sourcecode

stringb

character plot.R

```
#' function for plotting text
#' @export
#' @param x object of class rtext
#' @param y either NULL or a data.frame with columns "start", "end", "line"
#' @param col color for text
#' @param border border color for text
#' @param pattern_col color for text to be marked up via pattern or y option
#' @param pattern regular expression to be searched in text and marked up in plot
#' @param ... further parameters passed through to text_locate
plot.character <-
  function(
    x,
    y           = NULL,
    col    = "grey",
    border = "grey",
    pattern     = NULL,
    pattern_col = "#ED4C4C",
    ...
  ){
    string <- x
    # gen text data
    x <- nchar(x)
    y <- seq_along(x)
    maxy <- max(y)
    y    <- abs(y-maxy)+1
    # do empty plot
    graphics::plot(
      x    = x,
      y    = y,
      type = "n",
      ylab = "line",
      xlab = "char",
      xlim      = c(0, (ceiling(max(x)/10^nchar(max(x))*10))*(10^nchar(max(x))/10) ),
      ylim      = c(0, max(y)+1 ),
      axes=FALSE
    )
    # do text plot
    graphics::axis(1)
    graphics::axis(2,c(max(y),1),c(1,max(y)))
    graphics::box()
    graphics::rect(
      xleft=0,
      xright=x,
      ybottom=y-0.5,
      ytop=y+0.5,
```

```
      col = col,
      border = border,
      lty=0
    )
    # end or markup
    if( !is.null(pattern) ){
        found <- text_locate_all(string, pattern)
        for( i in seq_along(found) ){
          found[[i]]$line = i
        }
        found <- do.call(rbind, found)
        found <- found[!is.na(found$start),c("start", "end", "line")]
      graphics::rect(
        xleft   = found$start - 1,
        xright  = found$end,
        ybottom = length(string)+1 - found$line - 0.5,
        ytop    = length(string)+1 - found$line + 0.5,
        col     = pattern_col,
        border  = pattern_col#,
        #lty     = 0
      )
    }
  }
```

imports.r

```
#' imports
#'
#' @import backports
#' @keywords internal
#'
dummy_func <- function(){

}
```

text c.R

```
#' generic for concatenating strings
#'
#'
#' @param ... one or more texts to be concatonated (see also \link[base]{paste})
#' @param sep separator between concatonated elements (see also \link[base]{paste})
#' @param coll if texts (not only there elements) are to be collapsed as well,
#'        how should the be separated (see also \link[base]{paste})
#' @seealso \link[stringb:grapes-..-grapes]{\%..\%} and \link[stringb:grapes-.-grapes]{\%.\%}
#' @export
text_c <- function(..., sep="", coll=NULL){
  UseMethod("text_c")
}
```

```
#' text_c default
#' @rdname text_c
#' @method text_c default
#' @export
text_c.default <- function(..., sep="", coll=NULL){
  paste(..., sep=sep, collapse=coll)
}

#' concatenating strings operator
#'
#' @param a first text
#' @param b second text
#' @seealso \link{text_c} (and \link[base]{paste})
#' @export
`%.%`   <- function(a,b) text_c(a, b, sep="")


#' concatenating strings
#'
#' @param a first text
#' @param b first text
#' @seealso \link{text_c} (and \link[base]{paste})
#' @export
`%..%`  <- function(a,b) text_c(a, b, sep=" ")
```

text collapse.R

```
#' function for collapsing text vectors
#' @param x object to be collapsed
#' @param coll separator between collapsed text parts
#' @param ... additional parameter passed through to methods
#' @export
text_collapse <- function (x, coll="") {
  UseMethod("text_collapse")
}

#' default method for text_collapse()
#' @rdname text_collapse
#' @method text_collapse default
#' @export
text_collapse.default <- function(x, coll=""){
    paste0(unlist(x), collapse = coll)
}

#' text_collapse() mehtod for list
#' @rdname text_collapse
#' @method text_collapse list
#' @export
text_collapse.list <- function(x, coll=""){
  text_collapse(
    unlist(lapply(x, text_collapse, coll=coll)),
```

```
    coll)
}



#' text_collapse() method for data.frames
#' @export
#' @rdname text_collapse
#' @method text_collapse data.frame
text_collapse.data.frame <- function(x, coll=""){
    x <- apply(x, 1, text_collapse, coll=coll[1])
    x <- unlist(x, recursive = FALSE)
  if(length(coll)>1){
    coll <- coll[2]
  }
  text_collapse(x, coll=coll)
}


#' text_collapse() method for matrix
#' @export
#' @rdname text_collapse
#' @method text_collapse matrix
text_collapse.matrix <- function(x, coll=""){
  text_collapse(as.data.frame(x), coll)
}
```

text count.R

```
#' generic for counting pattern occurences
#' @param string text to search through
#' @param pattern regex to search for
#' @param vectorize should function be used in vectorized mode, i.e. should a
#'    pattern with length larger than 1 be allowed and if so, should it be
#'    matched to lines (with recycling if needed) instead of using on element on
#'    all lines
#' @param sum if true all element-wise counts will be summed up
#' @param ... further arguments passed through to \link[base]{gregexpr}
#' @export
text_count <- function(string, pattern, sum=FALSE, vectorize=FALSE, ...){
  UseMethod("text_count")
}

#' text_count defaul method
#' @rdname text_count
#' @method text_count default
#' @export
text_count.default <- function(string, pattern, sum=FALSE, vectorize=FALSE, ...){
  if(is.list(string)){
    tmp  <- lapply(string, text_count, pattern=pattern, sum=sum, vectorize=vectorize, ...)
    return(tmp)
  }
```

```
  if(vectorize){
    tmp <- mapply(gregexpr, pattern=pattern, text=string)
    names(tmp) <- NULL
    tmp <- vapply(tmp, function(tmp){sum(!is.na(tmp) & tmp!=-1)}, integer(1))
    tomp <-
      as.data.frame(
        do.call(
          rbind,
          mapply(c,i=seq_along(string), p=seq_along(pattern), SIMPLIFY = FALSE)
        )
      )
    tmp <- cbind(n=tmp, tomp)
    if(sum){
      return(sum(tmp$n))
    }else{
      return(tmp)
    }
  }else{
    tmp <- gregexpr(pattern, string, ...)
    tmp <- vapply(tmp, function(tmp){sum(!is.na(tmp) & tmp!=-1)}, integer(1))
    if(sum){
      return(sum(tmp))
    }else{
      return(tmp)
    }
  }
}
```

text detect.R

```
#' generic function to test if a regex can be found within a string
#' @param string text to be searched through
#' @param pattern regex to look for
#' @param ... further arguments passed through to \link[base]{grepl}
#' @export
text_detect <- function(string, pattern, ...){
  UseMethod("text_detect")
}


#' text_detect default method
#' @rdname text_detect
#' @method text_detect default
#' @export
text_detect.default <- function(string, pattern, ...){
  grepl(pattern=pattern, x=string, ...)
}



#' generic function to test if a regex can be found within a string
```

```
#' @rdname text_detect
#' @export
text_grepl <- function(string, pattern, ...){
  UseMethod("text_detect")
}
```

text dup.R

```
#' generic repeating text
#' @param string text to be repeated
#' @param times how many times shal string be repeated
#' @param vectorize should function be used in vectorized mode, i.e. should a
#'    pattern with length larger than 1 be allowed and if so, should it be
#'    matched to lines (with recycling if needed) instead of using on element on
#'    all lines
#' @param ... further arguments passed through
#' @export
text_rep <- function(string, times, vectorize=FALSE, ...){
  UseMethod("text_rep")
}

#' @export
#' @rdname text_rep
text_dup <- function(string, times, vectorize=FALSE, ...){
  UseMethod("text_rep")
}



#' text_rep defaul method
#' @rdname text_rep
#' @method text_rep default
#' @export
text_rep.default <- function(string, times, vectorize=FALSE, ...){
  # list handling
  if(is.list(string)){
    tmp  <- lapply(string, text_rep, times=times, vectorize=vectorize, ...)
    return(tmp)
  }
  # sanatize input
  times[times<0] <- 0
  # doing duty-to-do
  if(vectorize){
    # text
    tmp           <- mapply(text_rep, string, times)
    Encoding(tmp) <- "UTF-8"
    names(tmp)    <- NULL
```

```
    # data
    tomp <-
      as.data.frame(
        do.call(
          rbind,
          mapply(c,i=seq_along(string), p=seq_along(times), SIMPLIFY = FALSE)
        )
      )
    # return
    tmp <- data.frame(t=tmp, i=tomp$i, p=tomp$p)
    rownames(tmp) <- NULL
    return(tmp)
  }else{
    tmp <-
      vapply(
        X   = string,
        FUN = strrep,
        FUN.VALUE = "",
        times = times
      )
    Encoding(tmp) <- "UTF-8"
    return(tmp)
  }
}
```

text eval.R

```
```
Warning in readLines(stringb[i]): incomplete final line found on '../
stringb/R/text_eval.R'
```

```
#' wrapper function of eval() and parse() to evaluate character vector
#' @param x character vector to be parsed and evaluated
#' @param envir where to evaluate character vector
#' @param ... arguments passed through to eval()
#' @export
text_eval <- function(x, envir=parent.frame(), ...){
  eval(
    parse(text = x),
    envir = envir,
    ...
```
```

```
  )
}
```

text extract.R

```
#' extract regex matches
#'
#' wrapper function around regexec and regmatches
#'
#' @param x text from which to extract
#' @param pattern see \link{grep}
#' @param ignore.case see \link{grep}
#' @param perl see \link{grep}
#' @param fixed see \link{grep}
#' @param useBytes see \link{grep}
#' @param invert if TRUE non-regex-matches are extracted instead
#' @export
text_extract <-
  function(
    x,
    pattern,
    ignore.case = FALSE,
    perl        = FALSE,
    fixed       = FALSE,
    useBytes    = FALSE,
    invert      = FALSE
  ){
    regmatches(
      x,
       regexpr(
         pattern     = pattern,
         text        = x,
         ignore.case = ignore.case,
         perl        = perl,
         fixed       = fixed,
         useBytes    = useBytes
       ),
      invert = invert
    )
  }
```

text extract all.R

```
#' extract regex matches
#'
#' wrapper function around gregexec and regmatches
#'
```

```
#' @param x text from which to extract
#' @param pattern see \link{grep}
#' @param ignore.case see \link{grep}
#' @param perl see \link{grep}
#' @param fixed see \link{grep}
#' @param useBytes see \link{grep}
#' @param invert if TRUE non-regex-matches are extracted instead
#' @export
text_extract_all <-
  function(
    x,
    pattern,
    ignore.case = FALSE,
    perl        = FALSE,
    fixed       = FALSE,
    useBytes    = FALSE,
    invert      = FALSE
  ){
    regmatches(
      x,
      gregexpr(
        pattern     = pattern,
        text        = x,
        ignore.case = ignore.case,
        perl        = perl,
        fixed       = fixed,
        useBytes    = useBytes
      ),
      invert = invert
    )
  }
```

text extract group.R

```
#' generic for getting regex group matches
#'
#' @param string text from which to extract character sequence
#' @param pattern regex to be searched for
#' @param group integer vector to indicate those regex group matches to extract
#' @param invert whether or no matches or non-matches should be extracted
#' @param ... further parameter passed through to \link[base]{regexec}
#' @export
text_extract_group <- function(string, pattern, group, invert=FALSE, ...){
  UseMethod("text_extract_group")
}

#' text default
#' @rdname text_extract_group
#' @method text_extract_group default
#' @export
text_extract_group.default <- function(string, pattern, group=NULL, invert=FALSE, ...){
  tmp   <- regexec(pattern = pattern, text=string)
  found <- vapply(tmp, `[`, 1, 1)!=-1
```

```
  if(invert){
    for(i in seq_along(tmp) ){
      match_length <- attr(tmp[[i]], "match.length")
      use_bytes    <- attr(tmp[[i]], "useBytes")
      tmp[[i]]     <- tmp[[i]][-1]
      attr(tmp[[i]], "match.length") <- match_length[-1]
      attr(tmp[[i]], "useBytes")     <- use_bytes
    }
    res <- regmatches(string, tmp, invert = invert)
    res[!found] <- NA
    res <- as.data.frame( do.call(rbind, res) )
  }else{
    res <- regmatches(string, tmp, invert = invert)
    res[!found] <- NA
    res <- as.data.frame( do.call(rbind, res) )
    res[,1] <- NULL
  }
  if( dim(res)[2]>0 ){
    names(res) <- text_c("group", seq_len(dim(res)[2]))
  }
  if( !is.null(group) ){
    return( get_groups(res, group) )
  }else{
    return(res)
  }
}

#' helper function for text_extract_group
#' @param x text_extract_group result
#' @param groups groups to extract
#' @keywords internal
get_groups <- function(x, group){
  groups <- text_c("group", group)
  tmp <- list()
  for(i in seq_along(groups) ){
    if( is.null(x[[groups[i]]]) ){
      tmp[[groups[i]]] <- rep(NA, dim(x)[1])
    }else{
      tmp[[groups[i]]] <- x[[groups[i]]]
    }
  }
  tmp <- as.data.frame(tmp)
  return(tmp)
}



#' generic for getting all regex group matches
#'
#' @param string text from which to extract character sequence
#' @param pattern regex to be searched for
#' @param invert whether or no matches or non-matches should be extracted
#' @param ... further parameter passed through to \link[base]{gregexpr}
#' @param group integer vector to indicate those regex group matches to extract
```

```
#' @export
text_extract_group_all <- function(string, pattern, group=NULL, invert=FALSE, ...){
  UseMethod("text_extract_group_all")
}

#' text default
#' @rdname text_extract_group_all
#' @method text_extract_group_all default
#' @export
text_extract_group_all.default <-
  function(string, pattern, group=NULL, invert=FALSE, ...){
  snippets <- text_extract_all(string, pattern)
  groups   <- lapply(snippets, regexec, pattern=pattern)
  res      <- mapply(regmatches, m=groups, x=snippets)
  worker   <- function(x){
    tmp <-
      as.data.frame(
        do.call(rbind, x)
      )[, -1]
    names(tmp) <- text_c("group", seq_len(dim(tmp)[2]))
    tmp
  }
  res <- lapply(res, worker)
  # group option
  if(!is.null(group)){
    res <- lapply(res, get_groups, group=group)
  }
  # match option
  if(!is.null(group)){
    res <- lapply(res, get_groups, group=group)
  }
  return(res)
}
```

text length.R

```
#' wrapper around nchar to return text length
#' @param x see \link{nchar}
```

```
#' @param type see \link{nchar}
#' @param allowNA see \link{nchar}
#' @param keepNA see \link{nchar}
#' @export
text_nchar <- function(x, type = "chars", allowNA = FALSE, keepNA = TRUE){
  nchar(x, type, allowNA, keepNA)
}

#' wrapper around nchar to return text length
#' @param x see \link{nchar}
#' @param type see \link{nchar}
#' @param allowNA see \link{nchar}
#' @param keepNA see \link{nchar}
#' @param na.rm see \link{nchar}
#' @export
text_length <- function(x, type = "chars", allowNA = FALSE, keepNA = TRUE, na.rm=FALSE){
  sum(text_nchar(x, type, allowNA, keepNA), na.rm=na.rm)
}
```

text locate.R

```
#' helper function to get start, end, length form pattern match
#' @param string text to be searched through
#' @param pattern regex to look for
#' @param ... further options passed through to \link[base]{regexpr}
text_locate_worker <- function(string, pattern, ...){
  tmp <- regexpr(pattern, string, ...)
  regmatches2(tmp)
}

#' function to get start, end, length form pattern match
#' @param string text to be searched through
#' @param pattern regex to look for
#' @param vectorize should function be used in vectorized mode, i.e. should a
#'    pattern with length larger than 1 be allowed and if so, should it be
#'    matched to lines (with recycling if needed) instead of using on element on
#'    all lines
#' @param ... further options passed through to \link[base]{regexpr}
#' @export
text_locate <- function(string, pattern, vectorize=FALSE, ...){
  UseMethod("text_locate")
}

#' text_locate default
#' @rdname text_locate
#' @method text_locate default
#' @export
text_locate.default <- function(string, pattern, vectorize=FALSE, ...){
  if(is.list(string)){
    res <-
      lapply(
        X         = string,
        FUN       = text_locate,
```

```
        pattern   = pattern,
        vectorize = vectorize,
        ...
      )
    return(res)
  }
  if( length(pattern>1) & vectorize ){
    res <-
      mapply(
      text_locate_worker,
      string   = string,
      pattern  = pattern,
      MoreArgs = ...,
      SIMPLIFY = FALSE
    )
  }else{
    res <- text_locate_worker(string, pattern, ...)
  }
  if(vectorize){
    for(i in seq_along(res)){
      res[[i]]$i <- i
      p <- i %% length(pattern)
      p[p==0] <- length(pattern)
      res[[i]]$p <-p
    }
    res <- do.call(rbind, res)
    rownames(res) <- NULL
  }
  return(res)
}


#' helper function to get start, end, length form pattern match
#' @param string text to be searched through
#' @param pattern regex to look for
#' @param ... further options passed through to \link[base]{regexpr}
text_locate_all_worker <- function(string, pattern, ...){
  tmp <- gregexpr(pattern, string, ...)
  lapply(tmp, regmatches2)
}

#' function to get start, end, length form pattern match for all matches
#' @param string text to search through
#' @param pattern regex to search for
#' @param vectorize should function be used in vectorized mode, i.e. should a
#'    pattern with length larger than 1 be allowed and if so, should it be
#'    matched to lines (with recycling if needed) instead of using on element on
#'    all lines
#' @param simplify either getting back a list of results or all list elements
#'    merged into a data.frame with columns identifying original line (i) and
#'    pattern (p) number
#' @param ... further arguments passed through to \link[base]{gregexpr}
#' @export
text_locate_all <- function(string, pattern, vectorize=FALSE, simplify=FALSE, ...){
```

```
  UseMethod("text_locate_all")
}

#' text_locate_all default
#' @rdname text_locate_all
#' @method text_locate_all default
#' @export
text_locate_all.default <- function(string, pattern, vectorize=FALSE, simplify=FALSE, ...){
  if(is.list(string)){
    return(lapply(string, text_locate_all, pattern, ..., vectorize=vectorize))
  }
  if( length(pattern>1) & vectorize==TRUE ){
    res <-
        mapply(
          text_locate_all_worker,
          string   = string,
          pattern  = pattern,
          MoreArgs = ...,
          SIMPLIFY = FALSE
        )
    res <- unlist(res, recursive = FALSE)
  }else{
    res <- text_locate_all_worker(string, pattern, ...)
  }
  if(simplify){
    for(i in seq_along(res)){
      res[[i]]$i <- i
      p <- i %% length(pattern)
      p[p==0] <- length(pattern)
      res[[i]]$p <-p
    }
    res <- do.call(rbind, res)
  }
  rownames(res) <- NULL
  return(res)
}
```

text locate group.R

```
#' generic for getting positions regex groups
#' @inheritParams text_locate
#' @param group integer vector specifying groups to return
#' @export
text_locate_group <- function(string, pattern, group, ... ){
  UseMethod("text_locate_group")
}

#' text_locate_group default
#' @rdname text_locate_group
#' @method text_locate_group default
#' @export
```

```
text_locate_group.default <- function(string, pattern, group, ... ){
  positions    <- regexec(pattern = pattern, text=string, ...)
  positions    <- drop_non_group_matches(positions, group)
  regmatches2(positions)
}
```

text pad.R

```
#' padding text to specified width
#'
#' @param string text to be wrapped
#' @param width width text should have after padding; defaults to: max(nchar(string))
#' @param pad the character or character sequence to use for padding
#' @param side one of: c("left", "right", "both", "l", "r", "b", 1, 2, 3)
#' @export
text_pad <-
  function(
    string,
    width = max(nchar(string)),
    pad   = " ",
    side  = c("left", "right", "both", "l", "r", "b", 1, 2, 3)
  )
{
  UseMethod("text_pad")
}

#' text_wrap default
#' @rdname text_pad
#' @method text_pad default
#' @export
text_pad.default <-
  function(
    string,
    width = max(nchar(string)),
    pad   = " ",
    side  = c("left", "right", "both", "l", "r", "b", 1, 2, 3)
  )
{
  # input checks
  side <- side[1]
  stopifnot(side %in% c("left", "right", "both", "l", "r", "b", 1, 2, 3))
  if(side %in% c("left",  "l")){
    side <- 1
  }else if(side %in% c("right", "r")){
    side <- 2
  }else if(side %in% c("both",  "b")){
    side <- 3
  }
  # doing-duty-to-do
  if(side < 3 ){
    if( nchar(pad)==1 ){
      tmp <- text_dup(pad, width-nchar(string), vectorize = TRUE)$t
    }else{
```

```
      tmp <- text_snippet(text_dup(pad, width), length = width - nchar(string) )
    }
  }else{
    if( nchar(pad) == 1){
      tmpl <- text_dup(pad, floor(width-nchar(string)), vectorize = TRUE)$t
      tmpr <- text_dup(pad, ceiling(width-nchar(string)), vectorize = TRUE)$t
    }else{
      tmpl <- text_snippet(text_dup(pad, width), length = floor((width - nchar(string))/2) )
      tmpr <- text_snippet(text_dup(pad, width), length = ceiling((width - nchar(string))/2) )
    }
  }
  # return
  if(side == 1 ){
    return( text_c(tmp, string) )
  }
  if(side == 2 ){
    return( text_c(string, tmp) )
  }
  if(side == 3){
    return( text_c(tmpl, string, tmpr) )
  }
}
```

text read.R

```
#' read in text
#'
#' A wrapper to readLines() to make things more ordered and convenient. In
#' comparison to the wrapped up readLines() function text_read() does some
#' things differently: (1) If no encoding is given, it will always assume files
#' are stored in UTF-8 instead of the system locale. (2) it will always converts
#' text to UTF-8 instead of transforming it to the system locale. (3) in
#' addition to loading, it offers to tokenize the text using a regular expression
#' or NULL for no tokenization at all.
#'
#' @param file name or path to the file to be read in or a \link[base]{connection} object (see \link[bas
#' @param tokenize either
#'    NULL so that no splitting is done;
#'    a regular expression to use to split text into parts;
#'    or a function that does the splitting (or whatever other transformation)
#' @param encoding character encoding of file passed throught to \link[base]{readLines}
#' @param ... further arguments passed through to \link[base]{readLines} like:
#'   n, ok, warn, skipNul
#' @export

text_read <- function(file, tokenize="\n", encoding="UTF-8", ...)
{
  tmp <- readLines(file, ...)
  # transform to UTF-8 encoding
  tmp <- iconv(tmp, encoding, "UTF-8")
  # all within one vector element
  if( is.null(tokenize) ){
    return(paste0(tmp, collapse = "\n"))
```

```
  }
  # tokenized by function
  if(is.function(tokenize)){
    return( unlist(tokenize(paste0(tmp, collapse = "\n"))) )
  }
  # vector elements should correspond to lines
  if(tokenize == "\n"){
    return(tmp)
  }
  # tokenized by other pattern
  if(is.character(tokenize)){
    return(unlist(strsplit(paste0(tmp, collapse = "\n"), tokenize)))
  }
}
```

text replace.R

```
#' replacing patterns in string
#' @param string text to be replaced
#' @param pattern regex to look for
#' @param replacement replacement for pattern found
#' @param recycle should arguments be recycled if lengths do not match?
#' @param ... further parameter passed through to sub
#' @export
text_replace <- function(string, pattern=NULL, replacement=NULL, ...){
  UseMethod("text_replace")
}

#' replacing patterns default
#' @rdname text_replace
#' @method text_replace default
#' @export
text_replace.default <-
  function(string, pattern=NULL, replacement=NULL, recycle=FALSE, ...){
    if( (length(pattern) > 1 | length(replacement) > 1) & recycle ){
      mapply(sub, x=string, pattern=pattern, replacement=replacement, ..., USE.NAMES = FALSE)
    }else{
      sub(pattern=pattern, replacement=replacement, x=string, ...)
    }
  }


#' replacing patterns in string
#' @param string text to be replaced
#' @param pattern regex to look for
#' @param replacement replacement for pattern found
#' @param recycle should arguments be recycled if lengths do not match?
#' @param ... further parameter passed through to gsub
#' @export
text_replace_all <- function(string, pattern=NULL, replacement=NULL, ...){
  UseMethod("text_replace_all")
}
```

```
#' replacing patterns default
#' @rdname text_replace_all
#' @method text_replace_all default
#' @export
text_replace_all.default <-
  function(string, pattern=NULL, replacement=NULL, recycle=FALSE, ...){
    if( (length(pattern) > 1 | length(replacement) > 1) & recycle ){
      mapply(gsub, x=string, pattern=pattern, replacement=replacement, ..., USE.NAMES = FALSE)
    }else{
      gsub(pattern=pattern, replacement=replacement, x=string, ...)
    }
}



#' deleting patterns in string
#' @param string text to be replaced
#' @param pattern regex to look for and delete
#' @param ... further parameter passed through to sub
#' @export
text_delete <- function(string, pattern=NULL, ...){
  UseMethod("text_delete")
}

#' deleting patterns in string
#' @rdname text_delete
#' @method text_delete default
#' @export
text_delete.default <- function(string, pattern=NULL, ...){
  text_replace(string = string, pattern = pattern, replacement="")
}
```

