with linear model
  fit <- limma::lmFit(data, design)
  if (!formula == ~ 0 + group) {
    dataset(statistic_set(x)) <- fit
    message("identical(`formula`, ~ 0 + group) == F, ",
                    "stop downstream analysis.",
                    "\n\tuse `dataset(statistic_set(x))` ",
                    "to get `limma::limFit(data, design)` output object ",
                    "for custom 'limma' analysis.")
    return(x)
  }
  fit <- limma::contrasts.fit(fit, contrast)
  fit <- limma::eBayes(fit)
  dataset(statistic_set(x)) <- fit
  if (is.null(top_coef)) {
    return(x)
  } else if (any(top_coef == "all")) {
    top_coef <- as.list(1:ncol(contrast))
  } else if (!is.list(top_coef)) {
```

```
              top_coef <- list(top_coef)
            }
            lst <-
              lapply(top_coef,
                     function(i){
                       top <- limma::topTable(fit, coef = i, adjust = "BH",
                                              number = Inf)
                       top <- dplyr::mutate(top, .features_id = rownames(top))
                       dplyr::as_tibble(dplyr::relocate(top, .features_id))
                     })
            names(lst) <-
              vapply(top_coef, FUN.VALUE = "ch", USE.NAMES = F,
                     function(i){
                       paste0(colnames(contrast)[i], collapse = " & ")
                     })
            top_table(statistic_set(x)) <- lst
            return(x)
          })
```

21 File: extraMethods-collate_data.R

```
# ==========================================================================
# collate any dataset in target project without filtering or arranging,
# relative to class-project
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases collate_data
#'
#' @title Extract and format data from raw project directory
#'
#' @description
#' The primary method used to extract data from the raw project directory.
#' By specifying [subscript-class], this method reads all corresponding files,
#' followed by gathering and formating the data, then stores these data in the slot
#' (\code{dataset(project_dataset(object))}).
#'
#' @note Normally, users do not need to use this method for MCnebula2 analysis.
#' [filter_formula()], [filter_structure()], [filter_ppcp()]
#' provide more understandable usage.
#'
#' @details
#' This methods requires the name and path of the file in the raw project directory,
```

```r
#' as well as the reading function; These are recorded in [project-class].
#'
#' @name collate_data-methods
#'
#' @order 1
NULL
#> NULL

#' @importFrom dplyr mutate
#' @importFrom dplyr relocate
#' @importFrom dplyr select
#' @importFrom dplyr arrange
#' @importFrom dplyr distinct
#' @importFrom dplyr filter
#' @importFrom dplyr rename
#' @importFrom tibble as_tibble
#' @importFrom tibble tibble
#' @importFrom data.table rbindlist
#' @importFrom data.table fread
#' @importFrom stringr str_extract
#' @exportMethod collate_data
#' @description \code{collate_data()}: get the default parameters for the method
#' \code{collate_data}.
#' @rdname collate_data-methods
setMethod("collate_data",
          signature = setMissing("collate_data"),
          function(){
            list(fun_collate = .collate_data.msframe)
          })

#' @exportMethod collate_data
#' @description \code{collate_data(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{collate_data}.
#' @rdname collate_data-methods
setMethod("collate_data",
          signature = c(x = "ANY"),
          function(x, subscript, fun_collate, ...){
            reCallMethod("collate_data",
                         .fresh_param(collate_data()), ...)
          })
```

```
#' @exportMethod collate_data
#'
#' @aliases collate_data
#'
#' @param x [project-class] object or other class object inheriting it.
#' @param subscript character(1). See [subscript-class].
#' @param fun_collate function.
#' Used to extract and format the data from raw project directory.
#' The default is \code{MCnebula2:::.collate_data.msframe}.
#' @param ... Other parameters passed to the fun_collate.
#'
#' @rdname collate_data-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)
#'
#'   ## extract candidates data in SIRIUS project directory
#'   ## chemical structure
#'   test <- collate_data(test, ".f3_fingerid")
#'   latest(project_dataset(test))
#'
#'   ## chemical formula
#'   test <- collate_data(test, ".f2_formula")
#'   latest(project_dataset(test))
#'
#'   ## chemical classes
#'   test <- collate_data(test, ".f3_canopus")
#'   latest(project_dataset(test))
#'
#'   ## mz and rt
#'   test <- collate_data(test, ".f2_info")
#'   latest(project_dataset(test))
#'
#'   ## classification description
#'   test <- collate_data(test, ".canopus")
#'
#'   ## the extracted data in 'mcnebula'
#'   dataset(project_dataset(test))
#'   entity(dataset(project_dataset(test))$.f3_fingerid)
#'
#'   unlink(tmp, T, T)
#' }
setMethod("collate_data",
          signature = c(x = "ANY", subscript = "character",
                        fun_collate = "function"),
          function(x, subscript, fun_collate, ...){
            x <- get_metadata(x, subscript)
            msframe.lst <- extract_rawset(x, subscript, fun_collate, ...)
            project_dataset(x) <- add_dataset(project_dataset(x), msframe.lst)
            return(x)
          })

.collate_data.msframe <-
  function(x, subscript, reference){
    project_metadata <- extract_metadata(x, subscript)
    if (!missing(reference)) {
      df <- metadata(project_metadata)[[ subscript ]]
      df <- dplyr::mutate(df, .features_id = match.features_id(x)(upper),
                          .candidates_id = match.candidates_id(x)(files))
      df <- merge(reference, df, by = c(".features_id", ".candidates_id"))
      metadata(project_metadata)[[ subscript ]] <- df
    }
    read_data(x, project_metadata = project_metadata,
              subscript = subscript)
  }


# @exportMethod read_data
#' @description \code{read_data}: basic methods used to extract and format
#' data from raw project directory.
#'
```

```r
#' @param project_metadata [project_metadata-class] object. Specifying the files to read.
#' @param path character(1). The path of raw project directory.
#' @param .features_id character. ID for signing files in sub-directory of each 'features'.
#' @param .candidates_id character. ID for signing each candidates of 'features'.
#' @param fun_read function. Used to read files from raw project directory.
#' @param fun_format function. Used to format the data.
#'
#' @rdname collate_data-methods
#' @noRd
#'
setMethod("read_data",
          signature = setMissing("read_data",
                                 x = "ANY",
                                 project_metadata = "project_metadata",
                                 subscript = "character"),
          function(x, project_metadata, subscript){
            path.df <- metadata(project_metadata)[[ subscript ]]
            path <- paste0(project_path(x), "/", path.df$upper, "/", path.df$files)
            fun_read <- methods_read(x)[[ paste0("read", subscript) ]]
            fun_format <- methods_format(x)
            .features_id <- match.features_id(x)(path.df$upper)
            if (length(.features_id) == 0)
              .features_id <- subscript
            .candidates_id <- match.candidates_id(x)(path.df$files)
            .message_info("collate_data", "read_data", subscript)
            msframe <- read_data(path = path, fun_read = fun_read,
                                 subscript = subscript, fun_format = fun_format,
                                 .features_id = .features_id,
                                 .candidates_id = .candidates_id
            )
          })

# @exportMethod read_data
#' @rdname collate_data-methods
#' @noRd
setMethod("read_data",
          signature = setMissing("read_data",
                      subscript = "character", path = "character",
                      .features_id = "character", .candidates_id = "character",
                      fun_read = "function", fun_format = "function"),
          function(subscript, path,
                    .features_id, .candidates_id,
                    fun_read, fun_format){
             entity <- fun_read(path)
             if (is.data.frame(entity)) {
               ## a 'data.table' may cause error
               entity <- list(data.frame(entity))
               names(entity) <- subscript
             }
             entity <- mapply(entity, .features_id, .candidates_id,
                              SIMPLIFY = F,
                              FUN = function(df, .features_id, .candidates_id){
                                dplyr::mutate(df, .features_id = .features_id,
                                              .candidates_id = .candidates_id)
                              })
             entity <- dplyr::relocate(data.table::rbindlist(entity, fill = T),
                                       .features_id, .candidates_id)
             msframe <- new("msframe", subscript = subscript, entity = entity)
             fun_format(msframe)
           })

#' @export collate_used
#' @aliases collate_used
#' @description \code{collate_used}: Use [filter_structure()] and [create_reference()]
#' to build 'specific_candidate' data, then collate all used data of MCnebula workflow
#' from Project directory, for subsequent data processing.
#' @rdname collate_data-methods
collate_used <- function(x) {
  x <- filter_structure(x)
  x <- create_reference(x)
  sub1 <- c(".f2_formula",  ".canopus", ".f2_info", ".f2_msms")
  sub2 <- c(".f3_canopus", ".f3_spectra")
  for (i in sub1) {
    x <- collate_data(x, i)
    message()
  }
  for (i in sub2) {
    x <- collate_data(x, i, reference = specific_candidate(x))
    message()
  }
  return(x)
}
```

22 File: extraMethods-draw__nodes.R

```
# ==========================================================================
# draw all nodes (with annotation) for a specified child-nebula
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases draw_nodes
#'
#' @title Draw and visualize chemcial structures for Child-Nebulae
#'
#' @description
#' Methods used for drawing and visualizing nodes of 'features'
#' in Child-Nebulae (networks). The methods used to visualize 'features'
#' with annotations of:
#' - chemical structures
#' - chemical classification
#' - quantification data (peak area)
#' - ID of 'feature' (.features_id)
#'
#' @details
#' Those annotated visualizations are drawn in steps and then are put together.
#' In order to render the text as a graphical path (otherwise, the graphics
#' would not be compatible with too small fonts and would result in misplaced text),
#' the 'ggplot' object or 'grob' object is first exported as an SVG file,
#' which is subsequently read by [grImport2::readPicture()], followed by
#' [grImport2::grobify()] as 'grob' object, and then combined into
#' the final 'grob'. In general, this process is time consuming,
#' especially when there are a lot of 'features' for visualization.
#'
#' @seealso [grid::grid.draw()], [grid::grob()], [grImport2::readPicture()],
#' [grImport2::grobify()]...
#'
#' @name draw_nodes-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod draw_nodes
#' @description \code{draw_nodes()}: get the function for generating
#' default parameters for the method
#' \code{draw_nodes}.
#' @rdname draw_nodes-methods
```

```
setMethod("draw_nodes",
          signature = setMissing("draw_nodes",
                                 x = "missing"),
          function(){
            function(x) {
              if (!is.null(nebula_index(x)[[ "tracer_color" ]])) {
                nodes_color <- nebula_index(x)[[ "tracer_color" ]]
                names(nodes_color) <- nebula_index(x)[[ ".features_id" ]]
              } else {
                nodes_color <- "#FFF9F2"
              }
              list(nodes_color = nodes_color,
                   add_id_text = T,
                   add_structure = T,
                   add_ppcp = T,
                   add_ration = T
              )
            }
          })

#' @exportMethod draw_nodes
#' @description \code{draw_nodes(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{draw_nodes}.
#' @rdname draw_nodes-methods
setMethod("draw_nodes",
          signature = c(x = "mcnebula", nebula_name = "character"),
          function(x, nebula_name, nodes_color, add_id_text,
                   add_structure, add_ppcp, add_ration){
            reCallMethod("draw_nodes",
                         .fresh_param(draw_nodes()(x)))
          })

#' @importFrom grDevices colorRampPalette dev.off
#' @importFrom svglite svglite
#' @importFrom grid pushViewport
#' @importFrom grid viewport
#' @importFrom grid popViewport
#' @importFrom pbapply pblapply
#' @importFrom tibble as_tibble
#' @importFrom rsvg rsvg_svg
#' @exportMethod draw_nodes
```

```
#'
#' @aliases draw_nodes
#'
#' @param x [mcnebula-class] object.
#' @param nebula_name character(1). Chemical classes in 'nebula_index' data.
#' Specified to draw nodes (of network) of all the 'features' of that.
#'
#' @param nodes_color character with names or not. The Value is Hex color.
#' Specified colors for 'features' to draw nodes. If the number of the colors
#' were not enough, the rest 'features' would be fill with default color.
#' If [set_tracer()] has been run, the colors specified in 'nebula_index'
#' would be used preferentially.
#'
#' @param add_id_text logical. If \code{TRUE}, add ID (.features_id) for
#' 'features' inside the nodes.
#'
#' @param add_structure logical. If \code{TRUE}, draw chemical structures inside
#' the nodes. See [draw_structures()].
#'
#' @param add_ppcp logical. If \code{TRUE}, draw radical bar plot inside the nodes
#' for annotation of PPCP data. See [set_ppcp_data()] for custom modify the annotated
#' PPCP data. Hex colors in \code{palette_col(object)} would be used for fill the bar
#' plot (Used by [ggplot2::scale_fill_manual()]).
#'
#' @param add_ration logical. If \code{TRUE}, draw ring plot inside the nodes
#' for annotation of features quantification data. See [set_ration_data()] for custom
#' modify the annotated quantification data. Hex colors in \code{palette_stat(object)}
#' would be used for fill be ring plot.
#'
#' @rdname draw_nodes-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_child_nebulae(test1, 0.01)
#'   test1 <- create_child_layouts(test1)
#'   test1 <- activate_nebulae(test1)
#'
#'   ## set features quantification data
#'   ids <- features_annotation(test1)$.features_id
#'   quant. <- data.frame(
#'     .features_id = ids,
#'     sample_1 = rnorm(length(ids), 1000, 200),
#'     sample_2 = rnorm(length(ids), 2000, 500)
#'   )
#'   metadata <- data.frame(
#'     sample = paste0("sample_", 1:2),
#'     group = c("control", "model")
#'   )
#'   features_quantification(test1) <- quant.
#'   sample_metadata(test1) <- metadata
#'
#'   ## optional 'nebula_name'
#'   visualize(test1)
#'   ## a class for example
#'   class <- visualize(test1)$class.name[1]
#'   tmp <- export_path(test1)
#'   test1 <- draw_structures(test1, class)
#'   test1 <- draw_nodes(test1, class)
#'
#'   ## see results
#'   grobs <- nodes_grob(child_nebulae(test1))
#'   grobs
#'   grid::grid.draw(grobs[[1]])
#'   ## visualize with ID of 'feature' (.features_id)
#'   ## with legend
#'   ids <- names(grobs)
#'   x11(width = 9, height = 5)
#'   show_node(test1, ids[1])
#'
#'   ## default parameters
#'   draw_nodes()
```

```
#'
#'   unlink(tmp, T, T)
#' }
setMethod("draw_nodes",
          signature = c(x = "mcnebula", nebula_name = "character",
                        nodes_color = "character",
                        add_id_text = "logical",
                        add_structure = "logical",
                        add_ppcp = "logical",
                        add_ration = "logical"),
          function(x, nebula_name, nodes_color, add_id_text,
                   add_structure, add_ppcp, add_ration){
            if (add_ppcp) {
              if (length(ppcp_data(child_nebulae(x))) == 0)
                x <- set_ppcp_data(x)
            }
            if (add_ration) {
              if (length(ration_data(child_nebulae(x))) == 0)
                x <- set_ration_data(x)
            }
            if (add_structure) {
              if (length(ppcp_data(child_nebulae(x))) == 0)
                x <- draw_structures(x, nebula_name)
            }
            .features_id <-
              `[[`(tibble::as_tibble(tbl_graph(child_nebulae(x))[[nebula_name]]),
                   "name")
            .features_id <-
              unlist(lapply(.features_id,
                            function(id){
                              if (is.null(nodes_ggset(child_nebulae(x))[[id]]))
                                id
                            }))
            if (is.null(.features_id)) {
              return(x)
            }
            ggsets <- ggset_activate_nodes(x, .features_id, nodes_color,
                                           add_ppcp, add_ration)
            nodes_ggset(child_nebulae(x)) <-
              c(nodes_ggset(child_nebulae(x)), ggsets)
            path <- paste0(export_path(x), "/tmp/nodes")
            .check_path(path)
            .message_info("draw_nodes", "ggplot -> svg -> grob")
            grImport2:::setPrefix("")
            nodes_grob <-
              pbapply::pbsapply(names(ggsets), simplify = F,
                                function(id){
                                  file <- paste0(path, "/", id, ".svg")
                                  svglite::svglite(file, bg = "transparent")
                                  ggset <- modify_rm_legend(ggsets[[ id ]])
                                  ggset <- modify_set_margin(ggset)
                                  print(call_command(ggset))
                                  if (add_structure) {
                                    vp <- grid::viewport(width = 0.8, height = 0.8)
                                    grid::pushViewport(vp)
                                    show_structure(x, id)
                                    grid::popViewport()
                                  }
                                  if (add_id_text) {
                                    label <- paste0("ID: ", id)
                                    grid::grid.draw(.grob_node_text(label))
                                  }
                                  dev.off()
                                  rsvg::rsvg_svg(file, file)
                                  .cairosvg_to_grob(file)
                                })
            nodes_grob(child_nebulae(x)) <-
              c(nodes_grob(child_nebulae(x)), nodes_grob)
            return(x)
          })

#' @exportMethod show_node
#' @description \code{show_node()}: get the default parameters for the method
#' \code{show_node}.
#' @rdname draw_nodes-methods
setMethod("show_node",
          signature = setMissing("show_node",
                                 x = "missing"),
          function(){
            list(panel_viewport =
                 grid::viewport(0, 0.5, 0.4, 1, just = c("left", "centre")),
               legend_viewport =
                 grid::viewport(0.4, 0.5, 0.6, 1, just = c("left", "centre"))
             )
           })

#' @exportMethod show_node
#'
#' @aliases show_node
#'
#' @description Visualize the node of 'feature' which has been drawn
#' by methods [draw_nodes()] (or drawn by methods [annotate_nebula()]).
#' @description \code{show_node(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{show_node}.
#'
#' @param x [mcnebula-class] object.
#' @param .features_id character(1). ID of 'feature' to show node.
#' @param panel_viewport 'viewport' object. Create by [grid::viewport()].
#' @param legend_viewport 'viewport' object.
#'
#' @rdname draw_nodes-methods
#'
setMethod("show_node",
          signature = c(x = "ANY", .features_id = "character"),
          function(x, .features_id, panel_viewport, legend_viewport){
            args <- .fresh_param(show_node())
            args$.features_id <- .features_id
            do.call(.show_node, args)
          })

.show_node <-
  function(x, .features_id, panel_viewport, legend_viewport){
    grob <- nodes_grob(child_nebulae(x))[[.features_id]]
    if (is.null(grob))
      stop("the node of `.features_id` has not been drawn")
    .message_info_viewport("BEGIN")
    if (!is.null(panel_viewport)) {
      .check_class(panel_viewport, "viewport", "grid::viewport")
      grid::pushViewport(panel_viewport)
      upper <- T
    } else {
      upper <- F
    }
```

```r
    .message_info_viewport()
    grid::grid.draw(grob)
    if (upper) {
      grid::upViewport()
    } else {
      return(message(""))
    }
    if (!is.null(legend_viewport)) {
      .check_class(legend_viewport, "viewport", "grid::viewport")
      .message_info_viewport()
      grid::pushViewport(legend_viewport)
      p <- call_command(nodes_ggset(child_nebulae(x))[[.features_id]])
      grid::grid.draw(.get_legend(p))
    }
    .message_info_viewport("END")
  }

#' @description \code{ggset_activate_nodes}:
#' create the [ggset-class] object of node of specified 'feature'.
#' @rdname draw_nodes-methods
#' @export
ggset_activate_nodes <-
  function(x, .features_id, nodes_color = "#FFF9F2",
           add_ppcp = T, add_ration = T){
    if (add_ppcp) {
      .check_data(child_nebulae(x), list(ppcp_data = "set_ppcp_data"))
    }
    nodes_color <- .as_dic(nodes_color, .features_id, "#FFF9F2")
    set <- .prepare_data_for_nodes(x, .features_id, add_ppcp)
    ggsets <-
      sapply(.features_id, simplify = F,
             function(id) {
               names <- paste0(set[[id]]$rel.index)
               pal <- .as_dic(palette_col(x), names,
                              fill = F, as.list = F, na.rm = T)
               labels <- .as_dic(paste0("Bar: ", names, ": ", set[[id]]$class.name),
                                 names, fill = F, as.list = F)
               new_ggset(new_command(ggplot, set[[id]]),
                         .command_node_nuclear(nodes_color[[id]]),
                         .command_node_border(),
                         .command_node_radial_bar(),
                    .command_node_ylim(),
                    .command_node_polar(),
                    .command_node_fill(pal, labels),
                    new_command(theme_void),
                    .command_node_theme()
          )
        })
  if (add_ration) {
    .check_data(child_nebulae(x), list(ration_data = "set_ration_data"))
    axis.len <- vapply(set, function(df) tail(df$seq, n = 1), 1) + 1
    set <- .prepare_data_for_ration(x, .features_id, axis.len)
    group <- unique(sample_metadata(x)$group)
    pal.ex <- .as_dic(palette_stat(x), group,
                      fill = F, as.list = F, na.rm = T)
    labels.ex <- .as_dic(paste0("Ring: group: ", group), group,
                         fill = F, as.list = F)
    ggsets <-
      sapply(.features_id, simplify = F,
             function(id) {
               if (is.null(set[[id]]))
                 return(ggsets[[id]])
               ggset <- add_layers(
                 ggsets[[id]],
                 .command_node_ration(set[[id]]),
                 new_command(labs, fill = "Groups / Classes"),
                 ## this as a separator
                 new_command(geom_point, mapping = aes(x = seq, y = 0, fill = " "),
                   data = data.frame(seq = 1L), stroke = 0, size = 0, shape = 21
                 )
               )
               scale <- command_args(layers(ggset)$scale_fill_manual)
               command_args(layers(ggset)$scale_fill_manual)$values <-
                 c(pal.ex, c(" " = "white"), scale$values)
               command_args(layers(ggset)$scale_fill_manual)$labels <-
                 c(labels.ex, c(" " = " "), scale$labels)
               ## control sequence
               command_args(layers(ggset)$scale_fill_manual)$breaks <-
                 names(c(pal.ex, c(" " = "white"), scale$values))
               ggset
             })
  }
```

```
    ggsets
  }

#' @importFrom dplyr mutate
.prepare_data_for_nodes <-
  function(x, .features_id, add_ppcp = T){
    df <- data.frame(rel.index = -1L, pp.value = 0L, seq = 1:3)
    if (add_ppcp) {
      set <- ppcp_data(child_nebulae(x))
      set <- sapply(.features_id, simplify = F,
                    function(id) {
                      if (is.null(set[[id]]))
                        df
                      else
                        set[[id]]
                    })
    } else {
      set <- sapply(.features_id, function(id) df, simplify = F)
    }
    set
  }

.prepare_data_for_ration <-
  function(x, .features_id, axis.len){
    set <- ration_data(child_nebulae(x))
    sapply(.features_id, simplify = F,
           function(id) {
             if (is.null(set[[id]]))
               return()
             df <- set[[id]]
             max <- cumsum(df$value)
             min <- c(0, max[-length(max)])
             factor <- axis.len[[id]] / max(max)
             df$x <- (min + df$value / 2) * factor
             df$width <- df$value * factor
             df
           })
  }

#' @aliases set_ppcp_data
#'
```

```r
#' @title Custom specify PPCP data for visualization in nodes
#'
#' @description
#' Run before [annotate_nebula()] or [draw_nodes()].
#' Custom specify PPCP data for visualization in nodes.
#' All chemical classes exists in PPCP data could be specified.
#'
#' @seealso [annotate_nebula()], [draw_nodes()].
#'
#' @name set_ppcp_data-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod set_ppcp_data
#' @description \code{set_ppcp_data()}: get the function for generating
#' default parameters for the method
#' \code{set_ppcp_data}.
#' @rdname set_ppcp_data-methods
setMethod("set_ppcp_data",
          signature = setMissing("set_ppcp_data"),
          function(){
            function(x){
              list(classes = names(tbl_graph(child_nebulae(x))))
            }
          })

#' @importFrom dplyr filter
#' @importFrom dplyr mutate
#' @exportMethod set_ppcp_data
#' @description \code{set_ppcp_data(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{set_ppcp_data}.
#' @rdname set_ppcp_data-methods
setMethod("set_ppcp_data",
          signature = c(x = "mcnebula"),
          function(x, classes){
            reCallMethod("set_ppcp_data",
                         .fresh_param(set_ppcp_data()(x)))
          })
```

