Package 'geneviewer'

November 3, 2026

Type Package

Title Gene Cluster Visualizations

Version 0.2.0

Maintainer Niels van der Velden <n.s.j.vandervelden@gmail.com>

Description Provides tools for plotting gene clusters and transcripts by importing data from GenBank, FASTA, and GFF files. It performs global pairwise protein alignments and 'MUMmer' alignments [Needleman and Wunsch (1970) <doi:10.1016/0022-2836(70)90057-4>; Delcher et al. (1999) <doi:10.1093/nar/27.11.2369>] and displays results on gene arrow maps. Extensive customization options are available, including legends, labels, annotations, scales, colors, tooltips, and more.

License MIT + file LICENSE

SystemRequirements MUMmer (optional, used by mummer_alignment())

Encoding UTF-8

LazyData true

Language en-US

URL https://github.com/nvelden/geneviewer

BugReports https://github.com/nvelden/geneviewer/issues

Depends R (>= 3.5.0)

Imports htmlwidgets, fontawesome, magrittr, dplyr (>= 1.0.0), rlang, grDevices, stats, tools, utils

Suggests shiny, pwalign, Biostrings, chromote, htmltools, jsonlite, testthat (>= 3.2.0)

Config/testthat/edition 3

Config/Needs/website knitr, rmarkdown, tidyr, biomaRt

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Niels van der Velden [aut, cre], Mike Bostock [ctb, cph] (Bundled D3 library), Eli Grey [ctb, cph] (Bundled FileSaver.js library)

Repository CRAN

Date/Publication 2026-11-03 17:20:19 UTC

Contents

BRCA1_splice_variants . . . 3
coronavirus_genomes . . . 4
erythromycin_BlastP . . . 6
erythromycin_cluster . . . 7
gbk_features_to_df . . . 7
GC_align . . . 8
GC_annotation . . . 9
GC_chart . . . 11
GC_cluster . . . 12
GC_clusterFooter . . . 14
GC_clusterLabel . . . 15
GC_clusterTitle . . . 18
GC_color . . . 20
GC_coordinates . . . 21
GC_element_style . . . 23
GC_events . . . 25
GC_genes . . . 28
GC_grid . . . 30
GC_labels . . . 31
GC_legend . . . 33
GC_links . . . 36
GC_normalize . . . 40
GC_pattern . . . 41
GC_scale . . . 43
GC_scaleBar . . . 45
GC_sequence . . . 47
GC_theme . . . 49
GC_title . . . 52
GC_toolbar . . . 54
GC_tooltip . . . 56
GC_trackMouse . . . 57
GC_transcript . . . 58
GC_zoom . . . 61
GC_zoom_to . . . 63
genbank_to_fasta . . . 64
geneviewer-shiny . . . 65
geneviewer_palette . . . 66
geneviewer_theme . . . 67
get_introns . . . 68
hs_dystrophin_transcripts . . . 69
human_hox_genes . . . 69
mitochondrial_gene_order . . . 70

mummer_alignment 71
ophA_clusters 73
pQEB1_alignment 73
pQEB1_plasmids 74
protein_blast 75
read_bed 77
read_fasta 78
read_gbk 79
read_gff 80
tn7_crispr_elements 81

Index **83**

`BRCA1_splice_variants` *BRCA1 Splice Variants*

Description

This dataset contains detailed information on five different splice variants of the BRCA1 gene. It includes data on the types of transcript features (exon, intron, UTR), chromosomal positions, strand orientations, and transcript lengths.

Usage

```
BRCA1_splice_variants
```

Format

A data frame with 119 observations and 6 variables:

ensembl_transcript_id Unique Ensembl transcript identifiers. A character vector.

type Type of the transcript feature, indicating whether it is an "exon", or "UTR". A character vector.

start The start position of the feature on the chromosome. A numeric vector.

end The end position of the feature on the chromosome. A numeric vector.

strand The strand orientation of the feature, with -1 indicating the reverse strand. An integer vector.

transcript_length The total length of the transcript in base pairs. An integer vector.

Source

Derived from the Ensembl website

`coronavirus_genomes` *Genes of eleven coronaviruses from the four genera*

Description

The genes of eleven coronaviruses, the genomes of Fig. 1 of the reference: HCoV-229E and HCoV-NL63 (*Alphacoronavirus*), MHV, HCoV-OC43, HCoV-HKU1, MERS-CoV, SARS-CoV-1, SARS-CoV-2 and the bat coronavirus RaTG13 (*Betacoronavirus*), IBV (*Gammacoronavirus*) and PDCoV (*Deltacoronavirus*). They are in that order, by genus and subgenus, so that each genome sits next to its closest relatives here. Made by `data-raw/coronavirus_genomes.R`.

Usage

```
coronavirus_genomes
```

Format

A data frame with 148 rows and 15 columns:

cluster Short name of the virus, e.g. "SARS-CoV-2". A character vector.

genus Genus of the virus, e.g. "Betacoronavirus". A character vector.

subgenus Subgenus of the virus, e.g. "Sarbecovirus". A character vector.

virus Name of the virus. A character vector.

strain Strain or isolate, as in the record. A character vector.

host Host of the virus, e.g. "human" or "bat". A character vector.

accession GenBank or RefSeq accession of the genome. A character vector.

gene Gene name, e.g. "ORF1a", "S", "3a" or "9b", or "L" and "3'UTR" for the untranslated ends. A character vector.

product Description of the gene product, from the record. A character vector.

category Role of the gene: "5' UTR", "ORF1a", "ORF1b", "spike", "hemagglutinin-esterase", "envelope", "membrane", "nucleocapsid", "accessory" or "3' UTR". A character vector.

start Start position of the gene in the genome. A numeric vector.

end End position of the gene in the genome. A numeric vector.

strand Strand of the gene; all are "forward". A character vector.

homolog Chain of homologous genes across neighbouring genomes of the same genus, named after its first gene, e.g. "MHV 5a" for 5a of MHV, ns12.9 of HCoV-OC43 and 4 of HCoV-HKU1, for `GC_links(group = "homolog")`; `NA` for a gene without a homolog next to it. A character vector.

identity Protein identity, in percent, of the gene to its homolog in the genome above; `NA` for the first gene of a chain. A numeric vector.