text replace group.R

```
#' text_replace_locates default
#' @param string text for which to replace parts
#' @param found result of an call to text_locate_group or text_locate
#'        - i.e. a list of data.frames
#'        with two columns named 'start' and 'end' that mark character spans
#'        to be replaced within the text elements
#' @param group vector of integers identifying thos regex groups to be replaced
#' @param replacement character vector of replacements of length 1 or
#'        length(group) to replace regex group matches (marked character spans
#'        provided by the found parameter)
#' @param invert should character spans provided by found or their counterparts
#'        be replaced
#' @export
text_replace_locates <- function(string, found, replacement, group, invert){
  UseMethod("text_replace_locates")
}
```

```
#' text_replace_locates default
#' @method text_replace_locates default
#' @rdname text_replace_locates
#' @export
text_replace_locates.default <- function(string, found, replacement, group, invert){
  start  <- found$start
  end    <- found$end
  if( any(is.na(start)) ){
    tmp <- string
  }else{
    end2   <- c(start-1, nchar(string[1]))
    start2 <- c(0,end+1)
    df     <- data.frame(start=c(start,start2), end=c(end, end2))
    df     <- df[order(df$start, df$end),]
    tmp    <- substring(string,df$start,df$end)
    if(invert){
      tmp[ seq_along(tmp) %% 2 == 1 ][group] <- replacement
    }else{
      tmp[ seq_along(tmp) %% 2 == 0 ][group] <- replacement
    }
    tmp <- text_collapse(tmp)
  }
  return(tmp)
}




#' function for replacing regex group matches
#' generic for getting regex group matches
#'
#' @param string text from which to extract character sequence
#' @param pattern regex to be searched for
#' @param ... further parameter passed through to \link[base]{regexec}
#' @param group vector of integers identifying thos regex groups to be replaced
#' @param replacement character vector of replacements of length 1 or
#'        length(group) to replace regex group matches (marked character spans
#'        provided by the found parameter)
#' @param invert should character spans provided by found or their counterparts
#'        be replaced
#' @export
text_replace_group <-
  function(
    string,
    pattern,
    replacement,
    group=seq_along(replacement),
    invert=FALSE,
    ...
){
  UseMethod("text_replace_group")
}

#' text_replace_group default
```

```
#' @rdname text_replace_group
#' @method text_replace_group default
#' @export
text_replace_group.default <-
  function(
    string,
    pattern,
    replacement,
    group=TRUE,
    invert=FALSE,
    ...
  ){
    found <- text_locate_group(string, pattern, ...)
    mapply(
      text_replace_locates,
      string,
      found,
      MoreArgs =
        list(
          replacement = replacement,
          group       = group,
          invert      = invert
        ),
      USE.NAMES = FALSE
    )
  }
```

text show.R

```
#' showing text
#'
#' shows text or portions of the text via cat and the usage of text_snippet()
#' @param x text to be shown
#' @param length number of characters to be shown
#' @param from show from ith character
#' @param to show up to ith character
#' @param coll should x be collapsed using newline character as binding?
#' @param wrap should text be wrapped, or wrapped to certain width, or wrapped
#'    by certain function
#' @param ... further arguments passed through to \link[base]{cat}
#' @export
text_show = function(x, length=500, from=NULL, to=NULL, coll=FALSE, wrap=FALSE, ...){
  UseMethod("text_show")
}

#' text_show default
#' @rdname text_show
#' @method text_show default
#' @export
text_show.default = function(x, length=500, from=NULL, to=NULL, coll=FALSE, wrap=FALSE, ...){
  tmp       <- text_snippet(x, length, from, to, coll)
  diff_char <- sum(nchar(x)) - sum(nchar(tmp)) > 0
```

```
  diff_sum  <- sum(nchar(x)) - sum(nchar(tmp))
  diff_note <- ifelse(diff_char, paste0("\n[... ", format(diff_sum, big.mark = " "), " characters not sh
  if(wrap==FALSE){
    cat( tmp, diff_note)
  }else if(is.function(wrap)){
    cat(wrap(tmp), diff_note)
  }else{
    cat( unlist(strsplit(tmp, " ")), diff_note, fill=wrap, ...)
  }
}
```

text snippet.R

```
#' retrieving text snippet
#'
#' function will give back snippets of text via using length,
#' length and from, length and to, or from and to to specify the snippet
#' @param x character vector to be snipped
#' @param length length of snippet
#' @param from starting character
#' @param to last character
#' @param coll should a possible vector x with length > 1 collapsed with newline
#'    character as separator?
#' @describeIn text_snippet retrieving text snippet
#' @export
text_snippet <-function(x, length=max(nchar(x)), from=NULL, to=NULL, coll=FALSE){
  # input check
  stopifnot( length(length)!=0 | (length(from)!=0 & length(to)!=0) ) # any input
  # collapse before snipping?
  if(coll!=FALSE){
    if( identical(coll, TRUE) ){
      x <- paste0(x, collapse = "\n")
    }else{
      x <- paste0(x, collapse = coll)
    }
  }
  # snipping cases
  if( !is.null(from) & !is.null(to) ){                          # from + to
    return(substring(x, from, to))
  }else if( !is.null(from) & is.null(to) ){                     # from + length
    return(substring(x, from, from+length-1))
  }else if( is.null(from) & !is.null(to) ){                     # to + length
    return(substring(x, to-length, to))
  }else if( length(length)!=0 & is.null(from) & is.null(to) ){  # length
    return(substring(x, 0, length))
  }
}
```

text split.R

```
#' generic splitting strings
#' @param string text to search through
```

```
#' @param pattern regex to search for
#' @param vectorize should function be used in vectorized mode, i.e. should a
#'    pattern with length larger than 1 be allowed and if so, should it be
#'    matched to lines (with recycling if needed) instead of using on element on
#'    all lines
#' @param ... further arguments passed through to \link[base]{gregexpr}
#' @export
text_split <- function(string, pattern, vectorize=FALSE, ...){
  UseMethod("text_split")
}

#' text_split defaul method
#' @rdname text_split
#' @method text_split default
#' @export
text_split.default <- function(string, pattern, vectorize=FALSE, ...){
  if(is.list(string)){
    splits  <- lapply(string, text_split, pattern=pattern, vectorize=vectorize, ...)
    return(splits)
  }
  if(!vectorize & length(pattern)>1){
    warning("text_split : length of pattern > 1, only first element will be used")
    pattern <- pattern[1]
  }
  if(vectorize){
    splits <-  mapply(strsplit, split=pattern, x=string)
    info   <- mapply(c, i=seq_along(string), p=seq_along(pattern), SIMPLIFY = FALSE)
    for(i in seq_along(splits)){
      splits[[i]] <-
        data.frame(
          splits[[i]],
          info[[i]][[1]],
          info[[i]][[2]]
        )
    }
    splits <- do.call(rbind, splits)
    rownames(splits) <- NULL
    names(splits)    <- c("t", "i", "p")
    return(splits)
  }else{
    splits <- strsplit(x=string, split=pattern, ...)
    return(splits)
  }
}


#' generic splitting strings into pieces of length n
#' @param string text to search through
#' @param n length of pieces
#' @param vectorize should function be used in vectorized mode, i.e. should a
#'    pattern with length larger than 1 be allowed and if so, should it be
#'    matched to lines (with recycling if needed) instead of using on element on
#'    all lines
#' @export
```

```
text_split_n <- function(string, n, vectorize=FALSE){
  UseMethod("text_split_n")
}

#' text_split_n defaul method
#' @rdname text_split_n
#' @method text_split_n default
#' @export
text_split_n.default <- function(string, n, vectorize=FALSE){
  if(!vectorize & length(n)>1){
    warning("text_split : length of pattern > 1, only first element will be used")
    n <- n[1]
  }
  if(vectorize){
    splits <- mapply(text_split_n, n=n, string=string)
    return(splits)
  }else{
    splits <- gregexpr(text_c(".{0,",n,"}"), string)
    splits <- regmatches(string, splits)
    return(splits)
  }
}
```

text sub.R

```
#' generic for extracting characters sequences by position
#'
#' @param string text from which to extract character sequence
#' @param start first character position
#' @param end  last character position
#' @seealso \link{text_snippet}
#' @export
text_sub <- function(string, start=NULL, end = NULL){
  UseMethod("text_sub")
}


#' text_sub default
#' @rdname text_sub
#' @method text_sub default
#' @export
text_sub.default <- function(string, start=NULL, end = NULL){
  text_snippet(string, from=start, to=end)
}
```

text to lower.R

```
#' function for make text lower case
#' @param x text to be processed
#' @export
```

```
text_to_lower <- function (x) {
  UseMethod("text_to_lower")
}

#' default method for text_tolower()
#' @rdname text_to_lower
#' @method text_to_lower default
#' @export
text_to_lower.default <- function(x){
  if(is.list(x)){
    return(
      lapply(x, tolower)
    )
  }else{
    return(tolower(x))
  }
}


#' function for make text lower case
#' @param x text to be processed
#' @export
text_to_upper <- function (x) {
  UseMethod("text_to_upper")
}

#' default method for text_to_upper()
#' @rdname text_to_upper
#' @method text_to_upper default
#' @export
text_to_upper.default <- function(x){
  if(is.list(x)){
    return(
      lapply(x, toupper)
    )
  }else{
    return(toupper(x))
  }
}


#' function for make text lower case
#' @param x text to be processed
#' @export
text_to_title_case <- function (x) {
  UseMethod("text_to_title_case")
}

#' default method for text_to_title_case.()
#' @rdname text_to_title_case
#' @method text_to_title_case default
#' @export
text_to_title_case.default <- function(x){
  if(is.list(x)){
```

```
    return(
      lapply(x, tools::toTitleCase)
    )
  }else{
    return(tools::toTitleCase(x))
  }
}
```

text tokenize.R

```
#' generic for gregexpr wrappers to tokenize text
#' @param string text to be tokenized
#' @param regex regex expressing where to cut see (see \link[base]{gregexpr})
#' @param ignore.case whether or not reges should be case sensitive
#'    (see \link[base]{gregexpr})
#' @param fixed whether or not regex should be interpreted as is or as regular
#'    expression (see \link[base]{gregexpr})
#' @param perl whether or not Perl compatible regex should be used
#'    (see \link[base]{gregexpr})
#' @param useBytes byte-by-byte matching of regex or character-by-character
#'    (see \link[base]{gregexpr})
#' @param non_token should information for non-token, i.e. those patterns by
#'    which the text was splitted, be returned as well
#' @export
text_tokenize <- function (
   string,
   regex       = NULL,
   ignore.case = FALSE,
   fixed       = FALSE,
   perl        = FALSE,
   useBytes    = FALSE,
   non_token   = FALSE
 ){
```

```
  UseMethod("text_tokenize")
}

#' default method for text_tokenize generic
#' @rdname text_tokenize
#' @method text_tokenize default
#' @export
text_tokenize.default <-
  function(
    string,
    regex       = NULL,
    ignore.case = FALSE,
    fixed       = FALSE,
    perl        = FALSE,
    useBytes    = FALSE,
    non_token   = FALSE
  ){
    # recursion
    if(length(string)>1){
      lapply(
        string,
        text_tokenize,
        regex       = regex,
        ignore.case = ignore.case,
        fixed       = fixed,
        perl        = perl,
        useBytes    = useBytes,
        non_token   = non_token
      )
    }else{
      # special cases
      if( any(grepl(regex, "")==TRUE) ){
        tmp <- strsplit(string, regex)[[1]]
        token <- data.frame(
          from     = seq_along(tmp),
          to       = seq_along(tmp),
          token    = tmp,
          is_token = rep(TRUE, length(tmp))
        )
        return(stringb_arrange(token, "from", "to"))
      }
      if( is.null(regex) ){
        regex <- ".*"
      }
      # finding characters spans where to split
      tlength <- text_length(string)
      found_splitter <-
        gregexpr(
          pattern     = regex,
          text        = string,
          ignore.case = ignore.case,
          fixed       = fixed,
          useBytes    = useBytes
        )
```

```
found_splitter_from   <- found_splitter[[1]]
found_splitter_length <- attributes(found_splitter[[1]])$match.length
found_splitter_to     <- found_splitter_length+found_splitter_from-1

# infering tokens
char_splitter <-
  unique(
    unlist(
      mapply(seq, found_splitter_from, found_splitter_to, SIMPLIFY = FALSE)
    )
  )

# dev : not used anymore? ...
# char_token <-
#  sort(unique(seq_len(tlength)[!(seq_len(tlength) %in% char_splitter)]))

char_token_from     <- c(1,found_splitter_to+1)
char_token_to       <- c(ifelse(found_splitter[[1]]==1, 1, found_splitter[[1]]-1),tlength)

token <-
  data.frame(
    from  = char_token_from,
    to    = char_token_to
  )

token_false_positive_iffer <-
  !(token$from %in% char_splitter | token$to %in% char_splitter)

token <- subset(token, token_false_positive_iffer)

# handling special cases
if( tlength > 0  &  dim(token)[1] == 0  &  !all( found_splitter[[1]] > 0 ) ){
  token <- rbind(token, c(1, tlength))
  names(token) <- c("from", "to")
}

# filling with tokens
if( ignore.case ){
  tmp <- regmatches(string, found_splitter, invert = TRUE)[[1]]
}else{
  tmp <- unlist(strsplit(string, regex, fixed = fixed, perl = perl))
}
tmp <- subset(tmp, token_false_positive_iffer)

token$token    <- tmp[seq_along(token$from)]
token$is_token <- rep(TRUE, dim(token)[1])

# adding non-tokens
if( non_token == TRUE ){
  # handling special cases
  if( any(found_splitter_to<0) | any(found_splitter_from<0) ){
    found_splitter_to   <- integer(0)
    found_splitter_from <- integer(0)
  }
```

```
        # adding to token
        non_token <-
          data.frame(
            from     = found_splitter_from,
            to       = found_splitter_to,
            token    = regmatches(string, found_splitter)[[1]],
            is_token = rep(FALSE, length(found_splitter_to))
          )
        token <-
          rbind(token, non_token )
      }

      # return
      iffer <- is.na(token$token)
      if( sum(iffer) > 0 ){
        token[iffer, "token"] <- text_sub(string, token[iffer, "from"], token[iffer, "to"])
      }
      return(stringb_arrange(token, "from", "to"))
    }
  }

#' generic to tokenize text into words
#'
#' A wrapper to text_tokenize that tokenizes text into words.
#' Since using text_tokenize()'s option non_token might slow things
#' down considerably this one purpose wrapper is a little more clever
#' than the general implementation and hence much faster.
#'
#' @param string the text to be tokenized
#' @param non_token whether or not token as well as non tokens shall be returned.
#' @export
text_tokenize_words <- function(string, non_token = FALSE){
  UseMethod("text_tokenize_words")
}

#' text_tokenize default
#' @rdname text_tokenize_words
#' @method text_tokenize_words default
#' @export
text_tokenize_words.default <-
  function(
    string,
    non_token = FALSE
  ){
    res <- text_tokenize(string, "\\W+")
    if(non_token){
      tmp <- text_tokenize(string, "\\w+")
      tmp$is_token <- rep(FALSE, dim(tmp)[1])
      res <- rbind(res, tmp)
    }
    return(stringb_arrange(res, "from", "to"))
  }

#' generic to tokenize text into lines
```

```
#'
#'
#' @param string the text to be tokenized
#' @param non_token whether or not token as well as non tokens shall be returned.
#' @export
text_tokenize_lines <- function(string, non_token = FALSE){
  UseMethod("text_tokenize_lines")
}

#' text_tokenize default
#' @rdname text_tokenize_lines
#' @method text_tokenize_lines default
#' @export
text_tokenize_lines.default <-
  function(
    string,
    non_token = FALSE
  ){
    res <- text_tokenize(string, "\n")
    if(non_token){
      tmp <- text_tokenize(string, "[^\n]")
      tmp$is_token <- rep(FALSE, dim(tmp)[1])
      res <- rbind(res, tmp)
      res <- stringb_arrange(res, "from", "to")
    }
    return(res)
  }



#' generic to tokenize text into sentences
#'
#' @param string the text to be tokenized
#' @param non_token whether or not token as well as non tokens shall be returned.
#' @export
text_tokenize_sentences <- function(string, non_token=FALSE){
  UseMethod("text_tokenize_sentences")
}

#' text_tokenize default
#' @rdname text_tokenize_sentences
#' @method text_tokenize_sentences default
#' @export
text_tokenize_sentences.default <- function(string, non_token=FALSE){
    # find sentence boundaries
    sentence_boundaries_1       <- text_locate_all(string, "([\\.\\!\\?][ \n]+\\p{Lu})", perl=TRUE)[[1]]
    sentence_boundaries_1$start <- sentence_boundaries_1$start+1
    sentence_boundaries_1$end   <- sentence_boundaries_1$end-1
    sentence_boundaries_2       <- text_locate_all(string, "(\n ?\n+)", perl=TRUE)[[1]]

    sentence_boundaries  <- rbind(sentence_boundaries_1, sentence_boundaries_2)
    sentence_boundaries  <-
      subset(
        sentence_boundaries,
```

```
      !is.na(sentence_boundaries$start),
      -length
    )

  # invert to sentences
  sentences <-
    subset(
      invert_spans(sentence_boundaries, end=nchar(string)),
      TRUE,
      -length
    )
  names(sentences) <- c("from", "to")

  # get text
  sentences$token <- substring(string, sentences$from, sentences$to)
  sentences$is_token <- TRUE
  # non_token
  if( non_token ){
    names(sentence_boundaries) <- c("from", "to")
    sentence_boundaries$token  <- substring(string, sentence_boundaries$from, sentence_boundaries$to)
    sentence_boundaries$is_token <- FALSE
    sentences <- rbind(sentences, sentence_boundaries)
    sentences <- stringb_arrange(sentences, "from", "to")
  }
  # return
  return(sentences)
}
```

text trim.R

```
#' trim spaces
#' @param string text to be trimmed
#' @param pattern regex to look for
#' @param side defaults to both might also be left, right, both or b, r, l to
#'             express where to trim pattern away
#' @param ... further arguments passed through to text_replace()
#' @export
text_trim <- function(string, side=c("both","left","right"), pattern=" ", ...){
  UseMethod("text_trim")
}
```

```
#' trim spaces default
#' @rdname text_trim
#' @method text_trim default
#' @export
text_trim.default <- function(string, side=c("both","left","right"), pattern=" ", ... ){
  # sanatizing side
  stopifnot(all(side %in% c("both", "left", "right", "b", "l", "r")))
  side <- side[1]
  # pimping pattern to match series at start / end
  p_start <- paste0("^", pattern, "*")
  p_end   <- paste0(pattern, "*$")
  if(side == "both" | side == "b" | side=="left" | side=="l"){
    string <- text_replace(string, pattern = p_start, replacement = "", ...)
  }
  if(side == "both" | side == "b" | side=="right" | side=="r"){
    string <- text_replace(string, pattern = p_end, replacement = "", ...)
  }
  string
}


#' trim spaces list
#' @rdname text_trim
#' @method text_trim list
#' @export
text_trim.list <- function(string, side=c("both","left","right"), pattern=" ", ... ){
  lapply(string, text_trim, side=side, pattern=pattern, ...)
}


#' trim spaces numeric
#' @rdname text_trim
#' @method text_trim numeric
#' @export
text_trim.numeric <- function(string, side=c("both","left","right"), pattern=" ", ... ){
  text_trim(as.character(string), side=side, pattern=pattern, ...)
}
```

text which.R

```
#' generic function to know in which elements a pattern can be found
#' @param string the text to be searched through
#' @param pattern regex to look for
#' @param ... further arguments passed through to \link[base]{grepl}
```

```
#' @export
text_which <- function(string, pattern, ...){
  UseMethod("text_which")
}


#' text_which default method
#' @rdname text_which
#' @method text_which default
#' @export
text_which.default <- function(string, pattern, ...){
  grep(pattern=pattern, x=string, ...)
}

#' generic function to know in which elements a pattern can be found
#' @rdname text_which
#' @export
text_grep <- function(string, pattern, ...){
  UseMethod("text_which")
}


#' generic function to get whole elements in which pattern was found
#' @param string the character vector to be searched through
#' @param pattern regex to look for
#' @param ... further arguments passed through to \link[base]{grep}
#' @export
text_which_value <- function(string, pattern, ...){
  UseMethod("text_which_value")
}

#' generic function to get whole elements in which pattern was found
#' @rdname text_which_value
#' @export
text_grepv <- function(string, pattern, ...){
  UseMethod("text_which_value")
}

#' text_which_value default method
#' @rdname text_which_value
#' @method text_which_value default
#' @export
text_which_value.default <- function(string, pattern, ...){
  grep(pattern=pattern, x=string, value=TRUE, ...)
}


#' generic for subsetting/filtering vectors
#' @param string text to be subsetted
#' @param pattern regular expression to subset by
#' @param ... further arguments passed through to \link[base]{grep}
#' @export
text_subset <- function(string, pattern, ...){
  UseMethod("text_which_value")
```

```
}

#' generic for subsetting/filtering vectors
#' @param string text to be subsetted
#' @param pattern regular expression to subset by
#' @param ... further arguments passed through to \link[base]{grep}
#' @export
text_filter <- function(string, pattern, ...){
  UseMethod("text_which_value")
}
```

text wrap.R

```
#' wraping text to specified width
#'
#' @param string text to be wrapped
#' @param ... further arguments passed through to \link[base]{strwrap}
#' @seealso \link[base]{strwrap}
#' @export
text_wrap = function(string, ...){
  UseMethod("text_wrap")
}

#' text_wrap default
#' @rdname text_wrap
#' @method text_wrap default
#' @export
text_wrap.default = function(string, ...){
  strwrap(string)
}
```

text write.R

```
#' write text to file
#'
#' A generic function to write text to file (or a \link[base]{connection}) and
```

```
#'    accompanying methods that wrap \link[base]{writeLines} to do so. In contrast
#'    to vanilla writeLines() text_write() (1) is  a generic so methods, handling
#'    something else than character vectors, can be implemented (2) in contrast to
#'    writeLines()' default to transform to write text in the system locale
#'    text_write() will default to UTF-8 no matter the locale (3) furthermore this
#'    encoding can be changed to any encoding supported by \link[base]{iconv}
#'    (see also \link[base]{iconvlist})
#'
#' @param string text to be written
#' @param file file name or file path or an \link[base]{connection} object -
#'    passed through to writeLines()'s con argument
#' @param sep character to separate lines (i.e. vector elements) from each other
#' - passed through to writeLines()'s con argument
#' @param encoding encoding in which to write text to disk
#' @param ... further arguments that might be passed to methods
#' (not used at the moment)
#' @export
text_write <- function(string, file, sep="\n", encoding="UTF-8", ...){
  UseMethod("text_write")
}


#' text_write() default
#'
#' @rdname text_write
#' @method text_write default
#' @export
text_write.default <- function(string, file, sep="\n", encoding="UTF-8", ...){
  writeLines(
    text     = iconv(as.character(string),to=encoding),
    con      = file,
    sep      = sep,
    useBytes = TRUE
  )
}
```