```
#' @exportMethod set_ppcp_data
#'
#' @param x [mcnebula-class] object.
#' @param classes character. The names of chemical classes.
#' Use \code{classification(object)} to get optional candidates.
#'
#' @rdname set_ppcp_data-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_child_nebulae(test1, 0.01)
#'   test1 <- create_child_layouts(test1)
#'   test1 <- activate_nebulae(test1)
#'
#'   ## optional 'nebula_name'
#'   visualize(test1)
#'   ## a class for example
#'   class <- visualize(test1)$class.name[1]
#'   tmp <- export_path(test1)
#'   ## customize the chemical classes displayed
#'   ## in the radial bar plot in node.
#'   classes <- classification(test1)
#'   ## get some random classes
#'   set.seed(10)
#'   classes <- sample(classes$class.name, 50)
#'   classes
#'   test1 <- set_ppcp_data(test1, classes)
#'   test1 <- draw_nodes(test1, class,
#'     add_structure = F,
#'     add_ration = F
```

```r
#'   )
#'
#'   ## visualize with ID of 'feature' (.features_id)
#'   ## with legend
#'   ids <- names(nodes_grob(child_nebulae(test1)))
#'   x11(width = 15, height = 5)
#'   show_node(test1, ids[1])
#'
#'   ## get a function to generate default parameters
#'   set_ppcp_data()
#'   ## the default parameters
#'   set_ppcp_data()(test1)
#'
#'   unlink(tmp, T, T)
#' }
setMethod("set_ppcp_data",
          signature = c(x = "mcnebula", classes = "character"),
          function(x, classes){
            ppcp_data <-
              suppressMessages(latest(filter_ppcp(x, dplyr::filter,
                                                  class.name %in% classes)))
            ppcp_data <- dplyr::select(ppcp_data, rel.index, class.name,
                                       pp.value, .features_id)
            ppcp_data(child_nebulae(x)) <-
              lapply(split(ppcp_data, ~ .features_id),
                     function(df) {
                       dplyr::mutate(df, seq = 1:nrow(df))
                     })
            return(x)
          })

#' @aliases set_ration_data
#'
#' @title Custom specify the quantification data for visualization in nodes
#'
#' @description
#' Run before [annotate_nebula()] or [draw_nodes()].
#' Set whether to use the group average value to annotate the 'features'
#' quantification in nodes.
#' Before this methods, user should use \code{features_quantification<-} and
#' \code{sample_metadata<-} to set quantification data and metadata in
#' [mcnebula-class] object.
#'
#' @seealso [annotate_nebula()], [draw_nodes()].
#'
#' @name set_ration_data-methods
#'
#' @order 1
NULL
#> NULL

#' @importFrom tidyr gather
#' @importFrom tibble as_tibble
#' @importFrom dplyr group_by
#' @importFrom dplyr summarise
#' @importFrom dplyr ungroup
#' @exportMethod set_ration_data
#' @description \code{set_ration_data()}: get the default parameters for the method
#' \code{set_ration_data}.
#' @rdname set_ration_data-methods
setMethod("set_ration_data",
          signature = setMissing("set_ration_data"),
          function(){
            list(mean = T)
          })

#' @exportMethod set_ration_data
#' @description \code{set_ration_data(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{set_ration_data}.
#' @rdname set_ration_data-methods
setMethod("set_ration_data",
          signature = c(x = "mcnebula"),
          function(x, mean){
            reCallMethod("set_ration_data",
                         .fresh_param(set_ration_data()))
          })

#' @exportMethod set_ration_data
#'
#' @param x [mcnebula-class] object.
#' @param mean logical. If \code{TRUE}, calculate mean value for
#' all group of the samples.
```

```
#'
#' @rdname set_ration_data-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_child_nebulae(test1, 0.01)
#'   test1 <- create_child_layouts(test1)
#'   test1 <- activate_nebulae(test1)
#'
#'   ## set features quantification data
#'   ids <- features_annotation(test1)$.features_id
#'   quant. <- data.frame(
#'     .features_id = ids,
#'     sample_1 = rnorm(length(ids), 1000, 200),
#'     sample_2 = rnorm(length(ids), 2000, 500)
#'   )
#'   quant. <- dplyr::mutate(quant.,
#'     sample_3 = sample_1 * 1.5,
#'     sample_4 = sample_2 * 5
#'   )
#'   metadata <- data.frame(
#'     sample = paste0("sample_", 1:4),
#'     group = rep(c("control", "model"), c(2, 2))
#'   )
#'   features_quantification(test1) <- quant.
#'   sample_metadata(test1) <- metadata
#'
#'   ## a more convenient way to obtain simulation data
#'   # test1 <- MCnebula2:::.simulate_quant_set(test1)
#'
```

```
#'   ## optional 'nebula_name'
#'   visualize(test1)
#'   ## a class for example
#'   class <- visualize(test1)$class.name[1]
#'   tmp <- export_path(test1)
#'
#'   test1 <- set_ration_data(test1, mean = F)
#'   test1 <- draw_nodes(test1, class,
#'     add_structure = F,
#'     add_ppcp = F
#'   )
#'
#'   ## visualize with ID of 'feature' (.features_id)
#'   ## with legend
#'   ids <- names(nodes_grob(child_nebulae(test1)))
#'   x11(width = 15, height = 5)
#'   show_node(test1, ids[1])
#'
#'   ## the default parameters
#'   set_ration_data()
#'
#'   unlink(tmp, T, T)
#' }
setMethod("set_ration_data",
          signature = c(x = "mcnebula", mean = "logical"),
          function(x, mean){
            .check_data(x, list(features_quantification = "features_quantification",
                                sample_metadata = "sample_metadata"), "(x) <-")
            ration_data <-
              tidyr::gather(features_quantification(x),
                            key = "sample", value = "value", -.features_id)
            ration_data <-
              tibble::as_tibble(merge(ration_data, sample_metadata(x),
                                      by = "sample", all.x = T))
            if (mean) {
              ration_data <-
                dplyr::summarise(dplyr::group_by(ration_data, .features_id, group),
                                 value = mean(value, na.rm = T))
              ration_data <- dplyr::ungroup(ration_data)
            }
            ration_data(child_nebulae(x)) <-
```

```
              split(ration_data, ~.features_id)
            return(x)
          })
```

23 File: extraMethods-draw_structures.R

```
# ==========================================================================
# draw all chemical structures for a specified child-nebula
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases draw_structures
#'
#' @title Draw and visualize chemcial structure
#'
#' @description
#' Methods used for drawing and visualizing chemical structures of 'features'
#' in Child-Nebulae.
#' [ChemmineOB::convertToImage()] is the core function used for drawing chemical
#' structures.
#'
#' @seealso [ChemmineOB::convertToImage()].
#'
#' @name draw_structures-methods
#'
#' @order 1
NULL
#> NULL

#' @importFrom dplyr filter
#' @importFrom dplyr select
#' @importFrom tibble as_tibble
#' @importFrom tibble as_tibble
#' @exportMethod draw_structures
#'
#' @aliases draw_structures
#'
#' @param x [mcnebula-class] object.
#' @param nebula_name character(1). Chemical classes in 'nebula_index' data.
#' Specified to draw chemical structures of all the 'features' of that.
#' @param .features_id character(1). The ID of 'features'.
#' @param data data.frame. A 'data.frame' contains columns of '.features_id' and
#' 'smiles'.
#' @param ... ...
#'
#' @rdname draw_structures-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_child_nebulae(test1, 0.01)
#'   test1 <- create_child_layouts(test1)
#'   test1 <- activate_nebulae(test1)
#'
#'   ## optional 'nebula_name'
#'   visualize(test1)
#'   ## a class for example
#'   class <- visualize(test1)$class.name[1]
#'   tmp <- export_path(test1)
#'   test1 <- draw_structures(test1, class)
#'
#'   ## see results
#'   grobs <- structures_grob(child_nebulae(test1))
#'   grobs
#'   grid::grid.draw(grobs[[1]])
#'   ## visualize with ID of 'feature' (.features_id)
#'   ids <- names(grobs)
#'   show_structure(test1, ids[1])
#'
#'   unlink(tmp, T, T)
#' }
setMethod("draw_structures",
          signature = setMissing("draw_structures",
                                 x = "mcnebula",
                                 nebula_name = "character"),
          function(x, nebula_name, ...){
            .check_data(child_nebulae(x), list(tbl_graph = "create_child_layouts"))
            tidy <- tbl_graph(child_nebulae(x))[[nebula_name]]
            if (is.null(tidy))
              stop( "`nebula_name` not found in `tbl_graph(child_nebulae(x))`" )
            df <- dplyr::select(tibble::as_tibble(tidy), .features_id = name, smiles)
            draw_structures(x, data = df, ...)
          })

#' @exportMethod draw_structures
#' @rdname draw_structures-methods
setMethod("draw_structures",
          signature = setMissing("draw_structures",
                                 x = "mcnebula",
                                 .features_id = "character"),
          function(x, .features_id, ...){
            .check_data(x, list(features_annotation = "create_features_annotation"))
            df <- dplyr::select(features_annotation(x), .features_id, smiles)
            df <- dplyr::filter(df, .features_id %in% !!.features_id)
            draw_structures(x, data = df, ...)
          })

#' @exportMethod draw_structures
#' @rdname draw_structures-methods
setMethod("draw_structures",
          signature = setMissing("draw_structures",
                                 x = "mcnebula",
                                 data = "data.frame"),
          function(x, data, ...){
            sets <- structures_grob(child_nebulae(x))
            if (!is.null(sets)) {
              data <- dplyr::filter(data, !.features_id %in% names(sets))
            }
            if (!nrow(data) == 0) {
              structures_grob(child_nebulae(x)) <-
                c(sets, .draw_structures(data, paste0(export_path(x),
                      "/tmp/structure"), T, ...))
            }
            return(x)
          })
```

```r
#' @importFrom grid grid.draw
#' @exportMethod show_structure
#'
#' @description
#' \code{show_structure}: visualize the chemical structure of 'feature'
#' which has been drawn.
#'
#' @rdname draw_structures-methods
#'
setMethod("show_structure",
          signature = c(x = "ANY", .features_id = "character"),
          function(x, .features_id){
            .check_data(child_nebulae(x), list(structures_grob = "draw_structures"))
            grid::grid.draw(structures_grob(child_nebulae(x))[[.features_id]])
          })

#' @importFrom dplyr mutate
#' @importFrom pbapply pbapply
.draw_structures <-
  function(df, path, rm_background = F, fun_draw = .smiles_to_cairosvg){
    .check_columns(df, c(".features_id", "smiles"), "data.frame")
    .check_path(path)
    df <- dplyr::mutate(df, path = paste0(!!path, "/", .features_id, ".svg"))
    df <- dplyr::filter(df, !is.na(smiles))
    if (nrow(df) == 0)
      return(NULL)
    .message_info("draw_structures", "smiles -> svg -> grob")
    grImport2:::setPrefix("")
    lst <- pbapply::pbapply(dplyr::select(df, smiles, path), 1,
                            function(vec){
                              fun_draw(vec[["smiles"]], vec[["path"]])
                              .cairosvg_to_grob(vec[["path"]])
                            })
    names(lst) <- df$.features_id
    if (rm_background)
      lst <- lapply(lst, .rm_background)
    return(lst)
  }

.rm_background <-
  function(grob){
    if (is(grob$children[[1]]$children[[1]], "picRect")) {
      grob$children[[1]]$children[[1]] <- NULL
    }
    return(grob)
  }

#' @importFrom ChemmineOB convertToImage
#' @importFrom rsvg rsvg_svg
.smiles_to_cairosvg <-
  function(smile, path){
    ChemmineOB::convertToImage("SMI", "SVG", source = smile, toFile = path)
    rsvg::rsvg_svg(path, path)
  }
```

24 File: extraMethods-report.R

```r
# ==========================================================================
# some methods for class 'report', to fast generate layer of 'section' or
# 'code_block' etc.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases include_figure
#'
#' @title Easily embed figure into document
#'
#' @description
#' Creates a pre-defined [code_block_figure-class] object containing the codes of
#' [knitr::include_graphics()] for formatting display the figure in document.
#'
#' @seealso [code_block_figure-class], [report-class], [knitr::include_graphics()]...
#'
#' @name include_figure-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod include_figure
#'
#' @aliases include_figure
#'
#' @param file character(1). The path of file. See [knitr::include_graphics()] for
#' the supported image formats.
#' @param name character(1). For cross-reference in document.
#' See \url{https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref}.
#' @param caption character(1). Caption of figure display in document.
#'
#' @rdname include_figure-methods
#'
#' @examples
#' \dontrun{
#'   tmp <- paste0(tempdir(), "/test.pdf")
#'   pdf(tmp)
#'   plot(1:10)
#'   dev.off()
#'
#'   fig_block <- include_figure(
#'     tmp, "plot", "This is caption"
#'   )
#'   fig_block
#' }
setMethod("include_figure",
          signature = c(file = "character",
                        name = "character",
                        caption = "character"),
          function(file, name, caption){
            .check_file(file)
            codes <- paste0("knitr::include_graphics('", file, "')")
            args <- list(echo = F, eval = T, message = F)
            new_code_block_figure(name = name,
                                  caption = caption,
                                  codes = codes,
                                  args = args)
          })

#' @aliases include_table
#'
#' @title Easily embed table into document
#'
#' @description
#' Creates a pre-defined [code_block_table-class] object containing the codes of
#' [knitr::kable()] for formatting display the table in document.
#'
```

```r
#' @name include_table-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod include_table
#'
#' @aliases include_table
#'
#' @param data 'data.frame' object. The data of table to display in document.
#' @param name character(1). For cross-reference in document.
#' See \url{https://bookdown.org/yihui/rmarkdown-cookbook/cross-ref.html#cross-ref}.
#'
#' @param caption character(1). Caption of figure display in document.
#'
#' @rdname include_table-methods
#'
#' @examples
#' \dontrun{
#'   data <- data.frame(x = 1:10, y = 1:10)
#'   tab_block <- include_table(
#'     data, "table1",
#'     "This is caption"
#'   )
#'   tab_block
#' }
setMethod("include_table",
          signature = c(data = "data.frame",
                        name = "character",
                        caption = "character"),
          function(data, name, caption){
            var <- deparse(substitute(data))
            codes <- paste0("knitr::kable(", var, ", ",
                            "format = 'markdown', ",
                            "caption = '", caption, "')")
            args <- list(echo = F, eval = T, message = F)
            new_code_block_table(name = name,
                                 codes = codes,
                                 args = args)
          })
```

```
#' @aliases history_rblock
#'
#' @title Create 'code_block' object from history codes
#'
#' @description
#' Get codes from R history, then formatted as [code_block-class] object.
#'
#' @seealso
#' [code_block-class], [history()]...
#'
#' @name history_rblock-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod history_rblock
#' @description \code{history_rblock()}: get the default parameters for the method
#' \code{history_rblock}.
#' @rdname history_rblock-methods
setMethod("history_rblock",
          signature = setMissing("history_rblock"),
          function(){
            list(exclude = 1)
          })

#' @exportMethod history_rblock
#' @description \code{history_rblock(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{history_rblock}.
#' @rdname history_rblock-methods
setMethod("history_rblock",
          signature = c(nrow  = "numeric"),
          function(nrow, pattern_start, pattern_end, exclude){
            reCallMethod("history_rblock",
                         .fresh_param(history_rblock()))
          })

#' @exportMethod history_rblock
#'
#' @aliases history_rblock
#'
```

```r
#' @param nrow numeric(1). The number of lines of code to fetch.
#' @param pattern_start character(1).
#' The pattern string used to match the starting line of codes in R history.
#'
#' @param pattern_end character(1).
#' The pattern string used to match the ending line of codes in R history.
#'
#' @param exclude numeric(1). Used to exclude the last lines of code.
#'
#' @rdname history_rblock-methods
#'
#' @examples
#' \dontrun{
#'   test1 <- 1
#'   test2 <- 2
#'   test3 <- 3
#'
#'   block <- history_rblock(, "^test1", "^test3")
#'   block
#' }
setMethod("history_rblock",
          signature = setMissing("history_rblock",
                                 nrow = "numeric",
                                 exclude = "numeric"),
          function(nrow, exclude){
            his <- tail(get_history(exclude), n = nrow)
            args <- list(echo = T, eval = F)
            new_code_block(codes = his, args = args)
          })

#' @exportMethod history_rblock
#' @rdname history_rblock-methods
setMethod("history_rblock",
          signature = setMissing("history_rblock",
                                 pattern_start = "character",
                                 pattern_end = "character",
                                 exclude = "ANY"),
          function(pattern_start, pattern_end, exclude){
            if (missing(exclude))
              exclude <- 1
            his <- get_history(exclude)
            end <- tail(grep(pattern_end, his, perl = T), n = 1)
            start <- tail(grep(pattern_start, his[1:end]), n = 1)
            args <- list(echo = T, eval = F)
            new_code_block(codes = his[start:end], args = args)
          })

#' @export rblock
#' @aliases rblock
#' @title Eval the code as well create 'code_block' object
#'
#' @description \code{rblock}: Run or not run the code with formatting as [code_block-class]
#' object.
#' @param code The code to run or document. Braces ( '{}' ) must be used.
#' @param eval logical. Whether to eval the code.
#' @param envir environment. The  'environment'  in which the code is to be evaluated.
#' @param ... Other parameters passed to [new_code_block()].
#' @rdname rblock
#'
#' @examples
#' \dontrun{
#'   rblock({
#'     test1 <- 1
#'     test2 <- 2
#'     test3 <- 3
#'   })
#'
#'   rblock({
#'     test <- mcn_5features
#'     ## this annotation line would be ignored
#'     test1 <- filter_structure(test)
#'     test1 <- create_reference(test1)
#'     test1 <- filter_formula(test1, by_reference=T)
#'     test1 <- create_stardust_classes(test1)
#'   })
#' }
rblock <- function(code, eval = T, envir = parent.frame(), ...){
  code <- substitute(code)
  if (!rlang::is_call(code, "{")) {
    info <- paste0(c("The `code` argument must be a braced expression,",
                     "use this such as:",
                     styler::style_text("rblock({\ntest <- 1\ntest <- 2\n})")),
                    collapse = "\n")
    stop(info)
  }
  if (eval) {
    eval(code, envir = envir)
  }
  code <- unlist(lapply(2:length(code), function(n) deparse(code[[n]])))
  args <- list(...)
  block.args <- list(echo = T, eval = F)
  if (!is.null(args$args)) {
    block.args <- .fresh_param(block.args, args$args)
    args$args <- NULL
  }
  do.call(new_code_block, c(list(codes = code, args = block.args), args))
}
```

25 File: extraMethods-workflow.R

```
# ==========================================================================
# Quickly create analysis templates.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases workflow
#'
#' @title Quickly create analysis templates.
#'
#' @description Quickly create analysis templates and quickly construct a report.
#' This template contains a number of pre-defined sections, each with a fixed
#' description and code. The flexible approach is to use this method to obtain the
#' codes that form these sections and then modify these codes, which is more
#' likely to result in analysis results and reports that meet the requirements.
#'
#' @name workflow-methods
#'
#' @order 1
NULL
#> NULL

setOldClass("environment")

#' @exportMethod workflow
#' @description \code{workflow()}: get the available section names and
#' its heading level.
#' @rdname workflow-methods
setMethod("workflow",
  signature = setMissing("workflow"),
  function()
  {
    codes <- c(
      "workflow(sections = ",
      deparse(.workflow_name),
      ", mode = 'templ')"
    )
    writeLines(codes)
  }
)

#' @exportMethod workflow
#' @description \code{workflow(...)}: use the default parameters whatever 'missing'
#' while performing the method \code{workflow}.
#' @rdname workflow-methods
setMethod("workflow",
  signature = c(mode = "character"),
  function(sections, mode, envir, sirius_version, sirius_project, ion_mode, ...)
  {
    reCallMethod("workflow",
      .fresh_param(
        list(
          sections = eval(.workflow_name),
          envir = parent.frame(),
          sirius_version = "sirius.v4",
          sirius_project = ".",
          ion_mode = "pos"
        )), ...)
  }
)

#' @importFrom utils menu
#' @exportMethod workflow
#'
#' @param sections numeric with names. Use \code{workflow()}
#' to show the available sections.
#' @param mode character(1). "print" or "run". If "print", print the
```

```r
#' template of the select workflow; If "run", the codes would be eval,
#' and return with a [report-class] object.
#' @param envir The environment to eval the codes. Default is \code{parent.frame()}.
#' @param sirius_version character(1). Passed to [initialize_mcnebula()].
#' @param sirius_project character(1). Passed to [initialize_mcnebula()].
#' @param ion_mode character(1). Set this using \code{ion_mode<-}.
#' @param ... ...
#'
#' @rdname workflow-methods
setMethod("workflow",
  signature = setMissing("workflow",
    sections = "numeric", mode = "character", envir = "environment",
    sirius_version = "character", sirius_project = "character",
    ion_mode = "character"),
  function(sections, mode, envir, sirius_version, sirius_project, ion_mode, ...)
  {
    if (is.null(names(sections)))
      stop("`sections` must be numeric with names.")
    else if (!any(names(sections) %in% names(eval(.workflow_name))))
      stop("`sections` should be part or all of these in `workflow()`.")
    env.args <- as.environment(
      c(list(sirius_version = sirius_version,
          sirius_project = sirius_project,
          ion_mode = ion_mode), list(...))
    )
    sections <- lapply(names(sections),
      function(name) {
        ch <- .workflow_templ[[ name ]]
        vapply(ch, cl, character(1), .envir = env.args)
      })
    if (mode == "print") {
      lapply(sections,
        function(section) {
          writeLines(c(section, ""))
        })
      .workflow_gather()
      return(message("\n## Done"))
    } else if (mode == "run") {
      conflict <- gather_sections(get = F)
      if (length(conflict) > 0) {
        cl("Conflicting variable names detected in environment `envir`, ",
          "confirm to clear them?")
        input <- utils::menu(c("yes", "no"))
        if (input == 1)
          rm(list = conflict, envir = envir)
        else
          stop("Terminate the operation.")
      }
      lapply(sections,
        function(section) {
          eval(parse(text = section), envir = envir)
        })
      sections <- gather_sections(envir = envir)
      report <- do.call(new_report, sections)
      return(report)
    } else {
      return(NA)
    }
  }
)

.workflow_gather <- function(){
  codes <- substitute({
    sections <- gather_sections()
    report <- do.call(new_report, sections)
    render_report(report, file <- paste0(tempdir(), "/report.Rmd"))
    rmarkdown::render(file)
  })
  codes <- unlist(lapply(2:length(codes), function(n) deparse(codes[[n]])))
  writeLines(codes)
}

.workflow_name <-
  substitute(
    c("Abstract" = 1, "Introduction" = 1, "Set-up" = 1,
      "Integrate data and Create Nebulae" = 1,
      "Initialize analysis" = 2,
      "Filter candidates" = 2,
      "Filter chemical classes" = 2,
      "Create Nebulae" = 2,
      "Visualize Nebulae" = 2,
      "Session infomation" = 1
    ))

#' @importFrom glue glue
cl <- function(..., .open = "<<<", .close = ">>>", .envir = parent.frame()) {
  glue::glue(..., .open = .open, .close = .close, .envir = .envir)
}
```

26 File: functions-clear.R

```
# ==========================================================================
# Cleans up data in the mcnebula object that is no longer in use.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases clear
#'
#' @title Clean up data in the mcnebula object that is no longer in use
#'
#' @param x [mcnebula-class] object.
#'
#' @name clear
NULL
#> NULL

#' @export clear_dataset
#' @aliases clear_dataset
#' @description \code{clear_dataset}: The data (chemical formula,
#' chemical structure, chemical classes)
#' in \code{project_dataset(x)}, and data in \code{backtrack(mcn_dataset(mcn)}
#' would be clean up to reduced memory usage.
#' This may be best used after running the [create_nebula_index()],
#' if your machine doesn't have much Random Access Memory (RAM).
#' @note If this function has conducted, the PPCP dataset would not be
#' available for downstream methods, such as [set_ppcp_data()],
#' [annotate_nebula()]...
#' @rdname clear
clear_dataset <- function(x) {
  clear <- c(".f2_formula", ".f3_fingerid", ".f3_canopus")
  for (i in clear) {
    dataset(project_dataset(x))[[ i ]] <- NULL
  }
  backtrack(mcn_dataset(x)) <- list()
  return(x)
}

#' @export clear_nodes
#' @aliases clear_nodes
#' @description \code{clear_nodes}: Clear data ('grobs' and 'ggset') of 'nodes'
#' in slot \code{child_nebulae}.
#' @rdname clear
clear_nodes <- function(x) {
  nodes_ggset(child_nebulae(x)) <- list()
  nodes_grob(child_nebulae(x)) <- list()
  return(x)
}
```

27 File: functions-plot__msms__mirrors.R

```
# ==========================================================================
# draw mirror bar plot of MS/MS spectra
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @importFrom dplyr starts_with
#' @importFrom grid grid.draw
#' @importFrom grid upViewport
#' @importFrom grid downViewport
#' @aliases plot_msms_mirrors
#'
#' @title Draw MS/MS mirror bar plots
#'
#' @description
#' Draw MS/MS spectra as mirror bar plots using 'ggplot2' with [facet_wrap()].
#' The sub-panel of each 'features' would be found by [grid::grid.grep()]...
#' Then chemical structures would be drawn into sub-panel.
#'
#' @param x [mcnebula-class] object.
#' @param .features_id character. The ID of 'features'.
#' @param fun_modify function. Used to post modify the [ggset-class] object before
#' visualization. See [fun_modify].
#' @param structure_vp 'viewport' object. Created by [grid::viewport()]. The 'viewport'
#' to draw chemical structures in sub-panel. Can be `NULL`, this would return a 'ggplot'
#' object without chemical structure visualization.
#'
#' @seealso [facet_wrap()], [grid::grid.force], [grid::grid.grep()]...
#'
```

```r
#' @name plot_msms_mirrors
NULL
#> NULL

#' @export plot_msms_mirrors
#' @aliases plot_msms_mirrors
#' @rdname plot_msms_mirrors
plot_msms_mirrors <-
  function(x, .features_id, fun_modify = modify_set_labs_xy,
           structure_vp = grid::viewport(0.7, 0.3, 0.3, 0.3)
           ){
    .check_data(x, list(features_annotation = "create_features_annotation"))
    x <- collate_data(x, ".f2_msms")
    x <- collate_data(x, ".f3_spectra")

    ## raw peak
    raw_msms <- latest(x, "project_dataset", ".f2_msms")
    raw_msms <- dplyr::filter(raw_msms, .features_id %in% !!.features_id)
    raw_msms <- lapply(split(raw_msms, ~.features_id),
      dplyr::mutate, rel.int. = int. / max(int.) * 100
    )
    raw_msms <- lapply(raw_msms, dplyr::select,
      .features_id, raw_mz = mz, raw_rel.int. = rel.int.)