Details

The genes are read from the GenBank records and named as in the literature, e.g. "3a" for the "Orf3" of SARS-CoV-1 and the "NS3" of RaTG13. ORF1ab is split at its -1 ribosomal frameshift into ORF1a and ORF1b. Fourteen accessory ORFs that the records do not annotate are added from their position in the genome (`product` ends in "(not annotated in the record)"): the ORFs that lie inside N in another reading frame (Ia and Ib of HCoV-229E and HCoV-NL63, 9b and 9c of RaTG13 and SARS-CoV-2, 8c of MERS-CoV), 3b, 3c and 3d inside ORF3a of SARS-CoV-2 and 3b of RaTG13 (the SARS-CoV-2 ORFs are described by Jungreis et al., 2021), and ORF10 of RaTG13, which matches ORF10 of SARS-CoV-2. Each runs from an ATG to its first stop codon. MHV is strain JHM, as the records of strain A59 leave out HE, 4 and 5a.

Homologous genes of closely related neighbouring genomes share a `homolog`. Every gene is translated from the genome and aligned to the genes of the next genome with `protein_blast()`. Only neighbours whose ORF1b, which holds the polymerase and is the most conserved protein, is at least 70% identical are joined: the genomes of a subgenus and the two alphacoronaviruses (82%), but not the subgenera of the betacoronaviruses (62-66%) or the genera (50-56%), whose proteins are so far apart that their identity says little. The cut-off is chosen for these genomes, not an official one. Between close relatives, the core genes (ORF1a, ORF1b, S, HE, E, M and N) are homologous and are joined to the same gene of the next genome, however low their identity. Accessory genes are joined when each is the other's best hit, with at least 30% identity, or 50% when either protein is shorter than 50 amino acids, as short alignments reach a third identity by chance.

Each genome also has a row for its 5' untranslated region, named "L" after the leader sequence it starts with, from the start of the genome to ORF1a, and one for its 3' untranslated region ("3'UTR"), from the last gene to the end of the genome.

Source

GenBank accessions AF304460, AY567487, AY903460, AY597011, MN996532 and AY274119 and RefSeq accessions AC_000192, NC_019843, NC_045512, NC_001451 and NC_039208, read with `read_gbk()`.

References

Chazal N. Coronavirus, the King Who Wanted More Than a Crown: From Common to the Highly Pathogenic SARS-CoV-2, Is the Key in the Accessory Genes? Front Microbiol. 2021;12:682603.

Jungreis I, Sealfon R, Kellis M. SARS-CoV-2 gene content and COVID-19 mutation impact by comparing 44 Sarbecovirus genomes. Nat Commun. 2021;12:2642.

Examples

```
GC_chart(coronavirus_genomes, cluster = "cluster", group = "category",
         height = "700px") %>%
  GC_labels("gene") %>%
  GC_clusterLabel() %>%
  GC_zoom(range = c(0.75, 1))
```

`erythromycin_BlastP` *Erythromycin protein alignment results*

Description

This dataset contains detailed information on genes involved in the biosynthesis of Erythromycin, an antibiotic produced by the bacterium Saccharopolyspora erythraea and several homologous gene clusters identified by antiSMASH. It includes gene identifiers, chromosomal positions, orientations, and annotations regarding their products and functions, as well as identity and similarity scores from `protein_blast()`, which uses global pairwise protein alignments.

Usage

```
erythromycin_BlastP
```

Format

A data frame with 148 observations and 16 variables:

protein_id Unique protein identifiers. A character vector.

region The chromosomal region of the gene, indicating start and end positions and strand. A character vector.

translation Amino acid sequence of the protein encoded by the gene. A character vector.

cluster Identifier of the gene cluster to which the gene belongs. A character vector.

strand The strand orientation ("forward" or "complement") of the gene. A character vector.

start The start position of the gene on the chromosome. A numeric vector.

end The end position of the gene on the chromosome. A numeric vector.

rowID A unique identifier for each row in the dataset. An integer vector.

identity The percentage of identical positions in the global alignment with the best hit. A numeric vector.

similarity The percentage of identical or similar amino acids in the global alignment with the best hit. A numeric vector.

BlastP The protein_id of the best hit in the query cluster, or NA if there is none. A character vector.

score Synteny score of the cluster with the query cluster, based on the hits. A numeric vector.

Gene Gene name or identifier if available, otherwise NA. A character vector.

Position Formatted string indicating the gene's position and orientation on the chromosome. A character vector.

Product Description of the gene product. A character vector.

Functions Functional categorization of the gene. A character vector.

Source

Derived from `protein_blast()` alignments of Saccharopolyspora erythraea genes involved in Erythromycin production.

erythromycin_cluster *Erythromycin Gene Cluster Data*

Description

This dataset contains information about various genes involved in the biosynthesis of Erythromycin, an antibiotic produced by the bacterium Saccharopolyspora erythraea. It includes details such as gene identifiers, start and end positions of genes, and their associated functions and products.

Usage

```
erythromycin_cluster
```

Format

A data frame with 23 rows and 6 columns:

Identifiers Unique identifiers for the genes. A character vector.

Start Start positions of the genes on the chromosome. A numeric vector.

End End positions of the genes on the chromosome. A numeric vector.

Strand Strand orientation of the gene, either "+" or "-". A character vector.

Product Description of the gene products. A character vector.

Functions Functional categorization of the genes. A character vector.

gbk_features_to_df *Convert GenBank Features to Data Frame*

Description

This function processes a list of GenBank features (loaded by read_gbk()) and converts selected features into a data frame. It supports processing multiple gene clusters.

Usage

```
gbk_features_to_df(
  gbk_list,
  feature = "CDS",
  keys = NULL,
  process_region = TRUE
)
```

Arguments

gbk_list	A list of lists where each sub-list contains GenBank features for a specific gene cluster. Each sub-list is expected to have a named list of features, with each feature being a character vector.
feature	A string specifying the feature type to extract from each gene cluster's FEATURE list (e.g., "CDS" or "gene"). Defaults to "CDS".
keys	An optional vector of strings representing specific keys within the feature to retain in the final data frame. If NULL (the default), all keys within the specified feature are included.
process_region	A boolean flag; when set to TRUE (the default), special processing is performed on the 'region' key (if present) to extract 'strand', 'start', and 'end' information.