tools.R

```
#' function to invert spans to those numbers not covered
#' @param from vector of span starts
#' @param to vector of span ends
#' @param start minimum
#' @param end maximum value
invert_spans <- function(from, to=NULL, start=1, end=Inf){
  if( is.data.frame(from) & is.null(to) ){
    to   <- from$end
    from <- from$start
  }
  if(is.infinite(end)){
    tmp <- (start:(max(to)+1))[!(start:(max(to)+1) %in% sequenize(from, to))]
  }else{
    tmp <- (start:end)[!(start:end %in% sequenize(from, to))]
```

```
  }
  tmp <- de_sequenize(tmp)
  if(is.infinite(end)){
    tmp$end[length(tmp$end)] <- Inf
  }
  tmp$length <- tmp$end - tmp$start +1
  tmp$length[is.na(tmp$length)] <- Inf
  return(tmp)
}

#' helper function that turns cut points into spans
#' @param cuts where after to cut into pieces
#' @param end where does it all end
#' @keywords internal
cuts_to_spans <- function(cuts, start=1, end=Inf){
  cuts <- sort(cuts)
  # doing duty to do
  from  <- c(1, cuts + 1)
  to    <- c(cuts, end)
  tmp   <- data.frame(from, to)
  # consistency checks
  tmp <-
    subset(
      tmp,
      !(
        to > end |
          from > end |
          duplicated(tmp) |
          to < start |
          from < start
      )
    )
  return(tmp)
}


#' helper function to spans into sequences
#' @param start first number of sequence
#' @param end last number of sequence
#' @param simplify discard order, duplicaes etc?
#' @keywords internal
sequenize <- function(start, end=NULL, simplify=TRUE){
  if( is.null(end) ){
    if(is.matrix(start)){
      end   <- start[,2]
      start <- start[,1]
    }else{
      end   <- start[[2]]
      start <- start[[1]]
    }
  }
  tmp <- mapply(seq, start, end)
  if(simplify){
    tmp <- sort(unique(unlist(tmp)))
```

```
  }
  return(tmp)
}



#' helper function to transforms sequences into spans
#' @param x a bunch of numbers to urn into sequences
#' @keywords internal
de_sequenize <- function(x){
  x <- sort(unique(unlist(x)))
  xmin  <- min(x)
  xlead <- x[-1]
  xdiff <- c(xlead, NA) - x
  iffer <- is.na(xdiff) | xdiff > 1
  end   <- x[iffer]
  start <- c( xmin, xlead[iffer[seq_len(length(iffer)-1)]] )
  return(data.frame(start, end))
}



#' helper function for text_replace_group
#' @param x text_replace_group result
#' @param groups groups to extract
#' @keywords internal
get_groups <- function(x, group){
  groups <- text_c("group", group)
  tmp <- list()
  for(i in seq_along(groups) ){
    if( is.null(x[[groups[i]]]) ){
      tmp[[groups[i]]] <- rep(NA, dim(x)[1])
    }else{
      tmp[[groups[i]]] <- x[[groups[i]]]
    }
  }
  tmp <- as.data.frame(tmp)
  return(tmp)
}

#' helper function to standardize regexpr results
#' @param tmp regexpr or gregexpr result
#' @keywords internal
regmatches2 <- function(tmp, group=TRUE){
  if(is.list(tmp)){
    return(lapply(tmp, regmatches2, group=group))
  }
  # make data frame of match positions
  start            <- tmp
  start[start==-1] <- NA
  length               <- attr(start, "match.length")
  length[ length < 0]  <- NA
  end <- ifelse( length == 0, NA, start + length-1 )
  attributes(start) <- NULL
```

```
  df <- data.frame(start, end, length)
  # return
  return(df[group,])
}

#' helper for usage of regmatches
#' @param tmp result from regexec or gregexpr or regexpr
#' @keywords internal
drop_non_group_matches <-  function(tmp, group=TRUE){
  for(i in seq_along(tmp) ){
    if( !tmp[[i]][1]==-1 ){
      match_length <- attr(tmp[[i]], "match.length")
      use_bytes    <- attr(tmp[[i]], "useBytes")
      tmp[[i]]     <- tmp[[i]][-1][group]
      attr(tmp[[i]], "match.length") <- match_length[-1][group]
      attr(tmp[[i]], "useBytes")     <- use_bytes
    }
  }
  tmp
}

#' a stringsAsFactors=FALSE data.frame
#' @param ... passed through to data.frame
#' @param stringsAsFactors set to false by default
#' @keywords internal
data.frame <- function(..., stringsAsFactors=FALSE){
  base::data.frame(..., stringsAsFactors = stringsAsFactors)
}

#' a stringsAsFactors=FALSE as.data.frame
#' @param ... passed through to data.frame
#' @param stringsAsFactors set to false by default
#' @keywords internal
as.data.frame <- function(..., stringsAsFactors=FALSE){
  base::as.data.frame(..., stringsAsFactors = stringsAsFactors)
}

#' function to sort df by variables
#' @param df data.frame to be sorted
#' @param ... column names to use for sorting
stringb_arrange <- function(df, ...){
  sorters    <- as.character(as.list(match.call()))
  if( length(sorters)>2 ){
    sorters    <- sorters[-c(1:2)]
    sorters    <- paste0("df['",sorters,"']", collapse = ", ")
    order_call <- paste0("order(",sorters,")")
    return(df[eval(parse(text=order_call)), ])
  }else{
    return(df)
  }
}

#' text function: wrapper for system.file() to access test files
#' @param x name of the file
```

```
#' @keywords internal
test_file <- function(x=NULL){
  if(is.numeric(x)){
    return(test_file(test_file()[(x-1) %% length(test_file()) +1 ]))
  }
  if(is.null(x)){
    return(list.files(system.file("testfiles", package = "stringb")))
  }else if(x==""){
    return(list.files(system.file("testfiles", package = "stringb")))
  }else{
    return(system.file(paste("testfiles", x, sep="/"), package = "stringb") )
  }
}
```

zzz.R

```
#' #' function executet on loading the package
#' .onLoad <- function(libname, pkgname) {
#'   packageStartupMessage(
#'     "Please cite in any publication as:"
#'   )
#' }
```

rtext

imports.r

```
#' @importFrom R6 R6Class
#' @import hellno
#' @import stringb
#' @useDynLib rtext
#' @importFrom Rcpp sourceCpp
NULL
```

```
#' magrittr pipe
#' @importFrom magrittr %>%
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
NULL
```

prometheus.R

```
#' prometheus early version
#' @source https://de.wikisource.org/w/index.php?title=Prometheus_(Gedicht,_fr%C3%BChe_Fassung)&oldid=2
"prometheus_early"

#' prometheus late version
#' @source https://de.wikisource.org/w/index.php?title=Prometheus_(Gedicht,_sp%C3%A4te_Fassung)&oldid=1
"prometheus_late"
```

R6 rtext extended.r

```
#' extended R6 class
#'
#'
#' @docType class
#'
#' @name R6_rtext_extended
#'
#' @export
#'
#' @keywords data
#'
#' @return Object of \code{\link{R6Class}}
#'
#' @format \code{\link{R6Class}} object.
#'
#' @seealso \code{\link{rtext}}
#'
R6_rtext_extended <-
  R6::R6Class(

    #### class name ================================================================
    "R6_rtext_extended",

    #### private ===================================================================
    private = list(
      hashes = list(),
      hashed =
        function(name=NULL){
          # special case NULL
          if( is.null(name) ){
```

```
      name <- self$ls()$name
    }
    # recursion
    if( length(name)>1 ){
      tmp <- lapply(name, private$hashed)
      names(tmp) <- name
      return(tmp)
    }
    # doing-duty-to-do
    if( is.null(private$hashes[[name]]) ){
      private$hashes[[name]] <- private$hash(name)
    }
    return(
      private$hashes[[name]]
    )
  },
hash   =
  function(name=NULL){
    # special case NULL
    if( is.null(name) ){
      name <- self$ls()$name
    }
    # recursion
    if( length(name)>1 ){
      tmp <- lapply(name, private$hash)
      names(tmp) <- name
      return(tmp)
    }
    # doing-duty-to-do
    tmp <- self$get(name)
    tmp <- rtext_hash(tmp)
    private$hashes[[name]] <- tmp
    # return
    return(tmp)
  }
),

#### public ==========================================================
public = list(

  #### [ options ] #### ..................................................

  options =
    list(
      verbose = TRUE, # should message method print messages or not
      warning = TRUE  # should warnings pushed via self$warning() be reported
    ),

  #### [ get() ] #### ....................................................
  #      get stuff (private or public out of instance)
  get = function(name=NULL){
    # recursion
    if( length(name)>1 ){
      tmp <- lapply(name, self$get)
```

```
    names(tmp) <- name
    return(tmp)
  }
  if(is.null(name)){
    self$message("no input, returning NULL")
    return(NULL)
  }
  # self
  if(name=="self"){
    return(self)
  }
  # in self
  if( name %in% names(self) ){
    return(base::get(name, envir=self))
  }
  # private or in private
  if( exists("private") ){
    if(name=="private"){
      return(private)
    }else if(name %in% names(private) ){
      return(base::get(name, envir=private))
    }
  }
  # else
  self$message("name not found")
  return(NULL)
},

debug = function(pos=1){
  assign("self", self, envir = as.environment(pos))
  assign("private", private, envir = as.environment(pos))
  self$message("[self] and [private] assigned to global environment")
  # return self for piping
  return(invisible(self))
},

#### [ ls() ] #### ..................................................
#      list contents of instance
ls = function( what=c("self","private"), class=NULL){
  tmp_where   <- character(0)
    tmp_names <- character(0)
  tmp_classes <- character(0)
  df <- data.frame()
  if( "self" %in% what ){
    tmp_where   <- "self"
    tmp_names   <- ls(self)
    tmp_classes <-
      vapply(
        X         = tmp_names,
        FUN       =
          function(x){
            paste(class(self[[x]]), sep = ", ", collapse = ", ")
          },
        FUN.VALUE = character(1)
```

```
      )
    df <-
      rbind(
        data.frame(
          name  = tmp_names,
          where = tmp_where,
          class = tmp_classes
        ),
        df,
        make.row.names = FALSE
      )
  }
  if( "private" %in% what & exists("private") ){
    tmp_where   <- "private"
    tmp_names   <- ls(private)
    tmp_classes <-
      vapply(
        X         = tmp_names,
        FUN       =
          function(x){
            paste(class(private[[x]]), sep = ", ", collapse = ", ")
          },
        FUN.VALUE = character(1)
      )
    df <-
      rbind(
        data.frame(
          name  = tmp_names,
          where = tmp_where,
          class = tmp_classes
        ),
        df,
        make.row.names = FALSE
      )
  }
  if( dim(df)[1] > 0 ){
    df <- df[order(df$where, df$class, df$name), ]
  }
  if( !is.null(class) ){
    df <- df[grep(class, df$class), ]
  }
  return(df)
},

#### [ message() ] #### ..................................................
#      post a message (if verbose is set to TRUE)
message = function(x, ...){
  xname <- as.character(as.list(match.call()))[2]
  if(self$options$verbose){
    if(is.character(x)){
      message(class(self)[1], " : ", x, ...)
    }else{
      message(class(self)[1], " : ", xname, " : \n", x)
    }
```

```
      }
    },

    #### [ warning() ] #### ..............................................
    #      post a warning (if vwarning is set to TRUE)
    warning = function(x, ...){
      xname <- as.character(as.list(match.call()))[2]
      if(self$options$warning){
        if(is.character(x)){
          warning(class(self)[1], " : ", x, ...)
        }else{
          warning(class(self)[1], " : ", xname, " : \n", x)
        }
      }
    }
  )


)
```

RcppExports.R

```
# Generated by using Rcpp::compileAttributes() -> do not edit by hand
# Generator token: 10BE3573-1514-4C36-9D1C-5A225CD40393

#' (function to check which chars belong to which token)
#' takes a vector of xs to check if these lie between pairs of ys and if so
#' returning their index; assumes xs and ys are sorted; returns only the first
#' span index which enclosing the x
#' @param x a vector of type numeric; a number to be placed into a span
#' @param y1 a vector of type numeric; first element of span
#' @param y2 a vector of type numeric; last element of span
#' @keywords internal
which_token_worker <- function(x, y1, y2) {
    .Call('rtext_which_token_worker', PACKAGE = 'rtext', x, y1, y2)
}
```

rtext.R

```
#' R6 class - linking text and data
#'
#' @docType class
#' @name rtext
#' @export
#' @keywords data
#' @return Object of \code{\link{R6Class}}
#' @format An \code{\link{R6Class}} generator object.
#' @section The rtext class family:
#'
```

```
#' Rtext consists of an set of R6 classes that are conencted by inheritance.
#' Each class handles a different set of functionalities that are - despite
#' needing the data structure provided by rtext_base - independent.
#'
#' \describe{
#'    \item{R6_rtext_extended}{
#'      A class that has nothing to do per se with rtext
#'      but merely adds some basic features to the base R6 class (debugging,
#'      hashing, getting fields and handling warnings and messages as well as
#'      listing content)
#'    }
#'
#'    \item{rtext_base}{
#'    [inherits from R6_rtext_extended] The foundation of the rtext class.
#'      This class allows to load and store text, its meta data, as well as data
#'      about the text in a character by character level.
#'    }
#'
#'    \item{rtext_loadsave}{
#'      [inherits from rtext_base] Adds load and save methods for loading and saving
#'      rtext objects (text and data) into/from Rdata files.}
#'
#'    \item{rtext_export}{
#'      [inherits from rtext_loadsave] Adds methods to import and export from and
#'      to SQLite databases - like load and save but for SQLite.
#'    }
#'
#'    \item{rtext_tokenize}{
#'      [inherits from rtext_export] Adds methods to aggregate character level data
#'      onto token level. (the text itself can be tokenized via S3 methods from
#'      the stringb package - e.g. text_tokenize_words())
#'    }
#'
#'    \item{rtext}{
#'      [inherits from rtext_tokenize] Adds no new features at all but is just a
#'      handy label sitting on top of all the functionality provided by the
#'      inheritance chain.
#'    }
#' }
#'
#' @examples
#'
#' # initialize (with text or file)
#' quote_text <-
#' "Outside of a dog, a book is man's best friend. Inside of a dog it's too dark to read."
#' quote <- rtext$new(text = quote_text)
#'
#' # add some data
#' quote$char_data_set("first", 1, TRUE)
#' quote$char_data_set("last", quote$char_length(), TRUE)
#'
#' # get the data
#' quote$char_data_get()
#'
```

```
#' # transform text
#' quote$char_add("[this is an insertion] \n", 47)
#'
#' # get the data again (see, the data moved along with the text)
#' quote$text_get()
#' quote$char_data_get()
#'
#' # do some convenience coding (via regular expressions)
#' quote$char_data_set_regex("dog_friend", "dog", "dog")
#' quote$char_data_set_regex("dog_friend", "friend", "friend")
#' quote$char_data_get()
#'
#' # aggregate data by regex pattern
#' quote$tokenize_data_regex(split="(dog)|(friend)", non_token = TRUE, join = "full")
#'
#' # aggregate data by words
#' quote$tokenize_data_words(non_token = TRUE, join="full")
#'
#' # aggregate data by lines
#' quote$tokenize_data_lines()
#'
#' # plotting and data highlighting
#' plot(quote, "dog_friend")
#'
#' # adding further data to the plot
#' plot(quote, "dog_friend")
#' plot(quote, "first", col="steelblue", add=TRUE)
#' plot(quote, "last", col="steelblue", add=TRUE)
#'
rtext <-
  R6::R6Class(

    #### misc ============================================================
    classname    = "rtext",
    active       = NULL,
    inherit      = rtext_tokenize,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('rtext'),

    #### private =========================================================
    private = list(),

    #### public ==========================================================
    public = list()

)
```

rtext base.R

```
#' rtext_base : basic workhorse for rtext
#'
#' @docType class
#'
#' @name rtext_base
#'
#' @export
#'
#' @keywords data
#'
#' @return Object of \code{\link{R6Class}}
#'
#' @format \code{\link{R6Class}} object.
#'
#' @seealso \code{\link{rtext}}
#'
rtext_base <-
  R6::R6Class(

    #### misc ===========================================================================
    classname    = "rtext_base",
    active       = NULL,
    inherit      = R6_rtext_extended,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('rtext'),

    #### private ========================================================================
    private = list(
      text = function(){
        paste0(private$char, collapse = "")
      },

      char                = character(0),
      char_data           = list()
    ),

    #### public =========================================================================
    public = list(
```

```
#### puplic data fields ================================================
text_file  = as.character(NA),
encoding   = as.character(NA),
sourcetype = as.character(NA),
id         = NULL,
save_file  = {a <- NA; a <- as.character(a); a},

#### [ initialize ] #### ..............................................
initialize =
  function(
    text        = NULL,
    text_file   = NULL,
    encoding    = "UTF-8",
    id          = NULL,
    save_file   = NULL,
    verbose     = TRUE
  )
  {

    ##### Saving verbose option
    self$options$verbose <- verbose

    ##### Stating what is done
    self$message("initializing")

    ##### read in text // set field: sourcetype
    if(is.null(text) & is.null(text_file)){ # nothing at all
      private$char <- ""
      self$sourcetype <- "empty"
    }else if(is.null(text) & !is.null(text_file)){ # read from text_file
      private$char <- text_read(text_file, encoding = encoding, tokenize = "")
      self$sourcetype <- "text_file"
    }else{ # take text as supplied
      private$char <-
        unlist(strsplit(paste0(iconv(text, encoding, "UTF-8"), collapse = "\n"),""))
      self$sourcetype <- "text"
    }

    ##### set field: text_file
    if( !is.null(text_file) ){
      self$text_file <- text_file
    }

    ##### set field: save_file
    if( !is.null(save_file) ){
      self$save_file <- save_file
    }

    ##### Encoding
    Encoding(private$char) <- "UTF-8"
    self$encoding <- "UTF-8"

    ##### ID
```

```
    if( is.null(id) ){
      self$id <- rtext_hash(self)
    }else{
      self$id <- id
    }

    ##### Hashing again
    private$hash()
  }
,

#### methods =================================================================

# info
info = function(){
  res <-
    list(
      text_file  = self$text_file,
      character  = length(private$char),
      encoding   = self$encoding,
      sourcetype = self$sourcetype
    )
  return(res)
},

#### [ text_show ] #### ..................................................
text_show = function(length=500, from=NULL, to=NULL, coll=FALSE, wrap=FALSE){
  text_show(x=self$text_get(Inf), length=length, from=from, to=to, coll=coll, wrap=wrap)
},

#### [ text_get ] ........................................................
text_get = function(length=Inf, from=NULL, to=NULL, split=NULL){
  res <- rtext_get_character(chars=private$char, length=length, from=from, to=to)
  res <- paste0(res, collapse = "")
  Encoding(res) <- self$encoding
  if( !is.null(split) ){
    res <- unlist(strsplit(res, split = split))
    Encoding(res) <- self$encoding
  }
  return(res)
},

#### [ text_get_lines ] ..................................................
text_get_lines = function(length=Inf, from=NULL, to=NULL){
    # get text
    tmp_text  <- self$text_get(length=length, from=from, to=to)
    # split/tokenize
    tmp       <- text_tokenize(tmp_text, "\n")
    # gather other data
    tmp$is_token <- NULL
    line <- seq_along(tmp$from)
    n    <- nchar(tmp$token)
    #return
    return(
```

```
      data.frame(line, n, from=tmp$from, to=tmp$to, text=tmp$token)
    )
},

#### [ char_get ] #### ..................................................
char_get = function(length=Inf, from=NULL, to=NULL, raw=FALSE){
  if(raw | identical(length, TRUE) ){
    res <- private$char
    Encoding(res) <- self$encoding
    return(res)
  }
  res <- get_vector_element(vec=private$char, length=length, from=from, to=to)
  Encoding(res) <- self$encoding
  return(res)
},

#### [ char_add ] #### ..................................................
char_add = function(what=NULL, after=NULL){
  what        <- enc2utf8(what)
  what        <- unlist(strsplit(what,""))
  if( is.null(after) ) {
    private$char <- c(private$char, what)
  }else if ( after==0 ) {
    private$char <- c(what, private$char)
    # update char_data$i
    for( name_i in seq_along(names(private$char_data)) ){
      name <- names(private$char_data)[name_i]
      private$char_data[[name]]$i <- private$char_data[[name]]$i + length(what)
    }
  }else{
    index  <- seq_along(private$char)
    part1  <- private$char[index <= after]
    part2  <- private$char[index >  after]
    private$char <- c( part1, what, part2)
    # update char_data$i
    for( name_i in seq_along(names(private$char_data)) ){
      name <- names(private$char_data)[name_i]
      iffer <- private$char_data[[name]]$i > after
      private$char_data[[name]]$i[iffer] <- private$char_data[[name]]$i[iffer] + length(what)
    }
  }
  # necessary updates
  private$hash("char")
  # return for piping
  invisible(self)
},

#### [ char_delete ] #### ..................................................
char_delete = function(n=NULL, from=NULL, to=NULL){
  non_deleted  <- vector_delete(x = seq_along(private$char), n=n, from=from, to=to)
  private$char <- vector_delete(x = private$char, n=n, from=from, to=to)
  # update char_data$i (drop deletd data, update index)
  new_index           <- seq_along(non_deleted)
  for( name_i in seq_along(names(private$char_data)) ){
```

```
    name <- names(private$char_data)[name_i]
    private$char_data[[name]] <-
      subset(private$char_data[[name]], private$char_data[[name]]$i %in% non_deleted)
    private$char_data[[name]]$i <- new_index[match(private$char_data[[name]]$i, non_deleted)]
  }
  # necessary updates
  private$hash(c("char", "char_data"))
  # return for piping
  invisible(self)
},

#### [ char_replace ] #### ..............................................
char_replace = function(from=NULL, to=NULL, by=NULL){
  # check input
  stopifnot( !is.null(from), !is.null(to), !is.null(by) )
  by <- enc2utf8(by)
  # doing-duty-to-do
  index <- seq_along(private$char)
  private$char <-
    c(
      private$char[index < from],
      unlist(strsplit(by, "")),
      private$char[index > to]
    )
  # updata char_data
  for( name_i in seq_along(names(private$char_data)) ){
    name <- names(private$char_data)[name_i]
    private$char_data[[name]] <-
      subset(
        private$char_data[[name]],
        private$char_data[[name]]$i < from | private$char_data[[name]]$i > to
      )
    iffer <-
      private$char_data[[name]]$i > to
    private$char_data[[name]]$i[iffer] <-
      private$char_data[[name]]$i[iffer] + nchar(by) - to - from + 1
  }
  # necessary updates
  private$hash("char")
  # return for piping
  invisible(self)
},
char_length = function(){
  length(private$char)
},

#### [ char_data_set ] #### ..............................................
char_data_set = function(x=NULL, i=NULL, val=NA, hl = 0){
  # check input
  if( length(i) == 0 ){
    if( is.null(private$char_data[[x]]) ){
        tmp <-
          subset(data.frame(i=1, char="", x=val), FALSE)
        names(tmp)[3] <- x
```

```
      private$char_data[[x]]  <- tmp
  }
  return(invisible(self))
}
stopifnot( length(x) == 1 )
stopifnot( x != c("i", "char", "hl") )
if( is.null(x) | is.null(i) ){
  warning("char_data_set : no sufficient information passed for x, i - nothing coded")
  invisible(self)
}
if( any( i > self$char_length() | any( i < 1)) ){
  stop("char_data_set : i out of bounds")
}
# prepare input
if( length(val)==1 ){
  val <- rep(val, length(i))
}
if( length(hl)==1 ){
  hl <- rep(hl, length(i))
}
# check for coresponding lengths
stopifnot( length(i) == length(val) & length(val) == length(hl) )

# make sure there is a data frame to fill
if( is.null(private$char_data[[x]] ) ){
  private$char_data[[x]] <-
    subset(
      data.frame(
        i    = 1L,
        hl   = 0
      ),
      FALSE
    )
}

# split data
# - new i in old i and level is less or equal to new level
# -> already coded with lower level are discarded!
i_in_char_data  <-
  merge(
    data.frame(i=i),
    subset(private$char_data[[x]], TRUE,  c("i", "hl")),
    all.x = TRUE,
    by="i"
  )$hl <= hl
i_in_char_data[is.na(i_in_char_data)] <- FALSE

# - adding those not already coded
i_not_in_char_data     <- !(i %in% private$char_data[[x]]$i)

# assign data with i already in i
input_to_data_matcher <-
  match(i[i_in_char_data], private$char_data[[x]]$i)
```

```
  private$char_data[[x]][input_to_data_matcher, "i"] <-
    i[i_in_char_data]

  private$char_data[[x]][input_to_data_matcher, "hl"]   <-
    hl[i_in_char_data]

  private$char_data[[x]][input_to_data_matcher, x]   <-
    val[i_in_char_data]

  # code for i not already in char_data
  add_df <-
    data.frame(
      i  = i[i_not_in_char_data],
      hl = hl[i_not_in_char_data]
    )

  add_df[[x]] <-
    val[i_not_in_char_data]

  private$char_data[[x]] <-
    rbind_fill(
      private$char_data[[x]],
      add_df
    ) %>%
    dp_arrange("i")

  # necessary updates
  private$hash("char_data")

  # return for piping
  invisible(self)
},

#### [ char_data_set_regex ] #### ......................................
char_data_set_regex = function(x=NULL, pattern=NULL, val=NA, hl=0, ...){
  found_spans <- text_locate_all(private$text(), pattern, ...)[[1]]
  found_spans <- subset(found_spans, !is.na(start) & !is.na(end))
  found_is    <- unique(as.integer(unlist(mapply(seq, found_spans$start, found_spans$end))))
  self$char_data_set(x=x, i=found_is, val=val,  hl=hl)
},

#### [ char_data_get ] #### ........................................
char_data_get = function( from = 1, to = Inf, x = NULL, full=FALSE){
  if( from > length(private$char) | to < 1 | to < from ){
    return(data.frame())
  }
  # subset columns
  if( is.null(x) ){
    l_tbr <- private$char_data
  }else{
    l_tbr <- private$char_data[ x ]
    iffer <-
      vapply(
        l_tbr,
```

```
          function(x){
            if( is.null(x) ){ return(TRUE) }else{
              if( dim1(x)==0 ){ return(TRUE) }else{
                return(FALSE)
            }}
          },
          TRUE
        )
      l_tbr <- l_tbr[!iffer]
    }
    # something to return?
    if( length(l_tbr) == 0 ){
      res <- data.frame(i=seq(max(from, 1), min(to, length(private$char))))
    }else{
      # putting together data.frames
      l_tbr <- lapply(l_tbr, function(x){x["hl"] <- NULL;x})
      res <- Reduce(
        function(x, y){
          merge(x, y, by="i", all=TRUE)
        },
        l_tbr
      )
    }
    # subset according to: from and to
    res <- subset(res, res$i >= from & res$i <= to)
    # adding char
    char_i <- seq_along(private$char)
    iffer <- char_i >= from & char_i <= to
    char <- data.frame(char = private$char[iffer], i = char_i[iffer]  )
    res <-
      merge(
        char,
        subset(res, res$i >= from & res$i <=to),
        by  = "i",
        all.x = full,
        all.y = TRUE
      )
    # adding xs not found
    x_amiss <- x[!(x %in% names(res))]
    for( i in seq_along(x_amiss) ){
      res[x_amiss[i]] <- rep(NA, dim1(res))
    }
    # return
    return( res )
  },

  #### [ hash_get ] #### ..................................................
  hash_get = function(name=""){
    private$hashed(name)
  }
 )
)
```