    ## non-noise peak
    sig_msms <- latest(x, "project_dataset", ".f3_spectra")
    sig_msms <- dplyr::filter(sig_msms, .features_id %in% !!.features_id)
    sig_msms <- dplyr::select(sig_msms, .features_id,
                         sig_mz = mz, sig_rel.int. = rel.int.)
    sig_msms <- split(sig_msms, ~ .features_id)

    ## merge
    set <- sapply(.features_id, simplify = F,
                  function(id){
                    df <- tol_merge(raw_msms[[id]], sig_msms[[id]],
                                    "raw_mz", "sig_mz")
                    df <- dplyr::select(df, -dplyr::starts_with(".features_id"))
                    dplyr::mutate(df, sig_rel.int. = -sig_rel.int.)
                  })

    set <- data.table::rbindlist(set, idcol = T)
    set <- dplyr::rename(set, .features_id = .id)
    matched_set <- dplyr::filter(set, !is.na(sig_mz))

    ## precursor mz and rt
    anno <- dplyr::select(features_annotation(x),
                          tani.score, .features_id, pre.mz = mz, rt.secound)
    anno <- dplyr::filter(anno, .features_id %in% !!.features_id)
    export_name <- export_name(x)

    anno <-
      dplyr::mutate(anno,
                    tani.score = round(tani.score, 2),
                    pre.mz = round(pre.mz, 4),
                    rt.min = round(rt.secound / 60, 2),
                    x = 0, y = 65,
                    label = paste0(!!export_name[[ "pre.mz" ]],
                                   ": ", pre.mz, "\n",
                                   !!export_name[[ "rt.min" ]],
                                   ": ", rt.min, "\n",
                                   "TS: ", ifelse(is.na(tani.score), "-",
                                                  tani.score)
                                   ))

    ggset <-
      new_ggset(new_command(ggplot),
                .command_msms_rawPeak(set),
                .command_msms_sigPeak(matched_set),
                .command_msms_rawDot(matched_set),
                .command_msms_sigDot(matched_set),
                .command_msms_text(anno),
                .command_msms_y(),
                new_command(theme_minimal),
                .command_msms_theme(),
                .command_msms_facet()
      )
    p <- call_command(fun_modify(ggset))

    if (is.null(structure_vp))
      return(p)

    print(p, newpage = T)
```

```
    df.vp <- get_facet.wrap.vp(.features_id)

    .message_info_viewport()
    apply(df.vp, 1,
          function(lst){
            grob <- structures_grob(child_nebulae(x))[[ lst[[ "strip" ]] ]]
            if (!is.null(grob)) {
              grid::downViewport(paste0(lst[[ "vp" ]]))
              grid::pushViewport(structure_vp)
              grid.draw(grob)
              grid::upViewport(0)
            }
          })
    .message_info_viewport()

  }

.command_msms_rawPeak <-
  function(df, color = "black", size = 0.8, alpha = 0.8){
    df <- dplyr::mutate(df, mz = raw_mz, rel.int. = 0)
    new_command(geom_segment, data = df,
                aes(x = mz, xend = raw_mz, y = rel.int., yend = raw_rel.int.),
                color = color, size = size, alpha = alpha
    )
  }

.command_msms_sigPeak <-
  function(df, color = "#E6550DFF", size = 0.8, alpha = 1){
    new_command(geom_segment, data = df,
                aes(x = sig_mz, xend = sig_mz, y = 0, yend = sig_rel.int.),
                color = color, size = size, alpha = alpha
    )
  }

.command_msms_sigDot <-
  function(df, color = "#E6550DFF", size = 0.8, alpha = 1){
    new_command(geom_point, data = df,
                aes(x = sig_mz, y = sig_rel.int.),
                color = color, size = size, alpha = alpha
    )
  }
```

```r
.command_msms_rawDot <-
  function(df, color = "black", size = 0.8, alpha = 0.8){
    new_command(geom_point, data = df,
                aes(x = raw_mz, y = raw_rel.int.),
                color = color, size = size, alpha = alpha
    )
  }

.command_msms_text <-
  function(df, hjust = 0, fontface = "bold", family = .font){
    new_command(geom_text, data = df,
                aes(x = x, y = y, label = label),
                hjust = hjust, fontface = fontface, family = family
    )
  }

.command_msms_y <-
  function(){
    new_command(scale_y_continuous, limits = c(-100, 100))
  }

.command_msms_theme <-
  function(){
    new_command(theme,
                text = element_text(family = .font),
                strip.text = element_text(size = 12),
                panel.grid = element_line(color = "grey85"),
                plot.background = element_rect(
                  fill = "white", color = "transparent", size = 0
                )
    )
  }

.command_msms_facet <-
  function(){
    new_command(facet_wrap, ~ paste("ID:", .features_id), scales = "free")
  }

#' @importFrom dplyr bind_rows
tol_merge <-
  function(main,
           sub,
           main_col = "mz",
           sub_col = "mz",
           tol = 0.002,
           bin_size = 1
           ){
    if (main_col == sub_col) {
      new_name <- paste0(sub_col, ".sub")
      colnames(sub)[colnames(sub) == sub_col] <- new_name
      sub_col <- new_name
    }
    main$...seq <- 1:nrow(main)
    backup <- main
    ## to reduce computation, round numeric for limitation
    ## main
    main$...id <- round(main[[ main_col ]], bin_size)
    ## sub
    sub.x <- sub.y <- sub
    sub.x$...id <- round(sub.x[[ sub_col ]], bin_size)
    sub.y$...id <- sub.x$...id + ( 1 * 10^-bin_size )
    sub <- rbind(sub.x, sub.y)
    ## expand merge
    df <- merge(main, sub, by = "...id", all.x = T, allow.cartesian = T)
    df$...diff <- abs(df[[ main_col ]] - df[[ sub_col ]])
    df <- dplyr::filter(df, ...diff <= !!tol)
    ## get the non-merged
    backup <- backup[!backup$...seq %in% df$...seq, ]
    df <- dplyr::bind_rows(df, backup)
    ## remove the assist col
    dplyr::select(df, -...id, -...diff, -...seq)
  }

#' @importFrom grid grid.grep
#' @importFrom grid grid.force
get_facet.wrap.vp <-
  function(
           strip,
           grid.force = T
           ){
    if(grid.force){
      grid::grid.force()
```

```r
    }
    ## grep vp of panel
    panel <- grid::grid.grep("panel", grep = T, global= T, viewports = T, grobs = F)
    ## vp name
    panel <- sapply(panel, paste)
    ## the specific seq number of vp
    panel.seq <- stringr::str_extract(panel, "(?<=panel-)[0-9]{1,}(?=-)")
    panel.seq <- max(as.integer(panel.seq))
    ## number stat
    len <- length(strip)
    len.p <- length(panel)
    ## the number of blank panel
    na <- len.p - (len %% len.p)
    if(na == len.p)
      na <- 0
    ## as matrix
    mat <- matrix(c(sort(strip), rep(NA, na)), ncol = panel.seq, byrow = T)
    vec <- as.vector(mat)
    ## as data.frame
    df <- data.table::data.table(vp = panel, strip = vec)
    ## filter out the NA
    df <- dplyr::filter(df, !is.na(strip))
    return(df)
  }
```

28 File: functions-report.R

```r
# ==========================================================================
# Help for rendering report
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @export get_ref
#' @description \code{get_ref}: get the string for cross-reference.
#' @param object [code_block_figure-class] or [code_block_table-class] object.
#' @param type character. "fig" or "tab".
#' @rdname code_block-class
#'
#' @examples
#' \dontrun{
#'   ## general
#'   codes <- "df <- data.frame(x = 1:10)
#'     df<-dplyr::mutate(df,y=x*1.5)%>%
#'     dplyr::filter(x >= 5)
#'     p <- ggplot(df)+
#'     geom_point(aes(x=x,y=y))
#'     p"
#'   block <- new_code_block("r", codes, list(eval = T, echo = T, message = T))
#'   ## see results
#'   block
#'   call_command(block)
#'   writeLines(call_command(block))
#'
#'   ## figure
#'   fig_block <- new_code_block_figure(
#'     "plot1",
#'     "this is a caption",
#'     codes = codes
#'   )
#'   ## see results
#'   fig_block
#'   writeLines(call_command(fig_block))
#'   command_args(fig_block)
#'   cat(get_ref(fig_block), "\n")
#'
#'   ## table
#'   codes <- "df <- data.frame(x = 1:10) %>%
#'     dplyr::mutate(y = x, z = x * y)
#'     knitr::kable(df, format = 'markdown', caption = 'this is a caption') "
#'   tab_block <- new_code_block_table("table1", codes = codes)
#'   ## see results
#'   tab_block
#'   cat(get_ref(tab_block), "\n")
#'
#'   ## default parameters
#'   new_code_block()
#'
#' }
get_ref <-
  function(object, type = c("fig", "tab")){
    type <- match.arg(type)
    name <- sub("^r ", "", command_name(object))
    paste0("\\@ref(", type, ":", name, ")")
  }
```

```r
#' @export render_report
#' @aliases render_report
#' @title Convert 'report' as .Rmd file
#' @description \code{render_report}: Write down [report-class] onject
#' as .Rmd file, then [rmarkdown::render()] can be used to output any
#' available formation.
#'
#' @param x [report-class] object.
#' @param file character(1). File name to save as.
#' @param set_all_eval logical(1). If \code{TRUE}, all [code_block-class] object
#' or [section-class] object related with "r" would be set with \code{eval = T}.
#'
#' @rdname render_report
render_report <- function(x, file, set_all_eval = F) {
  if (set_all_eval) {
    layers(x) <- lapply(layers(x),
      function(obj) {
        if (is(obj, "code_block")) {
          command_args(obj)$eval <- T
        } else if (is(obj, "section")) {
          if (!is.null(code_block(obj))) {
            command_args(code_block(obj))$eval <- T
          }
        }
        return(obj)
      })
  }
  writeLines(call_command(x), file)
}

#' @export gather_sections
#' @aliases gather_sections
#' @title Quickly gather all the 'sections' in environment
#' @description \code{gather_sections}:
#' Gathers all eligible variable names in an environment by means of Regex
#' matches.  These variables must: have a uniform character prefix, and the first
#' character that follows must be a number. e.g., "s1", "s2", "s12.2",
#' "s15.5.figure"...
#'
#' @param prefix character(1). The character prefix of the variable name.
#' @param envir environment. The environment to get the variables.
#' @param sort logical(1). If \code{TRUE}, sort the variable names
#' according to the first number string that accompanies the prefix.
#' @param get logical(1). If \code{TRUE}, return with a list of the value of
#' the variables. If \code{FALSE}, return with the variable names.
#'
#' @rdname gather_sections
gather_sections <- function(prefix = "s", envir = parent.frame(),
  sort = T, get = T)
{
  objs <- ls(envir = envir)
  sections <- objs[ grepl(paste0("^", prefix, "[0-9]"), objs) ]
  if (sort) {
    num <- stringr::str_extract(
      sections, paste0("(?<=", prefix, ")[0-9]{1,}[.]{0,}[0-9]{0,}")
    )
    sections <- sections[order(as.numeric(num))]
  }
  if (get) {
    sections <- sapply(sections, get, envir = envir, simplify = F)
  }
  sections
}
```

29 File: main-generic.R

```r
# ==========================================================================
# Generic for main method supplied by MCnebula2
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
setGeneric("initialize_mcnebula",
           signature = c(ANY = "x",
                         "character" = "sirius_version",
                         "character" = "sirius_project",
                         "character" = "output_directory"
                         ),
           function(x, sirius_version, sirius_project, output_directory)
             standardGeneric("initialize_mcnebula"))

setGeneric("filter_formula",
           signature = c(mcnebula = "x",
                         "function" = "fun_filter",
                         "logical" = "by_reference"),
           function(x, fun_filter, ..., by_reference)
             standardGeneric("filter_formula"))
setGeneric("filter_structure",
           signature = c(mcnebula = "x",
                         "function" = "fun_filter",
                         "logical" = "by_reference"),
           function(x, fun_filter, ..., by_reference)
             standardGeneric("filter_structure"))
setGeneric("create_reference",
           signature = c(mcnebula = "x",
                         "character" = "from",
                         "character" = "subscript",
                         "data.frame" = "data",
                         "vector" = "columns",
                         "logical" = "fill",
                         "list" = "MoreArgs"
                         ),
           function(x, from, subscript, data, columns, fill, MoreArgs)
             standardGeneric("create_reference"))
setGeneric("filter_ppcp",
           signature = c(mcnebula = "x",
                         "function" = "fun_filter",
                         "logical" = "by_reference"
                         ),
           function(x, fun_filter, ..., by_reference)
             standardGeneric("filter_ppcp"))
setGeneric("create_hierarchy",
           signature = c(mcnebula = "x", "function" = "fun_organize"),
           function(x, fun_organize) standardGeneric("create_hierarchy"))
setGeneric("create_features_annotation",
           signature = c(mcnebula = "x", "data.frame" = "extra_data",
                         "ANY" = "column"),
           function(x, extra_data, column)
             standardGeneric("create_features_annotation"))
setGeneric("create_stardust_classes",
           signature = c(mcnebula = "x",
                         "numeric" = "pp.threshold",
                         "numeric" = "hierarchy_priority",
                         "logical" = "position_isomerism",
                         "logical" = "inherit_dataset"
                         ),
```

```r
           function(x, pp.threshold, hierarchy_priority,
                    position_isomerism, inherit_dataset)
             standardGeneric("create_stardust_classes"))

setGeneric("cross_filter_stardust",
           signature = c(mcnebula = "x"),
           function(x, min_number, max_ratio,
                    types, cutoff, tolerance,
                    hierarchy_range, identical_factor)
             standardGeneric("cross_filter_stardust"))
setGeneric("cross_filter_quantity",
           signature = c(mcnebula = "x",
                         "numeric" = "min_number", "numeric" = "max_ratio"),
           function(x, min_number, max_ratio)
             standardGeneric("cross_filter_quantity"))
setGeneric("cross_filter_score",
           signature = c(mcnebula = "x", "character" = "types",
                         "numeric" = "cutoff", "numeric" = "tolerance"),
           function(x, types, cutoff, tolerance)
             standardGeneric("cross_filter_score"))
setGeneric("cross_filter_identical",
           signature = c(mcnebula = "x", "numeric" = "hierarchy_range",
                         "numeric" = "identical_factor"),
           function(x, hierarchy_range, identical_factor)
             standardGeneric("cross_filter_identical"))
setGeneric("backtrack_stardust",
           signature = c(mcnebula = "x", "character" = "class.name",
                         "numeric" = "rel.index", "logical" = "remove"),
           function(x, class.name, rel.index, remove)
             standardGeneric("backtrack_stardust"))

setGeneric("create_nebula_index",
           signature = c(mcnebula = "x", "logical" = "force"),
           function(x, force)
             standardGeneric("create_nebula_index"))

setGeneric("compute_spectral_similarity",
           signature = c(mcnebula = "x",
                         "logical" = "within_nebula",
                         "logical" = "recompute",
                         "ANY" = "sp1",
                         "ANY" = "sp2"),
           function(x, within_nebula, recompute, sp1, sp2)
             standardGeneric("compute_spectral_similarity"))

setGeneric("create_parent_nebula",
           signature = c(mcnebula = "x",
                         "numeric" = "edge_cutoff",
                         "numeric" = "max_edge_number",
                         "logical" = "remove_isolate"),
           function(x, edge_cutoff, max_edge_number, remove_isolate)
             standardGeneric("create_parent_nebula"))
setGeneric("create_child_nebulae",
           signature = c(mcnebula = "x",
                         "numeric" = "edge_cutoff",
                         "numeric" = "max_edge_number",
                         "logical" = "use_tracer"),
           function(x, edge_cutoff, max_edge_number, use_tracer)
             standardGeneric("create_child_nebulae"))

setGeneric("create_parent_layout",
           signature = c(mcnebula = "x", "character" = "ggraph_layout",
                         "numeric" = "seed"),
           function(x, ggraph_layout, seed)
             standardGeneric("create_parent_layout"))
setGeneric("create_child_layouts",
           signature = c(mcnebula = "x", "character" = "ggraph_layouts",
                         "numeric" = "seeds", "ANY" = "grid_layout",
                         "list" = "viewports", "ANY" = "panel_viewport",
                         "ANY" = "legend_viewport"),
           function(x, ggraph_layouts, seeds, grid_layout,
                    viewports, panel_viewport, legend_viewport)
             standardGeneric("create_child_layouts"))

setGeneric("activate_nebulae",
           signature = c(mcnebula = "x",
                         "function" = "fun_default_parent",
                         "function" = "fun_default_child"),
           function(x, fun_default_parent, fun_default_child)
             standardGeneric("activate_nebulae"))

setGeneric("visualize",
           function(x, item, fun_modify, annotate)
             standardGeneric("visualize"))
setGeneric("visualize_all",
           signature = c("ANY" = "x", "logical" = "newpage",
                         "ANY" = "fun_modify",
                         "ANY" = "legend_hierarchy"),
           function(x, newpage, fun_modify, legend_hierarchy)
             standardGeneric("visualize_all"))

setGeneric("annotate_nebula",
           function(x, nebula_name)
             standardGeneric("annotate_nebula"))
```

30 File: methods-activate_nebulae.R

```
# ==========================================================================
# make up 'ggset' (a meta class for visualizing ggplot object),
# based on layout of parent nebula and child nebulae.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases activate_nebulae
#'
#' @title activate Nebulae for visualization
#'
#' @description
#' Based on layouts create by [create_parent_layout()] or [create_child_layouts()],
#' use functions to activate Nebulae as [ggset-class] object for [visualize()]
#' methods to draw them.
#'
#' @name activate_nebulae-methods
#'
#' @seealso [ggset-class], [create_parent_layout()], [create_child_layouts()]...
#'
#' @order 1
NULL
#> NULL

#' @exportMethod activate_nebulae
#' @description \code{activate_nebulae()}: get the default parameters for the method
#' \code{activate_nebulae}.
#' @rdname activate_nebulae-methods
setMethod("activate_nebulae",
```

```r
          signature = setMissing("activate_nebulae"),
          function(){
            list(fun_default_parent = ggset_activate_parent_nebula,
                 fun_default_child = ggset_activate_child_nebulae)
          })

#' @exportMethod activate_nebulae
#' @description \code{activate_nebulae(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{activate_nebulae}.
#' @rdname activate_nebulae-methods
setMethod("activate_nebulae",
          signature = c(x = "mcnebula"),
          function(x, fun_default_parent, fun_default_child){
            reCallMethod("activate_nebulae",
                         .fresh_param(activate_nebulae()))
          })

#' @exportMethod activate_nebulae
#'
#' @aliases activate_nebulae
#'
#' @param x [mcnebula-class] object.
#' @param fun_default_parent function. Passed to create [ggset-class] object
#' for Parent-Nebula. Default is \code{ggset_activate_parent_nebula}.
#' Normally not used.
#'
#' @param fun_default_child function. Passed to create [ggset-class] object
#' for Child-Nebulae. Default is \code{ggset_activate_child_nebulae}.
#' Normally not used.
#'
#' @rdname activate_nebulae-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)
#'
#'   ## default parameters
#'   activate_nebulae()
#'
#'   test1 <- activate_nebulae(test1)
#'   ## see results
#'   ggset(parent_nebula(test1))
#'   head(ggset(child_nebulae(test1)))
#'
#'   ## visualize
#'   call_command(ggset(parent_nebula(test1)))
#'   ## or
#'   visualize(test1, "parent")
#'   ## child nebula
#'   call_command(ggset(child_nebulae(test1))[[1]])
#'   ## or
#'   visualize(test1, 1)
#' }
setMethod("activate_nebulae",
          signature = c(x = "mcnebula",
                        fun_default_parent = "function",
                        fun_default_child = "function"),
          function(x, fun_default_parent, fun_default_child){
            .message_info_formal("MCnebula2", "activate_nebulae")
            if (!is.null(layout_ggraph(parent_nebula(x)))) {
              ggset(parent_nebula(x)) <- fun_default_parent(x)
              parent <- T
            } else {
              parent <- F
            }
            if (!is.null(layout_ggraph(child_nebulae(x)))) {
              ggset(child_nebulae(x)) <- fun_default_child(x)
          child <- T
        } else {
          child <- F
        }
        if (any(!(parent | child)))
          stop("nothing need to be activate")
        return(x)
      })

#' @description \code{ggset_activate_parent_nebula}:
#' create [ggset-class] object of Parent-Nebula.
#' @rdname activate_nebulae-methods
#' @export
#' @importFrom ggraph ggraph
ggset_activate_parent_nebula <-
  function(x){
    .check_data(parent_nebula(x), list(layout_ggraph = "create_parent_layout"))
    new_ggset(new_command(ggraph::ggraph, layout_ggraph(parent_nebula(x))),
              .command_parent_edge(),
              .command_parent_node(),
              .command_parent_edge_width(),
              .command_parent_fill(palette_gradient(x)),
              .command_scale_x(layout_ggraph(parent_nebula(x))),
              .command_scale_y(layout_ggraph(parent_nebula(x))),
              new_command(match.fun(theme_grey), name = "theme_grey"),
              .command_parent_theme()
    )
  }

#' @description \code{ggset_activate_child_nebulae}:
#' create lists of [ggset-class] object of Child-Nebulae.
#' @rdname activate_nebulae-methods
#' @export
#' @importFrom ggraph ggraph
ggset_activate_child_nebulae <-
  function(x){
    .check_data(child_nebulae(x), list(layout_ggraph = "create_child_layouts"))
    set <- layout_ggraph(child_nebulae(x))
    hierarchy <- .get_hierarchy(x)
    ggsets <-
      lapply(names(set),
             function(name){
               fill <- palette_label(x)[[ hierarchy[[name]] ]]
               new_ggset(new_command(ggraph::ggraph, set[[ name ]]),
                         .command_parent_edge("black"),
                         .command_parent_node(),
                         .command_parent_edge_width(),
                         .command_parent_fill(palette_gradient(x)),
                         .command_child_title(name),
                         .command_scale_x(set[[ name ]]),
                         .command_scale_y(set[[ name ]]),
                         new_command(match.fun(theme_grey), name = "theme_grey"),
                         .command_child_theme(fill)
               )
             })
    names(ggsets) <- names(set)
    return(ggsets)
  }

.get_node_attribute_range <- function(x, attr){
  .check_data(x, list("features_annotation" = "create_features_annotation"))
  range(features_annotation(x)[[ attr ]])
}

.get_hierarchy <-
  function(x){
    hierarchy <- as.list(hierarchy(x)[["hierarchy"]])
    names(hierarchy) <- hierarchy(x)[["class.name"]]
    return(hierarchy)
  }