Value

A data frame where each row corresponds to a feature from the input list. The data frame includes a 'cluster' column indicating the source gbk file.

Examples

```
gbk_file <- system.file("extdata", "BGC0000001.gbk", package = "geneviewer")
gbk <- read_gbk(gbk_file)
df <- gbk_features_to_df(gbk)

# To extract only specific keys within the "CDS" feature
df <- gbk_features_to_df(gbk, feature = "CDS", keys = c("gene", "region"))

# To disable special processing of the 'region' key
df <- gbk_features_to_df(gbk, process_region = FALSE)
```

GC_align *Align gene clusters*

Description

This function aligns clusters based on a specified gene id. The clusters are shifted so the gene sits at the same place in each of them, and all clusters keep the same scale: genes of the same length are drawn at the same width. The alignment is kept when GC_scale() is called afterwards, also with axis_type = "range".

Usage

```
GC_align(GC_chart, id_column, id, align = "left")
```

Arguments

GC_chart	A chart object containing genomic data along with clustering information.
id_column	The name of the column that contains the gene identifiers.
id	The specific identifier of the gene to be aligned.
align	The alignment method for the gene. Valid values are "left", "right", or "center". Defaults to "left".

Value

The modified GC_chart object with updated genomic coordinates.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5'),
  group = c('A', 'B', 'C', 'C', 'A'),
  cluster =  c(1, 1, 1, 2, 2)
)


GC_chart(genes_data, group = "group", cluster = "cluster", height = "150px") %>%
  GC_align("group", "A", align = "left") %>%
  GC_legend(FALSE)
```

GC_annotation *Add Annotations to a GC_chart*

Description

This function adds annotations to specified clusters within a GC chart. Annotations can be of various types and are positioned based on provided coordinates. The types of annotations available are: text, textMarker, line, arrow, symbol, rectangle, promoter, terminator and arc.

Usage

```
GC_annotation(GC_chart, type = "text", cluster = NULL, ...)
```

Arguments

GC_chart	A GC chart object to which the annotations will be added.
type	The type of annotation to add: one of "text" (default), "textMarker", "line", "arrow", "symbol", "rectangle", "promoter", "terminator" or "arc".
cluster	Numeric or character vector specifying the clusters to which annotations should be added.
...	Additional parameters for customization of annotations, depending on the type.

Value

Updated GC chart object with added annotations.

See Also

GC_trackMouse

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
)

# Adding annotations to a GC chart
GC_chart(genes_data, cluster = "cluster", group = "group", height = "220px") %>%
  GC_annotation(
    type = "textMarker",
    cluster = 1,
    position = 24,
    text = "Gene 1",
    arrowSize = 8
  ) %>%
  GC_annotation(
    type = "text",
    text = "feature 1",
    x = 91,
    y = 71
  ) %>%
  GC_annotation(
    type = "symbol",
    symbol = "triangle",
    x = 95,
    y = 64,
    size = 10,
    rotation = 180
  ) %>%
  GC_annotation(
    type = "terminator",
    x = 81
  ) %>%
  GC_annotation(
    type = "promoter",
    x = 49
  ) %>%
  # Convenience function to track mouse position on hoover
  GC_trackMouse()
```

GC_chart *Create a GC Chart Visualization*

Description

Generates an interactive GC chart for genomic data.

Usage

```
GC_chart(
  data,
  start = "start",
  end = "end",
  cluster = NULL,
  group = NULL,
  strand = NULL,
  width = "100%",
  height = "400px",
  style = list(),
  elementId = NULL,
  save_button = TRUE
)
```

Arguments

data	Data frame containing genomic information or the file path to a folder containing .gbk files. When providing a file path, the data is loaded and processed into a data frame internally.
start	Column name that indicates start positions. Default is "start".
end	Column name that indicates end positions. Default is "end".
cluster	Optional column name used for clustering purposes. Default is NULL.
group	Column name used for gene grouping to influence color aesthetics.
strand	Optional column name indicating strand orientation. Acceptable values include 1, 'forward', 'sense', or '+' to represent the forward strand, and -1, 0, 'reverse', 'antisense', "complement" or '-' to represent the reverse strand. Default is NULL, meaning strand information is not used.
width	Width specification for the chart, such as '100%' or 500. Default is unspecified.
height	Height specification for the chart, such as '400px' or 300. Default is unspecified.
style	A list of CSS styles to be applied to the chart container. Each element of the list should be a valid CSS property-value pair. For example, list(backgroundColor = "white", border = "2px solid black"). Default is an empty list.
elementId	Optional identifier string for the widget. Default is NULL.
save_button	Logical, whether to include a save button to save the chart as SVG or PNG. Default is TRUE.

Value

A GC chart widget.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
)
# Load from data.frame
GC_chart(genes_data, group = "group", cluster = "cluster", height = "200px") %>%
GC_labels("name")

# Load from a GenBank file (or a folder of .gbk files)
gbk_file <- system.file("extdata", "BGC0000001.gbk", package = "geneviewer")
GC_chart(gbk_file, height = "200px")
```

GC_cluster *Modify Cluster Settings*

Description

This function can switch prevention of gene overlap on, adjust the spacing between tracks and alter the styling of specified clusters.

Usage

```
GC_cluster(
  GC_chart,
  separate_strands = NULL,
  strand_spacing = NULL,
  prevent_gene_overlap = NULL,
  overlap_min = NULL,
  overlap_spacing = NULL,
  cluster = NULL,
  style = list(),
  ...
)
```

Arguments

GC_chart	The gene chart object to be modified.
separate_strands	Logical, indicating whether to vertically separate forward and reverse genes.

`strand_spacing` Numeric, specifies the spacing between genes on different strands. Used only if `separate_strands` is TRUE.

`prevent_gene_overlap`
Whether to put overlapping genes on separate tracks, below each other: `TRUE` for all genes that overlap, or `"nested"` for only the genes that lie inside another gene (e.g. an ORF inside another reading frame), which go on the tracks below; the other genes stay on the first track, also where they overlap. A gene goes on the first track where it overlaps no gene placed before it. The track of each gene is in the `geneTrack` column, for the `style_by` of `GC_genes()` and `GC_labels()`, e.g. to draw the genes of the second track smaller.

`overlap_min` The overlap two genes need to go on separate tracks with `prevent_gene_overlap`: a number of base pairs, or a share of the shorter gene as a string, e.g. `"10%"`. Genes that overlap by no more, e.g. two genes that share a base or a stop codon, stay on the same track. Default NULL: any overlap.

`overlap_spacing`
Numeric, specifies the spacing between overlapping genes Used only if `prevent_gene_overlap` is TRUE.

`cluster` Optional; used to specify which clusters in the chart should have tooltips.

`style` A list of CSS styles to be applied to the gene track. Each element of the list should be a valid CSS property-value pair. For example, list(backgroundColor = "red", color = "white").

`...` Additional arguments to be passed to the underlying functions.

Value

Returns the modified gene chart object.

Examples