rtext export.R

```
#' R6 class - linking text and data
#'
#' @docType class
#' @name rtext_export
#' @export
#' @keywords data
#' @return Object of \code{\link{R6Class}}
#' @format \code{\link{R6Class}} object.
#' @seealso \code{\link{rtext}}
#'
rtext_export <-
  R6::R6Class(

    #### misc ============================================================
    classname    = "rtext_export",
    active       = NULL,
    inherit      = rtext_loadsave,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('rtext'),

    #### private ===========================================================
    private = list(),

    #### public ============================================================
    public = list(

      #### [ export_csv ] #### ..................................................
      export_csv = function(folder_name = ""){
        stopifnot(file.info(folder_name)$isdir)
        "TBD"
      },

      #### [ import_csv ] #### ..................................................
      import_csv = function(folder_name = ""){
        stopifnot(file.info(folder_name)$isdir)
        "TBD"
      },

      #### [ export_sqlite ] #### ..............................................
      export_sqlite = function(db_name = ""){
        # establish connection
        if( is.character(db_name) ){
          con <- RSQLite::dbConnect( RSQLite::SQLite(), db_name)
```

```
    on.exit({
        RSQLite::dbDisconnect(con)
    })
  }else{
    con <- db_name
  }
  # preapare data to be exportd
  tb_exported <- private$prepare_save()
  # export data
  RSQLite::dbBegin(con)
    RSQLite::dbWriteTable(con, "meta",      tb_exported$meta, overwrite=TRUE)
    RSQLite::dbWriteTable(con, "hashes",    tb_exported$hashes, overwrite=TRUE)
    RSQLite::dbWriteTable(con, "char",      as.data.frame(tb_exported$char), overwrite=TRUE)
    for(i in seq_along(tb_exported$char_data) ){
      if(i==1){
        overwrite <- TRUE
        append    <- FALSE
      }else{
        overwrite <- FALSE
        append    <- TRUE
      }
      tmp_name <- names(tb_exported$char_data[[i]])[3]
      tmp <- tb_exported$char_data[[i]]
      names(tmp) <- c("i","hl","val")
      tmp[["var"]] <- tmp_name
      RSQLite::dbWriteTable(
        con,
        "char_data",
        tmp,
        overwrite=overwrite,
        append=append
      )
    }
  RSQLite::dbCommit(con)
  # return
  return(invisible(self))
},

#### [ import_sqlite ] #### ..................................................
import_sqlite = function(db_name = ""){
  # establish connection
  if( is.character(db_name) ){
    con <- RSQLite::dbConnect(RSQLite::SQLite(), db_name)
    on.exit({
      RSQLite::dbDisconnect(con)
    })
  }else{
    con <- db_name
  }
  # import data
  imported <- list()
  imported$meta   <- RSQLite::dbReadTable(con, "meta")
  imported$hashes <- RSQLite::dbReadTable(con, "hashes")
  imported$char   <- RSQLite::dbReadTable(con, "char")[[1]]
```

```
        # import char_data
        if( RSQLite::dbExistsTable(con, "char_data") ){
          tmp <- RSQLite::dbReadTable(con, "char_data")
          tmp <- split(tmp, f=tmp$var)
          for( i in seq_along(tmp) ) {
            nam <- tmp[[i]]$var[1]
            tmp[[i]][[4]] <- NULL
            names(tmp[[i]]) <- c("i", "hl", nam)
          }
        }else{
          tmp <- list()
        }
        imported$char_data   <- tmp
        # incorporate data
        private$execute_load(imported)
        # return self for piping
        return(invisible(self))
      }
    )
  )
```

rtext loadsave.R

```
#' R6 class - load and save methods for rtext
#'
#' @docType class
#' @name rtext_loadsave
#' @export
#' @keywords data
#' @return Object of \code{\link{R6Class}}
#' @format \code{\link{R6Class}} object.
#' @seealso \code{\link{rtext}}
#'
rtext_loadsave <-
  R6::R6Class(

    #### misc ============================================================================
    classname    = "rtext_loadsave",
    active       = NULL,
    inherit      = rtext_base,
    lock_objects = TRUE,
```

```
class        = TRUE,
portable     = TRUE,
lock_class   = FALSE,
cloneable    = TRUE,
parent_env   = asNamespace('rtext'),

#### private ========================================================
private = list(

  #### [ prepare_save ] #### ..........................................
  prepare_save = function(id=NULL){
    # handle id option
    if( is.null(id) ){
      id <- self$id
    }else if( id[1] == "hash"){
      tb_saved$meta$id <- self$hash()
    }else{
      tb_saved$meta$id <- id[1]
    }
    # put together information
    tb_saved <-
      list(
        meta = data.frame(
          id           = id,
          date         = as.character(Sys.time()),
          text_file    = self$text_file,
          encoding     = self$encoding,
          save_file    = ifelse(is.null(self$save_file), NA, self$save_file),
          sourcetype   = self$sourcetype,
          rtext_version= as.character(packageVersion("rtext")),
          r_version    = paste(version$major, version$minor, sep="."),
          save_format_version = 1
        ),
        hashes       = as.data.frame(private$hash()),
        char         = private$char,
        char_data    = private$char_data
      )
    class(tb_saved) <- c("rtext_save","list")
    # return
    return(tb_saved)
  },

  #### [ execute_load ] #### ..........................................
  execute_load = function(tmp){
    # setting public
    self$id         <- tmp$meta$id
    self$text_file  <- tmp$meta$text_file
    self$encoding   <- tmp$meta$encoding
    self$sourcetype <- tmp$meta$sourcetype
    self$save_file  <- tmp$meta$save_file

    # setting private
    private$char       <- tmp$char
    private$char_data  <- tmp$char_data
```

```
      # updating rest
      private$hash()

      # return for piping
      invisible(self)
    }
  ),

  #### public ==================================================================
  public = list(

  #### [ save ] #### ..........................................................
  save = function(file=NULL, id=NULL){
    rtext_save <- as.environment(private$prepare_save(id=id))
    # handle file option
    if( is.null(rtext_save$meta$save_file) & is.null(file) ){
      stop("rtext$save() : Neither file nor save_file given, do not know where to store file.")
    }else if( !is.null(file) ){
      file <- file
    }else if( !is.null(rtext_save$meta$save_file) ){
      file <- rtext_save$meta$save_file
    }
    # save to file
    base::save(
      list = ls(rtext_save),
      file = file,
      envir = rtext_save
    )
    # return for piping
    return(invisible(self))
  },

  #### [ load ] ................................................................
  load = function(file=NULL){
    # handle file option
    if( is.null(file) ){
      stop("rtext$load() : file is not given, do not know where to load file from.")
    }else{
      file <- file
    }
    # loading info
    tmp <- load_into(file)
    # applying loaded info to self
    private$execute_load(tmp)
    # return self for piping
    return(invisible(self))
  }
 )
)
```

rtext tokenize.R

```
#' R6 class - linking text and data
#'
#' @docType class
#' @name rtext_tokenize
#'
#' @export
#' @keywords data
#' @return Object of \code{\link{R6Class}}
#' @format \code{\link{R6Class}} object.
#' @seealso \code{\link{rtext}}
#'
rtext_tokenize <-
  R6::R6Class(

    #### misc ===========================================================================
    classname    = "rtext_tokenize",
    active       = NULL,
    inherit      = rtext_export,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('rtext'),

    #### private ========================================================================
    private = list(),

    #### public =========================================================================
    public = list(

      #### [ tokenize_data_regex ] #### ....................................
      tokenize_data_regex =
        function(
          split       = NULL,
          ignore.case = FALSE,
          fixed       = FALSE,
          perl        = FALSE,
          useBytes    = FALSE,
          non_token   = FALSE,
          join        = c("full", "left", "right", ""),
          aggregate_function = NULL,
          ...
        ){
          # tokenize text
          token <-
            text_tokenize(
              private$text(),
              regex       = split,
              ignore.case = ignore.case,
              fixed       = fixed,
              perl        = perl,
```

```
        useBytes    = useBytes,
        non_token   = non_token
      )

    # tokenize data and aggegation
    self$tokenize_data_sequences(token=token, join=join, aggregate_function=aggregate_function, .
  },

#### [ tokenize_data_sequences ] #### ................................
tokenize_data_sequences = function(
  token,
  join               = c("full", "left", "right", ""),
  aggregate_function = NULL,
  ...
){
  token$token_i <- seq_dim1(token)

  join <- ifelse(is.numeric(join), c("full", "left", "right", "")[join], join[1])

  # tokenize data and aggegation
  token_data <-
    data.frame(token_i=NULL, start=NULL, end=NULL)
  chardata <- self$char_data_get()

  if( !is.null( chardata$i) ){
    # datanize tokens
    token_i <-
      which_token(
        chardata$i,
        token$from,
        token$to
      )
    # aggregate data
    if( !is.null(aggregate_function) ){
      # user supplied functions and otpions
      token_data <-
        chardata[,-c(1,2)] %>%
        stats::aggregate(by = list( token_i=token_i ), FUN=aggregate_function, ... )
    }else{
      # standard
      token_data <-
        stats::aggregate(
          chardata[,-c(1:2)],
          by = list( token_i=token_i ),
          FUN="modus",
          multimodal=NA,
          warn=FALSE
        )
    }
    #      names(private$token_data)[-1] <- names(private$char_data)[-1]
  }
  # join token and data
  if( join=="full" ){
    res <- merge(token, token_data, all = TRUE)
```

```
  }else if( join=="left" ){
    res <- merge(token, token_data, all.x = TRUE)
  }else if( join=="right" ){
    res <- merge(token, token_data, all.y = TRUE)
  }else{
    res <- merge(token, token_data)
  }
  # return
  return(res)
},

#### [ tokenize_data_words ] #### ....................................
tokenize_data_words =
  function(
    split       = "\\W+",
    ignore.case = FALSE,
    fixed       = FALSE,
    perl        = FALSE,
    useBytes    = FALSE,
    non_token   = FALSE,
    join        = c("full", "left", "right", ""),
    aggregate_function = NULL,
    ...
  ){
    self$tokenize_data_regex(
      split       = split,
      ignore.case = ignore.case,
      fixed       = fixed,
      perl        = perl,
      useBytes    = useBytes,
      non_token   = non_token,
      join        = join,
      aggregate_function = aggregate_function,
      ...
    )
  },

#### [ tokenize_data_lines ] #### ....................................
tokenize_data_lines =
  function(
    split       = "\n",
    ignore.case = FALSE,
    fixed       = FALSE,
    perl        = FALSE,
    useBytes    = FALSE,
    non_token   = FALSE,
    join        = c("full", "left", "right", ""),
    aggregate_function = NULL,
    ...
  ){
    self$tokenize_data_regex(
      split       = split,
      ignore.case = ignore.case,
      fixed       = fixed,
```

```
            perl        = perl,
            useBytes    = useBytes,
            non_token   = non_token,
            join        = join,
            aggregate_function = aggregate_function,
            ...
          )
        }
    )
  )
```

rtext tools.R

```
#' function to get text from rtext object
#'
#' @param chars the chars field
#' @param length number of characters to be returned
#' @param from first character to be returned
#' @param to last character to be returned
#' @keywords internal
# #' @export
rtext_get_character <- function(chars, length=100, from=NULL, to=NULL){
  # helper functions
  bind_to_charrange <- function(x){bind_between(x, 1, length(chars))}
  bind_length       <- function(x){bind_between(x, 0, length(chars))}
  return_from_to    <- function(from, to, split){
    res  <- chars[seq(from=from, to=to)]
    return(res)
  }
  # only length
  if( !is.null(length) & ( is.null(from) & is.null(to) ) ){
    length <- max(0, min(length, length(chars)))
    length <- bind_length(length)
    if(length==0){
      return("")
    }
    from   <- 1
    to     <- length
    return(return_from_to(from, to, split))
  }
  # from and to (--> ignores length argument)
```

```
  if( !is.null(from) & !is.null(to) ){
    from <- bind_to_charrange(from)
    to   <- bind_to_charrange(to)
    return(return_from_to(from, to, split))
  }
  # length + from
  if( !is.null(length) & !is.null(from) ){
    if( length<=0 | from + length <=0 ){
      return("")
    }
    to   <- from + length-1
    if((to < 1 & from < 1) | (to > length(chars) & from > length(chars) )){
      return("")
    }
    to   <- bind_to_charrange(to)
    from <- bind_to_charrange(from)
    return(return_from_to(from, to, split))
  }
  # length + to
  if( !is.null(length) & !is.null(to) ){
    if( length<=0 | to - (length-1) > length(chars) ){
      return("")
    }
    from <- to - length + 1
    if((to < 1 & from < 1) | (to > length(chars) & from > length(chars) )){
      return("")
    }
    from <- bind_to_charrange(from)
    to   <- bind_to_charrange(to)
    return(return_from_to(from, to, split))
  }
  stop("rtext$get_character() : I do not know how to make sense of given length, from, to argument valu
}



#' function for plotting rtext
#' @export
#' @param x object of class rtext
#' @param y char_data to be plotted
#' @param lines vector of integer listing the lines to be plottted
#' @param col color of the char_data variable to be highlighted
#' @param add add data to an already existing plot?
#' @param ... further parameters passed through to initial plot
plot.rtext <-
  function(
    x,
    y         = NULL,
    lines     = TRUE,
    col       = "#ED4C4CA0",
    add       = FALSE,
    ...
  ){
    # preparing data
```

```
what       <- y
line_data  <- subset(x$text_get_lines(), lines)
plot_x     <- line_data$n
plot_y     <- line_data$line
max_plot_y <- max( plot_y )
plot_y     <- abs( plot_y - max_plot_y ) + 1
max_plot_x <- max( plot_x )

# plotting text lines
if(!add){
  graphics::plot(
    x    = plot_x,
    y    = plot_y,
    type = "n",
    ylab = "line",
    xlab = "char",
    xlim      = c(0, (ceiling(max_plot_x)/10^nchar(max_plot_x)*10)*(10^nchar(max_plot_x)/10) ),
    ylim      = c(0, max_plot_y + 1 ),
    ...,
    axes=FALSE
  )
  graphics::axis( 1 )
  graphics::axis( 2, c(max_plot_y, 1), c(1, max_plot_y) )
  graphics::box()
  graphics::rect(
    xleft   = 0,
    xright  = plot_x,
    ybottom = plot_y - 0.5,
    ytop    = plot_y + 0.5,
    col = "grey", border = "grey", lty=0
  )
}
# plotting char_data
if ( !is.null(what) ){
  char_data <-
    x$char_data_get(
      x    = what,
      from = min(line_data$from),
      to   = max(line_data$to)
    )

  index <- which_token( char_data$i, line_data$from, line_data$to)
  plot_what_x <- char_data$i - line_data[ index, ]$from
  plot_what_y <- line_data[ index, ]$line
  plot_what_y <- abs( plot_what_y - max_plot_y ) +1
  graphics::rect(
    xleft   = plot_what_x,
    xright  = plot_what_x + 1,
    ybottom = plot_what_y - 0.5,
    ytop    = plot_what_y + 0.5,
    col = col, border = col, lty=0
  )
}
# return
```

```
    if(!exists("char_data")){char_data<-NULL}
    return(
      invisible(
        list(
          line_data = line_data,
          char_data = char_data
          )
        )
      )
  }
```

text tools.R

```
#' function tokenizing rtext objects
#' @inheritParams stringb::text_tokenize
#' @method text_tokenize rtext
#' @export
text_tokenize.rtext <-   function(
  string,
  regex       = NULL,
  ignore.case = FALSE,
  fixed       = FALSE,
  perl        = FALSE,
  useBytes    = FALSE,
  non_token   = FALSE
){
  string$text_get() %>%
    text_tokenize(
      regex       = regex,
      ignore.case = ignore.case,
      fixed       = fixed,
      perl        = perl,
      useBytes    = useBytes,
      non_token   = non_token
    ) %>%
    dp_arrange("from","to") %>%
    return()
```

```
}
```