.get_textbox_fill <-
  function(x, class.name){
    if (missing(class.name)) {
      class.name <- names(igraph(child_nebulae(x)))
    }
    hierarchy <- .get_hierarchy(x)
    vapply(class.name, FUN.VALUE = "ch",
           function(name){
             palette_label(x)[[ hierarchy[[ name ]] ]]
           })
  }
```

```
#' @aliases set_nodes_color
#'
#' @title Custom defined nodes color in Nebulae (network)
#'
#' @description
#' Custom defined the nodes color for visualizing.
#' Run after [activate_nebulae()].
#'
#' @name set_nodes_color-methods
#'
#' @order 1
NULL
#> NULL

#' @importFrom dplyr select
#' @exportMethod set_nodes_color
#'
#' @aliases set_nodes_color
#'
#' @param x [mcnebula-class] object.
#' @param attribute character. The attribute specified to colorful the nodes.
#' Can be continues attribute or discrete attribute, exist in
#' 'layout_ggraph' object or data of \code{extra_data}.
#' Related with [ggplot2::scale_fill_gradientn()] and [ggplot2::scale_fill_manual()].
#' If the attribute is continues, colors in \code{palette_gradient(object)} would
#' be used. If the attribute is discrete, use colors in \code{palette_set(object)}.
#'
#' @param extra_data data.frame. Extra data used for setting nodes color.
#' The data.frame must contains column of '.features_id'.
#'
#' @param use_tracer logical. If \code{TRUE}, hightlight the 'top' 'features'
#' marked in 'nebula_index' data. See [set_tracer()].
#'
#' @seealso [activate_nebulae()], [set_tracer()]...
#'
#' @rdname set_nodes_color-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)
#'
#'   ids <- features_annotation(test1)$.features_id
#'   extra_data <- data.frame(
#'     .features_id = ids,
#'     attr_1 = rnorm(length(ids), 100, 50),
#'     attr_2 = sample(c("special", "normal"), 5, replace = T)
#'   )
#'
#'   test1 <- set_nodes_color(test1, "attr_1", extra_data)
#'   visualize(test1, 1)
#'   visualize_all(test1)
#'   ## set labal of the legend
#'   export_name(test1) <- c(
#'     export_name(test1),
#'     attr_1 = "Continuous attribute",
#'     attr_2 = "Discrete attribute"
#'   )
#'   visualize_all(test1)
#'
#'   test1 <- set_nodes_color(test1, "attr_2", extra_data)
#'   visualize(test1, 1)
#'   visualize_all(test1)
#'
#'   ## set colors for 'tracer'
#'   test1 <- set_tracer(test1, ids[1:2])
#'   ## re-build Child-Nebulae
#'   test1 <- create_child_nebulae(test1, 0.01)
#'   test1 <- create_child_layouts(test1)
#'   test1 <- activate_nebulae(test1)
#'   ## set color
#'   test1 <- set_nodes_color(test1, use_tracer = T)
#'   visualize_all(test1)
#' }
setMethod("set_nodes_color",
          signature = setMissing("set_nodes_color",
                                 x = "mcnebula",
                                 attribute = "character",
                                 extra_data = "data.frame"),
          function(x, attribute, extra_data){
            .check_data(child_nebulae(x), list(ggset = "activate_nebulae"))
            .check_data(x, list(features_annotation = "create_features_annotation"))
            if (length(attribute) != 1) {
              stop( "`attribute` must be a character `length(attribute) == 1`." )
            }
            if (any(!(c(".features_id", attribute) %in% colnames(extra_data)))) {
              stop("extra_data must contains columns of '.features_id' and the ",
                   "specified `attribute`.")
            }
            ## add `extra_data` into data 'features_annotation' as attributes
            features_annotation <- features_annotation(x)
            attr(features_annotation, "extra_data") <- extra_data
            reference(mcn_dataset(x))$features_annotation <- features_annotation
            if (is.numeric(extra_data[[ attribute ]])) {
              command_scale <- .command_parent_fill(palette_gradient(x))
            } else {
              command_scale <- .command_parent_fill2(palette_set(x))
            }
            attribute <- as.name(attribute)
            aes_ex <- aes(fill = !!attribute)
            ggset(child_nebulae(x)) <-
              lapply(ggset(child_nebulae(x)),
                     function(ggset) {
                       data <- command_args(layers(ggset)[[
                                                      "ggraph::ggraph"
                                                      ]])[[ "graph" ]]
                       mapped_attr <- .get_mapping2(ggset)
                       mapped_attr <- mapped_attr[names(mapped_attr) != "fill"]
                       mapped_attr <- mapped_attr[mapped_attr %in% colnames(data)]
```

```r
                       data <- dplyr::select(data, x, y, name,
                                             dplyr::all_of(unname(mapped_attr)))
                       data <- merge(data, extra_data, by.x = "name", by.y = ".features_id",
                                    all.x = T)
                       mapping <-
                         command_args(layers(ggset)[[ "ggraph::geom_node_point" ]])$mapping
                       mapping$fill <- NULL
                       mapping <- ggraph:::aes_intersect(mapping, aes_ex)
                       ggset <- mutate_layer(ggset, "ggraph::geom_node_point",
                                             data = data,
                                             mapping = mapping)
                       seq <- grep(paste0("^scale_fill|^ggplot2::scale_fill"),
                                   names(layers(ggset)))
                       layers(ggset)[[ seq ]] <- NULL
                       add_layers(ggset, command_scale)
                     })
            return(x)
          })

#' @exportMethod set_nodes_color
#' @rdname set_nodes_color-methods
setMethod("set_nodes_color",
          signature = setMissing("set_nodes_color",
                                 x = "mcnebula",
                                 attribute = "character"),
          function(x, attribute){
            attr <- as.name(attribute)
            aes_ex <- aes(fill = !!attr)
            ggset(child_nebulae(x)) <-
              lapply(ggset(child_nebulae(x)),
                     function(ggset) {
                       mapping <-
                         command_args(layers(ggset)[[
                                        "ggraph::geom_node_point"
                                        ]])$mapping
                       mapping$fill <- NULL
                       mapping <- ggraph:::aes_intersect(mapping, aes_ex)
                       ggset <- mutate_layer(ggset, "ggraph::geom_node_point",
                                             data = NULL,
                                             mapping = mapping)
                       data <- command_args(layers(ggset)[[
                                            "ggraph::ggraph"
                                            ]])[[ "graph" ]]
                       if (is.numeric(data[[ attribute ]])) {
                         command_scale <- .command_parent_fill(palette_gradient(x))
                       } else {
                         command_scale <- .command_parent_fill2(palette_set(x))
                       }
                       seq <- grep(paste0("^scale_fill|^ggplot2::scale_fill"),
                                   names(layers(ggset)))
                       layers(ggset)[[ seq ]] <- NULL
                       add_layers(ggset, command_scale)
                     })
            return(x)
          })

#' @exportMethod set_nodes_color
#' @rdname set_nodes_color-methods
setMethod("set_nodes_color",
          signature = setMissing("set_nodes_color",
                                 x = "mcnebula",
                                 use_tracer = "logical"),
          function(x, use_tracer){
            .check_data(x, list(nebula_index = "create_nebula_index"))
            if (use_tracer & is.logical(nebula_index(x)[[ "tracer" ]])) {
              data <- dplyr::distinct(nebula_index(x),
                                      .features_id, tracer_color, tracer)
              data <- dplyr::mutate(data, tracer = ifelse(tracer, .features_id,
                                                          "Others"))
              pal <- dplyr::distinct(data, tracer, tracer_color)
              palette_set(melody(x)) <-
                .as_dic(pal$tracer_color, pal$tracer, fill = F, as.list = F)
              x <- set_nodes_color(x, attribute = "tracer", extra_data = data)
            }
            return(x)
          })
```

31 File: methods-annotate_nebula.R

```
# ==========================================================================
# visualize the nebula and annotate it with multiple attributes
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
#' @aliases annotate_nebula
#'
#' @title Add multiple annotation data for visualization of Child-Nebula.
#'
#' @description
#' Use methods [draw_nodes()] and [draw_structures()] to standby visualization
#' of Child-Nebula with mutiple annotation: chemical classification,
#' 'features' quantification, chemical structure...
#' Run after [activate_nebulae()].
#'
#' @details
#' Primarily, remove the [ggraph::geom_node_point()] layer in [ggset-class] object
#' of Child-Nebula. The 'nodes' would be replaced with 'grob' object create by
#' [draw_nodes()]. The function of [ggimage::geom_subview()] is used to add
#' 'grob' object into 'ggplot' object.
#'
#' @name annotate_nebula-methods
#'
#' @order 1
NULL
#> NULL

#' @importFrom dplyr select
#' @importFrom gridExtra arrangeGrob
#' @exportMethod annotate_nebula
#'
#' @aliases annotate_nebula
#'
#' @param x [mcnebula-class] object.
#' @param nebula_name character(1). Chemical classes in 'nebula_index' data.
#'
#' @seealso [activate_nebulae()], [draw_nodes()], [draw_structures()],
#' [set_ppcp_data()], [set_ration_data()]...
#'
#' @rdname annotate_nebula-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_child_nebulae(test1, 0.01)
#'   test1 <- create_child_layouts(test1)
#'   test1 <- activate_nebulae(test1)
#'
#'   ## set features quantification data
#'   ids <- features_annotation(test1)$.features_id
#'   quant. <- data.frame(
#'     .features_id = ids,
#'     sample_1 = rnorm(length(ids), 1000, 200),
#'     sample_2 = rnorm(length(ids), 2000, 500)
#'   )
#'   metadata <- data.frame(
#'     sample = paste0("sample_", 1:2),
#'     group = c("control", "model")
#'   )
#'   features_quantification(test1) <- quant.
#'   sample_metadata(test1) <- metadata
#'
#'   ## optional 'nebula_name'
#'   visualize(test1)
#'   ## a class for example
#'   class <- visualize(test1)$class.name[1]
#'   tmp <- export_path(test1)
#'   test1 <- annotate_nebula(test1, class)
#'
#'   ## The following can be run before "annotate_nebula()"
#'   ## to customize the visualization of nodes.
#'   # test1 <- draw_structures(test1, "Fatty Acyls")
#'   ## set parameters for visualization of nodes
#'   # test1 <- draw_nodes(
#'   #   test1, "Fatty Acyls",
#'   #   add_id_text = T,
#'   #   add_structure = T,
#'   #   add_ration = T,
#'   #   add_ppcp = T
#'   # )
#'   # test1 <- annotate_nebula(test1, class)
#'
#'   ## see results
#'   ggset <- ggset_annotate(child_nebulae(test1))
#'   ggset[[class]]
#'   ## visualize 'ggset'
#'   call_command(ggset[[class]])
#' }
setMethod("annotate_nebula",
          signature = c(x = "ANY", nebula_name = "character"),
          function(x, nebula_name){
            .message_info_formal("MCnebula2", "annotate_nebula")
            data <- layout_ggraph(child_nebulae(x))[[nebula_name]]
            data <- dplyr::select(data, x, y,
                                  .features_id = name, size = tani.score)
            data <- dplyr::mutate(data, size = ifelse(is.na(size), .2, size))
            .features_id <- data$.features_id
            x <- draw_structures(x, nebula_name)
            x <- draw_nodes(x, nebula_name)
            nodes_grob <- nodes_grob(child_nebulae(x))
            nodes_grob <- lapply(.features_id,
                                 function(id) {
                                   gridExtra::arrangeGrob(nodes_grob[[id]])
                                 })
            ggset <- ggset(child_nebulae(x))[[ nebula_name ]]
            layers(ggset)[[ "ggraph::geom_node_point" ]] <- NULL
            ggset_annotate(child_nebulae(x))[[ nebula_name ]] <-
              add_layers(modify_annotate_child(ggset),
                         .command_node_annotate(data, nodes_grob))
            return(x)
          })
```

32 File: methods-backtrack_stardust.R

```
# ==========================================================================
# comparation after filtering; add or remove classes for stardust_classes
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases backtrack_stardust
```

```r
#'
#' @title Recover filtered chemical classses for 'stardust_classes'
#'
#' @description
#' These methods used for custom modify chemical classes in 'stardust_classes'
#' data. Users can use the method to recover classes which filtered out by
#' [cross_filter_stardust()] into 'stardust_classes' data.
#' In addition, users can use the method to delete chemical classes in
#' 'stardust_classes' data.
#'
#' @description
#' \code{backtrack_stardust(object)}: get the filtered chemical classes after
#' using [cross_filter_stardust()].
#'
#' Run after [cross_filter_stardust()].
#'
#' @seealso [cross_filter_stardust()]
#'
#' @name backtrack_stardust-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod backtrack_stardust
#'
#' @aliases backtrack_stardust
#'
#' @param x [mcnebula-class] object.
#' @param class.name character. The chemical classes name.
#' @param rel.index numeric. The index number of chemical classes.
#' See columns of 'rel.index' in 'nebula_index' or 'stardust_classes'.
#'
#' @param remove logical. If \code{TRUE}, remove the specified chemical
#' classes in 'stardust_classes' data. If \code{FALSE}, recover the
#' data of specified chemical classes into 'stardust_classes'; the classes
#' must in slot \code{backtrack(mcn_dataset(object))}.
#'
#' @rdname backtrack_stardust-methods
#'
setMethod("backtrack_stardust",
          signature = setMissing("backtrack_stardust",
                                 x = "mcnebula"),
          function(x){
            .message_info("backtrack_stardust", "no args found",
                      "\n\tget filtered classes")
            set <- dplyr::filter(backtrack(mcn_dataset(x))[[ "stardust_classes" ]],
                                 !rel.index %in% stardust_classes(x)[[ "rel.index" ]])
            stat <- table(set$rel.index)
            df <- merge(data.frame(rel.index = as.integer(names(stat)),
                                   features_number = as.integer(stat)),
                        dplyr::select(classification(x),
                                      rel.index, class.name, description),
                        by = "rel.index", all.x = T)
            tibble::as_tibble(df)
          })

#' @exportMethod backtrack_stardust
#'
#' @rdname backtrack_stardust-methods
#'
setMethod("backtrack_stardust",
          signature = setMissing("backtrack_stardust",
                                 x = "mcnebula",
                                 class.name = "character",
                                 remove = "ANY"),
          function(x, class.name, remove){
            if (missing(remove))
              remove <- F
            rel.index <-
              dplyr::filter(classification(x),
                            class.name %in% !!class.name)[[ "rel.index" ]]
            backtrack_stardust(x, rel.index = rel.index, remove = remove)
          })

#' @exportMethod backtrack_stardust
#'
#' @rdname backtrack_stardust-methods
#'
setMethod("backtrack_stardust",
          signature = setMissing("backtrack_stardust",
                                 x = "mcnebula",
                              rel.index = "numeric",
                              remove = "ANY"),
          function(x, rel.index, remove){
            if (missing(remove))
              remove <- F
            else if (!is.logical(remove))
              stop( "`remove` must be logical or as missing as `FALSE`" )
            .message_info("backtrack_stardust", paste0("remove == ", remove))
            .check_data(x, list(stardust_classes = "create_stardust_classes"))
            if (remove) {
              reference(mcn_dataset(x))[[ "stardust_classes" ]] <-
                dplyr::filter(stardust_classes(x), !rel.index %in% !!rel.index)
            } else {
              if (is.null(backtrack(mcn_dataset(x))[[ "stardust_classes" ]]))
                stop( "nothing in `backtrack(mcn_dataset(x))`" )
              set <- dplyr::filter(backtrack(mcn_dataset(x))[[ "stardust_classes" ]],
                                   rel.index %in% !!rel.index)
              if (nrow(set) == 0)
                stop( "no any record for specified classes in `backtrack(mcn_dataset(x))`" )
              reference(mcn_dataset(x))[[ "stardust_classes" ]] <-
                dplyr::distinct(dplyr::bind_rows(stardust_classes(x), set))
            }
            return(x)
          })
```

33 File: methods-compute_spectral_similarity.R

```
# ==========================================================================
# compute spectral similarity of features within each nebula
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases compute_spectral_similarity
#'
#' @title Compute MS2 spectral similarity
#'
#' @description These are methods stripped from [MSnbase::compareSpectra].
#' The unnecessary parts were removed, [compute_spectral_similarity()] only
#' calculate the 'dotproduct' for two spectra and get the similarity value.
#' It allows faster results for batch comparisons.
#'
#' @name compute_spectral_similarity-methods
#'
```

```r
#' @seealso [MSnbase::compareSpectra()].
#'
#' @order 1
NULL
#> NULL

#' @importFrom pbapply pbapply
#' @importFrom pbapply pblapply
#' @importFrom pbapply pbmapply
#' @exportMethod compute_spectral_similarity
#' @exportMethod compute_spectral_similarity
#' @description \code{compute_spectral_similarity()}: get the default parameters for the method
#' \code{compute_spectral_similarity}.
#' @rdname compute_spectral_similarity-methods
setMethod("compute_spectral_similarity",
          signature = setMissing("compute_spectral_similarity",
                                 x = "missing"),
          function(){
            list(within_nebula = T,
                 recompute = F
            )
          })

#' @exportMethod compute_spectral_similarity
#' @description \code{compute_spectral_similarity(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{compute_spectral_similarity}.
#' @rdname compute_spectral_similarity-methods
setMethod("compute_spectral_similarity",
          signature = c(x = "mcnebula"),
          function(x, within_nebula, recompute, sp1, sp2){
            reCallMethod("compute_spectral_similarity",
                         .fresh_param(compute_spectral_similarity()))
          })

#' @exportMethod compute_spectral_similarity
#' @rdname compute_spectral_similarity-methods
setMethod("compute_spectral_similarity",
          signature = setMissing("compute_spectral_similarity",
                                 sp1 = "lightSpectrum",
                                 sp2 = "lightSpectrum"),
          function(sp1, sp2){
            compareSpectra(sp1, sp2)
          })

#' @exportMethod compute_spectral_similarity
#' @rdname compute_spectral_similarity-methods
setMethod("compute_spectral_similarity",
          signature = setMissing("compute_spectral_similarity",
                                 sp1 = "data.frame",
                                 sp2 = "data.frame"),
          function(sp1, sp2){
            if (ncol(sp1) == 2 & ncol(sp2) == 2) {
              .message_info("compute_spectral_similarity", "ncol(sp) == 2",
                        "\n\tguess columns are c('mz', 'intensity')")
              colnames(sp1) <- colnames(sp2) <- c("mz", "intensity")
            } else {
              .message_info("compute_spectral_similarity", "ncol(sp) > 2",
                        "\n\t select columns of c('mz', 'intensity')"
              )
            }
            sp1 <- new("lightSpectrum", mz = sp1[[ "mz" ]],
                      intensity = sp1[[ "intensity" ]])
            sp2 <- new("lightSpectrum", mz = sp2[[ "mz" ]],
                      intensity = sp2[[ "intensity" ]])
            compareSpectra(sp1, sp2)
          })

#' @exportMethod compute_spectral_similarity
#'
#' @aliases compute_spectral_similarity
#'
#' @param x [mcnebula-class] object.
#' @param within_nebula logical. If \code{TRUE},
#' only 'features' that exist in a Child-Nebula are compared
#' for spectral similarity. Data of 'nebula_index' (\code{nebula_index(object)})
#' would be used for assigning and combining the 'features' of comparison.
#'
#' @param recompute logical. If \code{TRUE}, discard the existing data in the object,
#' and recompute the spectral similarity.
#'
#' @param sp1 data.frame. An additional channel for comparing two spectrum.
#' Contains 'mz' and 'intensity' for spectral comparison.
#' @param sp2 data.frame. An additional channel for comparing two spectrum.
#' Contains 'mz' and 'intensity' for spectral comparison.
#'
#' @rdname compute_spectral_similarity-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)
#'   ## see results
#'   spectral_similarity(test1)
#'   ## or
#'   reference(test1)$spectral_similarity
#'   ## or
#'   reference(mcn_dataset(test1))$spectral_similarity
#'
#'   ## compare the two spectra individually
#'   spectra <- latest(test1, "project_dataset", ".f3_spectra")
#'   data1 <- dplyr::select(
#'     dplyr::filter(spectra, .features_id == "gnps1537"),
#'     mz, int.
#'   )
#'   data2 <- dplyr::select(
#'     dplyr::filter(spectra, .features_id == "gnps1539"),
#'     mz, int.
#'   )
#'   e1 <- compute_spectral_similarity(sp1 = data1, sp2 = data2)
#'   e1
#'   # [1] 0.7670297
#'
```

```r
#'   ## MSnbase
#'   if (requireNamespace("MSnbase")) {
#'     MSnbase::compareSpectra
#'     spec1 <- new("Spectrum2", mz = data1$mz, intensity = data1$int.)
#'     spec2 <- new("Spectrum2", mz = data2$mz, intensity = data2$int.)
#'     e2 <- MSnbase::compareSpectra(spec1, spec2, fun = "dotproduct")
#'     identical(e1, e2)
#'   }
#' }
setMethod("compute_spectral_similarity",
          signature = setMissing("compute_spectral_similarity",
                                 x = "mcnebula",
                                 within_nebula = "logical",
                                 recompute = "logical"),
          function(x, within_nebula, recompute){
            .message_info_formal("MCnebula2", "compute_spectral_similarity")
            .check_data(x, list(nebula_index = "create_nebula_index"))
            if (!is.null(spectral_similarity(x))) {
              if (recompute) {
                .message_info("compute_spectral_similarity", "recompute == T")
              } else {
                .message_info("compute_spectral_similarity", "recompute == F",
                          "\n\tdata already existed.")
                return(x)
              }
            }
            ## collate spectra
            x <- collate_data(x, ".f3_spectra", reference = specific_candidate(x))
            all_spectra <- dplyr::select(latest(x, "project_dataset", ".f3_spectra"),
                                         .features_id, mz, rel.int.)
            lst_lightSpectrum <- lapply(split(all_spectra, ~.features_id),
                                        function(df){
                                          new("lightSpectrum", mz = df[[ "mz" ]],
                                              intensity = df[[ "rel.int." ]]
                                          )
                                        })
            ## combn and compute dotproduct of spectra
            if (within_nebula) {
              combn <- lapply(split(nebula_index(x), ~ rel.index),
                              function(df){
                                if (nrow(df) < 2)
```

```
                            return()
                          data.frame(t(combn(df[[ ".features_id" ]], 2)))
                        })
      combn <- t(apply(data.table::rbindlist(combn), 1, sort))
    } else {
      combn <- t(combn(unique(nebula_index(x)[[ ".features_id" ]]), 2))
    }
    combn <- dplyr::rename(dplyr::distinct(data.frame(combn)),
                           .features_id1 = 1, .features_id2 = 2)
    .message_info("compute_spectral_similarity", "compareSpectra")
    combn[[ "similarity" ]] <-
      pbapply::pbapply(combn, 1, function(vec){
                           compareSpectra(lst_lightSpectrum[[ vec[1] ]],
                                          lst_lightSpectrum[[ vec[2] ]])
                            })
    ## compute ionMass difference
    if (!is.null(features_annotation(x))) {
      mz <- dplyr::select(features_annotation(x),
                            .features_id, mz, rt.secound)
      combn <- merge(combn,
                       dplyr::rename(mz, .features_id2 = .features_id,
                                     mz2 = mz, rt.secound2 = rt.secound),
                       by = ".features_id2", all.x = T)
      combn <- merge(combn,
                       dplyr::rename(mz, .features_id1 = .features_id,
                                     mz1 = mz, rt.secound1 = rt.secound),
                       by = ".features_id1", all.x = T)
      combn <-
        dplyr::select(dplyr::mutate(combn,
                                      mass_difference = mz2 - mz1,
                                      rt.min_difference =
                                        round((rt.secound2 - rt.secound1) / 60, 2)
                                      ),
                        -mz1, -mz2, -rt.secound1, -rt.secound2)
    }
    reference(mcn_dataset(x))[[ "spectral_similarity" ]] <-
      dplyr::as_tibble(combn)
    return(x)
  })
```

34 File: methods-create__child__layouts.R