```
genes_data <- data.frame(
  start = c(1, 10, 200, 220, 600),
  end = c(10, 150, 180, 400, 400),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5'),
  group = c('A', 'A', 'A', 'A', 'A')
)


GC_chart(genes_data, group = "group", height = "150px") %>%
  GC_cluster(separate_strands=TRUE, strand_spacing = 0) %>%
  GC_legend(FALSE)

# Only the gene inside another one goes below, drawn smaller
nested_genes <- data.frame(
  start = c(1, 120, 300, 340), end = c(125, 400, 380, 600),
  name = c("A", "B", "B2", "C")
)
GC_chart(nested_genes, height = "150px") %>%
  GC_cluster(prevent_gene_overlap = "nested") %>%
  GC_genes(style_by = "geneTrack",
```

```
          groupStyle = list("2" = list(markerHeight = 8, arrowheadHeight = 8))) %>%
  GC_labels("name")
```

`GC_clusterFooter` *Add a Footer to Each Cluster in a GC Chart*

Description

This function allows you to add a footer to all or specific clusters within a GC chart. You can specify titles, subtitles, and control the display and styling.

Usage

```
GC_clusterFooter(
  GC_chart,
  title = NULL,
  subtitle = NULL,
  show = TRUE,
  cluster = NULL,
  subtitleFont = list(),
  titleFont = list(),
  align = NULL,
  spacing = NULL,
  ...
)
```

Arguments

`GC_chart`	A GC chart object that the footers will be added to.
`title`	Character vector or NULL. The title to be displayed in the footer. Multiple titles can be provided for different clusters, and they will be recycled if there are more clusters than titles. Default is NULL.
`subtitle`	Character vector or NULL. Subtitles to accompany the main titles. Default is NULL.
`show`	Logical vector. Controls the visibility of each footer. Default is TRUE for all clusters.
`cluster`	Numeric or character vector specifying which clusters should have footers added or updated. If NULL, all clusters will be updated. Default is NULL.
`subtitleFont`	List, styling options for the subtitle.
`titleFont`	List, styling options for the title.
`align`	Character vector. Horizontal alignment of the title and subtitle, "left", "center" or "right", recycled over the clusters. `x` moves them from this position. Default is NULL: the alignment of the theme, otherwise "left".
`spacing`	Numeric vector. Pixels between the title and the subtitle, recycled over the clusters. Default is NULL: as set by the theme, otherwise 12.
`...`	Additional arguments for further customization of the footers.

Value

A GC chart object with updated footer settings for each specified cluster.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

# Add a simple footer with subtitle to all clusters
GC_chart(genes_data, cluster = "cluster", group = "group") %>%
  GC_clusterFooter(
    title = "Cluster Footer",
    subtitle = "Cluster subtitle"
  )
# Add styling to the title and subtitle
GC_chart(genes_data, cluster = "cluster", group = "group") %>%
  GC_clusterFooter(
    title = "This is a footer",
    subtitle = "Subtitle for the footer",
    show = TRUE,
    cluster = 1,
    x = 6,
    y = -20,
    align = "center", # left / right
    spacing = 12,
    titleFont = list(
      fontSize = "12px",
      fontWeight = "bold",
      fontFamily = "sans-serif",
      fill = "black",
      cursor = "default"
    ),
    subtitleFont = list(
      fill = "grey",
      fontSize = "10px",
      fontStyle = "normal",
      fontFamily = "sans-serif",
      cursor = "default"
    )
  )
```

GC_clusterLabel *Set or Update Cluster Labels for a GC Chart*

Description

This function allows you to set or update the labels for specified clusters within a GC chart. It provides flexibility in terms of the title, visibility, width, position, and other additional customization options.

Usage

```
GC_clusterLabel(
  GC_chart,
  title = NULL,
  show = TRUE,
  width = NULL,
  cluster = NULL,
  position = NULL,
  align = NULL,
  wrapLabel = NULL,
  wrapOptions = list(),
  subtitle = NULL,
  subtitleFont = list(),
  spacing = NULL,
  ...
)
```

Arguments

GC_chart	A GC chart object.
title	Character vector. The title for the cluster label. Default is NULL.
show	Logical. Whether to show the cluster label. Default is TRUE.
width	Character. The width of the cluster label, which sets the left or right margin of the cluster. Default is NULL: a margin set with GC_grid(), otherwise the width of the theme, otherwise "100px".
cluster	Numeric or character vector. Clusters in the GC chart to update. Default is NULL.
position	Character. Position of the label, either "left" or "right". Default is NULL: the position of the theme, otherwise "left".
align	Character vector. Horizontal alignment of the lines of the label in its margin, "left", "center" or "right", recycled over the clusters. x moves the label from this position, e.g. x = 5 for some space at the left edge. Default is NULL: the alignment of the theme, otherwise "center".
wrapLabel	Logical. Indicates whether the label should be wrapped. Default is NULL: as set by the theme, otherwise TRUE.
wrapOptions	List. Specifies the wrapping options. Default is an empty List.
subtitle	Character vector. A second line below the title, e.g. the species, recycled over the clusters. Default is NULL: no subtitle.
subtitleFont	List. Settings for the subtitle font, e.g. list(fontStyle = "italic"). Default is an empty list: 11px, not bold.

spacing	Numeric vector. Pixels between the middle of the last line of the title and of the subtitle, recycled over the clusters. The title and subtitle are centred together on the cluster. Default is NULL: as set by the theme, otherwise 1.2 times the mean font size of the title and subtitle (14 for the default 12px and 11px).
...	Additional customization arguments for the cluster label, such as 'fontSize', 'fontStyle', 'fontWeight', 'fontFamily', 'cursor', etc.