tools.R

```
#' function to write csv files with UTF-8 characters (even under Windwos)
#' @param df data frame to be written to file
#' @param file file name / path where to put the data
#' @keywords internal
write_utf8_csv <-
  function(df, file){
    if ( is.null(df) ) df <- data.frame()
    firstline <- paste(  '"', names(df), '"', sep = "", collapse = " , ")
    char_columns <- seq_along(df[1,])[sapply(df, class)=="character"]
    #for( i in  char_columns){
    #  df[,i] <- toUTF8(df[,i])
    #}
    data <- apply(df, 1, function(x){paste('"', x,'"', sep = "",collapse = " , ")})
    writeLines( text=c(firstline, data), con=file , useBytes = T)
  }


#' function to read csv file with UTF-8 characters (even under Windwos) that
#' were created by write_U
#' @param file file name / path where to get the data
#' @keywords internal
read_utf8_csv <- function(file){
  if ( !file.exists(file) ) return( data.frame() )
  # reading data from file
  content <- readLines(file, encoding = "UTF-8")
  if ( length(content) < 2 ) return( data.frame() )
  # extracting data
  content <- stringb::text_split(content, " , ")
  content <- lapply(content, stringb::text_replace_all, '"', "")
  content_names <- content[[1]][content[[1]]!=""]
  content <- content[seq_along(content)[-1]]
  # putting it into data.frame
  df <- data.frame(dummy=seq_along(content), stringsAsFactors = F)
  for(name in content_names){
    tmp <- sapply(content, `[[`, dim(df)[2])
    Encoding(tmp) <- "UTF-8"
    df[,name] <- tmp
  }
  df <- df[,-1]
  # return
  return(df)
}



#' function to get hash for R objects
#' @param x the thing to hash
```

```
#' @keywords internal
rtext_hash <- function(x){
  digest::digest(x, algo="xxhash64")
}

#' text function: wrapper for system.file() to access test files
#' @param x name of the file
#' @param pattern pattern of file name
#' @keywords internal
testfile <- function(x=NULL, pattern=NULL, full.names=FALSE){
  if(is.numeric(x)){
    return(testfile(testfile()[(x-1) %% length(testfile()) +1 ]))
  }
  if(is.null(x)){
    return(
      list.files(
        system.file(
          "testfiles",
          package = "rtext"
        ),
        pattern = pattern,
        full.names = full.names
      )
    )
  }else if(x==""){
    return(
      list.files(
        system.file(
          "testfiles",
          package = "rtext"
        ),
        pattern = pattern,
        full.names = full.names
      )
    )
  }else{
    return(
      system.file(
        paste("testfiles", x, sep="/"),
        package = "rtext")
    )
  }
}


#' function used to delete parts from a vector
#' @param x input vector
#' @param n number of items to be deleted
#' @param from from which position onwards elements should be deleted
#' @param to up to which positions elements should be deleted
#' @keywords internal

vector_delete <- function(x, n=NULL, from=NULL, to=NULL){
  # shortcuts
```

```
  if( is.null(n) ){
    if(is.null(from) & is.null(to)){
      return(x)
    }
  }else{
    if( n==0){
      return(x)
    }
  }
  # iffer
  iffer <- TRUE
  if( is.null(from) & is.null(to)  & !is.null(n) ){ # only n
    iffer <- seq_along(x) > length(x) | seq_along(x) <= length(x)-n
  }else if( !is.null(from) & is.null(to)  & is.null(n) ){ # only from
    iffer   <- seq_along(x) < from
  }else if( is.null(from) & !is.null(to) & is.null(n) ){ # only to
    iffer   <- seq_along(x) > to
  }else if( !is.null(from) & !is.null(to)  & is.null(n) ){ # from + to
    iffer   <- seq_along(x) > to | seq_along(x) < from
  }else if( !is.null(from) & is.null(to)  & !is.null(n) ){ # from + n
    if( n > 0 ){
      n     <- bind_between(n-1, 0, length(x))
      iffer <- seq_along(x) > from+n | seq_along(x) < from
    }
  }else if( is.null(from) & !is.null(to)  & !is.null(n) ){ # to + n
    iffer <- seq_along(x) > to | seq_along(x) <= to-n
  }
  # return
  return( x[iffer] )
}




#' function that loads saved rtext
#' @param save_file a saved rtext object in Rdata format
#' @keywords internal

load_into <- function(save_file){
  tmp_env <- new.env(parent = emptyenv())
  load(save_file, envir = tmp_env)
  tmp <- lapply(tmp_env, I)
  class(tmp) <- NULL
  return(tmp)
}


#' function that shifts vector values to right or left
#'
#' @param x Vector for which to shift values
#' @param n Number of places to be shifted.
#'    Positive numbers will shift to the right by default.
#'    Negative numbers will shift to the left by default.
#'    The direction can be inverted by the invert parameter.
```

```
#' @param default The value that should be inserted by default.
#' @param invert Whether or not the default shift directions
#'    should be inverted.
#' @keywords internal

shift <- function(x, n=0, default=NA, invert=FALSE){
  n <-
    switch (
      as.character(n),
      right    =  1,
      left     = -1,
      forward  =  1,
      backward = -1,
      lag      =  1,
      lead     = -1,
      as.numeric(n)
    )
  if( length(x) <= abs(n) ){
    if(n < 0){
      n <- -1 * length(x)
    }else{
      n <- length(x)
    }
  }
  if(n==0){
    return(x)
  }
  n <- ifelse(invert, n*(-1), n)
  if(n<0){
    n <- abs(n)
    forward=FALSE
  }else{
    forward=TRUE
  }
  if(forward){
    return(c(rep(default, n), x[seq_len(length(x)-n)]))
  }
  if(!forward){
    return(c(x[seq_len(length(x)-n)+n], rep(default, n)))
  }
}

#' function forcing value to fall between min and max
#' @param x the values to be bound
#' @param max upper boundary
#' @param min lower boundary
#' @keywords internal
bind_between <- function(x, min, max){
  x[x<min] <- min
  x[x>max] <- max
  return(x)
}
```

```
#' function for binding data.frames even if names do not match
#' @param df1 first data.frame to rbind
#' @param df2 second data.frame to rbind
#' @keywords internal

rbind_fill <- function(df1=data.frame(), df2=data.frame()){
  names_df <- c(names(df1), names(df2))
  if( dim1(df1) > 0 ){
    df1[, names_df[!(names_df %in% names(df1))]] <- rep(NA, dim1(df1))
  }else{
    df1 <- data.frame()
  }
  if( dim1(df2) > 0 ){
    df2[, names_df[!(names_df %in% names(df2))]] <- rep(NA, dim1(df2))
  }else{
    df2 <- data.frame()
  }
  rbind(df1, df2)
}




#' function that checks is values are in between values
#' @param x input vector
#' @param y lower bound
#' @param z upper bound
#' @keywords internal
is_between <- function(x,y,z){
  return(x>=y & x<=z)
}


#' function that extracts elements from vector
#'
#' @param vec the chars field
#' @param length number of elements to be returned
#' @param from first element to be returned
#' @param to last element to be returned
#' @keywords internal
get_vector_element <-
  function(vec, length=NULL , from=NULL, to=NULL){
    # helper functions
    bind_to_vecrange <- function(x){bind_between(x, 1, length(vec))}
    bind_length      <- function(x){bind_between(x, 0, length(vec))}
    return_from_to   <- function(from, to, split){
      res  <- vec[seq(from=from, to=to)]
      return(res)
    }
    # only length
    if( !is.null(length) & ( is.null(from) & is.null(to) ) ){
      length <- max(0, min(length, length(vec)))
      length <- bind_length(length)
      if(length==0){
```

```
      return("")
    }
    from   <- 1
    to     <- length
    return(return_from_to(from, to, split))
  }
  # from and to (--> ignores length argument)
  if( !is.null(from) & !is.null(to) ){
    from <- bind_to_vecrange(from)
    to   <- bind_to_vecrange(to)
    return(return_from_to(from, to, split))
  }
  # length + from
  if( !is.null(length) & !is.null(from) ){
    if( length<=0 | from + length <=0 ){
      return("")
    }
    to   <- from + length-1
    if((to < 1 & from < 1) | (to > length(vec) & from > length(vec) )){
      return("")
    }
    to   <- bind_to_vecrange(to)
    from <- bind_to_vecrange(from)
    return(return_from_to(from, to, split))
  }
  # length + to
  if( !is.null(length) & !is.null(to) ){
    if( length<=0 | to - (length-1) > length(vec) ){
      return("")
    }
    from <- to - length + 1
    if((to < 1 & from < 1) | (to > length(vec) & from > length(vec) )){
      return("")
    }
    from <- bind_to_vecrange(from)
    to   <- bind_to_vecrange(to)
    return(return_from_to(from, to, split))
  }
  stop("get_vector_element() : I do not know how to make sense of given length, from, to argument valu
}



#' get first dimension or length of object
#' @param x object, matrix, vector, data.frame, ...

#' @keywords internal
dim1 <- function(x){
  ifelse(is.null(dim(x)[1]), length(x), dim(x)[1])
}


#' get first dimension or length of object
#' @param x object, matrix, vector, data.frame, ...
```

```
#' @keywords internal

dim2 <- function(x){
  dim(x)[2]
}


#' seq along first dimension / length
#' @param x x
#' @keywords internal

seq_dim1 <- function(x){
  seq_len(dim1(x))
}


#' function returning index of spans that entail x
#' @param x position of the character
#' @param y1 start position of the token
#' @param y2 end position of the token
#' @keywords internal

which_token <- function(x, y1, y2){
  # how to order x and y?
  order_x <- order(x)
  order_y <- order(y1)
  # order x and y! - which_token_worker expects inputs to be ordered
  ordered_x  <- x[order_x]
  ordered_y1 <- y1[order_y]
  ordered_y2 <- y2[order_y]
  # doing-duty-to-do
  index <- which_token_worker(ordered_x, ordered_y1, ordered_y2)
  # ordering back to input ordering
  index <- order_y[index[order(order_x)]]
  # return
  index
}




#' function giving back the mode

#' @param x vector to get mode for
#' @param multimodal wether or not all modes should be returned in case of more than one
#' @param warn should the function warn about multimodal outcomes?
#' @export
modus <- function(x, multimodal=FALSE, warn=TRUE) {
  x_unique <- unique(x)
  tab_x    <- tabulate(match(x, x_unique))
  res      <- x_unique[which(tab_x==max(tab_x))]
  if( identical(multimodal, TRUE) ){
    return(res)
  }else{
```

```
    if( warn & length(res) > 1 ){
      warning("modus : multimodal but only one value returned (use warn=FALSE to turn this off)")
    }
    if( !identical(multimodal, FALSE) & length(res) > 1 ){
      return(multimodal)
    }else{
      return(res[1])
    }
  }
}
```

```
#' function to get classes from e.g. lists
#' @param x list to get classes for
#' @keywords internal
classes <- function(x){
  tmp <- lapply(x, class)
  data.frame(name=names(tmp), class=unlist(tmp) , row.names = NULL)
}
```

```
#' function to sort df by variables
#' @param df data.frame to be sorted
#' @param ... column names to use for sorting
#' @keywords internal
dp_arrange <- function(df, ...){
  sorters    <- as.character(as.list(match.call()))
  if( length(sorters)>2 ){
    sorters    <- sorters[-c(1:2)]
    sorters    <- paste0("df['",sorters,"']", collapse = ", ")
    order_call <- paste0("order(",sorters,")")
    res        <- df[eval(parse(text=order_call)), ]
    if( is.data.frame(df) & !is.data.frame(res) ){
      res <- as.data.frame(res)
      names(res) <- names(df)
    }
    return(res)
  }else{
    return(df)
  }
}
```

zzz.R

```
 # function executet on loading the package
 #.onLoad <- function(libname, pkgname) {
   #packageStartupMessage("Please cite in any publication as:")
   #library(stringb)
#}
```

diffrprojects

diff align.R

```
if(getRversion() >= "2.15.1"){
  utils::globalVariables(
    c(
      "token_i_1", "token_i_2",
      "text1_tokenized", "text2_tokenized",
      "token", ".", "...", "res_token_i_1", "res_token_i_2",
      "min_dist_1"
    )
  )
}

#' algining texts
#'
#' Function aligns two texts side by side as a data.frame with change type and
#' distance given as well
#'
#' @param text1 first text
#' @param text2 second text
#' @param tokenizer defaults to NULL which will trigger linewise tokenization;
#'        accepts a function that turns a text into a token data frame;
#'        a token data frame has at least three columns:
#'        from (first character of token),
```

```
#'        to (last character of token)
#'        token (the token)
#' @param ignore defaults to NULL which means that nothing is ignored;
#'        function that accepts a token data frame (see above) and returns a
#'        possibly subseted data frame of hte same form
#' @param clean defaults to NULL which means that nothing cleaned; accepts a
#'        function that takes a vector of tokens and returns a vector of same
#'        length - potentially clean up
#' @param distance defaults to Levenshtein ("lv"); see \link[stringdist]{amatch},
#'        \link[stringdist]{stringdist-metrics}, \link[stringdist]{stringdist}
#' @param ... further arguments passed through to distance function
#' @param verbose should function report on its doings via messages or not
#' @inheritParams stringdist::stringdist
#'
#' @return dataframe with tokens aligned according to distance
#'
#' @export
diff_align <- function(
  text1     = NULL,
  text2     = NULL,
  tokenizer = NULL,
  ignore    = NULL,
  clean     = NULL,
  distance  = c("lv", "osa", "dl", "hamming", "lcs", "qgram", "cosine", "jaccard", "jw", "soundex"),
  useBytes = FALSE,
  weight = c(d = 1, i = 1, s = 1, t = 1),
  maxDist = 0,
  q = 1,
  p = 0,
  nthread = getOption("sd_num_thread"),
  verbose = TRUE,
  ...
){
  # checking input
  if( is.function(distance) ){ stop("using non standard distance functions is not implemented yet - sor
  stopifnot(
    !is.null(text1),
    is.character(text1),
    !is.null(text2),
    is.character(text2)
  )

  # assigning default options
  if( is.null(tokenizer) ){ tokenizer <- stringb::text_tokenize_lines }
  if( is.null(clean) ){     clean     <- function(x){x} }
  if( is.null(ignore) ){    ignore    <- function(x){x} }
  if( length(text1) > 1){ text1 <- text_collapse(text1) }
  if( length(text2) > 1){ text2 <- text_collapse(text2) }
  distance <- distance[1]
  if(maxDist == 0){ maxDist <- 1e-150}

  # tokenize
  if( verbose ){ message(" - tokenizing text") }
  text1_tokenized <- tokenizer(text1)[1:3]
```

```
text1_tokenized$token_i <- seq_along(text1_tokenized$token)

text2_tokenized <- tokenizer(text2)[1:3]
text2_tokenized$token_i <- seq_along(text2_tokenized$token)

# clean
if( verbose ){ message(" - cleaning token") }
text1_tokenized_prec <- text1_tokenized
text2_tokenized_prec <- text2_tokenized
text1_tokenized$token <- clean(text1_tokenized$token)
text2_tokenized$token <- clean(text2_tokenized$token)

# ignore
if( verbose ){ message(" - ignoring token") }
text1_tokenized_prei <- text1_tokenized
text2_tokenized_prei <- text2_tokenized
text1_tokenized <- ignore(text1_tokenized)
text2_tokenized <- ignore(text2_tokenized)

# column naming
text1_tokenized_prec <- stats::setNames(text1_tokenized_prec, c("from_1", "to_1", "token_1", "token_i
text2_tokenized_prec <- stats::setNames(text2_tokenized_prec, c("from_2", "to_2", "token_2", "token_i
text1_tokenized_prei <- stats::setNames(text1_tokenized_prei, c("from_1", "to_1", "token_1", "token_i
text2_tokenized_prei <- stats::setNames(text2_tokenized_prei, c("from_2", "to_2", "token_2", "token_i
text1_tokenized <- stats::setNames(text1_tokenized, c("from_1", "to_1", "token_1", "token_i_1"))
text2_tokenized <- stats::setNames(text2_tokenized, c("from_2", "to_2", "token_2", "token_i_2"))

# alignment and distances
if( verbose ){ message(" - doing distance calculation and alignment") }

# distance
a <-
  stringdist::amatch(
    text1_tokenized$token_1,
    text2_tokenized$token_2,
    method  = distance,
    useBytes = useBytes,
    weight = weight,
    maxDist = maxDist,
    q = q,
    p = p,
    nthread = nthread,
    matchNA = FALSE
  )

# alignment
alignment <-
  data.frame(
    text1_tokenized,
    text2_tokenized[a, ]
  )

alignment$distance <-
  stringdist::stringdist(
```

```
    alignment$token_1,
    alignment$token_2,
    method  = distance,
    useBytes = useBytes,
    weight = weight,
    q = q,
    p = p,
    nthread = nthread
  )

# type and distances
if( dim1(alignment) > 0 ){
  alignment$type <- ""
  alignment$type[alignment$distance == 0]<-"no-change"
  alignment$type[alignment$distance >  0]<-"change"

alignment <-
  rtext:::rbind_fill(
    alignment,
    text1_tokenized[
      !(text1_tokenized$token_i_1 %in% alignment$token_i_1),
      ]
  )

alignment <-
  rtext:::rbind_fill(
    alignment,
    text2_tokenized[
      !(text2_tokenized$token_i_2 %in% alignment$token_i_2),
      ]
  )

iffer <- is.na(alignment$token_2)
alignment[iffer, "type"]     <- "deletion"
alignment[iffer, "distance"] <-
  stringdist::stringdist(
    "",
    alignment[iffer, "token_1"],
    method  = distance,
    useBytes = useBytes,
    weight = weight,
    q = q,
    p = p,
    nthread = nthread
  )

iffer <- is.na(alignment$token_1)
alignment[iffer, "type"]     <- "insertion"
alignment[iffer, "distance"] <-
  stringdist::stringdist(
    "",
    alignment[iffer, "token_2"],
    method  = distance,
    useBytes = useBytes,
```

```
    weight = weight,
    q = q,
    p = p,
    nthread = nthread
  )

}

# non matches
if( dim1(text1_tokenized_prei)>0 ){
  tmp <-
    subset(
      cbind(text1_tokenized_prei, type="ignored"),
      !(text1_tokenized_prei$token_i_1 %in% alignment$token_i_1)
    )
  alignment <-
    rtext:::rbind_fill(alignment, tmp)
}

if( dim1(text2_tokenized_prei)>0 ){
tmp <-
  subset(
    cbind(text2_tokenized_prei, type="ignored"),
    !(text2_tokenized_prei$token_i_2 %in% alignment$token_i_2)
  )
alignment <-
  rtext:::rbind_fill(alignment, tmp)
}

# original token
if( dim1(alignment) > 0 ){
alignment$token_1 <-
  dplyr::left_join(
    subset(alignment, select="token_i_1"),
    subset(text1_tokenized_prec, select=c("token_i_1", "token_1") ),
    by=c("token_i_1"="token_i_1")
  )$token_1

alignment$token_2 <-
  dplyr::left_join(
    subset(alignment, TRUE, token_i_2),
    subset(text2_tokenized_prec, select=c("token_i_2", "token_2") ),
    by=c("token_i_2"="token_i_2")
  )$token_2
}

# column order and missing columns
if( !("type" %in% names(alignment)) ){
  alignment <- cbind(alignment, type=character(0))
}

alignment <-
  subset(
    alignment,
```

```
      select=c(
        "token_i_1", "token_i_2", "distance", "type",
        "from_1", "to_1", "from_2", "to_2",
        "token_1",  "token_2"
      )
    )

  # return
  return(alignment)
}
```

dp.R

```
#' class for diffrproject
#'
#' @docType class
#'
#' @name diffrproject
#'
#' @export
#'
#' @keywords data
#'
#' @return Object of \code{\link{diffrproject}}
#'
#' @format \code{\link{R6Class}} creator object.
#'
#' @section The diffrprojects class family:
#'
#' Diffrproject consists of an set of R6 classes that are conencted by inheritance.
#' Each class handles a different set of functionalities that are modular.
```

```
#'
#' \describe{
#'    \item{R6_rtext_extended}{
#'      A class that has nothing to do per se with diffrprojects.
#'      It merely adds some basic features to the base R6 class (debugging,
#'      hashing, getting fields and handling warnings and messages as well as
#'      listing content). This class is imported from rtext package
#'    }
#'
#'    \item{dp_base}{
#'      [inherits from rtext::R6_rtext_extended]
#'      This class forms the foundation of all diffrpojects (dp_xxx) classes by
#'      implementing data fields for meta data, texts, data on texts,
#'      links between texts, alignment of text tokens, and data on the alignment
#'      of text tokens. Furthermore it implements methods add, delete, code, and
#'      link texts or to aggregate text data on text token level.
#'    }
#'
#'    \item{dp_loadsave}{
#'      [inherits from dp_base]
#'      This class allows for loading and saving diffrprojects from and to Rdata
#'      files.
#'    }
#'
#'    \item{dp_export}{
#'      [inherits from dp_loadsave]
#'      This class provides methods for exporting and importing to and from
#'      RSQLite.
#'    }
#'
#'    \item{dp_align}{
#'      [inherits from dp_export]
#'      This is one of the workhorses of diffrprojects. The methods of this class
#'      allow for adding, deleting or computing alignments between text tokens
#'      (e.g. words or lines or sentences or characters or paragraphs, or some
#'      other way to split text into chunks). Furthermore it allows to also
#'      assign data to individual alignments (a connection beween two token of
#'      text from different text versions).
#'    }
#'
#'    \item{dp_inherit}{
#'      [inherits from dp_align]
#'      The text_data_inherit method added by this class allows to copy text
#'      data from one token of a text version to another token of another text
#'      version channeled through aligments with zero distance. Conflicting
#'      codings (a text might have multiple codings stemming from several links
#'      and from direct coding of the text) are resolved by the fact that text
#'      codings are accompanied by a hierarchy level that defaults to zero and
#'      gets decreased by one every time the coding is inherited by a token.
#'    }
#'
#'    \item{diffrproject}{
#'      [inherits from dp_inherit]
#'      Just a wrapper inheriting from dp_inherit to have a less technical name
```

```
#'      at the end of the inheritance chain.
#'    }
#' }
#'
#'
#'
#'
diffrproject <-
  R6::R6Class(

    #### class name ============================================================
    classname    = "diffrproject",

    #### misc ==================================================================
    active       = NULL,
    inherit      = dp_inherit,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('diffrprojects')

  )# closes R6Class
```