```
# # ==========================================================================
# for creating child layout, these layouts includes:
# layouts for nodes position of ggraph (layout_ggraph slot);
# layouts of size and position of grid panel (grid_layout slot);
# layouts of viewports of each child_nebula (in which grid panel) (viewports slot).
# layout of viewport of panel of overall child_nebulae (panel_viewport slot);
# layout of viewport of legend of child_nebulae (legend_viewport slot).
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_child_layouts
#'
#' @title Create layouts for visualization of Child-Nebulae
#'
#' @description
#' Create visual style of Child-Nebulae.
#' The 'style' means a variety of layouts for drawing the networks
#' (i.e. all Child-Nebulae). See details.
#'
#' @details
#' This method provides a flexible way to draw Child-Nebulae.
#' Users can create visual style based on default parameters.
#' For experienced users of 'grid' package,
#' the related functions such as [grid::grid.layout()], [grid::viewport()]
#' can be used to create customized visualizations.
#' The layouts for visualization of Child-Nebulae include:
#' - nodes position: \code{layout_ggraph}
#' - size and position of grid panel: \code{grid_layout}
#' - size and position of each Child-Nebula (inside the panel): \code{viewports}
#' - size and position of overall Child-Nebulae: \code{panel_viewport}
#' - size and position of overall legend: \code{legend_viewport}
#'
#' @name create_child_layouts-methods
#'
#' @seealso [grid::viewport()], [grid::grid.layout()],
#' [ggraph::create_layout()]...
#'
#' @order 1
NULL
#> NULL

#' @importFrom grid grid.layout
```

```r
#' @importFrom grid viewport
#' @importFrom grid pushViewport
#' @importFrom dplyr desc
#' @importFrom dplyr arrange
#' @importFrom dplyr select
#' @exportMethod create_child_layouts
#' @description \code{create_child_layouts()}: get the function for generating
#' default parameters for the method
#' \code{create_child_layouts}.
#' @rdname create_child_layouts-methods
setMethod("create_child_layouts",
          signature = setMissing("create_child_layouts",
                                 x = "missing"),
          function(){
            function(x){
              .check_data(child_nebulae(x), list(igraph = "create_child_nebulae"))
              len <- length(igraph(child_nebulae(x)))
              ggraph_layouts <-
                vapply(igraph(child_nebulae(x)), .propose_graph_layout, "ch")
              seeds <- rep(1, len)
              ncol <- round(sqrt(len))
              if (ncol ^ 2 < len) {
                nrow <- ncol + 1
              } else {
                nrow <- ncol
              }
              grid_layout <- grid::grid.layout(nrow, ncol)
              num_grid <- grid_layout$nrow * grid_layout$ncol
              if (num_grid < len) {
                stop( "`grid_layout` must contains enough sub-panels for child_nebulae." )
              }
              names <- as.character(sort(factor(names(igraph(child_nebulae(x))),
                                                levels = hierarchy(x)$class.name)))
              mtrx <- matrix(c(names, rep("...FILL", num_grid - len)),
                             ncol = grid_layout$ncol, byrow = T)
              viewports <-
                unlist(recursive = F,
                       mapply(apply(mtrx, 1, c, simplify = F), 1:nrow(mtrx),
                              SIMPLIFY = F,
                              FUN = function(names, row) {
                                lapply(1:length(names),
                                       function(n) {
                                         grid::viewport(layout = grid_layout,
                                                        layout.pos.row = row,
                                                        layout.pos.col = n)
                                       })
                              }))[1:len]
              names(viewports) <- names
              panel_viewport <-
                grid::viewport(0, 0.5, 0.8, 1, just = c("left", "centre"))
              legend_viewport <-
                grid::viewport(0.8, 0.5, 0.2, 1, just = c("left", "centre"))
              list(ggraph_layouts = ggraph_layouts,
                   seeds = seeds,
                   grid_layout = grid_layout,
                   viewports = viewports,
                   panel_viewport = panel_viewport,
                   legend_viewport = legend_viewport
              )
            }
          })

#' @exportMethod create_child_layouts
#'
#' @aliases create_child_layouts
#'
#' @description \code{create_child_layouts(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{create_child_layouts}.
#'
#' @param x [mcnebula-class] object.
#' @param ggraph_layouts character with names or not.
#' If with names, the names should be chemical classes in 'nebula_index' data.
#' The names used to specify layout for all or partial Child-Nebulae.
#' The value, see [ggraph::create_layout()].
#'
#' @param seeds numeric with names or not. The names, see parameter
#' \code{ggraph_layouts}. The values would passed to [set.seed()]
#'
#' @param grid_layout 'layout' object. Create by [grid::grid.layout()].
#'
#' @param viewports list with names or not.
#' Each element is a 'viewport' object create by [grid::viewport()]
#'
#' @param panel_viewport 'viewport' object.
#' Describe the size and position for drawing overall Child-Nebulae (panel).
#'
#' @param legend_viewport 'viewport' object.
#' Describe the size and position for drawing legend of Child-Nebulae.
#'
#' @rdname create_child_layouts-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_child_nebulae(test1, 0.01)
#'
#'   ## function to generate default parameters
#'   create_child_layouts()
#'   ## default parameters
#'   create_child_layouts()(test1)
#'
#'   test1 <- create_child_layouts(test1)
#'   ## see results (a object for 'ggraph' package to visualization)
#'   lapply(
#'     layout_ggraph(child_nebulae(test1)),
#'     tibble::as_tibble
#'   )
#' }
setMethod("create_child_layouts",
          signature = c(x = "mcnebula"),
          function(x, ggraph_layouts, seeds,
                   grid_layout, viewports,
                   panel_viewport, legend_viewport){
            .message_info_formal("MCnebula2", "create_child_layouts")
            do.call(.create_child_layouts,
                    .fresh_param(create_child_layouts()(x)))
          })

#' @importFrom tidygraph activate
.create_child_layouts <-
  function(x, ggraph_layouts, seeds,
           grid_layout, viewports,
           panel_viewport, legend_viewport
           ){
    set <- igraph(child_nebulae(x))
    ggraph_layouts <- .as_dic(ggraph_layouts, names(set), fill = F)
    seeds <- .as_dic(seeds, names(set))
    if (!is.null(grid_layout))
      .check_class(grid_layout)
    .check_names(viewports, set, "viewports", "igraph(child_nebulae(x))")
    if (length(viewports) != length(set)) {
      stop(paste0("`viewports` must be a list ",
                  "the same length as 'igraph(child_nebulae(x))'."))
    }
    viewports <- .as_dic(viewports, names(set), fill = F)
    .check_class(panel_viewport, "viewport", "grid::viewport")
    .check_class(legend_viewport, "viewport", "grid::viewport")
    tbl_graph(child_nebulae(x)) <- lapply(set,
      function(igraph) {
        tbl <- tidygraph::as_tbl_graph(igraph)
        edges <- tibble::as_tibble(tidygraph::activate(tbl, "edges"))
        if (nrow(edges) == 0) {
          tbl <- dplyr::mutate(
            tidygraph::activate(tbl, "edges"), similarity = double(0),
            mass_difference = double(0), rt.min_difference = double(0)
          )
        }
        return(tbl)
      }
    )
    layout_ggraph(child_nebulae(x)) <-
      lapply(names(tbl_graph(child_nebulae(x))),
             function(name){
              layout <- ggraph_layouts[[name]]
              if (is.null(layout))
                layout <- .propose_graph_layout(graph)
              set.seed(seeds[[name]])
              ggraph::create_layout(tbl_graph(child_nebulae(x))[[ name ]],
                                    layout = layout)
            })
    names(layout_ggraph(child_nebulae(x))) <-
      names(tbl_graph(child_nebulae(x)))
    grid_layout(child_nebulae(x)) <- grid_layout
    viewports(child_nebulae(x)) <- viewports
    panel_viewport(child_nebulae(x)) <- panel_viewport
    legend_viewport(child_nebulae(x)) <- legend_viewport
    return(x)
  }

.propose_graph_layout <-
  function(graph){
    if (length(graph) >= 300 | length(graph) <= 10)
      "kk"
    else
      "fr"
  }
```

35 File: methods-create_child_nebulae.R

```
# ==========================================================================
# use features annotation and spectral similarity data to create network
# for child-nebula
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_child_nebulae
#'
#' @title Gather data to create Child-Nebulae
#'
#' @description
#' Similar to [create_parent_nebula()], gather 'spectral_similarity' data and
#' and 'features_annotation' data; but additionally, use 'nebula_index' data
#' to group 'features' by chemical classes. Each chemical classes in 'nebula_index'
#' data would lead to a 'igraph' object.
#'
#' @seealso [compute_spectral_similarity()], [create_features_annotation()],
#' [create_nebula_index()],
#' [igraph::graph_from_data_frame()].
#'
#' @name create_child_nebulae-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_child_nebulae
#' @description \code{create_child_nebulae()}: get the default parameters for the method
#' \code{create_child_nebulae}.
#' @rdname create_child_nebulae-methods
setMethod("create_child_nebulae",
          signature = setMissing("create_child_nebulae"),
          function(){
            list(edge_cutoff = 0.5,
                 max_edge_number = 5,
                 use_tracer = T)
          })

#' @exportMethod create_child_nebulae
#' @description \code{create_child_nebulae(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{create_child_nebulae}.
#' @rdname create_child_nebulae-methods
setMethod("create_child_nebulae",
          signature = c(x = "mcnebula"),
          function(x, edge_cutoff, max_edge_number, use_tracer){
            reCallMethod("create_child_nebulae",
                         .fresh_param(create_child_nebulae()))
          })

#' @exportMethod create_child_nebulae
#'
#' @aliases create_child_nebulae
#'
#' @param x [mcnebula-class] object.
#' @param edge_cutoff numeric(1). Value in (0,1). Set a threshold to
#' create edges upon similarity value of 'spectral_similarity' data.
#'
```

```
#' @param max_edge_number numeric(1).
#' For nodes (features) in each Child-Nebulae (i.e. network), the maximum number of
#' edges link with. If the number exceeds the limitation, only edges representing higher
#' spectral similarity would be retained.
#'
#' @param use_tracer logical.
#' If \code{TRUE}, 'tracer' in 'nebula_index' data would be used to filter out
#' Child-Nebulae: a Child-Nebula without any 'feature' being marked as 'tracer',
#' this Child-Nebula would be filtered out. See [create_nebula_index()].
#'
#' @rdname create_child_nebulae-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)
#'
#'   ## default parameters
#'   create_child_nebulae()
#'
#'   test1 <- create_child_nebulae(test1, 0.01)
#'   ## see results
#'   igraph(child_nebulae(test1))
#'   ## write output for 'Cytoscape' or other network software
#'   tmp <- paste0(tempdir(), "/child_nebulae/")
#'   dir.create(tmp)
#'   res <- igraph(child_nebulae(test1))
#'   lapply(
#'     names(res),
#'     function(name) {
#'       igraph::write_graph(
#'         res[[name]],
#'          file = paste0(tmp, name, ".graphml"),
#'          format = "graphml"
#'        )
#'      }
#'    )
#'    list.files(tmp)
#'
#'    unlink(tmp, T, T)
#' }
setMethod("create_child_nebulae",
          signature = setMissing("create_child_nebulae",
                                 x = "mcnebula",
                                 edge_cutoff = "numeric",
                                 max_edge_number = "numeric",
                                 use_tracer = "logical"),
          function(x, edge_cutoff, max_edge_number, use_tracer){
            .message_info_formal("MCnebula2", "create_child_nebulae")
            .check_data(x, list(features_annotation = "create_features_annotation",
                                spectral_similarity = "compute_spectral_similarity",
                                nebula_index = "create_nebula_index"
                                ))
            if (max_edge_number < 1) {
              stop( "`max_edge_number` must be a numeric greater or equal to 1" )
            }
            features <- features_annotation(x)
            edges <- dplyr::filter(spectral_similarity(x),
                                   similarity >= edge_cutoff)
            if (use_tracer & is.logical(nebula_index(x)[[ "tracer" ]])) {
              classes <- unique(dplyr::filter(nebula_index(x), tracer)$class.name)
              nebula_index <- dplyr::filter(nebula_index(x), class.name %in% classes)
            } else {
              nebula_index <- nebula_index(x)
            }
            igraph(child_nebulae(x)) <-
              lapply(split(nebula_index, ~ class.name),
                     function(meta) {
                       features <- dplyr::filter(features, .features_id %in%
                                                 meta$.features_id)
                       edges <- dplyr::filter(edges, .features_id1 %in% meta$.features_id &
                                              .features_id2 %in% meta$.features_id)
                       if (nrow(edges) > max_edge_number) {
                         edges <- .decrease_edges(edges, max_edge_number)
                       }
                       igraph::graph_from_data_frame(edges, directed = T,
                                                     vertices = features)
                     })
            return(x)
          })
```

36 File: methods-create_features_annotation.R

```
# ==========================================================================
# create features annotation data.frame, involves formula and structure,
# based on `specific_candidate`
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_features_annotation
#'
#' @title merge annotation for 'features'
#'
#' @description
#' According to \code{specific_candidate(object)} data, merge the latest
#' filtered chemical formulae annotation, structural annotation. The ion mass
#' and retention time for each 'feature' would also be gathered.
#' User can also pass custom annotation for each 'feature', as long as the
#' 'data.frame' with column of '.features_id'.
#'
#' @details
#' The 'features_annotation' data created from:
#' - The 'specific_candidate' data: \code{specific_candidate(object)}
#' - The filtered chemical formula data: \code{latest(object, subscript = ".f2_formula")}
#' - The filtered structural data: \code{latest(object, subscript = ".f3_fingerid")}
#' - The ion mass and retention time (m/z and RT): latest(object, "project_dataset", ".f2_info")
#'
#' The last would be collated via: \code{collate_data(object, subscript = ".f2_info")}
#'
#' @name create_features_annotation-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_features_annotation
```

```r
#' @rdname create_features_annotation-methods
setMethod("create_features_annotation",
          signature = setMissing("create_features_annotation",
                                 x = "mcnebula",
                                 extra_data = "data.frame",
                                 column = "numeric"),
          function(x, extra_data, column){
            colnames(extra_data)[column] <- ".features_id"
            create_features_annotation(x, extra_data)
          })

#' @exportMethod create_features_annotation
#' @rdname create_features_annotation-methods
setMethod("create_features_annotation",
          signature = setMissing("create_features_annotation",
                                 x = "mcnebula",
                                 extra_data = "data.frame"),
          function(x, extra_data){
            if (is.null(features_annotation(x)))
              x <- create_features_annotation(x)
            if ( !".features_id" %in% colnames(extra_data) )
              stop( "id column not found" )
            reference(mcn_dataset(x))[[ "features_annotation" ]] <-
              merge(features_annotation(x), extra_data,
                    by = ".features_id", all.x = T)
            return(x)
          })

#' @exportMethod create_features_annotation
#'
#' @aliases create_features_annotation
#'
#' @param x [mcnebula-class] object.
#' @param extra_data data.frame.
#' @param column numeric(1). If name of columns not contain ".features_id",
#' used to specify ID column for 'features'.
#'
#' @rdname create_features_annotation-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)
#'   ## see results
#'   features_annotation(test1)
#'   ## or
#'   reference(test1)$features_annotation
#'   ## or
#'   reference(mcn_dataset(test1))$features_annotation
#'
#'   ## merge additional data
#'   ids <- features_annotation(test1)$.features_id
#'   data <- data.frame(.features_id = ids, quant. = rnorm(length(ids), 1000, 200))
#'   test1 <- create_features_annotation(test1, extra_data = data)
#' }
setMethod("create_features_annotation",
          signature = setMissing("create_features_annotation",
                                 x = "mcnebula"),
          function(x){
            .message_info_formal("MCnebula2", "create_features_annotation")
            .check_data(x, list(specific_candidate = "create_reference"))
            ref <- specific_candidate(x)
            ## formula dataset
            subscript <- c(".f2_formula", ".f3_fingerid")
            lst <- lapply(subscript, function(sub){
                            set <- latest(x, subscript = sub)
                            idcol <- dplyr::select(set, .features_id, .candidates_id)
                            check <- dplyr::distinct(dplyr::bind_rows(idcol, ref),
                                                     .features_id, .candidates_id)
                            if (any( duplicated(check[[ ".features_id" ]]) )) {
                              name <- gsub("^.*_", "", sub)
                              stop( "the filtered \"", sub, "\" set in `x` must match ",
                                   "with the id columns ",
                                   "(.features_id, .candidates_id) ",
                                   "in `specific_candidate(x)`, ",
                              "use `filter_", name,
                              "(x, by_reference = T)` previously.")
                        } else {
                          set <- merge(dplyr::select(ref, .features_id),
                                       dplyr::select(set, -.candidates_id),
                                       by = ".features_id", all.x = T)
                          return(set)
                        }
                             })
            res <- checkColMerge(lst[[1]], lst[[2]], by = ".features_id", all = T)
            res <- merge(ref, res, by = ".features_id", all.x = T)
            ## add ionMass and retention time for features
            x <- collate_data(x, subscript = ".f2_info")
            mz_rt <- dplyr::select(latest(x, "project_dataset", ".f2_info"),
                                   .features_id, mz, rt.secound)
            reference(mcn_dataset(x))[[ "features_annotation" ]] <-
              dplyr::as_tibble(merge(res, mz_rt, by = ".features_id", all.x = T))
            return(x)
          })
```

37 File: methods-create_hierarchy.R

```
# ==========================================================================
# collate and build classification hierarchy annotation data
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_hierarchy
#'
#' @title Create hierarchy data of chemical classification
#'
#' @description
#' Methods used to create hierarchy data of chemical classification.
#' Annotate all chemical classes with hierarchy number.
#'
#' @name create_hierarchy-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_hierarchy
#' @description \code{create_hierarchy()}:
```

```
#' get the default parameters for the method
#' \code{create_hierarchy}.
#' @rdname create_hierarchy-methods
setMethod("create_hierarchy",
          signature = setMissing("create_hierarchy"),
          function(){
            list(fun_organize = .build_hierarchy)
          })

#' @exportMethod create_hierarchy
#' @description \code{create_hierarchy(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{create_hierarchy}.
#' @rdname create_hierarchy-methods
setMethod("create_hierarchy",
          signature = c(x = "mcnebula"),
          function(x, fun_organize){
            reCallMethod("create_hierarchy",
                         .fresh_param(create_hierarchy()))
          })

#' @exportMethod create_hierarchy
#'
#' @aliases create_hierarchy
#'
#' @param x [mcnebula-class] object.
#' @param fun_organize function. Normally not used.
#' Default is \code{MCnebula2:::.build_hierarchy}.
#'
#' @rdname create_hierarchy-methods
#'
setMethod("create_hierarchy",
          signature = c(x = "mcnebula", fun_organize = "function"),
          function(x, fun_organize){
            class <- classification(x)
            if (is.null(class)) {
              x <- collate_data(x, ".canopus")
              class <- classification(x)
            }
            reference(mcn_dataset(x))[[ "hierarchy" ]] <-
              fun_organize(class)
            return(x)
          })

.build_hierarchy <-
  function(data){
    data <- dplyr::select(data, rel.index, chem.ont.id,
                          class.name, parent.chem.ont.id)
    root <- data[data$parent.chem.ont.id == "", ]
    list <- list()
    length(list) <- 12
    n <- 1
    list[[n]] <- root
    df <- data[data$parent.chem.ont.id %in% root$chem.ont.id, ]

    while(nrow(df) > 0){
      n <- n + 1
      list[[n]] <- df
      df <- data[data$parent.chem.ont.id %in% df$chem.ont.id, ]
    }
    data <- data.table::rbindlist(list, idcol = T)
    data$.id <- data$.id - 1
    dplyr::rename(dplyr::as_tibble(data), hierarchy = .id)
  }

#' @export get_parent_classes
#' @aliases get_parent_classes
#' @description \code{get_parent_classes}: For chemical classes to get
#' its parent chemical classes.
#' @param classes character. Names of chemical classes.
#' @param hierarchy_cutoff. numeric(1). The highest hierarchy of parent chemical classes
#' that needs to be searched.
#' @param re_class_no_parent logical(1). If \code{TRUE}, once a chemical class find with
#' no parent, the chemical class itself would be returned.
#' @rdname create_hierarchy-methods
get_parent_classes <-
  function(classes, x,
           hierarchy_cutoff = 3,
           re_class_no_parent = F
           ){
    .check_data(x, list(hierarchy = "create_hierarchy"))
    db <- dplyr::filter(hierarchy(x), hierarchy >= hierarchy_cutoff)
    ## as 'dictionary'
    name2id <- .as_dic(db$chem.ont.id, db$class.name, fill = F)
    id2parent <- .as_dic(db$parent.chem.ont.id, db$chem.ont.id, fill = F)
    id2name <- .as_dic(db$class.name, db$chem.ont.id, fill = F)
    sapply(classes, simplify = F,
           function(class){
             set <- c()
             parent <- 0
             id <- name2id[[class]]
             test <- try(id2parent[[id]], silent = T)
             if (inherits(test, "try-error"))
               if(re_class_no_parent)
                 return(class)
               else
                 return()
             while(!is.null(parent)){
               if(parent != 0){
                 set <- c(set, id2name[[parent]])
                 id <- parent
               }
               parent <- id2parent[[id]]
             }
             if(length(set) == 0){
               if(re_class_no_parent)
                 return(class)
             }
             return(set)
           })
  }
```

38 File: methods-create_nebula_index.R

```
# ==========================================================================
# create nebula index from filtered stardust classes
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_nebula_index
#'
#' @title Set down the chemical classes for visualization
#'
#' @description
#' Arrange the filtered 'stardust_classes' data as 'nebula_index' data.
```

```r
#' The chemical classes in 'nebula_index' data would be visualized as Child-Nebulae.
#' Run after [cross_filter_stardust()].
#'
#' @name create_nebula_index-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_nebula_index
#' @description \code{create_nebula_index()}: get the default parameters for the method
#' \code{create_nebula_index}.
#' @rdname create_nebula_index-methods
setMethod("create_nebula_index",
          signature = setMissing("create_nebula_index"),
          function(){
            list(force = F)
          })

#' @exportMethod create_nebula_index
#' @description \code{create_nebula_index(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{create_nebula_index}.
#' @rdname create_nebula_index-methods
setMethod("create_nebula_index",
          signature = c(x = "mcnebula"),
          function(x, force){
            reCallMethod("create_nebula_index",
                         .fresh_param(create_nebula_index()))
          })

#' @exportMethod create_nebula_index
#'
#' @aliases create_nebula_index
#'
#' @param x [mcnebula-class] object.
#' @param force logical. The number of chemical classes in 'stardust_classes' data
#' would be checked. The maximum is 120. If there were too many classes, return
#' with error. Set to \code{FALSE}, escape from maximum check.
#'
#' @rdname create_nebula_index-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)
#'   ## see results
#'   nebula_index(test1)
#'   ## or
#'   reference(test1)$nebula_index
#'   ## or
#'   reference(mcn_dataset(test1))$nebula_index
#' }
setMethod("create_nebula_index",
          signature = c(x = "mcnebula", force = "logical"),
          function(x, force){
            .message_info_formal("MCnebula2", "create_nebula_index")
            .check_data(x, list(stardust_classes = "create_stardust_classes"))
            if (!force) {
              class_num <- length(unique(stardust_classes(x)[[ "rel.index" ]]))
              if (class_num > 120)
                stop("too many classes; ",
                     "length(unique(stardust_classes(x)$rel.index)) > 120")
            }
            reference(mcn_dataset(x))[[ "nebula_index" ]] <-
              dplyr::select(stardust_classes(x), rel.index, class.name,
                            hierarchy, .features_id)
            return(x)
          })

#' @aliases set_tracer
#'
#' @title Mark top 'features' in 'nebula_index' data
#'
```

```r
#' @description
#' Custom defined the specific 'features' in 'nebula_index' data.
#' Mark these 'features' for subsequent visualization with eye-catching highlighting
#' ([set_nodes_color()]).
#' Run after [create_nebula_index()].
#'
#' @name set_tracer-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod set_tracer
#' @description \code{set_tracer()}: get the function for generating
#' default parameters for the method
#' \code{set_tracer}.
#' @rdname set_tracer-methods
setMethod("set_tracer",
          signature = setMissing("set_tracer"),
          function(){
            function(x, .features_id){
              if (length(.features_id) > length(palette_set(x)))
                stop("too much specified features; ",
                     "use 'palette_set<-' to set more colors")
              colors <- .as_dic(palette_set(x), .features_id,
                                as.list = F, na.rm = T)
              list(.features_id = .features_id,
                   colors = unname(colors),
                   rest = "#D9D9D9"
              )
            }
          })

#' @exportMethod set_tracer
#' @description \code{set_tracer(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{set_tracer}.
#' @rdname set_tracer-methods
setMethod("set_tracer",
          signature = c(x = "mcnebula"),
          function(x, .features_id, colors, rest){
            args <- as.list(environment())
            args$.features_id <- .features_id
            reCallMethod("set_tracer",
                         .fresh_param(set_tracer()(x, .features_id), args))
          })