Value

Updated GC chart with new or modified cluster labels.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

# Set cluster labels
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
GC_clusterLabel(title = unique(genes_data$cluster))

# Set label for a specific cluster
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
GC_clusterLabel(title = "Cluster 1", cluster = 1)

# Add a subtitle below each label
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
  GC_clusterLabel(
    title = c("pQEB1", "pQEB1_inv1"),
    subtitle = c("E. cloacae", "K. oxytoca"),
    subtitleFont = list(fontStyle = "italic"),
    spacing = 14
  )

# Style labels
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
  GC_clusterLabel(
    title = c("Cluster 1", "Cluster 2"),
    width = "100px",
    x = 0,
    y = 0,
    position = "left",
    align = "left", # center / right
    wrapLabel = TRUE,
    wrapOptions = list(
      dyAdjust = 0,
      lineHeightEms = 1.05,
      lineHeightSquishFactor =  1,
```

```
      splitOnHyphen =  TRUE,
      centreVertically = TRUE
     ),
    fontSize = "12px",
    fontStyle = "normal",
    fontWeight = "bold",
    fontFamily = "sans-serif",
    cursor = "default"
  )
```

GC_clusterTitle *Update cluster Title of a GC Chart Cluster*

Description

Modify the cluster title and subtitle of specified clusters within a GC chart and adjust the display settings.

Usage

```
GC_clusterTitle(
  GC_chart,
  title = NULL,
  subtitle = NULL,
  subtitleFont = list(),
  titleFont = list(),
  show = TRUE,
  height = NULL,
  cluster = NULL,
  align = NULL,
  spacing = NULL,
  ...
)
```

Arguments

GC_chart	A GC chart object.
title	Character vector. Titles to set for the clusters.
subtitle	Character vector. Subtitles to set for the clusters.
subtitleFont	List. Settings for the subtitle font.
titleFont	List. Settings for the title font.
show	Logical. Whether to display the title. Default is TRUE.
height	Character. Height for the title (e.g., "40px").
cluster	Numeric or character vector. Clusters in the GC chart to update.

align	Character vector. Horizontal alignment of the title and subtitle, "left", "center" or "right", recycled over the clusters. x moves them from this position. Default is NULL: the alignment of the theme, otherwise "center".
spacing	Numeric vector. Pixels between the title and the subtitle, recycled over the clusters. Default is NULL: as set by the theme, otherwise 20.
...	Additional customization arguments for title and subtitle.

Value

Updated GC chart with new title settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
)

# Basic usage
GC_chart(genes_data, cluster ="cluster", group = "group", height = "400px") %>%
GC_labels("name") %>%
GC_clusterTitle(
  title = "Cluster 1 Data",
  subtitle = "Detailed View",
  show = TRUE,
  cluster = 1
)

# Customizing title style
GC_chart(genes_data, cluster = "cluster", group = "group", height = "400px") %>%
  GC_labels("name") %>%
  GC_clusterTitle(
    title = "Cluster 1 Data",
    subtitle = "Detailed View",
    show = TRUE,
    cluster = 1,
    x = 0,
    y = 5,
    align = "center",
    spacing = 20,
    titleFont = list(
      fontSize = "16px",
      fontStyle = "normal",
      fontWeight = "bold",
      textDecoration = "normal",
      fontFamily = "sans-serif",
      cursor = "default",
      fill = "black"
      # Any other CSS styles
```

```
  ),
  subtitleFont = list(
    fontSize = "14px",
    fontStyle = "normal",
    fontWeight = "bold",
    textDecoration = "normal",
    fontFamily = "sans-serif",
    cursor = "default",
    fill = "black"
    # Any other CSS styles
  )
)
```

GC_color *Update Color Scheme in Gene Chart*

Description

This function updates the color scheme of the legend and genes in a gene chart.

Usage

```
GC_color(GC_chart, colorScheme = NULL, customColors = NULL)
```

Arguments

GC_chart	The gene chart object to be modified.
colorScheme	Optional; character or NULL, the name of a predefined color scheme to apply to the genes. Acceptable values include D3.js's built-in color schemes like "schemeCategory10", "schemeAccent", "schemeTableau10", and the geneviewer palettes like "okabeIto" (see geneviewer_palette()).
customColors	Either NULL, a list of color values, or a named list of color values by group. Groups left out of a named list keep the colours of colorScheme, the palette of the theme or the default colours, e.g. customColors = list(Other = "grey") only changes the colour of "Other".

Value

Returns the gene chart object with updated color settings for the genes.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
```

```
)

GC_chart(genes_data, cluster = "cluster", group = "group", height = "100px") %>%
  GC_color(colorScheme = "schemeCategory10")

GC_chart(genes_data, cluster = "cluster", group = "group", height = "100px") %>%
  GC_color(customColors = c("red", "orange", "green"))

GC_chart(genes_data, cluster = "cluster", group = "group", height = "100px") %>%
  GC_color(customColors = list(A = "yellow", B = "pink", C =  "purple"))

# Set the colour of one group; the others keep the colours of the scheme
GC_chart(genes_data, cluster = "cluster", group = "group", height = "100px") %>%
  GC_color(colorScheme = "schemeSet2", customColors = list(C = "grey"))
```

GC_coordinates *Modify Coordinates in a GC Chart*

Description

This function updates a GC chart by modifying the coordinates settings. It allows for showing or hiding tick values, applying custom tick values for the top and bottom axes, and supports several other customizations for specific or all clusters in the chart.

Usage