dp align.R

```
#' class for dp_align
#'
#' @docType class
#'
#' @name dp_align
#'
#' @export
#'
#' @keywords data
#'
#' @return Object of \code{\link{dp_align}}
#'
#' @format \code{\link{R6Class}} object.
#'
#' @seealso \code{\link{diffrproject}}
#'
dp_align <-
  R6::R6Class(

    #### class name ============================================================
    classname    = "dp_align",

    #### misc ==================================================================
    active       = NULL,
```

```
inherit      = dp_export,
lock_objects = TRUE,
class        = TRUE,
portable     = TRUE,
lock_class   = FALSE,
cloneable    = TRUE,
parent_env   = asNamespace('diffrprojects'),

#### public ==================================================================
public = list(

  #### data ====================================================================

  #### methods =================================================================

  #### [ text_align() ] ========================================================

  text_align = function(
      t1=NULL,
      t2=NULL,
      tokenizer = NULL,
      ignore = NULL,
      clean = NULL,
      distance = c("lv", "osa", "dl", "hamming", "lcs", "qgram", "cosine", "jaccard", "jw", "soundex
      useBytes = FALSE,
      weight = c(d = 1, i = 1, s = 1, t = 1),
      maxDist = 0,
      q = 1,
      p = 0,
      nthread = getOption("sd_num_thread"),
      verbose = self$options$verbose,
      ...
    ){

    if( is.null(t1) & is.null(t2) ){
      # check again
      # if(interactive() & self$options$ask){
      #   y <- readline("Alignment for ALL files? \nyes / no : ")
      #   if( !any(grepl("y", y)) ){
      #     return(FALSE)
      #   }
      # }
      for(i in seq_along(self$link) ){
        self$text_align(
          self$link[[i]]$from,
          self$link[[i]]$to,
          tokenizer = tokenizer,
          ignore = ignore,
          clean=clean,
          distance=distance,
          useBytes = useBytes,
          weight=weight,
          maxDist = maxDist,
          q=q,
```

```
          p=p,
          nthread = nthread,
          verbose = verbose,
          ...
        )
      }
    }else{
      self$message("- doing alignment")

      tt1 <-
        self$text[[t1]]$text_get()
      tt2 <-
        self$text[[t2]]$text_get()

      alignment <-
        diff_align(
          tt1, tt2,
          tokenizer = tokenizer,
          ignore = ignore,
          clean=clean,
          distance=distance,
          useBytes = useBytes,
          weight=weight,
          maxDist = maxDist,
          q=q,
          p=p,
          nthread = nthread,
          verbose = verbose,
          ...
        )

      self$alignment_add(
        alignment,
        link = stringb::text_c(t1, "~", t2)
      )
    }
    # return
    return(invisible(self))
  },

#### [ alignment_add() ] ==============================================

alignment_add = function(x, link){

  # fetching link name if necessary
  if( !is.character(link) ){
    link <- names(self$link)[link]
  }

  # alignment_i
  alignment_i <- self$alignment[[link]]$alignment_i
  if( length(alignment_i) > 0){
    max_a <- max(alignment_i)
    alignment_i <-
```

```
      seq_len( max(alignment_i) )[ !(seq_len( max(alignment_i)) %in% alignment_i)]
  }else{
    max_a <- 0
  }
  x$alignment_i <-
    c( alignment_i, seq_len( dim1(x) - dim1(alignment_i)) + max_a )

  selection <-
    c(
      "alignment_i",
      "token_i_1", "token_i_2",
      "distance", "type",
      "from_1", "to_1",
      "from_2", "to_2"
    )
  x <- subset(x, select = selection[selection %in% names(x)] )

  # adding alignments
  self$alignment[[link]]
  tmp <-
    rbind_fill(
      self$alignment[[link]],
      x
    )
  self$alignment[[link]] <-
    subset(
      tmp,
      !duplicated(
        subset(tmp, select=c(from_1, to_1, from_2, to_2))
      )
    )

  # return for piping
  return(invisible(self))
},

#### [ alignment_delete() ] ============================================

alignment_delete =
  function(
    link=NULL, alignment_i=NULL, from_1=NULL, to_1=NULL, from_2=NULL, to_2=NULL, type=NULL
  ){

  # check input
  stopifnot( !is.null(link) )
  stopifnot(
    !is.null(alignment_i) |
    !is.null(from_1) | !is.null(to_1) |
    !is.null(from_2) | !is.null(to_2) |
    !is.null(type)
  )

  # recursion
  if( length(link)>1 ){
```

```
  for(i in seq_along(link)){
    self$alignment_delete(
      link        = link[i],
      alignment_i = alignment_i,
      from_1      = from_1,
      to_1        = to_1,
      from_2      = from_2,
      to_2        = to_2,
      type        = type
    )
  }
}else{ # no recursion

  if( !is.null(alignment_i) & (!is.null(from_1) | !is.null(to_1) | !is.null(from_2) | !is.null(to_
      self$warning("alignment_i and other arguments supplied - I cannot use bot groups at the same
    }

  # fetching link name if necessary
  if( !is.character(link) ){
    link <- names(self$link)[link]
  }

  # finish because link does not exist
  if(is.null(link)){
    self$warning("link not found")
    return(invisible(self))
  }

  # doing-duty-to-do
  iffer <- list()
  if( !is.null(alignment_i)){
    iffer[[1]] <- self$alignment[[link]]$alignment_i %in% alignment_i
  }
    iffer[[2]] <- self$alignment[[link]]$from_1 <= from_1
    iffer[[3]] <- self$alignment[[link]]$to_1   >= to_1
    iffer[[4]] <- self$alignment[[link]]$from_2 <= from_2
    iffer[[5]] <- self$alignment[[link]]$to_2   >= to_2
    iffer[[6]] <- as.character(self$alignment[[link]]$type)   == type

    f <- function(x){ if( length(x) == 0 ){ x<-rep(NA, dim1(self$alignment[[link]])) }; return(x)
    g <- function(x){
      if( all( is.na(x) ) ){
        return(FALSE)
      }
      if( all( is.na(x) | x ) ){
        return(TRUE)
      }
      FALSE
    }

  iffer  <- iffer %>% lapply(f) %>%  as.data.frame() %>% apply(1,g)
  wiffer <- self$alignment[[link]][!iffer, ]$alignment_i

  # update alignment_data
```

```
    for(i in seq_along(self$alignment_data[[link]]) ){
      iffer_tmp <- self$alignment_data[[link]][[i]]$alignment_i %in% wiffer
      self$alignment_data[[link]][[i]] <- self$alignment_data[[link]][[i]][iffer_tmp,]
    }

    # update alignments
    self$alignment[[link]] <- self$alignment[[link]][!iffer, ]
  }

  # update hashes
  private$hash("alignment")

  # return
  return(invisible(self))
},

#### [ alignment_code() ] ============================================

alignment_code =
  function(
    link=NULL, alignment_i=NULL, x=NULL, val=NA, hl = 0,
    pattern=NULL, pattern1=NULL, pattern2=NULL, invert=FALSE,
    from_1=NULL, to_1=NULL,
    from_2=NULL, to_2=NULL,
    type=NULL
  ){
    # check inputs
    stopifnot(!is.null(link), !is.null(x))

    # fetching link name if necessary
    if( !is.character(link) ){
      link <- names(self$link)[link]
    }

    # doing-duty-to-do
    iffer <- list()
    if( !is.null(alignment_i)){
      iffer[[1]] <- self$alignment[[link]]$alignment_i %in% alignment_i
    }
    iffer[[2]] <- self$alignment[[link]]$from_1 <= from_1
    iffer[[3]] <- self$alignment[[link]]$to_1   >= to_1
    iffer[[4]] <- self$alignment[[link]]$from_2 <= from_2
    iffer[[5]] <- self$alignment[[link]]$to_2   >= to_2
    iffer[[6]] <- as.character(self$alignment[[link]]$type)   == type

    if( !is.null(pattern) ){
      token_1 <-
        text_sub(
          self$text[[self$link[[link]]$from]]$text_get(),
          self$alignment[[link]]$from_1,
          self$alignment[[link]]$to_1
        )
      token_2 <-
        text_sub(
```

```
      self$text[[self$link[[link]]$to  ]]$text_get(),
      self$alignment[[link]]$from_2,
      self$alignment[[link]]$to_2
    )
  iffer[[7]] <-
    stringb::text_detect(token_1, pattern) |
    stringb::text_detect(token_2, pattern)
}

if( !is.null(pattern1) ){
  token_1 <-
    text_sub(
      self$text[[self$link[[link]]$from]]$text_get(),
      self$alignment[[link]]$from_1,
      self$alignment[[link]]$to_1
    )
  iffer[[8]] <-
    stringb::text_detect(token_1, pattern1)
}

if( !is.null(pattern2) ){
  token_2 <-
    text_sub(
      self$text[[self$link[[link]]$to  ]]$text_get(),
      self$alignment[[link]]$from_2,
      self$alignment[[link]]$to_2
    )
  iffer[[9]] <-
    stringb::text_detect(token_2, pattern2)
}

# combining iffer
f <- function(x){ if( length(x) == 0 ){ x<-rep(NA, dim1(self$alignment[[link]])) }; return(x) }
g <- function(x){
  if( all( is.na(x) ) ){
    return(FALSE)
  }
  if( all( is.na(x) | x ) ){
    return(TRUE)
  }
  FALSE
}

iffer  <- iffer %>% lapply(f) %>%  as.data.frame() %>% apply(1,g)
if( invert ){
  wiffer <- self$alignment[[link]][!iffer, ]$alignment_i
}else{
  wiffer <- self$alignment[[link]][iffer, ]$alignment_i
}

# setting values
self$alignment_data_set(
  link        = link,
  alignment_i = wiffer,
```

```
      val         = val,
      x           = x,
      hl          = hl
    )

    # return
    return(invisible(self))
},

#### [ alignment_set ] #### ..........................................
alignment_data_set = function(
  link=NULL, alignment_i=NULL, x=NULL, val=NA, hl = 0
){
  # check input
  stopifnot( length(x) == 1 )
  if( any(x == c("alignment_i", "link", "hl", "x")) ){
    stop("Reserved names used: alignment_i, link, hl, and x are reserved names - use another name!")
  }
  if( is.null(x) | is.null(alignment_i) | is.null(link) ){
    warning("char_data_set : no sufficient information passed for x, i - nothing coded")
    return(invisible(self))
  }
  if(
    any(
      alignment_i > max(self$alignment[[link]]$alignment_i) |
      any( alignment_i < 1)
    )
  ){
    stop("text_alignement_set : alignment_i out of bounds")
  }

  # fetching link name if necessary
  if( !is.character(link) ){
    link <- names(self$link)[link]
  }

  # prepare input
  if( length(val)==1 ){
    val <- rep(val, length(alignment_i))
  }
  if( length(hl)==1 ){
    hl <- rep(hl, length(alignment_i))
  }
  # check for coresponding lengths
  stopifnot( length(alignment_i) == length(val) & length(val) == length(hl) )

  # make sure there is a data frame to fill
  if( is.null(self$alignment_data[[link]][[x]] ) ){
    self$alignment_data[[link]][[x]] <-
      subset(
        data.frame(
          alignment_i    = 1L,
          hl   = 0
        ),
```

```
      FALSE
    )
}

# split data
# - new i in old i and level is less or equal to new level
# -> already coded with lower level are discarded!
i_in_data  <-
  merge(
    data.frame(alignment_i=alignment_i),
    subset(self$alignment_data[[link]][[x]], select=c("alignment_i", "hl")),
    all.x = TRUE,
    by="alignment_i"
  )$hl <= hl
i_in_data[is.na(i_in_data)] <- FALSE

# - adding those not already coded
i_notin_data     <- !(alignment_i %in% self$alignment_data[[link]][[x]]$alignment_i)

# assign data with i already in i
input_to_data_matcher <-
  match(alignment_i[i_in_data], self$alignment_data[[link]][[x]]$alignment_i)

self$alignment_data[[link]][[x]][input_to_data_matcher, "alignment_i"] <-
  alignment_i[i_in_data]

self$alignment_data[[link]][[x]][input_to_data_matcher, "hl"]   <-
  hl[i_in_data]

self$alignment_data[[link]][[x]][input_to_data_matcher, x]   <-
  val[i_in_data]

# code for i not already in char_data
add_df <-
  data.frame(
    alignment_i  = alignment_i[i_notin_data],
    hl = hl[i_notin_data]
  )

add_df[[x]] <-
  val[i_notin_data]

self$alignment_data[[link]][[x]] <-
  rbind_fill(
    self$alignment_data[[link]][[x]],
    add_df
  ) %>%
  dp_arrange("alignment_i")

# necessary updates
private$hash("alignment_data")

# return for piping
return(invisible(self))
```

```
},

#### [ text_code_alignment_token() ] ====================================================

text_code_alignment_token = function(link=NULL, alignment_i=NULL, text1=FALSE, text2=FALSE, x=NULL,
  # fetching link name if necessary
  if( !is.character(link) ){
    link <- names(self$link)[link]
  }

  tbc <-
    self$alignment[[link]] %>%
      subset(
        subset = self$alignment[[link]]$alignment_i %in% alignment_i,
        select = c("from_1","to_1", "from_2", "to_2")
      )

  l <- dim1(tbc)
  if( l != length(val) ){ val <- rep(val, l)[seq_len(l)] }
  if( l != length(hl ) ){ hl  <- rep(hl,  l)[seq_len(l)] }

  tbc$val <- val
  tbc$hl  <- hl
  tbc_split <- split(tbc, seq_dim1(tbc))

   if( text1 ){
     res <-
       do.call(
          rbind,
          lapply(tbc_split, function(x){
            if( !is.na(x$from_1) & !is.na(x$to_1) ){
              res <-
                data.frame(
                i   = seq(x$from_1, x$to_1),
                val = x$val,
                hl  = x$hl
              )
            }else{
              res <- subset(data.frame(i=0,val=NA,hl=0), FALSE)
            }
          return(res)
        })
       )

     self$text_code(self$link[[link]]$from, x=x, i=res$i, val=res$val, hl=res$hl)
  }

  if( text2 ){
    res <-
      do.call(
        rbind,
        lapply(tbc_split, function(x){
          if( !is.na(x$from_2) & !is.na(x$to_2) ){
            res <-
```

```
              data.frame(
                i   = seq(x$from_2, x$to_2),
                val = x$val,
                hl  = x$hl
              )
            }else{
              res <- subset(data.frame(i=0,val=NA,hl=0), FALSE)
            }
            return(res)
          })
      )

    self$text_code(self$link[[link]]$to, x=x, i=res$i, val=res$val, hl=res$hl)
  }
  return(invisible(self))
},

#### [ text_code_alignment_token_regex() ] ============================================

text_code_alignment_token_regex = function(link=NULL, alignment_i, text1=TRUE, text2=TRUE, x=NULL, p

},

#### [ alignment_data_full ] #### ..........................................
alignment_data_full = function(link=NULL, data_only=TRUE){

  # fetching link name if necessary
  if( is.null(link) ){
    link <- seq_along(self$link)
  }
  if( !is.character(link) ){
    link <- names(self$link)[link]
  }

  if(data_only){
    tmp <-
      self$alignment %>%
      as.data.frame() %>%
      dplyr::right_join(as.data.frame(self$alignment_data))
  }else{
    tmp <-
      self$alignment %>%
      as.data.frame() %>%
      dplyr::left_join(as.data.frame(self$alignment_data))
  }
  tmp <-
    tmp %>%
    dplyr::rename(
      var_name = name,
      var_value = val
    ) %>%
      dplyr::left_join(as.data.frame(self$link)) %>%
      dplyr::rename(
        text_from = from,
```

```
        text_to = to
      )

  for( i in seq_along(unique(tmp$text_from)) ){
    tf    <- unique(tmp$text_from)[i]
    iffer <- tmp$text_from == tf
    tmp[iffer, "token_1"] <-
      self$text[[tf]]$text_get() %>%
      stringb::text_sub(tmp$from_1[iffer],tmp$to_1[iffer])
  }
  for( i in seq_along(unique(tmp$text_to)) ){
    tf    <- unique(tmp$text_to)[i]
    iffer <- tmp$text_to == tf
    tmp[iffer, "token_2"] <-
      self$text[[tf]]$text_get() %>%
      stringb::text_sub(tmp$from_2[iffer],tmp$to_2[iffer])
  }

  if( !("token_2" %in% names(tmp)) ){
    tmp$token_1 <- rep(NA, nrow(tmp))
    tmp$token_2 <- rep(NA, nrow(tmp))
  }

  tmp <-
    dplyr::select(tmp, link, alignment_i, type, distance, alignment_i:token_2)

  # return
  return(tmp)
 }

 ) # closes public
)# closes R6Class
```

dp base.R

```
#' class for dp_base
#'
#' @docType class
#'
#' @name dp_base
#'
#' @export
#'
#' @keywords data
#'
#' @return Object of \code{\link{dp_base}}
#'
#' @format \code{\link{R6Class}} object.
#'
#' @seealso \code{\link{diffrproject}}
#'
dp_base <-
  R6::R6Class(

    #### class name ============================================================
    classname    = "dp_base",

    #### misc ==================================================================
    active       = NULL,
    inherit      = rtext::R6_rtext_extended,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('diffrprojects'),

    #### private ===============================================================
    private = list(),
```

```
#### public ======================================================================
public = list(

  #### data ========================================================================
  meta           =
    list(
      ts_created   = "",
      db_path      = "",
      file_path    = "",
      project_id   = ""
    ),
  alignment      = structure(list(), class=c("alignment_list","list")),
  alignment_data = structure(list(), class=c("alignment_data_list","list")),
  text           = list(),
  link           = structure(list(), class=c("alignment_list","list")),

  #### methods =====================================================================

  #### [ initialize() ] ============================================================

  initialize =
    function(
      project_id = digest::digest( list(sessionInfo(), Sys.time()) ) ,
      ask          = TRUE,
      ts_created   = force(as.POSIXct(as.numeric(Sys.time()), origin = "1970-01-01", tz="UTC")),
      db_path      = "./diffrproject.db"
    ){
    self$options$ask       <- ask
    self$meta$project_id   <- project_id
    self$meta$ts_created   <- ts_created
    self$meta$db_path      <- db_path
  },

  #### [ add text() ] ==============================================================

  text_add = function(text=NULL, text_file=NULL, rtext=NULL, name=NULL, ...){

    # case: < rtext >
    if( !is.null(rtext) ){
      text_add_worker(
        self,
        rtext,
        name = name
      )
      # return
      return(invisible(self))
    }

    # case: < text >
    if( !is.null(text) ){
      stopifnot(class(text) %in% c("character", "list"))
      if( is.null(text_file) ){
        for(i in seq_along(text) ){
```

```
        text_add_worker(
          self,
          rtext=rtext::rtext$new(text = text[[i]], ..., verbos=self$options$verbose),
          name = name[i]
        )
      }
    }else{
      for(i in seq_along(text) ){
        text_add_worker(
          self,
          rtext=rtext::rtext$new(text = text[[i]], text_file = text_file[i], ..., verbos=self$opti
          name = name[i]
        )
      }
    }
    # return
    return(invisible(self))
  }

  # case: < text_file >
  if( !is.null(text_file) ){
    for(i in seq_along(text_file) ){
      text_add_worker(
        self,
        rtext::rtext$new(text_file = text_file[i], ...),
        name = ifelse(is.null(name), basename(text_file[i]), name[i])
      )
    }
    # return
    return(invisible(self))
  }

  # case: < nothing added >
  warning("no file added")
  # return
  return(invisible(self))
},

#### [ text_delete() ] ====================================================

text_delete = function(name=NULL, id=NULL){
  if( is.null(name) & is.null(id) ){
    name <- length(self$text)
    self$text[[name]] <- NULL
  }else if( !is.null(id) & is.null(name) ){
      name <- vapply(self$text, `[[`, "", "id")==id
      self$text[name] <- NULL
  }else{
    self$text[[name]] <- NULL
  }
  # return self for piping
  return(invisible(self))
},
```

```
#### [ text_meta_data() ] ============================================================

text_meta_data = function(){
  dp_text_base_data(self)
},

#### [ text_data() ] ==================================================
text_data = function(text=NULL, var){
  tmp <- list()
  if( is.null(text) ){
    is <- seq_along(self$text)
  }else{
    is <- text
  }
  for(i in is){
    tmp[[i]] <- self$text[[i]]$char_data_get()
    tmp[[i]]$name <- names(self$text)[i]
  }
  tmp <- do.call(rbind_fill, tmp)
  return(tmp)
},

#### [ tokenize_text_data_lines() ] ===================================
tokenize_text_data_lines = function(
  text               = NULL,
  join               = c("full", "left", "right", ""),
  aggregate_function = NULL
){
  tmp <- list()
  if( is.null(text) ){
    is <- seq_along(self$text)
  }else{
    is <- text
  }
  for(i in is){
    tmp[[i]] <- self$text[[i]]$tokenize_data_lines()
    tmp[[i]]$name <- names(self$text)[i]
  }
  tmp <- do.call(rbind_fill, tmp)
  return(tmp)
},

#### [ tokenize_text_data_words() ] ===================================
tokenize_text_data_words = function(
  text        = NULL,
  join        = c("full", "left", "right", ""),
  aggregate_function = NULL
){
  tmp <- list()
  if( is.null(text) ){
    is <- seq_along(self$text)
  }else{
    is <- text
  }
```

```
  for(i in is){
    tmp[[i]] <-
      self$text[[i]]$
        tokenize_data_words(
          join               = join,
          aggregate_function = aggregate_function
        )
    tmp[[i]]$name <- names(self$text)[i]
  }
  tmp <- do.call(rbind_fill, tmp)
  return(tmp)
},

#### [ tokenize_text_data_regex() ] ======================================
tokenize_text_data_regex = function(
  split       = NULL,
  ignore.case = FALSE,
  fixed       = FALSE,
  perl        = FALSE,
  useBytes    = FALSE,
  non_token   = FALSE,
  join        = c("full", "left", "right", ""),
  aggregate_function = NULL
){
  tmp <- list()
  if( is.null(text) ){
    is <- seq_along(self$text)
  }else{
    is <- text
  }
  for(i in is){
    tmp[[i]] <-
      self$text[[i]]$
      tokenize_data_regex(
        split       = NULL,
        ignore.case = FALSE,
        fixed       = FALSE,
        perl        = FALSE,
        useBytes    = FALSE,
        non_token   = FALSE,
        join        = c("full", "left", "right", ""),
        aggregate_function = NULL
      )
    tmp[[i]]$name <- names(self$text)[i]
  }
  tmp <- do.call(rbind_fill, tmp)
  return(tmp)
},

#### [ text_code() ] =====================================================

text_code = function(text=NULL, x=NULL, i=NULL, val=NA, hl = 0){
  if( is.null(text) ){
    warning("no text selected, so I will code nothing")
```

```
    }else{
      text <- self$text[[text]]
      text$char_data_set( x=x, i=i, val=val, hl=0)
    }
    return(invisible(self))
  },

  #### [ text_code_regex() ] ============================================

  text_code_regex = function(text=NULL, x=NULL, pattern=NULL, val=NA, hl=0, ...){
    if( is.null(text) ){
      warning("no text selected, so I will code nothing")
    }else{
      text <- self$text[[text]]
      text$char_data_set_regex(x=x, pattern=pattern, val=val, hl=hl, ...)
    }
    return(invisible(self))
  },

  #### [ text_link ] ====================================================

  text_link = function(from=NULL, to=NULL, delete=FALSE){
    if( is.null(from) & is.null(to) ){
      from <- shift(names(self$text), 1, NULL)
      to   <- shift(names(self$text), -1, NULL)
    }
    from <- names(self$text[from])
    to   <- names(self$text[to])
    linker <- function(from, to, delete){
      name <- text_c(from, "~", to)
      if(delete){
        self$link[name] <- NULL
      }else{
        self$link[[name]] <- list(from=from, to=to)
      }
    }
    mapply(linker, from, to, delete=delete)
    invisible(self)
  }

 )# closes public
)# closes R6Class
```

dp export.R

```
#' R6 class - linking text and data
#'
#' @docType class
#' @name dp_export
#' @export
#' @keywords data
#' @return Object of \code{\link{R6Class}}
```

```
#' @format \code{\link{R6Class}} object.
#' @seealso \code{\link{diffrproject}}
#'
dp_export <-
  R6::R6Class(

    #### misc ============================================================================
    classname    = "dp_export",
    active       = NULL,
    inherit      = dp_loadsave,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('diffrprojects'),

    #### private =========================================================================
    private = list(),

    #### public ==========================================================================
    public = list(

      #### [ export_csv ] #### ..........................................................
      export_csv = function(folder_name = ""){
        stopifnot(file.info(folder_name)$isdir)
        "TBD"
      },

      #### [ import_csv ] #### ..........................................................
      import_csv = function(folder_name = ""){
        stopifnot(file.info(folder_name)$isdir)
        "TBD"
      },

      #### [ export_sqlite ] #### ......................................................
      export_sqlite = function(db_name = ""){
        # establish connection
        if( is.character(db_name) ){
          if( db_name != ""){
            con <- RSQLite::dbConnect( RSQLite::SQLite(), db_name)
          }else{
            con <- RSQLite::dbConnect( RSQLite::SQLite(), self$meta$db_path)
          }
          on.exit({
            RSQLite::dbDisconnect(con)
          })
        }else{
          con <- db_name
        }

        # preapare data to be exportd
        tb_exported <- private$prepare_save()
```

```
# export data
RSQLite::dbBegin(con)

  # meta
  meta <- as.data.frame(tb_exported$meta)
  rownames(meta) <- NULL
  RSQLite::dbWriteTable(con, "meta", meta, overwrite=TRUE)

  # link
  link <- as.data.frame(tb_exported$link)
  rownames(tb_exported$link) <- NULL
  RSQLite::dbWriteTable(con, "link", link, overwrite=TRUE)

  # alignment
  alignment <- as.data.frame(tb_exported$alignment)
  rownames(alignment) <- NULL
  RSQLite::dbWriteTable(con, "alignment", alignment, overwrite=TRUE)

  # alignment data
  alignment_data <- as.data.frame(tb_exported$alignment_data)
  rownames(alignment_data) <- NULL
  RSQLite::dbWriteTable(con, "alignment_data", alignment_data, overwrite=TRUE)

  # hashes
  RSQLite::dbWriteTable(con, "hashes",    tb_exported$hashes, overwrite=TRUE)

  # text_meta
  text_meta <-
    cbind(
      do.call(
        rbind,
          lapply(tb_exported$text, function(x){x$meta} )
      ),
      text_name = names(self$text)
    )
  rownames(text_meta) <- NULL
  RSQLite::dbWriteTable(con, "text_meta", text_meta, overwrite=TRUE)

  # text_char
  char <- lapply(tb_exported$text, function(x){ data.frame(char=x$char, i=seq_along(x$char) )} )
  write_numerous_parts_to_table(
    x          = char ,
    con        = con,
    table_name = "text_char",
    meta =
      data.frame(
        text_name = names(tb_exported$text),
        text_id = vapply(tb_exported$text, function(x){x$meta$id}, "")
      )
  )

  # text char_data
  char_data <- lapply( tb_exported$text, function(x){x$char_data})
  text_names <- names(char_data)
```

```
    for( i in seq_along(char_data) ){
      for( k in seq_along(char_data[[i]]) ){
        char_data[[i]][[k]]$variable <- names(char_data[[i]][[k]])[3]
        names(char_data[[i]][[k]])[3] <- "value"
        char_data[[i]][[k]]$text_name <- text_names[i]
        char_data[[i]][[k]]$text_id <- tb_exported$text[[i]]$meta$id
      }
    }
    char_data <- unlist(char_data, recursive = FALSE)
    write_numerous_parts_to_table(
      x = char_data,
      con = con,
      table_name = "text_char_data"
    )

  RSQLite::dbCommit(con)

  # return
  return(invisible(self))
},

#### [ import_sqlite ] #### ..................................................
import_sqlite = function(db_path = ""){
  # establish connection
  if( is.character(db_path) ){
    if( db_path == "" ){
      db_path <- self$meta$db_path
    }
    con <- RSQLite::dbConnect(RSQLite::SQLite(), db_path)
    on.exit({
      RSQLite::dbDisconnect(con)
    })
  }else{
    con <- db_path
  }
  # import data
  imported <- list()

  imported$meta           <- RSQLite::dbReadTable(con, "meta")

  imported$alignment      <- RSQLite::dbReadTable(con, "alignment")
  imported$alignment      <- split(imported$alignment, f=imported$alignment$link)
  imported$alignment      <- lapply(imported$alignment, subset, select=-link)

  imported$alignment_data <- RSQLite::dbReadTable(con, "alignment_data")
  imported$alignment_data <- split(imported$alignment_data, imported$alignment_data$link)
  imported$alignment_data <- lapply(imported$alignment_data, subset, select=-link)
  imported$alignment_data <- lapply(imported$alignment_data, function(x){split(x, x[[3]])})
  for( i in seq_along(imported$alignment_data) ) {
    for( k in seq_along(imported$alignment_data[[i]]) ){
      names(imported$alignment_data[[i]][[k]])[4] <- imported$alignment_data[[i]][[k]]$name[1]
      imported$alignment_data[[i]][[k]][[3]] <- NULL
    }
```

```
      }
      class(imported$alignment_data) <- c("alignment_data_list", "list")

      # import char
      imported$text_char      <- RSQLite::dbReadTable(con, "text_char")
      imported$text_char      <- split(imported$text_char, f=imported$text_char$text_name)
      imported$text_char      <- lapply(imported$text_char, subset, select=char, drop=TRUE)

      imported$text_meta      <- RSQLite::dbReadTable(con, "text_meta")

      # import char_data
      if( RSQLite::dbExistsTable(con, "text_char_data") ){
        imported$text_char_data <- RSQLite::dbReadTable(con, "text_char_data")
        imported$text_char_data <- split(imported$text_char_data, f=imported$text_char_data$text_name
        imported$text_char_data <- lapply( imported$text_char_data, function(x){ split(x, f=x$variabl

        for( i in seq_along(imported$text_char_data) ) {
          for( k in seq_along(imported$text_char_data[[i]]) ){
            nam <- imported$text_char_data[[i]][[k]]$variable[1]
            imported$text_char_data[[i]][[k]][["text_name"]] <- NULL
            imported$text_char_data[[i]][[k]][["text_id"]] <- NULL
            names(imported$text_char_data[[i]][[k]])[3] <- nam
            imported$text_char_data[[i]][[k]][["variable"]] <- NULL
          }
        }
      }else{
        imported$text_char_data <- list()
      }

      text_names <- names(imported$text_char_data)
      text_meta  <-
        lapply(split(imported$text_meta, seq_len(dim(imported$text_meta)[1]) ), as.list)
      names(text_meta) <- text_names

      for(i in seq_along(text_names)){
        imported$text[[text_names[i]]]$char      <- imported$text_char[[text_names[i]]]
        imported$text[[text_names[i]]]$char_data <- imported$text_char_data[[text_names[i]]]
        imported$text[[text_names[i]]]$meta      <- text_meta[[text_names[i]]]
      }

      # incorporate data
      private$execute_load(imported)

      # return self for piping
      return(invisible(self))
    }
  )
)
```