#' @exportMethod set_tracer
#'
#' @aliases set_tracer
#'
#' @param x [mcnebula-class] object.
#' @param .features_id character. The ID of 'features' to mark.
#' @param colors character. Hex color.
#' @param rest character(1). Hex color.
#'
#' @seealso [create_nebula_index()], [set_nodes_color()].
#'
#' @rdname set_tracer-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)
#'
#'   ids <- features_annotation(test1)$.features_id
#'   test1 <- set_tracer(test1, ids[1:2])
#'   ## see results
#'   nebula_index(test1)
#'
#'   ## see examples in 'set_nodes_color()'
#' }
setMethod("set_tracer",
          signature = c(x = "mcnebula", .features_id = "character",
                        colors = "character", rest = "character"),
          function(x, .features_id, colors, rest){
            .check_data(x, list(nebula_index = "create_nebula_index"))
            if (length(.features_id) != length(colors))
              stop("length(.features_id) != length(colors))")
            tracer_color <- data.frame(.features_id = .features_id,
                                       tracer_color = colors)
            nebula_index <- nebula_index(x)
            nebula_index$tracer <- NULL
            nebula_index$tracer_color <- NULL
            nebula_index <- merge(nebula_index, tracer_color,
                                  by = ".features_id", all.x = T)
            nebula_index <-
              dplyr::mutate(nebula_index,
                            tracer = ifelse(is.na(tracer_color), F, T),
                            tracer_color = ifelse(tracer, tracer_color, rest))
            reference(mcn_dataset(x))[[ "nebula_index" ]] <-
              dplyr::arrange(dplyr::relocate(tibble::as_tibble(nebula_index),
                                             .features_id,
                                             .after = hierarchy), rel.index)
            return(x)
          })
```

39 File: methods-create_parent_layout.R

```
# ==========================================================================
# for creating parent layout, these layouts includes:
# layouts for nodes position of ggraph.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_parent_layout
#'
#' @title Create layout for visualization of Parent-Nebula
#'
#' @description
#' These methods use functions of [tidygraph::as_tbl_graph()] and
#' [ggraph::create_layout()] to create 'layout_ggraph' (data.frame)
#' object to standby visualization of Parent-Nebula.
#' Run after [create_parent_nebula()].
#'
#' @name create_parent_layout-methods
#'
#' @seealso [tidygraph::as_tbl_graph()], [ggraph::create_layout()].
```

```
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_parent_layout
#' @description \code{create_parent_layout()}:
#' get the default parameters for the method
#' \code{create_parent_layout}.
#' @rdname create_parent_layout-methods
setMethod("create_parent_layout",
          signature = setMissing("create_parent_layout"),
          function(){
            list(ggraph_layout = "kk", seed = 1)
          })

#' @exportMethod create_parent_layout
#' @description \code{create_parent_layout(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{create_parent_layout}.
#' @rdname create_parent_layout-methods
setMethod("create_parent_layout",
          signature = c(x = "mcnebula"),
          function(x, ggraph_layout, seed){
            reCallMethod("create_parent_layout",
                         .fresh_param(create_parent_layout()))
          })

#' @exportMethod create_parent_layout
#'
#' @aliases create_parent_layout
#'
#' @param x [mcnebula-class] object.
#' @param ggraph_layout character(1). Layout name. See [ggraph::create_layout()].
#' @param seed numeric(1). Passed to [set.seed()].
#'
#' @rdname create_parent_layout-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)
#'
#'   ## default parameters
#'   create_parent_layout()
#'
#'   test1 <- create_parent_layout(test1)
#'   ## see results (a object for 'ggraph' package to visualization)
#'   tibble::as_tibble(layout_ggraph(parent_nebula(test1)))
#' }
setMethod("create_parent_layout",
          signature = c(x = "mcnebula", ggraph_layout = "character",
                        seed = "numeric"),
          function(x, ggraph_layout, seed){
            .message_info_formal("MCnebula2", "create_parent_layout")
            .check_data(parent_nebula(x), list(igraph = "create_parent_nebula"))
            tbl_graph(parent_nebula(x)) <-
              tidygraph::as_tbl_graph(igraph(parent_nebula(x)))
            set.seed(seed)
            layout_ggraph(parent_nebula(x)) <-
              ggraph::create_layout(tbl_graph(parent_nebula(x)),
                                    layout = ggraph_layout)
            return(x)
          })
```

40 File: methods-create_parent_nebula.R

```
# ==========================================================================
# use features annotation and spectral similarity data to create network
# for parent-nebula
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_parent_nebula
```

```
#'
#' @title Gather data to create Parent-Nebula
#'
#' @description
#' Gather 'spectral_similarity' data and 'features_annotation' data
#' to create 'igraph' object use function of [igraph::graph_from_data_frame()].
#'
#' @name create_parent_nebula-methods
#'
#' @seealso [compute_spectral_similarity()], [create_features_annotation()],
#' [igraph::graph_from_data_frame()].
#'
#' @order 1
NULL
#> NULL

#' @importFrom igraph graph_from_data_frame
#' @exportMethod create_parent_nebula
#' @description \code{create_parent_nebula()}: get the default parameters for the method
#' \code{create_parent_nebula}.
#' @rdname create_parent_nebula-methods
setMethod("create_parent_nebula",
          signature = setMissing("create_parent_nebula"),
          function(){
            list(edge_cutoff = 0.5,
                 max_edge_number = 5,
                 remove_isolate = T)
          })

#' @exportMethod create_parent_nebula
#' @description \code{create_parent_nebula(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{create_parent_nebula}.
#' @rdname create_parent_nebula-methods
setMethod("create_parent_nebula",
          signature = c(x = "mcnebula"),
          function(x, edge_cutoff, max_edge_number, remove_isolate){
            reCallMethod("create_parent_nebula",
                         .fresh_param(create_parent_nebula()))
          })
```

```r
#' @exportMethod create_parent_nebula
#'
#' @aliases create_parent_nebula
#'
#' @param x [mcnebula-class] object.
#' @param edge_cutoff numeric(1). Value in (0,1). Set a threshold to
#' create edges upon similarity value of 'spectral_similarity' data.
#'
#' @param max_edge_number numeric(1).
#' For nodes (features) in each Parent-Nebulae (i.e. network), the maximum number of
#' edges link with. If the number exceeds the limitation, only edges representing higher
#' spectral similarity would be retained.
#'
#' @param remove_isolate logical. If \code{TRUE}, remove the isolate 'features'
#' (in network, i.e. the nodes without edge)
#'
#' @rdname create_parent_nebula-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)
#'
#'   ## default parameters
#'   create_parent_nebula()
#'
#'   test1 <- create_parent_nebula(test1, 0.01)
#'   ## see results
#'   igraph(parent_nebula(test1))
#'   ## write output for 'Cytoscape' or other network software
#'   tmp <- tempdir()
#'   igraph::write_graph(
#'     igraph(parent_nebula(test1)),
#'     file = paste0(tmp, "/parent_nebula.graphml",
#'       format = "graphml"
#'     )
#'   )
#'
#'   unlink(tmp, T, T)
#' }
setMethod("create_parent_nebula",
          signature = setMissing("create_parent_nebula",
                                 x = "mcnebula",
                                 edge_cutoff = "numeric",
                                 max_edge_number = "numeric",
                                 remove_isolate = "logical"),
          function(x, edge_cutoff, max_edge_number, remove_isolate){
            .message_info_formal("MCnebula2", "create_parent_nebula")
            .check_data(x, list(features_annotation = "create_features_annotation",
                                spectral_similarity = "compute_spectral_similarity"
                                ))
            edges <- dplyr::filter(spectral_similarity(x),
                                   similarity >= edge_cutoff)
            if (nrow(edges) > max_edge_number) {
              edges <- .decrease_edges(edges, max_edge_number)
            }
            if (remove_isolate) {
              features <-
                dplyr::filter(features_annotation(x), .features_id %in%
                              unique(c(edges[[ ".features_id1" ]],
                                       edges[[ ".features_id2" ]]))
                )
            } else {
              features <- features_annotation(x)
            }
            igraph(parent_nebula(x)) <-
              igraph::graph_from_data_frame(edges, directed = T,
                                            vertices = features)
            return(x)
          })
```

41 File: methods-create_reference.R

```r
# ==========================================================================
# create reference data based on mcn_dataset
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_reference
#'
#' @title Establish 'specific candidate' for each 'feature'
#'
#' @description
#' According to the filtered data, whether obtained by [filter_formula()],
#' [filter_structure()] or [filter_ppcp()],
#' establishing specific candidate of each 'feature' for subsequent data filtering.
#' This step is an important intermediate link for the three part of data filtering,
#' makes the final filtered results of chemical formula, structure and classification
#' consistent.
#'
#' @details
#' \bold{Establish reference upon top candidate}
#' Suppose we predicted a potential compound represented by LC-MS/MS spectrum,
#' and obtained the candidates of chemical molecular formula,
#' structure and chemical class.
#' These candidates include both positive and negative results:
#' for chemical molecular formula and chemical structure,
#' the positive prediction was unique; for chemical class,
#' multiple positive predictions that belong to various classification were involved.
#' We did not know the exact negative and positive.
#' Normally, we ranked and filtered these according to the scores.
#' There were numerious scores, for isotopes, for mass error, for structural similarity,
#' for chemical classes...
#' Which score selected to rank candidates depends on the purpose
#' of research. Such as:
#' - To find out the chemical structure mostly be positive, ranking the candidates
#' by structural score.
#' - To determine whether the potential compound may be of a certain chemical classes,
#' ranking the candidates by the classified score.
#'
#' Ether by [filter_formula()], [filter_structure()] or [filter_ppcp()], the
#' candidate with top score can be obtained.
#' However, for the three module (formula, structure, classes), sometimes
#' thier top score candidates were not in line with each other.
#' That is, thier top score towards different chemical molecular formulas.
#' To find out the corresponding data in other modules,
#' \code{create_reference} should be performed to establish the
#' 'specific_candidate' for subsequent filtering.
#'
#' @name create_reference-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_reference
#'
#' @aliases create_reference
#'
#' @param x [mcnebula-class] object.
#' @param from character(1). "structure", "formula" or "ppcp".
#' @param subscript character(1). ".f3_fingerid", ".f2_formula" or ".f3_canopus".
#' See [subscript-class].
#' @param data data.frame. An external channel for user to specify candidate customarily.
#' Normally not used.
#' @param columns character(2) or numeric(2). Specify the key columns in the parameter
#' of data. Normally not used.
#' @param fill logical. If \code{TRUE}, run post modification.
#' Run \code{filter_formula(object)}, and use its results to fill the data
#' \code{specific_candidate} for 'features' without specified top candidate.
#' Only useful when the data \code{specific_candidate} were
#' based on scores of chemical structure or classes, as for some 'features'
#' there may be no chemical structural
#' or classified candidates but candidates for chemical formula.
#' @param MoreArgs list. Used only \code{fill = T}. Parameters passed to [filter_formula()].
#'
#' @rdname create_reference-methods
#'
#' @examples
#' \dontrun{
#'   test <- mcn_5features
#'
#'   ## set specific candidate
#'   ## ---------------------------------------
#'   ## from chemical structure
#'   test1 <- filter_structure(test)
#'   test1 <- create_reference(test1)
#'   ## see results
#'   specific_candidate(test1)
#'   ## or
#'   reference(test1)$specific_candidate
#'   ## or
#'   reference(mcn_dataset(test1))$specific_candidate
#'   ## 'create_reference(test1)' equals to
#'   test1 <- create_reference(test1, from = "structure", fill = T)
#'   e1 <- specific_candidate(test1)
#'
#'   ## the above equals to following:
#'   data <- latest(filter_structure(test1))
#'   test1 <- create_reference(test1, data = data, fill = T)
#'   e2 <- specific_candidate(test1)
#'   identical(e1, e2)
#'
#'   ## the 'specific_candidate' data used for filtering
#'   test1 <- filter_formula(test1, by_reference = T)
#'
#'   ## -------------------------------------
#'   ## from chemical formula
#'   test1 <- filter_formula(test1)
#'   test1 <- create_reference(test1, from = "formula")
#'
#'   ## -------------------------------------
#'   ## from chemical classes
#'   ## A complex example:
#'   ## suppose there were some classes we were interested in
#'   all_classes <- latest(test1, "project_dataset", ".canopus")$class.name
#'   set.seed(1)
#'   classes <- sample(all_classes, 50)
#'   classes
#'   test1 <- filter_ppcp(test1,
#'     dplyr::filter,
#'     class.name %in% classes,
#'     pp.value > 0.5,
#'     by_reference = F
#'   )
#'   data <- latest(test1)
#'   data
#'   ## 'feature' have a plural number of candidates.
#'   ids <- data$.features_id
#'   id <- unique(ids[duplicated(ids)])
#'   ## get the candidate of top chemical structural score.
#'   `%>%` <- magrittr::`%>%`
#'   candidates <- filter_structure(test1, dplyr::filter, .features_id %in% id) %>%
#'     latest() %>%
#'     dplyr::filter(.candidates_id %in% data$.candidates_id) %>%
#'     dplyr::arrange(.features_id, dplyr::desc(csi.score)) %>%
#'     dplyr::distinct(.features_id, .keep_all = T)
#'   ## for refecrence
#'   data <- data %>%
#'     dplyr::filter(
#'       .features_id != candidates$.features_id |
#'         (.features_id == candidates$.features_id &
#'           .candidates_id == candidates$.candidates_id)
#'     )
#'   test1 <- create_reference(test1, data = data, fill = T)
#'   specific_candidate(test1)
#'
#' }
setMethod("create_reference",
          signature = c(x = "mcnebula", fill = "logical"),
          function(x, from, subscript, data, columns, fill, MoreArgs){
            args <- as.list(environment())
            args <- args[!names(args) %in% c("fill", "MoreArgs")]
            args <- args[ !vapply(args, is.name, T) ]
            if (length(args) == 1)
              x <- create_reference(x, "structure")
            else
              x <- do.call(create_reference, args)
            if (fill) {
              .message_info("create_reference", "fill == T",
                            "\n\tfilling missing features with filtered formula")
              if (!missing(MoreArgs))
                x <- do.call(filter_formula, c(x, MoreArgs))
              else
                x <- filter_formula(x)
              if (any(duplicated(latest(x)$.features_id)))
                stop("the filtered formula must unique in `.features_id`")
              .ref <- specific_candidate(create_reference(x, data = latest(x)))
              reference(mcn_dataset(x))[[ "specific_candidate" ]] <-
                dplyr::distinct(dplyr::bind_rows(specific_candidate(x), .ref),
                                .features_id, .keep_all = T)
            }
            return(x)
          })

#' @exportMethod create_reference
#' @description \code{create_reference()}: get the default parameters for the method
#' \code{create_reference}.
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference"),
          function(){
            list(from = "structure",
                 fill = T)
          })

#' @exportMethod create_reference
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference",
                                 x = "mcnebula"),
          function(x){
            create_reference(x, "structure", fill = T)
          })

#' @exportMethod create_reference
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference",
                                 x = "mcnebula", from = "character"),
          function(x, from){
            subscript <- switch(from,
                                structure = ".f3_fingerid",
                                formula = ".f2_formula",
                                ppcp = ".f3_canopus"
            )
            create_reference(x, subscript = subscript)
          })
```

```r
#' @exportMethod create_reference
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference",
                                 x = "mcnebula",
                                 subscript = "character"),
          function(x, subscript){
            .message_info_formal("MCnebula2", "create_reference")
            data <- try(entity(dataset(mcn_dataset(x))[[ subscript ]]), silent = T)
            if (inherits(data, "try-error")) {
              stop(paste0("the specified dataset not exists. use, e.g., ",
                          "`filter_structure(x)` previously."))
            }
            if (subscript == ".f3_canopus") {
              data <- dplyr::distinct(data, .features_id, .candidates_id)
            }
            create_reference(x, data = data)
          })

#' @exportMethod create_reference
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference",
                                 x = "mcnebula",
                                 data = "data.frame",
                                 columns = "character"),
          function(x, data, columns){
            if (length(columns) != 2)
              stop( "length(`columns`) != 2" )
            colnames(data)[which(colnames(data) == columns)] <-
              c(".features_id", ".candidates_id")
            create_reference(x, data = data)
          })

#' @exportMethod create_reference
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference",
                                 x = "mcnebula",
                                 data = "data.frame",
                                 columns = "integer"),
          function(x, data, columns){
            if (length(columns) != 2)
              stop( "length(`columns`) != 2" )
            colnames(data)[columns] <- c(".features_id", ".candidates_id")
            create_reference(x, data = data)
          })

#' @exportMethod create_reference
#' @rdname create_reference-methods
setMethod("create_reference",
          signature = setMissing("create_reference",
                                 x = "mcnebula",
                                 data = "data.frame"),
          function(x, data){
            if (any( duplicated(data[[ ".features_id" ]]) ))
              stop("`.features_id` in `data` were not unique")
            fun <- methods_match(project_api(x))[[ "generate_candidates_id" ]]
            data <- format_msframe(data, fun_format = fun)
            reference(mcn_dataset(x))[[ "specific_candidate" ]] <-
              dplyr::as_tibble(dplyr::select(data, .features_id, .candidates_id))
            return(x)
          })
```

42 File: methods-create__stardust__classes.R

```
# ==========================================================================
# filter classification for each features, as stardust classes
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases create_stardust_classes
#'
#' @title 'Inner' filter for PPCP data
#'
#' @description
#' Perform 'inner' filter for PPCP
#' (posterior probability of classification prediction) data of each 'feature',
#' then gathered as 'stardust_classes' data.
#' Run after [create_reference()].
#' Standby for next step [cross_filter_stardust()].
#'
#' @details
#' The PPCP data for each 'feature' contains the prediction of thousands of classes
#' for the potential compound (even if the chemical structure was unknown).
#' See \url{http://www.nature.com/articles/s41587-020-0740-8}
#' for details about the prediction.
#' The data contains attributes of:
#' - \code{class.name}: name of classes.
#' - \code{pp.value}: value of posterior probability.
#' - \code{hierarchy}: hierarchy of classes in the taxonomy.
#' See \url{https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0174-y}
#' for details about hierarchy and taxonomy of chemical classification.
#' - ...
#'
#' The method [create_stardust_classes()] use these inner attributes to
#' filter classes candidates for each 'feature'.
#'
#' @name create_stardust_classes-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod create_stardust_classes
#' @description \code{create_stardust_classes()}: get the default parameters for the method
#' \code{create_stardust_classes}.
#' @rdname create_stardust_classes-methods
setMethod("create_stardust_classes",
          signature = setMissing("create_stardust_classes",
                                 x = "missing"),
          function(){
            list(pp.threshold = 0.5,
                 hierarchy_priority = 5:2,
                 position_isomerism = T,
                 inherit_dataset = F)
          })

#' @exportMethod create_stardust_classes
#' @description \code{create_stardust_classes(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{create_stardust_classes}.
#' @rdname create_stardust_classes-methods
setMethod("create_stardust_classes",
          signature = c(x = "mcnebula"),
          function(x, pp.threshold, hierarchy_priority,
                   position_isomerism, inherit_dataset){
            reCallMethod("create_stardust_classes",
                         .fresh_param(create_stardust_classes()))
          })

#' @exportMethod create_stardust_classes
#'
#' @aliases create_stardust_classes
#'
#' @param x [mcnebula-class] object.
#' @param pp.threshold numeric(1) Threshold for PPCP. \code{pp.threshold = 0.5} may
#' work well.
#'
#' @param hierarchy_priority numeric. The specified hierarchy of classes to retain.
#' The other hierarchy would be filtered out. The hierarchy:
#' - n: ...
#' - 5: Classes of Level 5.
#' - 4: Classes of Subclass.
#' - 3: Classes of Class.
#' - 2: Classes of Super Class.
#' - ...
#'
#' @param position_isomerism logical. If \code{TRUE}, use pattern match
#' to filter out all classes names contains Arabic numerals.
#' Generally, these classes describe about the position of chemical functional group,
#' which were too subtle for machine to predict from LC-MS/MS spectrum.
#'
#' @param inherit_dataset logical. If \code{TRUE}, use latest PPCP data
#' formed by [filter_ppcp()]. i.e., data of:
#' - latest(x, subscript = ".f3_canopus")
#'
#' Else, run [filter_ppcp()].
#'
#' @rdname create_stardust_classes-methods
#'
#' @examples
#' \dontrun{
#'   test <- mcn_5features
#'
#'   ## the previous steps
#'   test1 <- filter_structure(test)
#'   test1 <- create_reference(test1)
#'
#'   test1 <- create_stardust_classes(test1)
#'   ## see results
#'   stardust_classes(test1)
#'   ## or
#'   reference(test1)$stardust_classes
#'   ## or
#'   reference(mcn_dataset(test1))$stardust_classes
#'
#'   ## the default parameters
#'   create_stardust_classes()
#' }
setMethod("create_stardust_classes",
          signature = c(x = "mcnebula",
                        pp.threshold = "numeric",
                        hierarchy_priority = "numeric",
                        position_isomerism = "logical",
                        inherit_dataset = "logical"),
          function(x, pp.threshold, hierarchy_priority,
                   position_isomerism, inherit_dataset){
            .message_info_formal("MCnebula2", "create_stardust_classes")
            if (is.null(hierarchy(x)))
              x <- create_hierarchy(x)
            hierarchy <- hierarchy(x)
            if (inherit_dataset) {
              dataset <- latest(x, subscript = ".f3_canopus")
              check <- dplyr::distinct(dataset, .features_id, .candidates_id)
              if (any( duplicated(check[[ ".features_id" ]]) ))
                stop("`.candidates_id` for features in ppcp dataset were not unique")
            } else {
              x <- filter_ppcp(x, pp.threshold = pp.threshold)
              dataset <- latest(x)
            }
            dataset <-
              merge(dplyr::select(dataset, .features_id, .candidates_id,
                                  pp.value, rel.index),
                    hierarchy, by = "rel.index", all.x = T)
            dataset <- dplyr::filter(dataset, hierarchy %in% hierarchy_priority)
            if (position_isomerism) {
```

```
          dataset <- dplyr::filter(dataset, !grepl("[0-9]", class.name))
        }
        reference(mcn_dataset(x))[[ "stardust_classes" ]] <-
          dplyr::relocate(dplyr::as_tibble(dataset), .features_id, .candidates_id)
        return(x)
      })
```

43 File: methods-cross_filter_stardust.R

```
# ==========================================================================
# across attributes of each other features to filter classes
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases cross_filter_stardust
#'
#' @title 'Cross' filter for 'stardust_classes' data
#'
#' @description
#' 'Cross' filter for 'stardust_classes' data.
#' Use 'features_annotation' data and 'stardust_classes' data for
#' chemical classes filtering. Run after [create_stardust_classes()].
#' Methods \code{cross_filter_stardust} are integration of the following three method:
#' - \code{cross_filter_quantity}
#' - \code{cross_filter_score}
#' - \code{cross_filter_identical}
#'
#' @details
#' Compared to the chemical class filtering within PPCP data by [create_stardust_classes()],
#' the filtering within 'stardust_classes' data by [cross_filter_stardust()] is
#' fundamentally different.
#'
#' - For [create_stardust_classes()],
#' the PPCP data belongs to each 'feature'. When performing the filtering,
#' only simple threshold conditions or absolute conditions
#' are set to filter the chemical classes;
#' there is no crossover between the different attributes and
#' no crossover between the 'features'.
#' Therefore, we consider this as 'inner' filtering.
#' - For [cross_filter_stardust()],
#' the data of the chemical classes and their classified 'features', i.e.
#' 'stardust_classes' data, were combined and then grouped upon the chemical classes.
#' After grouping, each chemical class has a certain quantity of "features".
#' When filtering, statistics may be performed on 'features' data within a group;
#' statistics may be performed on these data in conjunction with 'features_annotation' data;
#' and statistics may be performed to compare groups with each other.
#' As its crossover, we consider this as 'cross' filtering.
#'
#' @param x [mcnebula-class] object.
#' @param min_number numeric(1). Value in (1, ).
#' For classified 'features' of chemical classes, minimum quantity.
#'
#' @param max_ratio numeric(1). Value in (0, 1].
#' For classified 'features' of chemical classes,
#' maximum proportion: the 'features' quantity versus
#' all 'features' (unique) quantity of all classes.
#'
#' @param types character.
#' The target attributes for Goodness assessment. See details.
#' There can be plural ones.
#'
#' @param cutoff numeric.
#' For Goodness assessment of target attributes.
#' The size of the value depends on the target attribute.
#' Note, the \code{cutoff} must be 'vector' the same length as \code{types}.
#'
#' @param tolerance numeric. Value in (0, 1).
#' For Goodness assessment of target attributes. The thresholds of Goodness.
#' Note, the \code{tolerance} must be 'vector' the same length as \code{types}.
#'
#' @param hierarchy_range numeric(2).
#' The hierarchy range of chemical classification passed for similarity assessment
#' of chemical classes. The hierarchy:
#' - 10: ...
#' - n: ...
#' - 5: Classes of Level 5.
#' - 4: Classes of Subclass.
#' - 3: Classes of Class.
#' - 2: Classes of Super Class.
#' - 1: ...
#' - 0: ...
#'
#' @param identical_factor numeric(1). Value in (0, 1).
#' Threshold value for classes similarity assessment.
```