```
GC_coordinates(
  GC_chart,
  show = TRUE,
  tickValuesTop = NULL,
  tickValuesBottom = NULL,
  ticksFormat = NULL,
  tickStyle = list(),
  textStyle = list(),
  cluster = NULL,
  groupStyle = NULL,
  style_by = NULL,
  ...
)
```

Arguments

GC_chart	The GC chart object to be modified.
show	Logical, whether to show the tick values or not. Can be a single value or a vector.
tickValuesTop	Numeric vector or NULL, custom tick values to be used at the top of the cluster. If NULL, the default tick values are used.

tickValuesBottom
: Numeric vector or NULL, custom tick values to be used at the bottom of the cluster. If NULL, the default tick values are used.

ticksFormat
: Format for tick labels. Default is ",.0f".

tickStyle
: List, styling options for the ticks.

textStyle
: List, styling options for the text.

cluster
: Numeric or character vector or NULL; specifies which clusters to generate coordinates for. If NULL, labels will be applied to all clusters. Default is NULL.

groupStyle
: Named list of styles by group, applied to the coordinates of all genes of that group (the group of GC_chart(), or the column given in style_by). show = FALSE leaves out their coordinates, styles in tick go to their ticks and the other styles to their text, e.g. list(other = list(show = FALSE), tnsA = list(fill = "red")). Only for the coordinates computed from the genes, not for tickValuesTop or tickValuesBottom.

style_by
: Column name or NULL; the column whose values groupStyle is matched against. Default is NULL: the group of GC_chart(). the coordinate modifications. If NULL, applies to all clusters.

...
: Additional arguments to be passed to the coordinate options.

Value

Returns the GC chart object with updated coordinates.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5'),
  cluster = c(1, 1, 2, 2, 2)
)

# Add coordinates to all clusters
GC_chart(genes_data, cluster = "cluster", group = "name", height = "200px") %>%
GC_coordinates()

# Modify coordinates of a specific cluster
GC_chart(genes_data, cluster = "cluster", group = "name", height = "200px") %>%
GC_coordinates() %>%
GC_coordinates(
  cluster = 2,
  show = TRUE,
  tickValuesTop = c(130, 170, 210, 240),
  tickValuesBottom = c(160, 200),
  ticksFormat = ",.0f", # ".2s",
  rotate = -45,
  yPositionTop = 55,
  yPositionBottom = 45,
  overlapThreshold = 20,
```

```
  tickStyle = list(
  stroke = "black",
  strokeWidth = 1,
  lineLength = 6
  ),
  textStyle = list(
  fill = "black",
  fontSize = "12px",
  fontFamily = "Arial",
  cursor = "default"
  )
)
```

GC_element_style *Style an Individual Element*

Description

Adds or updates an override for one element of a chart, without replacing the styles of other elements: a gene, a gene label, a gene coordinate, or the title or subtitle of a cluster title, footer or label. Use this in an editor and in the R code generated by it. Legend labels and markers can also be edited individually by group and part.

Usage

```
GC_element_style(
  GC_chart,
 component = c("genes", "labels", "coordinates", "clusterTitle", "clusterFooter",
    "clusterLabel", "legend", "title", "scaleBar", "scale"),
  row_id = NULL,
  ...,
  cluster = NULL,
  part = NULL,
  end = NULL,
  reset = FALSE,
  group = NULL
)
```

Arguments

GC_chart	A GC chart object.
component	The component to edit: "genes", "labels", "coordinates", "clusterTitle", "clusterFooter", "clusterLabel", "legend", "title", "scaleBar" or "scale".
row_id	For genes, labels and coordinates: the gene's rowID in GC_chart$x$data, also returned in input$<outputId>_select$data$rowID and double-click event data.

...	Named element options, using JavaScript names, e.g. `fill`, `stroke`, `strokeWidth`, `marker`, `markerSize`, `fontSize`, `dx`, `dy` or `rotate`. Labels, titles and subtitles also accept `text` to override their displayed text. Gene labels, chart titles, cluster text, whole scale bars and axes, and legend labels and markers (each on its own) accept `moveX` and `moveY`, pixel offsets from their configured position. Coordinates accept `show = FALSE` to leave them out and `tick`, a list of options for their tick.
`cluster`	For cluster titles, footers, labels, scale bars and axes: the name or number of the cluster.
`part`	For chart and cluster titles, footers and labels: `"title"` or `"subtitle"`; for the legend: `"label"` or `"marker"`.
`end`	For coordinates: `"start"` or `"end"` for the coordinate of the start or end of the gene, or `NULL` (default) for both.
`reset`	Logical, remove this element's override instead of updating it.
`group`	For the legend: its original group value, also returned as `data` in click, selection and double-click events. Editing label text does not change this value or the input data. Marker edits affect only the legend patch.

Details

Overrides of genes, labels and coordinates match the row ID rather than the element's drawing index, so sorting, reverse clusters, resize and repeated gene names do not change their target. Row IDs are assigned by `GC_chart()` after filtering missing coordinates. Reproducing an edit requires the same input rows in the same order; replacing or reordering the input data requires remapping edits. These overrides take precedence over group styles and index-based styles. Reset removes the override, restoring underlying styles. Call this after the component's normal configuration (e.g. `GC_clusterTitle()`), which replaces its overrides. Legend overrides also take precedence over `GC_legend(groupStyle = ...)`; resetting one part restores its group and global styles.

Value

The chart with the element override updated.

Examples

```
genes <- data.frame(start = c(90, 10), end = c(130, 50), name = c("a", "a"))
GC_chart(genes) %>%
  GC_labels("name") %>%
  GC_clusterTitle("Cluster", subtitle = "Subtitle") %>%
  GC_element_style("genes", row_id = 1, fill = "red", marker = "box") %>%
  GC_element_style("labels", row_id = 1, text = "Edited", fontSize = "16px") %>%
  GC_element_style("clusterTitle", cluster = 1, part = "subtitle",
                   fontStyle = "italic", fill = "grey")

GC_chart(data.frame(start = 10, end = 40, family = "A"), group = "family") %>%
  GC_element_style("legend", group = "A", part = "label", text = "Family A") %>%
  GC_element_style("legend", group = "A", part = "marker", fill = "red")
```

GC_events *Report Chart Interactions and Completed Updates*

Description

Makes a chart report clicks to Shiny, so an app can react to the part of the chart that was clicked: show the details of a gene, open the settings of the clicked part, and so on. Charts are not clickable unless this is set, and click reporting has no effect outside Shiny.