dp inherit.R

```
#' class for dp_inherit
#'
#' @docType class
#'
#' @name dp_inherit
#'
#' @export
#'
#' @keywords data
#'
#' @return Object of \code{\link{dp_align}}
#'
#' @format \code{\link{R6Class}} object.
#'
#' @seealso \code{\link{diffrproject}}
#'
dp_inherit <-
  R6::R6Class(

    #### class name ============================================================
    classname    = "dp_inherit",

    #### misc ==================================================================
    active       = NULL,
    inherit      = dp_align,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('diffrprojects'),

    #### public ================================================================
    public = list(

      #### data ================================================================

      #### methods =============================================================

      #### [ text_data_inherit ] #### ......................................

      text_data_inherit = function(
        link=NULL,
```

```
  direction = c("both", "forward", "backward")
){

  # checking inputs and setting defaults
  direction <- direction[1]
  if( is.null(link) ){
    link <- names(self$link)
  }

  # fetching link name if necessary
  if( !is.character(link) ){
    link <- names(self$link)[link]
  }

  # cycling through links
  links      <- self$link[link]

  # getting directions right
  if( direction == "forward" ){
    directions <- rep(direction, length(links))
  }

  if( direction == "backward" ){
    links      <- rev(links)
    directions <- rep(direction, length(links))
  }

  if( direction == "both" ){
    directions <- c(rep("forward", length(links)), rep("backward", length(links)))
    links <- c(links, rev(links))
  }

  # cycling through link list
  for( link_i in seq_along(links) ){
    # current link
    current_link_name <- names(links)[link_i]
    current_link      <- links[[link_i]]

    # current direction
    direction <- directions[link_i]

    # gathering zero distance alignments
    text1_tokens <-
      self$alignment[[current_link_name]] %>%
        subset(subset=distance==0) %>%
        subset(select=c(from_1, to_1))

    text2_tokens <-
      self$alignment[[current_link_name]] %>%
      subset(subset=distance==0) %>%
      subset(select=c(from_2, to_2))

    # getting direction right
    if( direction == "backward" ){
```

```
          text1 <- self$text[[current_link$to]]
          text2 <- self$text[[current_link$from]]
          tmp <- text1_tokens
          text1_tokens <- text2_tokens
          text1_tokens <- tmp
        }else{
          text1 <- self$text[[current_link$from]]
          text2 <- self$text[[current_link$to]]
        }

        # pushing data from one text to the other
        for(i in seq_len(nrow(text1_tokens)) ){
          push_text_char_data(
            from_text  = text1,
            to_text    = text2,
            from_token = text1_tokens[i, ],
            to_token   = text2_tokens[i, ],
            warn       = self$options$warning
          )
        }
      }
      return(invisible(self))
    } # end of text_data_inherit()
  ) # closes public
)# closes R6Class
```

dp loadsave.R

```
#' class for dp_base
#'
#' @docType class
#' @name dp_loadsave
#' @export
#' @keywords data
#' @return Object of \code{\link{dp_loadsave}}
#' @format \code{\link{R6Class}} object.
#' @seealso \code{\link{diffrproject}}
#'
#'
dp_loadsave <-
  R6::R6Class(

    #### misc ========================================================================
    classname    = "rtext_loadsave",
    active       = NULL,
    inherit      = dp_base,
    lock_objects = TRUE,
    class        = TRUE,
    portable     = TRUE,
    lock_class   = FALSE,
    cloneable    = TRUE,
    parent_env   = asNamespace('diffrprojects'),

    #### private =====================================================================
    private = list(

      #### [ prepare_save ] #### ..................................................
      prepare_save = function(){

        # meta
        meta <-
          list(
            project_id   = self$meta$project_id,
            db_path      = self$meta$db_path,
            file_path    = self$meta$file_path,
            ts_created   = self$meta$ts_created,
            n_texts      = length(self$text),
            nchar_text   = sum(unlist(lapply(self$text, function(x){x$char_length()}))),
            dp_version   = as.character(packageVersion("diffrprojects")),
```

```
      rtext_version= as.character(packageVersion("rtext")),
      save_format_version = 1
    )

  # texts
  text <- lapply( self$text, function(x){ get_private(x)$prepare_save() })

  # alignment
  alignment <- self$alignment

  # alignment_data
  alignment_data <- self$alignment_data

  # link
  link <- self$link

  # put together information
  tb_saved <-
    list(
      meta = meta ,
      hashes       =
        data.frame(
          name = names(private$hash()),
          hash = unlist(private$hash()),
          row.names=NULL
        ),
      text           = text,
      alignment      = alignment,
      alignment_data = alignment_data,
      link           = link
    )
  class(tb_saved) <- c("dp_save","list")
  # return
  return(tb_saved)
},

#### [ execute_load ] #### ..........................................
execute_load = function(tmp){

  #  meta
  self$meta$db_path      <- tmp$meta$db_path
  self$meta$file_path    <- tmp$meta$file_path
  self$meta$project_id   <- tmp$meta$project_id
  if( "numeric" %in% class(tmp$meta$ts_created)  ){
    self$meta$ts_created   <- as.POSIXct(tmp$meta$ts_created, origin = "1970-01-01", tz="UTC")
  }else if( "character" %in% class(tmp$meta$ts_created) ){
    self$meta$ts_created   <- as.POSIXct(tmp$meta$ts_created, tz="UTC")
  }else{
    self$meta$ts_created   <- tmp$meta$ts_created
  }

  # alignment
  self$alignment_data <- tmp$alignment_data
```

```
    # alignment data
    self$alignment      <- tmp$alignment

    # texts
    self$text <- list()
    text_names <- names(tmp$text)

    for(i in seq_along(text_names)){
      self$text[[text_names[i]]] <- rtext$new()
      self$text[[text_names[i]]]$get("private")$execute_load(tmp$text[[i]])
    }

    # hash update
    private$hashes <- private$hash()

    # return for piping
    invisible(self)
  }
),

#### public ==================================================================
public = list(

  #### [ save ] #### ..........................................................
  save = function(file=NULL, id=NULL){
    dp_save <- as.environment(private$prepare_save())
    # handle file option
    if( is.null(dp_save$meta$save_path) & is.null(file) ){
      if( self$options$warning ){
        warning("dp$save() : Neither file nor save_path given: storing in default location: ")
      }
      file <- "./diffrproject.RData"
    }else if( !is.null(file) ){
      file <- file
    }
    # save to file
    base::save(
      list = ls(dp_save),
      file = file,
      envir = dp_save
    )
    # return for piping
    return(invisible(self))
  },

  #### [ load ] ...............................................................
  load = function(file=NULL){
    # handle file option
    if( is.null(file) ){
      stop("dp$load() : file is not given, do not know where to load file from.")
    }else{
      file <- file
    }
    # loading info
```

```
        tmp <- rtext:::load_into(file)
        # applying loaded info to self
        private$execute_load(tmp)
        # return self for piping
        return(invisible(self))
      }
    )
  )
```

dp tools.R

```
#' function writing numerous parts of table to database
#'
#' @param x parts to be written
#' @param table_name of the table
#' @param meta additional information to be attachesd to table parts
#' @param con connection to database
#'
#' @export
#'
write_numerous_parts_to_table <- function(x, con, table_name, meta=data.frame() ){
  for( i in seq_along(x) ){
    if(i==1){
      overwrite <- TRUE
      append    <- FALSE
    }else{
      overwrite <- FALSE
      append    <- TRUE
    }
    for(k in seq_len(ncol(meta)) ){
      x[[i]][[ names(meta)[k] ]] <- meta[ i, k ]
    }
    RSQLite::dbWriteTable(
      con,
      table_name,
      x[[i]],
      overwrite=overwrite,
      append=append
    )
  }
}

#' function adding rtext objects to diffrprojects
#' @param self an object of class dp
#' @param rtext an object of class rtext
#' @param name an optional name for the text to stick to the text within the
#'        diffrproject corpus - if none is supplied the function will try to
#'        infere a reasonable name from the rtext$text_file field, if that is
#'        not given it will get the name noname_x where x is a running integer
#' @keywords internal
text_add_worker = function(self, rtext=NULL, name = NULL ){
  # input check
```

```
  stopifnot( "rtext"  %in% class(rtext) )
  # working variable creation
  names <- names(self$text)
  ids   <- vapply(self$text, `[[`, "", "id")
  id    <- rtext$id
  # doing-duty-to-do
  if( is.null(name) ){
    name <-
      tryCatch(
        basename(rtext$text_file), error=function(e){NA}
      )
    if( is.na(name) ){
      next_num <- max(c(as.numeric(text_extract(names, "\\d+")),0))+1
      name     <- text_c( "noname_", next_num)
    }
  }
  self$text[[name]]    <- rtext
  i <- 0
  while( rtext$id %in% ids ){
    rtext$id <- text_c(id, "_", i)
    i <- i+1
  }
}


#' function providing basic information on texts within diffrproject
#' @param dp a diffrproject object
#' @export
dp_text_base_data <- function(dp){
  df <- data.frame(NA)
  rt <- rtext$new("", verbose=FALSE)$info()
  names <- names(rt)
  for(i in seq_along(names) ){
    df[seq_along(dp$text), names[i]] <- NA
  }
  df <- df[,-1]
  for( i in seq_along(dp$text) ){
    df[i,] <- get("info", dp$text[[i]])()
  }
  df$name <- names(dp$text)
  if( all(is.na(df)) ){
    df <- subset(df, FALSE)
  }
  return(df)
}




#' as.data.frame method for for named lists of data.frames
#' @inheritParams base::as.data.frame
#' @param dfnamevar in which variable should list item names be saved
#' @method as.data.frame named_df_list
#' @export
```

```
as.data.frame.named_df_list <- function(x, row.names=NULL, optional=FALSE, dfnamevar="name", ...){
  if( any(unlist(lapply(x, class)) == "list") ){
    x <- lapply(x, as.data.frame.named_df_list)
  }
  # prepare variable
  each <- unlist(lapply(x, dim1))
  var <- names(x)
  var <- unlist(mapply(rep, var, each, SIMPLIFY=FALSE))
  # doing-duty-to-do
  if( class(x[[1]])!="data.frame" ){
    x<-as.data.frame(x)
  }else{
    names(x) <- NULL
    x <- do.call(rbind_fill, x)
    # add link variable
    x[[dfnamevar]] <- var
  }
  # return
  return(x)
}

#' as.data.frame method for for named lists of data.frames
#' @inheritParams as.data.frame.named_df_list
#' @method as.data.frame alignment_list
#' @export
as.data.frame.alignment_list <- function(x, row.names=NULL, optional=FALSE, ...){
  as.data.frame.named_df_list(
    x,
    row.names = row.names,
    optional = optional,
    dfnamevar = "link",
    ...
  )
}



#' as.data.frame method for for named lists of data.frames
#' @inheritParams as.data.frame.named_df_list
#' @method as.data.frame alignment_data_list
#' @export
as.data.frame.alignment_data_list <- function(x, row.names=NULL, optional=FALSE, ...){
  if(length(x) > 0 ){
    tmp <- as.data.frame.named_df_list(
      x,
      row.names = row.names,
      optional = optional,
      dfnamevar = "link",
      ...
    )
  }else{
    tmp <-
      data.frame("",1,1,"") %>%
      dplyr::filter(FALSE) %>%
```

```
      stats::setNames(c("link", "alignment_i", "hl", "name"))
  }
  cols <- which(names(tmp) %in% c("link", "alignment_i", "hl", "name"))
  val <- subset( tmp, select = -c(cols) )
  tmp <- subset( tmp, select = cols )
  tmp$val <- unlist(apply(val, 1, function(x){ x[!is.na(x)][1] } ))
  tmp
}


#' push char_data of one rtext objet to another
#'
#' Function that takes a rtext object pulls specific char_data from it and
#' pushes this information to another rtext object.
#'
#' Note, that this is an intelligent function.
#'
#' It will e.g. always decrease the hierarchy level (hl) found when pulling and
#' decrease it before pushing it forward therewith allowing that already present
#' coding might take priority over those pushed.
#'
#' Furthermore, the function will only push values if the pulled values are all
#' the same. Since, character index lengths that are used for pulling and
#' pushing might differ in length there is no straight forward rule to translate
#' non uniform value sequences in value sequnces of differing length. Note, that
#' of cause the values might differ between char_data variables but not within.
#' In case of non-uniformity the function will simply do nothing.
#'
#'
#' @param from_text text to pull data from
#' @param to_text text to push data to
#' @param from_token token of text to pull data from
#'        (e.g.: data.frame(from=1, to=4))
#' @param to_token token of text to push data to
#'        (e.g.: data.frame(from=1, to=4))
#' @param from_i index of characters to pull data from
#' @param to_i index of characters to push data to
#' @param x name of the char_data variable to pull and push -
#'        defaults to NULL which will result in cycling through all availible
#'        variables
#' @param warn should function warn about non-uniform pull values (those will
#'        not be pushed to the other text)
#'
#' @return NULL
#' @export

push_text_char_data <-
  function(
    from_text  = NULL,
    to_text    = NULL,
    from_token = NULL,
    to_token   = NULL,
    from_i     = NULL,
    to_i       = NULL,
```

```
  x          = NULL,
  warn       = TRUE
){
  # check input
  stopifnot(
    !is.null(from_text), !is.null(to_text),
    !is.null(from_token) | is.null(from_i),
    !is.null(to_token) | is.null(to_i)
  )
  # prepare variables
  if( is.null(from_i) ){
    from_i <- stringb:::sequenize(from_token)
  }
  if( is.null(to_i) ){
    to_i <- stringb:::sequenize(to_token)
  }

  char_data <- from_text$get("char_data")

  if( is.null(x) ){
    from_names <- names(char_data)
  }else{
    from_names <- names(char_data)[names(char_data) %in% x]
  }

  # push data
  for(i in seq_along(from_names) ){
    iffer <- char_data[[from_names[i]]]$i %in% from_i
    name  <- from_names[i]
    value <-
      subset(
        char_data[[from_names[i]]],
        subset = iffer,
        select = name
      ) %>%
      unlist()

    if( length(unique(value)) ==1 ){
      to_text$char_data_set(
        x   = from_names[i],
        i   = to_i,
        val = value[1],
        hl  = min(char_data[[from_names[i]]]$hl)-1
      )
    }else if( warn & length(unique(value)) > 1 ){
        warning("push_text_char_data() pulled non uniform values, nothing pushed")
    }
  }
  # return
  return(invisible(NULL))
}
```

```
#' transform rtext text data into a data.frame
#' @param x rtext object
#' @keywords internal
rtext_char_data_to_data_frame <- function(x){
  cd <- x$get("char_data")
  for( i in seq_along(cd) ){
    names(cd[[i]])[3] <- "value"
    cd[[i]]$value <- as.character(cd[[i]]$value)
    cd[[i]]$variable <- names(cd)[i]
  }
  df <- rbind_list(cd)
  return(df)
}

#' transform alignment_data list into data.frame
#' @param x alignment_data list
#' @keywords internal
alignment_data_to_data_frame <- function(x){
  for( i in seq_along(x) ){
    names(x[[i]])[3] <- "value"
    x[[i]]$value <- as.character(x[[i]]$value)
    x[[i]]$variable <- names(x)[i]
  }
  df <- rbind_list(x)
  return(df)
}




#' function sorting alignment data according to token index
#'
#' @param x data.frame to be sorted
#' @param ti1 either NULL (default): first column of x is used as first token
#'        index for sorting; a character vector specifying the column to be used
#'        as first token index; or a numeric vector of length nrow(x) to be use
#'        as first token index
#' @param ti2 either NULL (default): second column of x is used as second token
#'        index for sorting; a character vector specifying the column to be used
#'        as second token index; or a numeric vector of length nrow(x) to be use
#'        as second token index
#' @param first should first text or second text be given priority
#'
#' @export
#'
sort_alignment <- function(x, ti1 = NULL, ti2 = NULL, first = TRUE){
  # processing input
  if( is.null(ti1) ){
    ti1 <- x[,1]
  }else if( is.numeric(ti1) ){
    ti1 <- ti1
  }else if( is.character(ti1) ){
    ti1 <- x[, ti1]
  }
```

```
  if( is.null(ti2) ){
    ti2 <- x[,2]
  }else if( is.numeric(ti2) ){
    ti2 <- ti2
  }else if( is.character(ti2) ){
    ti2 <- x[, ti2]
  }

  # preparing loop
  if ( first == T  ){
    var1 <- ti1
    var2 <- ti2
  }else{
    var1 <- ti2
    var2 <- ti1
  }

  looper   <- seq_len(max(ti1, ti2, na.rm=T))
  data_nr  <- seq_along(x[,1])
  sorter <- NULL

  # loop
  for ( i in looper ){
    sorter <- c(  sorter                                                ,
                  data_nr[ i==var1 & !is.na(var1) &   is.na(var2) ] ,
                  data_nr[ i==var1 & !is.na(var1) &  !is.na(var2) ] ,
                  data_nr[ i==var2 &  is.na(var1) &  !is.na(var2) ] )
  }
  # return
  return(x[sorter,])
}
```

imports.r

```
#' imports
#' @importFrom R6 R6Class
#' @import hellno
#' @import stringb
#' @import rtext
#' @useDynLib diffrprojects
dummyimport <- function(){
  R6::R6Class()
  1 %>% magrittr::add(1)
}

#' @importFrom magrittr %>%
#' @export
magrittr::`%>%`
```

methods of comparison.R

```
#' method of comparison
#' @export
```

moc.R

```
#' if(getRversion() >= "2.15.1"){
#'   utils::globalVariables(
#'     c(
#'       "text1_tokenized", "text2_tokenized", "token_i"
#'     )
#'   )
#' }
#'
#' #' stub
#' #' @keywords internal
#' moc <- function(
#'   text1     = NULL,
#'   text2     = NULL,
#'   tokenizer = function(text){text_tokenize_lines(text)},
#'   ignore    = function(...){FALSE},
#'   clean     = function(token){token},
#'   distance  = function(token1, token2){matrix(0,nrow = length(token1), ncol = length(token2))},
#'   alignment = function(m){}
#' ){
```

```
#'   # alignment and distances
#'
#'   #### trivial matches -- unique equal token matches
#'   message(" - trivial matching")
#'   res <-
#'     moc_helper_trivial_matches( tt1 = text1_tokenized, tt2 = text2_tokenized )
#'
#'
#'   #### easy matches -- text1 non-unique equal token matches
#'   message(" - easy matching 1")
#'   res <-
#'     rbind(
#'       res,
#'       moc_helper_easy_matches( tt1 = text1_tokenized, tt2 = text2_tokenized, res= res, type=1)
#'     )
#'
#'
#'   #### easy matches -- text2 non-unique equal token matches
#'   message(" - easy matching 2")
#'   res <-
#'     rbind(
#'       res,
#'       moc_helper_easy_matches( tt1 = text1_tokenized, tt2 = text2_tokenized, res= res, type=2)
#'     )
#'
#'   #### easy matches -- text2 non-unique equal token matches
#'   message(" - easy matching 3")
#'
#'   # prepare tt1 and tt2 as lists of data.frames
#'   tt1 <-
#'     text1_tokenized %>%
#'     dplyr::filter( !(token_i %in% res$token_i_1) )
#'
#'   tt2 <-
#'     text2_tokenized %>%
#'     dplyr::filter( !(token_i %in% res$token_i_2) )
#'
#'   tt1_split <- split_tt_by_length(tt1)
#'   tt2_split <- split_tt_by_length(tt2)
#'
#'   tt_names <- unique(c(names(tt1_split), names(tt2_split)))
#'
#'   # do the matches
#'   for( i in rev(seq_along(tt_names)) ) {
#'     cat(i, " ", append=TRUE)
#'     res <-
#'       moc_helper_easy_matches(
#'         tt1 = tt1_split[[tt_names[i]]],
#'         tt2 = tt2_split[[tt_names[i]]],
#'         res=res,
#'         type=3
#'       )
#'   }
#'   cat("\n")
```

```
#'
#'   # finishing matching of no-change type
#'   res$type <- "no-change"
#'   res$diff <- 0
#' }
```

moc helper.R

```
#' if(getRversion() >= "2.15.1"){
#'   utils::globalVariables(
#'     c(
#'       "token_i_1", "token_i_2",
#'       "text1_tokenized", "text2_tokenized",
#'       "token", ".", "...", "res_token_i_1", "res_token_i_2",
#'       "min_dist_1"
#'     )
#'   )
#' }
#'
#' #' splitting a tokenized text
#' #' @param tt tokenized text
#' #' @keywords internal
#' split_tt_by_length <- function(tt){
#'   tt %>%
#'     dplyr::mutate( token_length = nchar(token) ) %>%
#'     split( .$token_length ) %>%
#'     lapply( dplyr::mutate, token_length = NULL ) %>%
#'     lapply( as.data.table  ) %>%
#'     lapply( setkey, "token", "token_i"  )
#' }
#'
#'
#' #' trivial matches
#' #'
#' #' method of comparison helper function
#' #' @param tt1 tokenized text number 1
#' #' @param tt2 tokenized text number 2
#' #' @keywords internal
#' moc_helper_trivial_matches <- function(tt1, tt2){
#'   # preparation
#'   tt1 <- subset( tt1, is_unique(token), select=c("token", "token_i"))
#'   tt1 <- data.table::as.data.table(tt1)
#'   data.table::setkey("tt1", "token")
#'
#'   tt2 <- subset( tt2, is_unique(token), select=c("token", "token_i"))
#'   tt2 <- data.table::as.data.table(tt2)
#'   data.table::setkey("tt2", "token")
#'
#'   # merge / join
#'   matches <- suppressWarnings(dplyr::inner_join(tt1, tt2, by="token"))
#'              data.table::setkey(matches, "token_i.x", "token_i.y")
#'
#'   # clean up names
```

```
#'   names(matches) <-
#'     names(matches) %>%
#'     stringb::text_replace("\\.", "_") %>%
#'     stringb::text_replace("x", "1") %>%
#'     stringb::text_replace("y", "2")
#'
#'   # return
#'   return(matches)
#' }
#'
#' #' easy matches 1
#' #'
#' #' method of comparison helper function
#' #' @param tt1 tokenized text number 1
#' #' @param tt2 tokenized text number 2
#' #' @keywords internal
#' moc_helper_easy_matches <- function(tt1, tt2, res, type=c(1,2), fullreturn=TRUE){
#'   # check input
#'   if( is.null(tt1) | is.null(tt2) ){
#'     # return
#'     if( fullreturn ){
#'       return(res)
#'     }else{
#'       return(data.frame())
#'     }
#'   }
#'   # preparation
#'   tt1_tmp <-
#'     tt1 %>%
#'     subset(select = c("token", "token_i") ) %>%
#'     dplyr::filter(
#'       !(token_i %in% res$token_i_1)
#'     ) %>%
#'     as.data.table()
#'   setkey(tt1_tmp, "token_i")
#'
#'   tt2_tmp <-
#'     tt2 %>%
#'     dplyr::select(token, token_i) %>%
#'     dplyr::filter(
#'       !(token_i %in% res$token_i_2)
#'     ) %>%
#'     as.data.table()
#'   setkey(tt2_tmp, "token_i")
#'
#'   # decide which tokens (from text1 or from text2) should be unique
#'   if( type == 1){
#'     tt1_tmp <- tt1_tmp %>%  dplyr::filter( is_unique(token) )
#'   }else if( type == 2){
#'     tt2_tmp <- tt2_tmp %>%  dplyr::filter( is_unique(token) )
#'   }
#'
#'   # get and order possible matches
#'   matches <-
```

```
#'     suppressWarnings(
#'       moc_helper_get_options_ordered_by_dist(tt1_tmp, tt2_tmp, res)
#'     )
#'
#'   # process optional matches
#'   chosen <-
#'     choose_options(matches$token_i_1, matches$token_i_2, res$token_i_1, res$token_i_2) %>%
#'     as.data.table() %>%
#'     setkey("token_i_1")
#'
#'   # add token to get it rbind-ed to res
#'   tt1_tmp <- stats::setNames(tt1_tmp, c("token", "token_i_1"))
#'   chosen <- dplyr::left_join(chosen, tt1_tmp, by="token_i_1")
#'
#'   # return
#'   if( fullreturn ){
#'     return( rbind(res, data.table(chosen), fill=TRUE) )
#'   }else{
#'     return(chosen)
#'   }
#' }
#'
#'
#' #' get options for machtches
#' #'
#' #' method of comparison helper function
#' #' @param tt1 tokenized text number 1
#' #' @param tt2 tokenized text number 2
#' #' @param res data.frame of already matched
#' #' @import data.table
#' #' @keywords internal
#' moc_helper_get_options_ordered_by_dist <- function(tt1, tt2, res){
#'   # distance between availible token positions and positions of tokens already matched
#'   dist           <- which_dist_min_absolute(tt1$token_i, res$token_i_1)
#'   tt1$min_dist_1 <- dist$minimum
#'   # preapare information from res
#'   res_tmp <-
#'     res[dist$location, ] %>%
#'     dplyr::select(token_i_1, token_i_2) %>%
#'     stats::setNames( paste0("res_",names(.)) )
#'   # combine res with info from tt1
#'   tt1_tmp <-
#'     tt1 %>%
#'     dplyr::select(token, token_i, min_dist_1) %>%
#'     cbind(res_tmp)
#'   # join tt1 and tt2
#'   tt2_tmp <- dplyr::select(tt2, token, token_i)
#'   tt1_tmp <-
#'     tt1_tmp %>%
#'     dplyr::inner_join(tt2_tmp, by="token")
#'   names(tt1_tmp)[names(tt1_tmp)=="token_i.x"] <- "token_i_1"
#'   names(tt1_tmp)[names(tt1_tmp)=="token_i.y"] <- "token_i_2"
#'   tt1_tmp <- data.table::as.data.table(tt1_tmp)
#'   # delete columns
```

```
#'   tt1_tmp[, token := NULL]
#'   tt1_tmp[, res_token_i_1 := NULL]
#'   # add token_i_2 position distance
#'   tt1_tmp$min_dist_2 <- 0L
#'   tt1_tmp$min_dist_2 <- abs(tt1_tmp$res_token_i_2 - tt1_tmp$token_i_2)
#'   # delete columns
#'   tt1_tmp[, res_token_i_2 := NULL]
#'   # sort
#'   data.table::setorder(tt1_tmp, "min_dist_1", "min_dist_2", "token_i_1", "token_i_2")
#'   # delete columns
#'   tt1_tmp[, "min_dist_1" := NULL]
#'   tt1_tmp[, "min_dist_2" := NULL]
#'   # return
#'   return(tt1_tmp)
#' }
```