```
#'
#' @name cross_filter_stardust-methods
#'
#' @order 1
#'
#' @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)
#'
#'   ## the default parameters
#'   cross_filter_stardust()
#'   # This is a simulated dataset with only 5 'features',
#'   # so the default parameters are meaningless for it.
#'
#'   # Note that real datasets often contain thousands of "features"
#'   # and the following 'min_number' and 'max_ratio' parameter values are not suitable.
#'   test1 <- cross_filter_stardust(
#'     test1,
#'     min_number = 2,
#'     max_ratio = 1
#'   )
#'   ## see results
#'   stardust_classes(test1)
#'   ## or
#'   reference(test1)$stardust_classes
#'   ## or
#'   reference(mcn_dataset(test1))$stardust_classes
#'
#'   e1 <- stardust_classes(test1)
#'
#'   ## see the filtered classes
#'   backtrack_stardust(test1)
#'
#'   ## reset the 'stardust_classes'
#'   test1 <- create_stardust_classes(test1)
#'
#'   ## customized filtering
#'   # Note that real datasets often contain thousands of "features"
#'   # and the following 'min_number' and 'max_ratio' parameter values are not suitable.
#'   test1 <- cross_filter_quantity(test1, min_number = 2, max_ratio = 1)
#'   test1 <- cross_filter_score(test1,
#'     types = "tani.score",
#'     cutoff = 0.3,
#'     tolerance = 0.6
#'   )
#'   test1 <- cross_filter_identical(
#'     test1,
#'     hierarchy_range = c(3, 11),
#'     identical_factor = 0.7
#'   )
#'   e2 <- stardust_classes(test1)
#'
#'   identical(e1, e2)
#'
#'   ## reset
#'   test1 <- create_stardust_classes(test1)
#'   ## targeted plural attributes
#'   test1 <- cross_filter_stardust(
#'     test1,
#'     min_number = 2,
#'     max_ratio = 1,
#'     types = c("tani.score", "csi.score"),
#'     cutoff = c(0.3, -150),
#'     tolerance = c(0.6, 0.3)
#'   )
#' }
NULL
#> NULL

#' @exportMethod cross_filter_stardust
#' @description \code{cross_filter_stardust()}:
#' get the default parameters for the method
#' \code{cross_filter_stardust}.
#' @rdname cross_filter_stardust-methods
setMethod("cross_filter_stardust",
```

```r
          signature = setMissing("cross_filter_stardust",
                                 x = "missing"),
          function(){
            list(min_number = 30,
                 max_ratio = 0.1,
                 types = "tani.score",
                 cutoff = 0.3,
                 tolerance = 0.6,
                 hierarchy_range = c(3, 11),
                 identical_factor = 0.7
            )
          })

#' @exportMethod cross_filter_stardust
#' @description \code{cross_filter_stardust(x, ...)}:
#' use the default parameters whatever 'missing'
#' while performing the method \code{cross_filter_stardust}.
#'
#' @rdname cross_filter_stardust-methods
setMethod("cross_filter_stardust",
          signature = c(x = "mcnebula"),
          function(x, min_number, max_ratio,
                   types, cutoff, tolerance,
                   hierarchy_range, identical_factor){
            .message_info_formal("MCnebula2", "cross_filter_stardust")
            .check_data(x, list(stardust_classes = "create_stardust_classes"))
            new_args <- .fresh_param(cross_filter_stardust())
            methods <- c("cross_filter_quantity", "cross_filter_score",
                         "cross_filter_identical")
            for (i in methods) {
              args <- new_args[names(new_args) %in% formalArgs(i)]
              new_args[[ "x" ]] <- do.call(match.fun(i), args)
            }
            backtrack(mcn_dataset(new_args[[ "x" ]]))[[ "stardust_classes" ]] <-
              stardust_classes(x)
            return(new_args[[ "x" ]])
          })

#' @exportMethod cross_filter_quantity
#'
#' @aliases cross_filter_quantity
```

```
#'
#' @description
#' \code{cross_filter_quantity}:
#' Filter chemical classes in 'stardust_classes' data according to:
#' - Absolute quantity: the classified 'features' of the class.
#' - Relative proportion: the classified 'features' of the class
#' comparing with all features of all classes.
#'
#' @details
#' \bold{Cross_filter_quantity}
#' Set 'features' quantity limitation for each group.
#' The groups with too many 'features' or too few 'features' would be filtered out.
#' This means the chemical class would be filtered out.
#' These thresholds are about:
#' - Minimum quantity: the 'features'.
#' - Maximum proportion: the 'features' quantity versus
#' all 'features' (unique) quantity of all groups.
#'
#' The purpose of this step is to filter out chemical classes that have
#' too large or too subtle a conceptual scope. For example, 'Organic compounds',
#' which covers almost all compounds that can be detected in metabolomics data,
#' is too large in scope to be of any help to our biological research.
#' The setting of parameters is not absolute, and there is no optimal solution.
#' Users can draw up thresholds according to the necessity of the study.
#'
#' @rdname cross_filter_stardust-methods
#'
setMethod("cross_filter_quantity",
          signature = setMissing("cross_filter_quantity",
                                 x = "mcnebula", min_number = "numeric",
                                 max_ratio = "numeric"),
          function(x, min_number, max_ratio){
            .message_info("cross_filter_stardust", "quantity")
            if (min_number < 1) {
              stop( "`min_number` must be a numeric greater or equal to 1" )
            }
            if (!(max_ratio <= 1 & max_ratio > 0)) {
              stop( "`max_ratio` must be a numeric within (0, 1]" )
            }
            sum <- length( unique(stardust_classes(x)[[ ".features_id" ]]) )
            set <- split(stardust_classes(x), ~ rel.index)
            set <- lapply(set, function(df)
                          if (nrow(df) >= min_number &
                              nrow(df) / sum <= max_ratio) df)
            reference(mcn_dataset(x))[[ "stardust_classes" ]] <-
              dplyr::as_tibble(data.table::rbindlist(set))
            return(x)
          })

#' @exportMethod cross_filter_score
#'
#' @aliases cross_filter_score
#'
#' @description
#' \code{cross_filter_score}
#' Filter chemical classes in 'stardust_classes' data according to the attributes
#' in 'features_annotation' data.
#' Set cut-off value of attributes for all 'features', then inspect overall
#' satisfaction of the classified 'features' of the class.
#'
#' @details
#' \bold{Cross_filter_score}
#' This step associate 'stardust_classes' data with 'features_annotation' data.
#' For each group, the Goodness assessment is performed
#' for each target attribute (continuous attribute, generally be a scoring
#' attribute of compound identification, such as 'tani.score'). If the group met all the
#' expected Goodness, the chemical class would be retained; otherwise,
#' the chemical class would be filtered out.
#' The Goodness (G) related with the 'features' within the group:
#' - n: the quantity of 'features' of which target attributes satisfied with the cut-off.
#' - N: the quantity of all 'features'.
#'
#' The Goodness: G = n / N.
#'
#' The assessment of Goodness is related to the parameters of \code{cutoff}
#' and \code{tolerance}:
#' - Expected Goodness, i.e. value of \code{tolerance}.
#' - Actual Goodness, related to parameter \code{cutoff}. G = n / N.
#'
#' Goodness assessment can be given to plural target attributes.
#' Note that the chemical class would retained only if
#' it passed the Goodness assessment of all target attributes.
```

```r
#'
#' The main purpose of this step is to filter out those chemical classes with
#' too many 'features' of low structural identification.
#'
#' @rdname cross_filter_stardust-methods
#'
setMethod("cross_filter_score",
          signature = setMissing("cross_filter_score",
                                 x = "mcnebula", types = "character",
                                 cutoff = "numeric", tolerance = "numeric"),
          function(x, types, cutoff, tolerance){
            .message_info("cross_filter_stardust", "score")
            .check_data(x, list(features_annotation = "create_features_annotation"))
            set <- split(stardust_classes(x), f = ~ rel.index)
            features <- features_annotation(x)
            res <- mapply(types, cutoff, tolerance,
                          SIMPLIFY = F, USE.NAMES = F,
                          FUN = function(type, cutoff, tolerance){
                            if (!is.numeric(features[[ type ]]))
                              stop("the columns of `types` were not numeric")
                            express <- parse(text = type)
                            ref <- dplyr::filter(features, eval(express) >= cutoff)
                            ref <- ref[[ ".features_id" ]]
                            vapply(set, FUN.VALUE = T, USE.NAMES = F,
                                   function(data){
                                     ref <- ref[ref %in% data[[ ".features_id" ]]]
                                     if (length(ref) / nrow(data) >= tolerance)
                                       return(T)
                                     else
                                       return(F)
                                   })
                          })
            if (length(res) == 1) {
              set <- set[unlist(res)]
            } else {
              logic  <- res[[1]]
              for (i in res[2:length(res)]) {
                logic <- logic & i
              }
              set <- set[logic]
            }
            reference(mcn_dataset(x))[[ "stardust_classes" ]] <-
              dplyr::as_tibble(data.table::rbindlist(set))
            return(x)
          })

#' @exportMethod cross_filter_identical
#'
#' @aliases cross_filter_identical
#'
#' @description
#' \code{cross_filter_identical}
#' Filter chemical classes in 'stardust_classes' data by comparing the classified 'features'.
#'
#' @details
#' \bold{Cross_filter_identical}
#' A similarity assessment of chemical classes.
#' Set a hierarchical range for chemical classification and let groups (
#' each group, i.e. a chemical class with its classified 'features')
#' within this range be compared for similarity to each other. For two groups,
#' if the classified 'features' almost identical to each other, the chemical
#' class represented by one of the groups would be discarded.
#' The assessment of identical degree of two groups (A and B):
#' - x: ratio of the classified 'features' of A belonging to B
#' - y: ratio of the classified 'features' of B belonging to A
#' - i: value of parameter \code{identical_factor}
#'
#' If x > i and y > i, the two groups would be considered as identical.
#' Then the group with fewer 'features' would be discarded.
#'
#' The purpose of this step is to filter out classes that may incorporate
#' each other and are similar in scope. The in silico prediection approach may not be able
#' to distinguish which class the potential compound belongs to from the LC-MS/MS spectra.
#'
#' @rdname cross_filter_stardust-methods
#'
setMethod("cross_filter_identical",
          signature = setMissing("cross_filter_identical",
                                 x = "mcnebula", hierarchy_range = "numeric",
                                 identical_factor = "numeric"),
          function(x, hierarchy_range, identical_factor){
            .message_info("cross_filter_stardust", "identical")
            set <- dplyr::filter(stardust_classes(x),
                                 hierarchy %in% hierarchy_range)
            set <- split(set, f = ~ rel.index)
            ids <- lapply(set, `[[`, ".features_id")
            groups <- combn(1:length(ids), 2, simplify = F)
            discard <- lapply(groups,
                          function(group){
                            if (any( ids[group[1]] %in% ids[group[2]] )) {
                              p <- mapply(c(1, 2), c(2, 1),
                                          SIMPLIFY = F,
                                          FUN = function(x, y){
                                            table(ids[group[x]] %in% ids[group[y]])[[ "TRUE" ]]
                                          })
                              if (p[[1]] > identical_factor & p[[2]] > identical_factor) {
                                if (length(ids[group[1]]) < length(ids[group[2]]))
                                  return(group[1])
                                else
                                  return(group[2])
                              }
                            }
                          })
            discard_index <-
              unique( data.table::rbindlist( set[unlist(discard)] )[[ "rel.index" ]]
              )
            reference(mcn_dataset(x))[[ "stardust_classes" ]] <-
              dplyr::filter(stardust_classes(x), !rel.index %in% discard_index)
            return(x)
          })
```

44 File: methods-filter_formula.R

```
# ==========================================================================
# collate formula dataset in sirius project and do filtering
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
#' @aliases filter_formula
#'
#' @title Collate and filter candidates of chemical formula for each 'feature'
#'
#' @description This methods provide an approach to
#' collate and filter chemical formula candidates data in baches for each
#' 'feature'.
```

```
#'
#' @details In SIRIUS project directory, if the computation job has done,
#' each 'feature' has multiple prediction candidates whether for chemical formula,
#' structure, or classification. This method provides an approach to collate
#' and filter these data in baches. See \link{MCnebula2} for details of chemical
#' formula, structure and classification.
#'
#' @name filter_formula-methods
#'
#' @order 1
NULL
#> NULL

#' @exportMethod filter_formula
#' @description \code{filter_formula()}: get the default parameters for the method
#' \code{filter_formula}.
#' @rdname filter_formula-methods
setMethod("filter_formula",
          signature = setMissing("filter_formula",
                                 x = "missing"),
          function(){
            list(fun_filter = .rank_by_default,
                 by_reference = F
            )
          })

#' @exportMethod filter_formula
#' @description \code{filter_formula(x, ...)}: use the default parameters whatever 'missing'
#' while performing the method \code{filter_formula}.
#' @rdname filter_formula-methods
setMethod("filter_formula",
          signature = c(x = "mcnebula"),
          function(x, fun_filter, ..., by_reference){
            reCallMethod("filter_formula",
                         .fresh_param(filter_formula()), ...)
          })

#' @exportMethod filter_formula
#'
#' @aliases filter_formula
#'
```

```
#' @param x [mcnebula-class] object.
#'
#' @param fun_filter function. Used to filter data.frame. The function would
#' run for candidates data (data.frame) for each 'features'. Such as:
#' - \code{lapply(split(all_data, ~.features_id), fun_filter, ...)}.
#'
#' This parameter provides an elegant and flexible way to filter data.
#' Users can pass function [dplyr::filter()] to specify
#' any attributes condition to filter the data.
#'
#' @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
```

```r
#'      !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
  }
```

45 File: methods-filter_ppcp.R

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

46 File: methods-filter_structure.R

```
# ==========================================================================
# 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)
```

```r
#'
#'   ## 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)
          })
```

47 File: methods-initialize_mcnebula.R

```r
# ==========================================================================
# 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)
          })
```

```r
#' @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)
          })
```

48 File: methods-visualize.R

```r
# ==========================================================================
# extract and visualize 'ggset' in 'mcnebula' object
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```r
#' @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))`" )
              }
            }
          })
```

```r
#' @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
```

```r
#' 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)
    }
    if (any(n <- grepl("scale_fill_manual", names(layers(ggset))))) {
      pal <- command_args(layers(ggset)[n][[1]])$values
      if (!is.null(pal)) {
        data <- data.frame(tracer = names(pal))
        layer <- new_command(ggplot2::geom_point,
          data = data, mapping = aes(x = 0L, y = 0L, fill = tracer),
          shape = 21, stroke = 0
        )
        ggset <- add_layers(ggset, layer)
      }
    }
    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))
```

```
}
```

49 File: project.sirius.v4.R

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

```r
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))
      }
    )
  }
```

50 File: project.sirius.v5.R

```
# ==========================================================================
# 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 ]])
```

```r
  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
```

51 File: tools-colors.R

```r
# ==========================================================================
# 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)
  }
```

52 File: tools-default_visualize.R

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

```r
.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)
  }
```

```r
.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))
  }
```

53 File: tools-export.R

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

54 File: tools-methods.R

```
# ==========================================================================
# 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
  }
```

55 File: tools-modify_ggset.R

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

```r
#' @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()]).
```

```r
#'
#' @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
```

```r
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
```

```r
                    if (!is.null(mapping)) {
                      vapply(mapping, FUN.VALUE = "ch",
                             function(m) tail(paste0(m), 1))
                    }
                  }))
  }
```

56 File: tools-MSnbase-MODIFIED__compareSpectra.R

```r
# ==========================================================================
# generic and methods of `compareSpectra` stripped from package of
# MSnbase and ProtGenerics
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
## from ProtGenerics
setGeneric("compareSpectra", function(x, y, ...)
    standardGeneric("compareSpectra"))
setGeneric("mz", function(object, ...) standardGeneric("mz"))
setGeneric("intensity", function(object, ...) standardGeneric("intensity"))

## from MSnbase and modified
setClass("lightSpectrum",
         representation =
           representation(mz = "numeric",
                          intensity = "numeric"),
         prototype =
           prototype(mz = numeric(),
                     intensity = numeric())
)
setMethod("mz", "lightSpectrum",
          function(object) object@mz)
setMethod("intensity", "lightSpectrum",
          function(object) object@intensity)
setMethod("compareSpectra", c("lightSpectrum", "lightSpectrum"),
          function(x, y) {
            binnedSpectra <- bin_Spectra(x, y)
            do.call(dotproduct, binnedSpectra)
          })
```

```r
# ==========================================================================
# function
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```r
bin_Spectra <-
  function(object1, object2, binSize = 1L,
           breaks = seq(floor(min(c(mz(object1), mz(object2)))),
                        ceiling(max(c(mz(object1), mz(object2)))),
                        by = binSize)) {
    breaks <- .fix_breaks(breaks, range(mz(object1), mz(object2)))
    list(bin_Spectrum(object1, breaks = breaks),
         bin_Spectrum(object2, breaks = breaks))
}

bin_Spectrum <-
  function(object, binSize = 1L, breaks, fun = sum) {
    ints <- .bin_values(object@intensity, object@mz, binSize = binSize,
                        breaks = breaks, fun = fun)
    return(ints)
}

#' The function aggregates `x` for `toBin` falling into bins defined
#' by `breaks` using the `fun` function.
#'
#' @details
#' This is a combination of the code from the former bin_Spectrum.
#'
#' @param x `numeric` with the values that should be binned.
#'
#' @param toBin `numeric`, same length than `x`, with values to be used for the
#'     binning.
#'
#' @param binSize `numeric(1)` with the size of the bins.
#'
#' @param breaks `numeric` defining the breaks/bins.
#'
#' @param fun `function` to be used to aggregate values of `x` falling into the
#'     bins defined by `breaks`.
#'
#' @return `list` with elements `x` and `mids` being the aggregated values
#'     of `x` for values in `toBin` falling within each bin and the bin mid
#'     points. --been modified, only return `x`.
#'
#' @noRd
```

```
.bin_values <-
  function(x, toBin, binSize = 1L, breaks, fun) {
    breaks <- .fix_breaks(breaks, range(toBin))
    nbrks <- length(breaks)
    idx <- findInterval(toBin, breaks)
    ## Ensure that indices are within breaks.
    idx[which(idx < 1L)] <- 1L
    idx[which(idx >= nbrks)] <- nbrks - 1L

    ints <- double(nbrks - 1L)
    ints[unique(idx)] <- unlist(lapply(base::split(x, idx), fun),
                                use.names = FALSE)
    return(ints)
}

#' Simple function to ensure that breaks (for binning) are span al leat the
#' expected range.
#'
#' @param brks `numeric` with *breaks* such as calculated by `seq`.
#'
#' @param rng `numeric(2)` with the range of original numeric values on which
#'     the breaks were calculated.
#'
#' @noRd

.fix_breaks <-
  function(brks, rng) {
    ## Assuming breaks being sorted.
    if (brks[length(brks)] <= rng[2]) {
      brks <- c(brks, max((rng[2] + 1e-6),
                          brks[length(brks)] + mean(diff(brks))))
    }
    brks
}

#' calculate the dot product between two vectors
#'
#' Stein, S. E., and Scott, D. R. (1994).
#' Optimization and testing of mass spectral library search algorithms for
#' compound identification.
#' Journal of the American Society for Mass Spectrometry, 5(9), 859-866.
```

```r
#' doi: https://doi.org/10.1016/1044-0305(94)87009-8
#'
#' Lam, H., Deutsch, E. W., Eddes, J. S., Eng, J. K., King, N., Stein, S. E.
#' and Aebersold, R. (2007)
#' Development and validation of a spectral library searching method for peptide
#' identification from MS/MS.
#' Proteomics, 7: 655-667.
#' doi: https://doi.org/10.1002/pmic.200600625
#'
#' @param x double
#' @param y double
#' @return double, length == 1
#' @noRd

dotproduct <- function(x, y) {
  as.vector(x %*% y) / (sqrt(sum(x*x)) * sqrt(sum(y*y)))
}
```

57 File: tools-report.R

```r
# ==========================================================================
# 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
```

58 File: tools-xcms-feature_detection.R

```
# ==========================================================================
# use XCMS to perform Feature Dectection
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# the following classes would import while used...
# @importClassesFrom MSnbase OnDiskMSnExp
# @importClassesFrom MSnbase MSpectra
# @importClassesFrom S4Vectors DFrame
# @importClassesFrom xcms XCMSnExp

# setClass("OnDiskMSnExp")
# setClass("MSpectra")
# setClass("DFrame")
# setClass("XCMSnExp")

#' @exportClass detectFlow
#'
#' @aliases detectFlow
#'
#' @title Steps in sequence to perform Feature Detection
#'
#' @description A class inherits from [layerSet-class] to store steps for
#' Feature Detection.
```

```
#'
#' @seealso [layerSet-class], [mcmass-class]
#'
#' @slot layers list. a list of objects [command-class].
#'
#' @rdname detectFlow-class
#'
#' @examples
#' \dontrun{
#' new('detectFlow', ...)
#' }
.detectFlow <-
  setClass("detectFlow",
    contains = "layerSet",
    representation = representation(),
    prototype = NULL
  )

#' @exportMethod show_layers
#' @description See \code{show_layers} in [ggset-class].
#' @rdname detectFlow-class
setMethod("show_layers",
  signature = c(x = "detectFlow"),
  function(x){
    selectMethod("show_layers", "ggset")(x)
  })

#' @exportClass toBeEval
#'
#' @aliases toBeEval
#'
#' @description character(1) to be eval while perform [run_lcms()].
#' @rdname detectFlow-class
toBeEval <- setClass("toBeEval", contains = c("character"), prototype = "pro_data(x)")

#' @exportClass mcmass
#'
#' @aliases mcmass
#'
#' @title storage of XCMS processed data of 'feature detection'
#'
```

```r
#' @description This class provides a process template for pre-processing
#' non-targeted mass spectrometry data with package \code{xcms}.
#' In default, the steps of Feature Detection were refer to:
#' \link{https://github.com/DorresteinLaboratory/XCMS3_FeatureBasedMN}
#'
#' @slot raw_data [OnDiskMSnExp-class] object.
#' @slot pro_data [XCMSnExp-class] object.
#' @slot sample_metadata data.frame. User supplied data about the mass data
#' files to process.
#' @slot features_defination [DFrame-class] object. Non user supplied data.
#' See \code{xcms::featureDefinitions}.
#' @slot features_quantification data.frame. Non user supplied data.
#' Peak area were used to quantify the feature level.
#' @slot ms2_spectra [MSpectra-class] object. Non user supplied data.
#' See \code{xcms::filteredMs2Spectra}.
#' @slot parameter_set list. Passed to repalace the parameters in slot
#' \code{detectFlow} of [mcmass-class] for performing [run_lcms()].
#' @slot detectFlow [detectFlow-class] object.
#'
#' @rdname mcmass-class
#'
#' @examples
#' \dontrun{
#' new('mcmass', ...)
#' }
.mcmass <-
  suppressWarnings(setClass("mcmass",
    contains = character(),
    representation = representation(
      raw_data = "OnDiskMSnExp",
      pro_data = "XCMSnExp",
      sample_metadata = "data.frame",
      features_defination = "DFrame",
      features_quantification = "data.frame",
      ms2_spectra = "MSpectra",
      parameter_set = "list",
      detectFlow = "detectFlow"
      ),
    prototype = prototype(detectFlow = .detectFlow())
  ))
```

```
# ==========================================================================
# methods (getter or setter)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

setGeneric("raw_data",
  function(x) standardGeneric("raw_data"))
setGeneric("raw_data<-",
  function(x, value) standardGeneric("raw_data<-"))

#' @exportMethod raw_data
#' @aliases raw_data
#' @description \code{raw_data}, \code{raw_data<-}: getter and setter
#' for the \code{raw_data} slot of the object.
#' @rdname mcmass-class
setMethod("raw_data",
  signature = c(x = "mcmass"),
  function(x){
    x@raw_data
  })

setGeneric("pro_data",
  function(x) standardGeneric("pro_data"))
setGeneric("pro_data<-",
  function(x, value) standardGeneric("pro_data<-"))