Usage

```
GC_events(
  GC_chart,
  click = TRUE,
  highlight = FALSE,
  rendered = FALSE,
  render_id = NULL,
  dblclick = FALSE,
  select = FALSE,
  legend_click = TRUE,
  drag = FALSE,
  zoom = FALSE
)
```

Arguments

GC_chart	A GC chart object.
click	Logical, whether to report clicks to Shiny.
highlight	Logical, whether to outline the part of the chart under the mouse, to show what a click selects.
rendered	Logical, whether to report successful chart updates.
render_id	Optional single character string or finite number returned with the rendered event, to identify the update. The default, NULL, generates a unique ID for each update in the browser. It is not a DOM element ID; the event's chartId identifies the chart itself. A new render_id also marks a new load for GC_zoom(), which then starts at its range again, also when rendered is FALSE.
dblclick	Logical, whether to report double-clicks for editing.
select	Logical, whether to enable individual element selection.
legend_click	Logical, whether legend clicks are reported to Shiny.
drag	Logical, enable dragging gene labels, chart titles and cluster titles, footers, labels, whole scale bars and axes, and legend labels and markers, each on its own. Off by default. Releasing a drag reports input$<outputId>_drag and geneviewer:drag, with position containing moveX and moveY offsets in whole SVG pixels and action set to "drag". Apply them with GC_element_style() to preserve the position on redraw. Escape cancels an active drag. Moves can be undone and redone, see Details.

zoom — Logical, whether to report the window of the zoom slider of GC_zoom() as input$<outputId>_zoom and geneviewer:zoom: mode, start and end of the window in the units of the mode (whole base pairs, or shares to 0.0001), zoomed (FALSE when everything shows), range, the window to pass to GC_zoom(range =) (NULL when everything shows), clusters, the cluster and the first and last base pair it shows, and action: "drag", "keyboard" or "reset" by the user, "set" by GC_zoom_to(), or "render" when a render changed the window, e.g. new data. It is reported on the first render and after every change, so an app always knows the window.

Details

Each click sets input$<outputId>_click, e.g. input$chart_click for GC_chartOutput("chart"), to a list with:

component The part of the chart that was clicked: "genes", "transcript", "labels", "coordinates", "links", "legend", "scale", "scaleBar", "sequence", "title" (the chart title), "clusterTitle", "clusterFooter", "clusterLabel" or "annotations".

cluster The cluster that was clicked, or NULL for the chart title, the legend and links between clusters.

data The data of the clicked element, or NULL: the row of the gene for genes, transcripts and labels, the row of the link for links, and the group for the legend.

Every part of the chart is marked with a data-component attribute and each cluster with data-cluster, also without this function, so they can be selected with CSS or JavaScript.

With dblclick = TRUE, double-clicks independently report the same payload to input$<outputId>_dblclick and the bubbling browser event geneviewer:dblclick. Browsers also send clicks during a double-click: use clicks for selection, and double-clicks for editing, rather than opening different settings panels on each event.

With select = TRUE, a click selects an individual editable element and outlines it. Clicking it again, empty chart space or a noneditable part of the chart, or pressing Escape, clears selection. Shift-, Ctrl- or Cmd-click adds an element to the selection or takes it out, and dragging on empty chart space draws a box that selects the elements entirely inside it (with Shift, Ctrl or Cmd, in addition to the selection). With drag = TRUE as well, dragging one of the selected elements moves all of them that can be moved, and the arrow keys move them together; such a move of several elements is one step to undo, reported as input$<outputId>_moves. Selection reports the payload of the element selected last (or NULL when cleared) to input$<outputId>_select and geneviewer:select, and a list of the payloads of all selected elements, in the order they were selected, to input$<outputId>_selection and geneviewer:selection. It survives redraws of the same data and resize. When a redraw changes the selected element's style, its updated payload is reported again. Selected items can be opened with Enter when dblclick = TRUE; items are keyboard focusable in selection mode. Use click = FALSE to avoid legacy single-click settings handlers. legend_click = FALSE excludes legend clicks from click reporting without enabling legend click editing. Legend labels and patches are individually selectable, with part set to "label" or "marker" and data set to the original group value. Legend clicks edit instead of toggling visibility when selection or double-click events are enabled, or when click reporting is enabled with legend_click = TRUE. Otherwise they hide/show genes. Double-click and selection browser events also work outside Shiny.

Click, double-click and selection payloads of editable elements also contain `style`: how it is drawn, with the names of `GC_element_style()` options, so an editor can show the current values. For a gene these are `fill`, `opacity`, `stroke`, `strokeWidth`, `strokeDasharray`, `marker`, `markerSize`, `markerHeight`, `arrowheadWidth`, `arrowheadHeight`, `cornerRadius` and `y`; for a label `fill`, `opacity`, `text`, `fontSize` (in pixels), `fontWeight`, `fontStyle` and `fontFamily`. Colours are hex codes, except transparent colours, `"none"` and patterns. Legend labels report the same text options; legend markers report `fill`, `opacity`, `stroke`, `strokeWidth` and `strokeDasharray`.

With `drag = TRUE`, the undo, redo and reset icons next to the download icon undo and redo moves, and move every element back to where it was first drawn, so positions set with `GC_element_style()` are kept. They show while the mouse is over the chart, once there is a move. After clicking the chart, Ctrl+Z (Cmd+Z on a Mac) undoes and Ctrl+Shift+Z or Ctrl+Y redoes. With `select = TRUE` as well, the arrow keys move the selected text or scale by a pixel, or 10 pixels with Shift. Each drag is one step to undo, arrow-key moves of one element within a second are one step, and a reset is one step. A step that moves one element reports it like a drag, with `action` set to `"undo"`, `"redo"` or `"nudge"`. A step that moves more, such as a reset, reports `input$<outputId>_moves` and `geneviewer:moves` once, with `action` (`"reset"`, `"undo"` or `"redo"`) and `moves`, a list of drag payloads, so an app that keeps drags keeps these too. Steps whose element was since moved by something else are dropped. Outside Shiny, moves are kept when the chart is resized.

With `rendered = TRUE`, each successful update also sets `input$<outputId>_rendered` to a list containing `id` (the supplied `render_id`, or a generated ID), `chartId` (the chart's DOM element ID), `width` and `height` (the widget size in pixels). When `render_id` is `NULL`, a new ID is generated in the browser for each successful update, unique across charts on the page. `chartId` stays the same when that chart updates; generated render IDs change. Explicit IDs, including `0` and `""`, are kept. It fires after the chart, legend, links and click handlers have been built, once per data update, including repeated identifiers, but not on resize or a failed draw. It does not wait for browser paint or external image loading.