RcppExports.R

```
# Generated by using Rcpp::compileAttributes() -> do not edit by hand
# Generator token: 10BE3573-1514-4C36-9D1C-5A225CD40393

#' (choose from a number of pre-sorted options)
#' takes a vector pair of toki1 / toki2 and a vector pair of res_token_i_1 /
#' res_token_i_2 and chooses so that each 1st and exh 2nd value only is used
#' where res_token_i_x identiefies already used items.
#' @param toki1 first number of number pair to choose from
#' @param toki2 second number of number pair to choose from
#' @param res_token_i_1 already used first numbers
#' @param res_token_i_2 already used second numbers
#' // @keywords internal
choose_options <- function(toki1, toki2, res_token_i_1, res_token_i_2) {
    .Call('diffrprojects_choose_options', PACKAGE = 'diffrprojects', toki1, toki2, res_token_i_1, res_t
}

#' (function to calculate distance matrix of integers)
#' takes vector of size n and vector of size m and gives back matrix of n rows and m columns
#' @param x a vector of type numeric
#' @param y a vector of type numeric
#' @keywords internal
dist_mat_absolute <- function(x, y) {
    .Call('diffrprojects_dist_mat_absolute', PACKAGE = 'diffrprojects', x, y)
}

#' (function to calculate minimum and position of minimum)
#' takes vector of size n and vector of size m and gives back list with
#' vectors of size n (minimum distance and location of minimum in y)
#' @param x a vector of type integer
#' @param y a vector of type integer
#' @keywords internal
which_dist_min_absolute <- function(x, y) {
    .Call('diffrprojects_which_dist_min_absolute', PACKAGE = 'diffrprojects', x, y)
}
```

text diff.R

```
#' function for calculating distance matrix between two texts
```

texts.R

```
#' text_version_1 a first version of a text
#' @source Source of Text: Diff. (2014, August 26). In Wikipedia, The Free Encyclopedia. Retrieved 10:14
"text_version_1"

#' text_version_2 a second version of a text
#' @source Source of Text: Diff. (2014, August 26). In Wikipedia, The Free Encyclopedia. Retrieved 10:14
"text_version_2"
```

tools.R

```
#' accessing private from R6 object
#'
#' @param x R6 object to access private from
#'
#' @source http://stackoverflow.com/a/38578080/1144966
#'
#' @export
#'
get_private <- function(x) {
  x[['.__enclos_env__']]$private
}


#' which are minima in vector
#' @param x vector to check
#' @param unique defaults to false
#' @keywords internal
is_minimum <- function(x, unique=FALSE){
  if(unique){
    return(
      min(x) == x & !duplicated(x)
    )
  }else{
    return(
      min(x) == x
    )
  }
}

#' checking if value is uniqe in set
#' @param x vector to check
#' @keywords internal
is_unique <- function(x){
  tmp <- !is_duplicate(x)
  tmp[is.na(x)] <- NA
  tmp
```

```
}

#' checking if value is duplicated in set
#' @param x vector to check
#' @keywords internal
is_duplicate <- function(x){
  x %in% x[duplicated(x)]
}

#' extract specific item from each list element
#' @param l list
#' @param item name or index of item to extract
#' @param unlist defaults to TRUE, whether to unlist results or leave as list
#' @keywords internal
get_list_item <- function(l, item, unlist=TRUE){
  tmp <-
    lapply(
      l,
      function(x, item){
        tryCatch(
          x[[item]],
          error = function(e){NULL}
        )
      },
      item
    )
  index <- vapply(tmp, is.null, TRUE)
  tmp[index] <- NA
  if( unlist ){
    return(unlist(tmp))
  }else{
    return(tmp)
  }
}



#' function rbinding list elements
#' @param l list
#' @keywords internal
rbind_list <- function(l){
  tmp <- do.call(rbind, l)
  rownames(tmp) <- NULL
  as.data.frame(tmp, stringsAsFactors = FALSE)
}


#' function that shifts vector values to right or left
#'
#' @param x Vector for which to shift values
#' @param n Number of places to be shifted.
#'    Positive numbers will shift to the right by default.
#'    Negative numbers will shift to the left by default.
#'    The direction can be inverted by the invert parameter.
```

```
#' @param default The value that should be inserted by default.
#' @param invert Whether or not the default shift directions
#'    should be inverted.
#' @keywords internal

shift <- function(x, n=0, default=NA, invert=FALSE){
  n <-
    switch (
      as.character(n),
      right    =  1,
      left     = -1,
      forward  =  1,
      backward = -1,
      lag      =  1,
      lead     = -1,
      as.numeric(n)
    )
  if( length(x) <= abs(n) ){
    if(n < 0){
      n <- -1 * length(x)
    }else{
      n <- length(x)
    }
  }
  if(n==0){
    return(x)
  }
  n <- ifelse(invert, n*(-1), n)
  if(n<0){
    n <- abs(n)
    forward=FALSE
  }else{
    forward=TRUE
  }
  if(forward){
    return(c(rep(default, n), x[seq_len(length(x)-n)]))
  }
  if(!forward){
    return(c(x[seq_len(length(x)-n)+n], rep(default, n)))
  }
}


#' function forcing value to fall between min and max
#' @param x the values to be bound
#' @param max upper boundary
#' @param min lower boundary
#' @keywords internal
bind_between <- function(x, min, max){
  x[x<min] <- min
  x[x>max] <- max
  return(x)
}
```

```
#' function for binding data.frames even if names do not match
#' @param df1 first data.frame to rbind
#' @param df2 second data.frame to rbind
#' @keywords internal
rbind_fill <- function(df1=data.frame(), df2=data.frame()){

  # get union of names
  names_df <- c(names(df1), names(df2))

  # prepare empty data.frame
  empty_frame <- data.frame(lapply(names_df, as.data.frame))
  names(empty_frame) <- names_df
  if(length(names_df)>0){
    empty_frame <- subset(empty_frame, FALSE)
  }

  # filling up
  if( dim1(df1) > 0 ){
    df1[, names_df[!(names_df %in% names(df1))]] <- rep(NA, dim1(df1))
  }else{
    df1 <- empty_frame
  }

  if( dim1(df2) > 0 ){
    df2[, names_df[!(names_df %in% names(df2))]] <- rep(NA, dim1(df2))
  }else{
    df2 <- empty_frame
  }

  # doing-duty-to-do
  rbind(df1, df2)
}




#' function that checks is values are in between values
#' @param x input vector
#' @param y lower bound
#' @param z upper bound
#' @keywords internal
is_between <- function(x,y,z){
  return(x>=y & x<=z)
}


#' function that extracts elements from vector
#'
#' @param vec the chars field
#' @param length number of elements to be returned
#' @param from first element to be returned
#' @param to last element to be returned
#' @keywords internal
```

```
get_vector_element <-
  function(vec, length=NULL, from=NULL, to=NULL){
    # helper functions
    bind_to_vecrange <- function(x){bind_between(x, 1, length(vec))}
    bind_length      <- function(x){bind_between(x, 0, length(vec))}
    return_from_to   <- function(from, to, split){
      res  <- vec[seq(from=from, to=to)]
      return(res)
    }
    # only length
    if( !is.null(length) & ( is.null(from) & is.null(to) ) ){
      length <- max(0, min(length, length(vec)))
      length <- bind_length(length)
      if(length==0){
        return("")
      }
      from   <- 1
      to     <- length
      return(return_from_to(from, to, split))
    }
    # from and to (--> ignores length argument)
    if( !is.null(from) & !is.null(to) ){
      from <- bind_to_vecrange(from)
      to   <- bind_to_vecrange(to)
      return(return_from_to(from, to, split))
    }
    # length + from
    if( !is.null(length) & !is.null(from) ){
      if( length<=0 | from + length <=0 ){
        return("")
      }
      to   <- from + length-1
      if((to < 1 & from < 1) | (to > length(vec) & from > length(vec) )){
        return("")
      }
      to   <- bind_to_vecrange(to)
      from <- bind_to_vecrange(from)
      return(return_from_to(from, to, split))
    }
    # length + to
    if( !is.null(length) & !is.null(to) ){
      if( length<=0 | to - (length-1) > length(vec) ){
        return("")
      }
      from <- to - length + 1
      if((to < 1 & from < 1) | (to > length(vec) & from > length(vec) )){
        return("")
      }
      from <- bind_to_vecrange(from)
      to   <- bind_to_vecrange(to)
      return(return_from_to(from, to, split))
    }
    stop("get_vector_element() : I do not know how to make sense of given length, from, to argument valu
  }
```

```
#' get first dimension or length of object
#' @param x object, matrix, vector, data.frame, ...
#' @keywords internal
dim1 <- function(x){
  ifelse(is.null(dim(x)[1]), length(x), dim(x)[1])
}


#' get first dimension or length of object
#' @param x object, matrix, vector, data.frame, ...
#' @keywords internal
dim2 <- function(x){
  dim(x)[2]
}


#' seq along first dimension / length
#' @param x x
#' @keywords internal
seq_dim1 <- function(x){
  seq_len(dim1(x))
}




#' function giving back the mode

#' @param x vector to get mode for
#' @param multimodal wether or not all modes should be returned in case of more than one
#' @param warn should the function warn about multimodal outcomes?
#' @keywords internal
modus <- function(x, multimodal=FALSE, warn=TRUE) {
  x_unique <- unique(x)
  tab_x    <- tabulate(match(x, x_unique))
  res      <- x_unique[which(tab_x==max(tab_x))]
  if( identical(multimodal, TRUE) ){
    return(res)
  }else{
    if( warn & length(res) > 1 ){
      warning("modus : multimodal but only one value returned (use warn=FALSE to turn this off)")
    }
    if( !identical(multimodal, FALSE) & length(res) > 1 ){
      return(multimodal)
    }else{
      return(res[1])
    }
  }
}
```

```
#' function to get classes from e.g. lists

#' @param x list to get classes for
#' @keywords internal
classes <- function(x){
  tmp <- lapply(x, class)
  data.frame(name=names(tmp), class=unlist(tmp) , row.names = NULL)
}
```

```
#' function to sort df by variables
#' @param df data.frame to be sorted
#' @param ... column names to use for sorting
#' @keywords internal
dp_arrange <- function(df, ...){
  sorters    <- as.character(as.list(match.call()))
  if( length(sorters)>2 ){
    sorters    <- sorters[-c(1:2)]
    sorters    <- paste0("df['",sorters,"']", collapse = ", ")
    order_call <- paste0("order(",sorters,")")
    res        <- df[eval(parse(text=order_call)), ]
    if( is.data.frame(df) & !is.data.frame(res) ){
      res <- as.data.frame(res)
      names(res) <- names(df)
    }
    return(res)
  }else{
    return(df)
  }
}
```

zzz.R

```
.onLoad <- function(libname, pkgname) {
  #library(stringb)
  #library(rtext)
  ##packageStartupMessage()
}
```

diffrprojectswidget

dp prepare data.R

```
if(getRversion() >= "2.15.1"){
  utils::globalVariables(
    c(
      "name", "val", "hl"
    )
  )
}

#' function for preparing data for tabulation
#'
#' @param dp an object of type diffrproject
#' @param link which link to produce table for
#' @param align_var either a character vector of variable names or TRUE for all
#' @param text_var either a character vector of variable names or TRUE for all
#' @param aggregate_function a function able to resolve conflicts if for a
#'    specific variable for a token of text severla values exist, if NULL it
#'    defaults to modus() but could also be e.g. paste or something alike
#' @param ... further arguments passed through to aggregate_function
#'
#' @export
#'
dp_prepare_data_table <-
  function(
    dp,
    link               = NULL,
    align_var          = TRUE,
    text_var           = TRUE,
    aggregate_function = NULL,
    ...
```

```
){
  # check input
  if( is.null(link) ){
    if( length(dp$link) == 1  ){
      link <- 1
    }else{
      stop("No link/alignment choosen, please specify link/alignment to render.")
    }
  }

  # get link name and text names
  if( is.numeric(link) ){
    link  <- names(dp$link)[link]
  }

  text_name_1 <- dp$link[[link]]$from
  text_name_2 <- dp$link[[link]]$to

  # prepare alignment
  alignment <-
    dp$alignment[[link]] %>%
    diffrprojects::sort_alignment(ti1 = "token_i_1", ti2 = "token_i_2")

  # prepare alignment_data
  alignment_data <-
    dp$alignment[[link]][, "alignment_i", drop=FALSE] %>%
    dplyr::left_join(
      tidyr::spread(
        diffrprojects:::as.data.frame.alignment_data_list(
          dp$alignment_data[link]
        ),
        name,
        val
      ),
      by = c("alignment_i"="alignment_i")
    ) %>%
    dplyr::select(-hl,-link, -alignment_i)

  if( any(align_var != TRUE) ){
    alignment_data <- alignment_data[, names(alignment_data) %in% align_var, drop = FALSE]
  }

  # preapare text_data
  tokens <-
    alignment %>%
    dplyr::select(from_1, to_1) %>%
    stats::setNames(c("from","to"))
  text1_data <-
    dp$text[[text_name_1]]$
    tokenize_data_sequences(
      token = tokens,
      aggregate_function = aggregate_function,
      ...
    ) %>%
```

```
      dplyr::select(-from, -to, -token_i)

    tokens <-
      alignment %>%
      dplyr::select(from_2, to_2) %>%
      stats::setNames(c("from","to"))
    text2_data <-
      dp$text[[text_name_2]]$
      tokenize_data_sequences(
        token = tokens,
        aggregate_function = aggregate_function,
        ...
      ) %>%
      dplyr::select(-from, -to, -token_i)

    if( any(text_var != TRUE) ){
      text1_data <- text1_data[, names(text1_data) %in% text_var, drop=FALSE]
      text2_data <- text2_data[, names(text2_data) %in% text_var, drop=FALSE]
    }

    # return
    return(
      list(
        alignment                = alignment,
        alignment_vars           = names(alignment),
        text1                    = dp$text[[text_name_1]]$text_get(),
        text2                    = dp$text[[text_name_2]]$text_get(),
        alignment_data           = alignment_data,
        alignment_data_vars      = names(alignment_data),
        alignment_text1_data     = text1_data,
        alignment_text2_data     = text2_data,
        alignment_text_data_vars  = names(text1_data)
      )
    )
  }

#' function for preparing data for tabulation
#'
#' @param dp an object of type diffrproject
#' @param link which link to produce table for
#' @param align_var either a character vector of variable names or TRUE for all
#' @param text_var either a character vector of variable names or TRUE for all
#' @param aggregate_function a function able to resolve conflicts if for a
#'    specific variable for a token of text severla values exist, if NULL it
#'    defaults to modus() but could also be e.g. paste or something alike
#' @param ... further arguments passed through to aggregate_function
#' @param minimize make data small and comlicated
#'
#' @export
#'
dp_prepare_data_vis <-
  function(
    dp,
    link               = NULL,
```

```
  align_var          = TRUE,
  text_var           = TRUE,
  aggregate_function = NULL,
  minimize           = FALSE,
  ...
){
  # check input
  if( is.null(link) ){
    if( length(dp$link) == 1  ){
      link <- 1
    }else{
      stop("No link/alignment choosen, please specify link/alignment to render.")
    }
  }

  # get link name and text names
  if( is.numeric(link) ){
    link  <- names(dp$link)[link]
  }

  text_name_1 <- dp$link[[link]]$from
  text_name_2 <- dp$link[[link]]$to

  # prepare alignment
  alignment <-
    dp$alignment[[link]] %>%
    diffrprojects::sort_alignment(ti1 = "token_i_1", ti2 = "token_i_2")

  # prepare alignment_data
  alignment_data <-
    dp$alignment[[link]][, "alignment_i", drop=FALSE] %>%
    dplyr::left_join(
      tidyr::spread(
        diffrprojects:::as.data.frame.alignment_data_list(
          dp$alignment_data[link]
        ),
        name,
        val
      ),
      by = c("alignment_i"="alignment_i")
    ) %>%
    dplyr::select(-hl,-link, -alignment_i)

  if( any(align_var != TRUE) ){
    alignment_data <- alignment_data[, names(alignment_data) %in% align_var, drop = FALSE]
  }

  # preapare text1_data
  text1 <-
    alignment %>%
    dplyr::select(token_i_1, from_1, to_1) %>%
    stats::setNames(c("token_i", "from","to")) %>%
    dplyr::filter(!duplicated(token_i), !is.na(token_i)) %>%
    dplyr::arrange(token_i)
```

```
# get text data by tokenizing character level data
# / aggregating it to character-span level
text1_data <-
  dp$text[[text_name_1]]$
  tokenize_data_sequences(
    token = text1[,c("from","to")],
    aggregate_function = aggregate_function,
    ...
  ) %>%
  dplyr::select(-from, -to, -token_i)

# add text to text
f <- dp$text[[text_name_1]]$text_get
text1$text <- mapply(f, from=text1$from, to=text1$to)
text1$tnr  <- 1

# preapare text2_data
text2 <-
  alignment %>%
  dplyr::select(token_i_2, from_2, to_2) %>%
  stats::setNames(c("token_i", "from","to")) %>%
  dplyr::filter(!duplicated(token_i), !is.na(token_i)) %>%
  dplyr::arrange(token_i)

# get text data by tokenizing character level data
# / aggregating it to character-span level
text2_data <-
  dp$text[[text_name_2]]$
  tokenize_data_sequences(
    token = text2[,c("from","to")],
    aggregate_function = aggregate_function,
    ...
  ) %>%
  dplyr::select(-from, -to, -token_i)

# add text to text
f <- dp$text[[text_name_2]]$text_get
text2$text <- mapply(f, from=text2$from, to=text2$to)
text2$tnr  <- 2

if( any(text_var != TRUE) ){
  text1_data <- text1_data[, names(text1_data) %in% text_var, drop=FALSE]
  text2_data <- text2_data[, names(text2_data) %in% text_var, drop=FALSE]
}

# drop unwanted variables from alignment
 alignment <-
   alignment %>%
   dplyr::select_("token_i_1", "token_i_2", "distance", "type")

# should data be minimized?
jsonify <-
    function(x){
```

```
          htmlwidgets::JS(
            jsonlite::toJSON(
              x, "values", pretty = TRUE, na="null"
            )
          )
        }

    # return
    return(
      list(
        alignment                = jsonify(alignment),
        alignment_vars           = jsonify(names(alignment)),
        text                     = jsonify(rbind(text1, text2)),
        text_vars                = jsonify(names(text1)),
        alignment_data           = jsonify(alignment_data),
        alignment_data_vars      = jsonify(names(alignment_data)),
        text1_data               = jsonify(text1_data),
        text2_data               = jsonify(text2_data),
        text_data_vars           = jsonify(names(text1_data))
      )
    )
  }
```

dp table.R

```
if(getRversion() >= "2.15.1"){
  utils::globalVariables(
    c(
      "token_i_1", "token_i_2",
      "from", "to", "from_1", "to_1", "ti", "to_2", "from_2",
      "alignment_i",
      "var_name", "var_value", "token_i"
    )
  )
}


#' function for tabulation
#'
#' @param dp an object of type diffrproject
#' @param link which link to produce table for
#' @param width width of widget
#' @param height heigth of widget
#' @param align_var either a character vector of variable names or TRUE for all
#' @param text_var either a character vector of variable names or TRUE for all
#' @param aggregate_function a function able to resolve conflicts if for a
#'    specific variable for a token of text severla values exist, if NULL it
#'    defaults to modus() but could also be e.g. paste or something alike
#' @param ... further arguments passed through to aggregate_function
#'
#' @export
#'
```

```
dp_table <- function(
  dp,
  link               = NULL,
  align_var          = FALSE,
  text_var           = FALSE,
  aggregate_function = NULL,
  ...,
  width = "100%",
  height = "400px"
) {

  # pass the data and settings using 'x'
  x <-
    dp_prepare_data_table(
      dp,
      link               = NULL,
      align_var          = align_var,
      text_var           = text_var,
      aggregate_function = NULL
    )

  # create a list that contains the settings
  x$options <- list( )

  # create the widget
  htmlwidgets::createWidget(
    "dp_table",
    x,
    width = width,
    height = height,
    package= "diffrprojectswidget"
  )
}



#' dp_table shiny output function
#'
#' @param outputId I have no idea
#' @param width width
#' @param height height
#'
#' @export
#'
dp_tableOutput <- function(outputId, width = "100%", height = "400px") {
  htmlwidgets::shinyWidgetOutput(outputId, "dp_table", width, height, package = "diffrprojectswidget")
}


#' dp_table shiny render function
#'
#' @param expr expr
#' @param env env
#' @param quoted quoted
```

```
#'
#'
#' @export
#'
renderDp_table <- function(expr, env = parent.frame(), quoted = FALSE) {
  if (!quoted) { expr <- substitute(expr) } # force quoted
  htmlwidgets::shinyRenderWidget(expr, dp_tableOutput, env, quoted = TRUE)
}
```

dp vis.R

```
#' function for visualization
#'
#' @param dp an object of type diffrproject
#' @param link which link to produce vis for
#' @param width width of widget
#' @param height heigth of widget
#' @param align_var either a character vector of variable names or TRUE for all
#' @param text_var either a character vector of variable names or TRUE for all
#' @param aggregate_function a function able to resolve conflicts if for a
#'    specific variable for a token of text severla values exist, if NULL it
#'    defaults to modus() but could also be e.g. paste or something alike
#' @param ... further arguments passed through to aggregate_function
#'
#' @export
#'
dp_vis <- function(
  dp,
  link      = NULL,
  align_var = TRUE,
  text_var  = TRUE,
  aggregate_function = NULL,
  ...,
  width     = "100%",
  height    = "400px"
) {

  # pass the data and settings using 'x'
  x <-
    dp_prepare_data_vis(
      dp,
      link               = NULL,
      align_var          = align_var,
      text_var           = text_var,
      aggregate_function = aggregate_function,
      minimize = TRUE,
      ...
    )
```

```
  # create a list that contains the settings
  x$options <- list( )

  # create the widget
  htmlwidgets::createWidget(
    "dp_vis",
    x,
    width = width,
    height = height,
    package= "diffrprojectswidget"
  )
}


#' dp_vis shiny output function
#'
#' @param outputId I have no idea
#' @param width width
#' @param height height
#'
#' @export
#'
dp_visOutput <- function(outputId, width = "100%", height = "400px") {
  htmlwidgets::shinyWidgetOutput(outputId, "dp_vis", width, height, package = "diffrprojectswidget")
}


#' dp_vis shiny render function
#'
#' @param expr expr
#' @param env env
#' @param quoted quoted
#'
#'
#' @export
#'
renderDP_vis <- function(expr, env = parent.frame(), quoted = FALSE) {
  if (!quoted) { expr <- substitute(expr) } # force quoted
  htmlwidgets::shinyRenderWidget(expr, dp_visOutput, env, quoted = TRUE)
}
```

imports.r

```
#' @importFrom magrittr %>%
#' @export
magrittr::`%>%`


#' imports
```

```
#'
#' @import hellno
dummyimport <- function(){
  1 %>% magrittr::add(1)
}
```