#' @exportMethod pro_data
#' @aliases pro_data
#' @description \code{pro_data}, \code{pro_data<-}: getter and setter
#' for the \code{pro_data} slot of the object.
#' @rdname mcmass-class
setMethod("pro_data",
  signature = c(x = "mcmass"),
  function(x){
    x@pro_data
  })

#' @exportMethod pro_data<-
#' @aliases pro_data<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("pro_data",
  signature = c(x = "mcmass"),
  function(x, value){
    initialize(x, pro_data = value)
  })

#' @exportMethod raw_data<-
#' @aliases raw_data<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("raw_data",
  signature = c(x = "mcmass"),
  function(x, value){
    initialize(x, raw_data = value)
  })

#' @exportMethod sample_metadata
#' @aliases sample_metadata
#' @rdname mcmass-class
setMethod("sample_metadata",
  signature = c(x = "mcmass"),
  function(x){
    x@sample_metadata
  })

#' @exportMethod sample_metadata<-
#' @aliases sample_metadata<-
#' @rdname mcmass-class
setReplaceMethod("sample_metadata",
  signature = c(x = "mcmass"),
  function(x, value){
    .check_columns(value, list("file", "sample", "group"),
      "sample_metadata")
    x@sample_metadata <- value
    return(x)
  })

#' @exportMethod features_quantification
#' @aliases features_quantification
#' @description \code{features_quantification}, \code{features_quantification<-}: getter and setter
#' for the \code{features_quantification} slot of the object.
#' @rdname mcmass-class
```

```r
setMethod("features_quantification",
  signature = c(x = "mcmass"),
  function(x){
    x@features_quantification
  })

#' @exportMethod features_quantification<-
#' @aliases features_quantification<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("features_quantification",
  signature = c(x = "mcmass", value = "data.frame"),
  function(x, value){
    initialize(x, features_quantification = value)
  })

setGeneric("features_defination",
  function(x) standardGeneric("features_defination"))
setGeneric("features_defination<-",
  function(x, value) standardGeneric("features_defination<-"))

#' @exportMethod features_defination
#' @aliases features_defination
#' @description \code{features_defination}, \code{features_defination<-}: getter and setter
#' for the \code{features_defination} slot of the object.
#' @rdname mcmass-class
setMethod("features_defination",
  signature = c(x = "mcmass"),
  function(x){
    x@features_defination
  })

#' @exportMethod features_defination<-
#' @aliases features_defination<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("features_defination",
  signature = c(x = "mcmass"),
  function(x, value){
    initialize(x, features_defination = value)
  })
```

```r
setGeneric("ms2_spectra",
  function(x) standardGeneric("ms2_spectra"))
setGeneric("ms2_spectra<-",
  function(x, value) standardGeneric("ms2_spectra<-"))

#' @exportMethod ms2_spectra
#' @aliases ms2_spectra
#' @description \code{ms2_spectra}, \code{ms2_spectra<-}: getter and setter
#' for the \code{ms2_spectra} slot of the object.
#' @rdname mcmass-class
setMethod("ms2_spectra",
  signature = c(x = "mcmass"),
  function(x){
    x@ms2_spectra
  })

#' @exportMethod ms2_spectra<-
#' @aliases ms2_spectra<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("ms2_spectra",
  signature = c(x = "mcmass"),
  function(x, value){
    initialize(x, ms2_spectra = value)
  })

setGeneric("parameter_set",
  function(x) standardGeneric("parameter_set"))
setGeneric("parameter_set<-",
  function(x, value) standardGeneric("parameter_set<-"))

#' @exportMethod parameter_set
#' @aliases parameter_set
#' @description \code{parameter_set}, \code{parameter_set<-}: getter and setter
#' for the \code{parameter_set} slot of the object.
#' @rdname mcmass-class
setMethod("parameter_set",
  signature = c(x = "mcmass"),
  function(x){
    x@parameter_set
  })
```

```
#' @exportMethod parameter_set<-
#' @aliases parameter_set<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("parameter_set",
  signature = c(x = "mcmass", value = "list"),
  function(x, value){
    initialize(x, parameter_set = value)
  })

setGeneric("detectFlow",
  function(x) standardGeneric("detectFlow"))
setGeneric("detectFlow<-",
  function(x, value) standardGeneric("detectFlow<-"))

#' @exportMethod detectFlow
#' @aliases detectFlow
#' @description \code{detectFlow}, \code{detectFlow<-}: getter and setter
#' for the \code{detectFlow} slot of the object.
#' @rdname mcmass-class
setMethod("detectFlow",
  signature = c(x = "mcmass"),
  function(x){
    x@detectFlow
  })

#' @exportMethod detectFlow<-
#' @aliases detectFlow<-
#' @param value The value for the slot.
#' @rdname mcmass-class
setReplaceMethod("detectFlow",
  signature = c(x = "mcmass", value = "detectFlow"),
  function(x, value){
    initialize(x, detectFlow = value)
  })
```

```
# ==========================================================================
# main methods
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

setGeneric("run_lcms",
```

```r
  function(x) standardGeneric("run_lcms"))

#' @exportMethod run_lcms
#'
#' @title Perform Feature Detection with steps in 'detectFlow'
#' @description This would use package mainly 'xcms' to perform
#' Feature Detection.
#'
#' @aliases run_lcms
#'
#' @param x [mcmass-class] object.
#'
#' @rdname run_lcms-methods
#'
#' @examples
#' \dontrun{
#' run_lcms(...)
#' }
setMethod("run_lcms",
  signature = c(x = "mcmass"),
  function(x){
    .message_info_formal("run_lcms", "")
    steps <- layers(detectFlow(x))
    .message_info("run_lcms", command_name(steps[[ 1 ]]))
    raw_data(x) <- call_command(steps[[ 1 ]])
    env <- environment()
    for (i in 2:length(steps)) {
      command_args(steps[[ i ]]) <- freshToBe(command_args(steps[[ i ]]), env)
      name <- command_name(steps[[ i ]])
      param <- parameter_set(x)[[ name ]]
      if (is.null(param)) {
        param <- parameter_set(x)[[ sub("^[.a-zA-Z0-9_]*::", "", name) ]]
      }
      if (!is.null(param)) {
        for (j in names(param)) {
          if (any(j == names(command_args(steps[[ i ]])))) {
            command_args(steps[[ i ]])[[ j ]] <- param[[ j ]]
          }
        }
      }
      .message_info("run_lcms", name)
      pro_data(x) <- call_command(steps[[i]])
    }
    layers(detectFlow(x)) <- steps
    return(x)
  })

freshToBe <- function(lst, envir) {
  lapply(lst,
    function(obj) {
      if (is(obj, "toBeEval")) {
        eval(parse(text = obj), envir = envir)
      } else {
        obj
      }
    })
}

#' @export run_export
#' @aliases run_export
#' @description \code{run_export}: get \code{features_quantification}
#' and \code{ms2_spectra} in [mcmass-class] object.
#' @param keep_onlyWithMs2 logical(1). If \code{TRUE}, the data
#' \code{features_quantification} only keep features which possess MS2.
#' @param saveMgf NULL or character(1). Use \code{MSnbase::writeMgfData} to
#' output the slot \code{ms2_spectra} of [mcmass-class] as .mgf file.
#' @param mzd passed to \code{MSnbase::combineSpectra}
#' @param minProp passed to \code{MSnbase::combineSpectra}
#' @param ppm passed to \code{MSnbase::combineSpectra}
#' @param ... passed to \code{MSnbase::combineSpectra}
#' @rdname run_lcms-methods
run_export <- function(x, keep_onlyWithMs2 = T,
  saveMgf = NULL, mzd = 0, minProp = .3, ppm = 20, ...)
{
  ## mass level 2
  ms2 <- xcms::featureSpectra(pro_data(x), return.type = "MSpectra")
  ms2 <- MSnbase::clean(ms2, all = TRUE)
  .message_info("run_export", "MSnbase::combineSpectra")
  ms2_spectra(x) <- MSnbase::combineSpectra(
    ms2, fcol = "feature_id", method = MSnbase::consensusSpectrum,
    mzd = 0, minProp = 0.3, ppm = 20, ...
  )
  if (!is.null(saveMgf)) {
    MSnbase::writeMgfData(ms2_spectra(x), saveMgf)
  }
  ## mass level 1
  features_defination(x) <- xcms::featureDefinitions(pro_data(x))
  quant <- xcms::featureValues(pro_data(x), value = "into")
  quant <- data.frame(quant)
  colnames(quant) <- sample_metadata(x)$sample
  quant$.features_id <- rownames(quant)
  quant <- dplyr::relocate(tibble::as_tibble(quant), .data$.features_id)
  if (keep_onlyWithMs2) {
    quant <- dplyr::filter(
      quant, .features_id %in% ms2_spectra(x)@elementMetadata$feature_id
    )
  }
  features_quantification(x) <- quant
  return(x)
}
```

```r
# ==========================================================================
# default detectFlow
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @export default_detectFlow
#' @aliases default_detectFlow
#' @param x [mcmass-class] object.
#' @param snthresh passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param noise passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param peakwidth passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param ppm passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param minFraction passed to \code{xcms::PeakDensityParam} for
#' \code{xcms::fillChromPeaks}
#' @description \code{default_detectFlow}: Create [detectFlow-class] object for
#' Feature Detection.
#' @rdname detectFlow-class
default_detectFlow <- function(x,
  snthresh = 5, noise = 50000, peakwidth = c(3, 30), ppm = 20,
  minFraction = .1)
```

```r
{
  if (!is(x, "mcmass")) {
    stop("`x` must be a 'mcmass' object.")
  }
  args <- list(
    new_command(MSnbase::readMSData,
      files = sample_metadata(x)$file,
      centroided. = TRUE, mode = "onDisk",
      pdata = new("NAnnotatedDataFrame", sample_metadata(x))
      ),
    new_command(xcms::findChromPeaks, object = toBeEval("raw_data(x)"),
      param = xcms::CentWaveParam(snthresh = snthresh,
        noise = noise, peakwidth = peakwidth, ppm = ppm)
      ),
    new_command(xcms::adjustRtime, object = toBeEval(),
      param = xcms::ObiwarpParam()
      ),
    new_command(xcms::groupChromPeaks, object = toBeEval(),
      param = xcms::PeakDensityParam(
        sampleGroups = sample_metadata(x)$group,
        minFraction = minFraction)
      ),
    new_command(xcms::fillChromPeaks, object = toBeEval(),
      param = xcms::ChromPeakAreaParam()
    )
  )
  names(args) <- vapply(args, command_name, "ch")
  new("detectFlow", layers = args)
}

#' @importFrom methods new
#' @importFrom methods getClass
#' @export new_mcmass
#' @aliases new_mcmass
#' @description \code{new_mcmass}: Create a [mcmass-class] object.
#' @param sample_metadata data.frame. Contains columns of 'file', 'sample',
#' 'group'
#' @param snthresh passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param noise passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param peakwidth passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param ppm passed to \code{xcms::CentWaveParam} for
#' \code{xcms::findChromPeaks}
#' @param minFraction passed to \code{xcms::PeakDensityParam} for
#' \code{xcms::fillChromPeaks}
#' @rdname mcmass-class
new_mcmass <- function(sample_metadata,
  snthresh = 5, noise = 50000, peakwidth = c(3, 30),
  ppm = 20, minFraction = .1)
{
  .suggest_bio_package("xcms")
  mcm <- .mcmass(
    raw_data = new(getClass("OnDiskMSnExp", where = "MSnbase")),
    pro_data = new(getClass("XCMSnExp", where = "xcms")),
    features_defination = new(getClass("DFrame", where = "S4Vectors")),
    ms2_spectra = new(getClass("MSpectra", where = "MSnbase")),
    sample_metadata = sample_metadata,
    detectFlow = .detectFlow()
  )
  detectFlow(mcm) <- default_detectFlow(
    mcm, snthresh = snthresh, noise = noise, peakwidth = peakwidth,
    ppm = ppm, minFraction = minFraction
  )
  return(mcm)
}

#' @export set_biocParallel
#' @aliases set_biocParallel
#' @description \code{set_biocParallel}: Set global parrallel processing for
#' package \code{xcms} or relative packages. See \code{BiocParallel::register}.
#' @param workers integer(1). See \code{BiocParallel::MulticoreParam} or
#' \code{BiocParallel::SnowParam} for help
#' @param ... Other Parameters passed to \code{BiocParallel::MulticoreParam} or
#' \code{BiocParallel::SnowParam}.
#' @rdname run_lcms-methods
set_biocParallel <- function(workers, ...) {
  if (.Platform$OS.type == "unix") {
    BiocParallel::register(
      BiocParallel::bpstart(
        BiocParallel::MulticoreParam(workers, ...)))
  } else {
    BiocParallel::register(
      BiocParallel::bpstart(
        BiocParallel::SnowParam(workers, ...)))
  }
}
```

```r
# ==========================================================================
# the following functions were source from
# https://raw.githubusercontent.com/jorainer/xcms-gnps-tools/master/customFunctions.R
# but revised
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

#' @title Format MS2 spectra for export in GNPS-MGF format
#'
#' @description
#'
#' Re-format MS2 spectrum information for export of the data in Mgf format
#' supported by GNPS. In detail, the function replaces the acquisition number
#' of each spectrum with the feature ID (expected to be present in the
#' `"feature_id"` column of `mcols(x)`) converted to an integer by removing
#' the ID's leading `"FT"`.
#'
#' @param x `Spectra`.
#'
#' @return `Spectra` with the acquisition number replaced.
#'
#' @author Johannes Rainer
#'
#' @noRd
formatSpectraForGNPS <- function(x) {
  fids <- S4Vectors::mcols(x)$feature_id
  if (!length(fids))
    stop("No column named 'feature_id' present in 'mcols(x)'")
  fids <- as.integer(sub("^FT", "", fids))
  S4Vectors::mendoapply(x, fids, FUN = function(z, id) {
    z@acquisitionNum <- id
    z
  })
}

#' @title Plot multiple spectra into the same plot
```

```r
#'
#' @description
#'
#' Plot multiple spectra into the same plot.
#'
#' @param x `Spectra` that should be plotted.
#'
#' @param col color to be used for the individual peaks.
#'
#' @param type `character(1)` defining the plot type. Defaults to `"h"` to plot
#'     vertical lines. For more details see documentation of `plot`.
#'
#' @param main `character(1)` defining the title.
#'
#' @param ... additional arguments to be passed to `points`.
#'
#' @author Johannes Rainer
#'
#' @noRd
plotSpectra <- function(x, col = "#00000040", type = "h", main, ...) {
  xcms::plot(3, 3, pch = NA, xlab = "m/z", ylab = "intensity",
    xlim = range(ProtGenerics::mz(x)),
    ylim = range(ProtGenerics::intensity(x)), main = main)
  tmp <- lapply(x,
    function(z) {
      graphics::points(ProtGenerics::mz(z), ProtGenerics::intensity(z),
        type = type, col = col,
        ...)
    })
}

#' @title Select spectrum with maximal intensity from a list of spectra
#'
#' @description
#'
#' `maxTic` can be used with the `combineSpectra` method to select the spectrum
#' with the largest overall signal from a list of spectra.
#'
#' @param z `Spectra` object.
#'
#' @return `Spectrum`
```

```r
#'
#' @author Johannes Rainer
#'
#' @noRd
maxTic <- function(z) {
  z[[BiocGenerics::which.max(lapply(ProtGenerics::intensity(z), sum))]]
}

#' @title Convert CAMERA output to an edge list for GNPS
#'
#' @description
#'
#' `getEdgelist` takes the output from the `getPeaklist` function from `CAMERA`
#' and converts it to a `data.frame` with edge definitions for *GNPS*. Each
#' row in that `data.frame` contains in columns `"ID1"` and `"ID2"` the
#' identifiers (i.e. `rownames` of the input `data.frame`) of the features.
#' Column `"EdgeType"` is always `"MS1 annotation"` and column `"Score"` `NA`
#' (since no score can be exported from `CAMERA`). Columns `"Annotation"`
#' contains the adduct names and their difference in m/z if **both** edges
#' (features) were predicted to be an adduct of the **same** compound. If
#' isotope annotations are available, these are also added to the column.
#' Column `"CorrelationGroup"` provides the information which
#' features were grouped by `CAMERA` into the same group.
#'
#' @param peaklist `data.frame` as returned by the [getPeaklist()] function
#'   from `CAMERA` package or an `xsAnnotate` object.
#'
#' @return `data.frame` with edge definitions (see description for more
#'     details).
#'
#' @author Mar Garcia-Aloy
#'
#' @examples
#'
#' res <- getEdgelist(getPeaklist(xsaFA))
#' @noRd
getEdgelist <- function(peaklist) {
  if (is(peaklist, "xsAnnotate")) {
    peaklist <- CAMERA::getPeaklist(peaklist)
    if (!nrow(peaklist))
      stop("Got an empty peak list.")
  }
  pl <- split(peaklist, factor(peaklist$pcgroup,
      levels = unique(peaklist$pcgroup)))
  res <- do.call(rbind, lapply(pl, .process_pcgroup))
  rownames(res) <- NULL
  res
}

#' @title Extract feature annotations from CAMERA results
#'
#' @description
#'
#' Similar to the `getEdgelist` function, this function extracts information
#' from a `CAMERA` result for use in GNPS.
#'
#' @param x `xsAnnotate` object after calling `findAdducts`.
#'
#' @return
#'
#' `data.frame` with columns:
#' - `"annotation network number"`: ion identity network (IIN) number. All
#'   features predicted by `CAMERA` to be an adduct of a (co-eluting) compound
#'   with the same mass are part of this IIN. If a feature was predicted to be
#'   an adduct of two different compounds (with different masses) the ID of the
#'   larger network is reported. All features for which no adduct annotation is
#'   available will have an `NA` in this column.
#' - `"best ion"`: the adduct definition of the feature.
#' - `"correlation group ID"`: this corresponds to the `"pcgroup"` column in
#'   `getPeaklist(x)`.
#' - `"auto MS2 verify"`: always `NA`.
#' - `"identified by n="`: the size of the IIN.
#' - `"partners"`: all other features (rows in the feature table) of this IIN.
#' - `"neutral M mass"`: the mass of the compound.
#'
#' @author Johannes Rainer
#'
#' @noRd
getFeatureAnnotations <- function(x) {
  if (!length(x@annoID))
    stop("No adduct information present. Please call 'findAdducts' on ",
      "the object.")
```

```
corr_group <- rep(seq_along(x@pspectra), lengths(x@pspectra))
corr_group <- corr_group[order(unlist(x@pspectra, use.names = FALSE))]

## Get the all ids (feature rows) for which an adduct was defined
ids <- unique(x@annoID[, "id"])

## Note: @pspectra contains the "correlation groups", @annoID the
## adduct groups, but it can happen that two ids are in the same adduct
## group without being in the same correlation group!

## loop through the adduct definitions and build the output data.frame
adduct_def <- lapply(ids, function(id) {
  ## IDs of the same correlation group
  ids_pcgroup <- x@pspectra[[corr_group[id]]]
  ## ID of the adduct annotation groups this id is part of
  anno_grp <- x@annoID[x@annoID[, "id"] == id, "grpID"]
  ## Subset the adduct annotation to rows matching the annotation group
  ## of the present ID and to ids present in the same correlation group.
  adduct_ann <- x@annoID[x@annoID[, "id"] %in% ids_pcgroup &
    x@annoID[, "grpID"] %in% anno_grp, , drop = FALSE]
  ## if we have more than one annotation group, select the bigger one
  if (length(anno_grp) > 1) {
    cnts <- BiocGenerics::table(adduct_ann[, "grpID"])
    adduct_ann <- adduct_ann[
      adduct_ann[, "grpID"] ==
        names(cnts)[order(cnts, decreasing = TRUE)][1], ,
      drop = FALSE]
  }
  grp_id <- adduct_ann[1, "grpID"]
  ## different adduct rules can match the same m/z - we're just taking
  ## the first one (with lower ruleID)
  rule_id <- adduct_ann[adduct_ann[, "id"] == id, "ruleID"][1]
  ids_grp <- adduct_ann[, "id"]
  df <- data.frame(
    `row ID` = id,
    `annotation network number` = unname(grp_id),
    `best ion` = as.character(x@ruleset[rule_id, "name"]),
    `identified by n=` = nrow(adduct_ann),
    `partners` = paste0(ids_grp[ids_grp != id], collapse = ";"),
    `neutral M mass` = unname(
      x@annoGrp[x@annoGrp[, "id"] == grp_id, "mass"]),
```

```r
      stringsAsFactors = FALSE, check.names = FALSE)
      })
  adduct_def <- do.call(rbind, adduct_def)
  res <- adduct_def[rep("other", length(corr_group)), ]
  res[, "row ID"] <- seq_along(corr_group)
  rownames(res) <- as.character(res[, "row ID"])
  res[ids, ] <- adduct_def
  res$`correlation group ID` <- corr_group
  res$`auto MS2 verify` <- NA
  res
}

#' Helper function to extract the adduct annotation from a pair of adducts
#' from the same *pcgroup*.
#'
#' @author Mar Garcia-Aloy
#'
#' @noRd
.define_annot <- function(y) {
  if (any(y$adduct == "")) return(NA)
  mass_1 <- .extract_mass_adduct(y$adduct[1])
  mass_2 <- .extract_mass_adduct(y$adduct[2])
  mass <- BiocGenerics::intersect(mass_1, mass_2)
  if (length(mass)) {
    def1 <- unlist(strsplit(y$adduct[1], " "))
    def2 <- unlist(strsplit(y$adduct[2], " "))
    paste0(def1[grep(mass[1], def1) - 1], " ",
      def2[grep(mass[1], def2) - 1], " dm/z=",
      round(abs(y$mz[1] - y$mz[2]), 4))
  } else NA
}

#' Helper function to extract the isotope annotation from a pair of adducts
#' from the same *pcgroup*.
#'
#' @author Mar Garcia-Aloy
#'
#' @noRd
.define_isotop <- function(w) {
  if (any(w$isotopes == "")) return(NA)
  if (unlist(strsplit(w$isotopes[1], '\\]\\[') )[1] ==
    unlist(strsplit(w$isotopes[2], '\\]\\[') )[1]) {
    a = paste0("[", do.call(rbind, strsplit(w$isotopes, "\\]\\["))[, 2],
      collapse = " ")
    b = paste0(unlist(strsplit(w$isotopes[2], '\\]') )[1], "]")
    paste0(b, a, " dm/z=", round(abs(w$mz[1] - w$mz[2]), 4))
  } else NA
}

#' Simple helper to extract the mass(es) from strings such as
#' [M+NH4]+ 70.9681 [M+H]+ 87.9886
#'
#' @author Johannes Rainer
#'
#' @noRd
#'
#' @examples
#'
#' .extract_mass_adduct("[M+NH4]+ 70.9681 [M+H]+ 87.9886")
#' .extract_mass_adduct("some 4")
.extract_mass_adduct <- function(x) {
  if (!length(x) || x == "") return(NA)
  spl <- unlist(strsplit(x, " ", fixed = TRUE))
  spl[seq(2, by = 2, length.out = length(spl)/2)]
}

#' Helper function to process features from the same *pcgroup*
#'
#' @author Mar Garcia-Aloy
#'
#' @noRd
.process_pcgroup <- function(x) {
  if (nrow(x) > 1) {
    res <- combn(seq_len(nrow(x)), 2, FUN = function(z) {
      anno <- .define_annot(x[z, ])
      iso <- .define_isotop(x[z, ])
      if (is.na(anno[1])) anno <- character()
      if (is.na(iso[1])) iso <- character()
      data.frame(ID1 = rownames(x)[z[1]],
        ID2 =  rownames(x)[z[2]],
        EdgeType = if (length(anno) || length(iso)) "MS1 annotation" else "MS1 correlation",
        Score = 0.0,
        Annotation = paste0(anno, iso, collapse = " "),
        CorrelationGroup = x$pcgroup[1],
        stringsAsFactors = FALSE)
      }, simplify = FALSE)
    do.call(rbind, res)
  } else NULL
}
```

59 File: tools-yaml.R

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

60 File: utils.R

```
# ==========================================================================
# 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){
```

```r
    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)
```

```r
      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 <-
```

```r
  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.")
    }
  }
```

```r
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
```

```r
.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)]
           })
  }
```

```r
.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)
}
```

61 File: zzz.R

```
## 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]
  # }
# }
```

```r
# .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") {
  unlockBinding(".font", topenv())
  assign(".font", font, env = topenv())
  options(mcnebulaFont = font)
}
.font <- "Times"
```