The widget also dispatches a bubbling `geneviewer:rendered` browser event with the same fields in `event.detail`, including outside Shiny. A loading indicator can listen to it without inspecting the widget payload. Use a different `render_id` for each load and ignore events for older loads.

Value

The GC chart object with the event settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130),
  end = c(40, 80, 120, 160),
  name = c("Gene 1", "Gene 2", "Gene 3", "Gene 4"),
  group = c("A", "A", "B", "C")
)

if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    GC_chartOutput("chart", height = "200px"),
    verbatimTextOutput("clicked")
  )
```

```
  server <- function(input, output, session) {
    output$chart <- renderGC_chart(
      GC_chart(genes_data, group = "group") %>%
        GC_labels("name") %>%
        GC_events(highlight = TRUE)
    )
    output$clicked <- renderPrint(str(input$chart_click))
  }
  shinyApp(ui, server)
}
```

GC_genes *Modify Gene Characteristics within a Chart*

Description

This function updates a gene chart with specific characteristics for genes based on the given parameters. It can show/hide genes, apply a color scheme, assign custom colors, filter by cluster, and accept additional options.

Usage

```
GC_genes(
  GC_chart,
  group = NULL,
  marker = NULL,
  marker_size = NULL,
  show = TRUE,
  colorScheme = NULL,
  customColors = NULL,
  cluster = NULL,
  itemStyle = list(),
  groupStyle = NULL,
  style_by = NULL,
  corner_radius = NULL,
  ...
)
```

Arguments

GC_chart	The gene chart object to be modified.
group	Column name used for gene grouping to influence color aesthetics.
marker	Character or NULL, type of marker to represent genes on the chart. Allowed values are 'arrow', 'boxarrow', 'box', 'cbox', and 'rbox'.
marker_size	Character or NULL, size category of the marker ('small', 'medium', 'large').
show	Logical, whether to show the genes or not.

colorScheme	Character or NULL, the name of the color scheme to use: a d3 scheme (e.g. "schemeSet2") or a geneviewer palette (e.g. "okabeIto", see geneviewer_palette()). When no cluster is selected, the legend uses the same colours. NULL keeps the colours set by GC_color().
customColors	List or NULL, custom colors to apply to the genes: a vector of colours, recycled over the groups, or a named list of colours by group. Groups left out of a named list keep the colours of the colour scheme. When no cluster is selected, the legend uses the same colours.
cluster	Numeric or character, the specific cluster to filter genes by.
itemStyle	List, a list of styles to apply to individual items in the chart.
groupStyle	Named list of styles by group, applied to every gene of that group (of group, or of the column given in style_by), e.g. list(promoter = list(marker = "box", fill = "grey")). Takes the same options as itemStyle; the itemStyle of a gene takes precedence over the style of its group.
style_by	Column name or NULL; the column whose values groupStyle is matched against, to style genes by another column than the one that sets their colours, or "geneTrack" for the track of a gene with GC_cluster()(prevent_gene_overlap =), e.g. groupStyle = list("2" = list(markerHeight = 8)) to draw the genes of the second track smaller. Default is NULL: group.
corner_radius	Numeric or NULL, radius in pixels of the rounded corners of 'arrow', 'boxarrow', 'box' and 'rbox' markers, e.g. 3 for soft corners. The inner corners of arrows stay sharp. Default is NULL: sharp corners, and 5 for 'rbox'.
...	Additional arguments to be passed to the gene options.

Value

Returns the modified gene chart object.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

# Change the appearance of a specific gene cluster
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
  GC_genes(
    group = "group",
    show = TRUE,
    marker = "arrow",
    marker_size = "medium",
    colorScheme = NULL, # One of D3.js build in colorSchemes
                        # (eg. "schemeCategory10",
                        # "schemeAccent", "schemeTableau10")
```

```
    customColors = NULL, # A vector of color names
    prevent_overlap = FALSE,
    gene_overlap_spacing = 40,
    cluster = 1, # Specify a specific cluster
    x = 1,
    y = 50,
    stroke = "black",
    strokeWidth = 1,
    arrowheadWidth = NULL,
    arrowheadHeight = NULL,
    arrowHeight = NULL,
    markerHeight = NULL # overwrites marker_size
   )

# Change the appearance of a specific gene
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
GC_genes(
  cluster = 2,
  itemStyle = list(list(index = 2, fill = "red", stroke = "blue")
  )
 )

# Change the appearance of all genes of a group
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
  GC_genes(groupStyle = list(C = list(marker = "box", stroke = "red")))
```

GC_grid *Update Grid Display of a GC Chart Cluster*

Description

Modify the grid display of specified clusters within a GC chart. This function allows users to adjust the margins, width, and height of the grid for each cluster.

Usage

```
GC_grid(
  GC_chart,
  margin = NULL,
  width = NULL,
  height = NULL,
  direction = "column",
  cluster = NULL
)
```

Arguments

GC_chart	A GC chart object.

margin	A list specifying top, right, bottom, and left margins.
width	Numeric or character. Width of the grid. If numeric, will be considered as percentage.
height	Numeric. Height of the grid.
direction	Character. Layout direction of the grid, either "column" (default) for vertical or "row" for horizontal.
cluster	Numeric or character vector. Clusters in the GC chart to update.

Value

Updated GC chart with new grid display settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

# Set Margin of clusters
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
GC_grid(margin = list(left = "50px", right = "0px"))

# Set height of a specific cluster
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
GC_grid(height = "120px", cluster = 2)
```

GC_labels *Add Labels to Each Cluster in a GC Chart*

Description

This function adds labels to each cluster within a GC chart. It provides the option to show or hide the labels and supports customization of label properties. It can automatically pick up group names as labels from the GC_chart object if not provided.

Usage

```
GC_labels(
  GC_chart,
  label = NULL,
  show = TRUE,
  cluster = NULL,
  itemStyle = list(),
```

```
  groupStyle = NULL,
  style_by = NULL,
  ...
)
```

Arguments

GC_chart	A GC chart object to which labels will be added.
label	Character vector, logical, or NULL. Specific labels for the clusters. If NULL and GC_chart has group names, those will be used as labels.
show	Logical; controls the visibility of labels. Default is TRUE.
cluster	Numeric or character vector or NULL; specifies which clusters should be labeled. If NULL, labels will be applied to all clusters. Default is NULL.
itemStyle	List, a list of styles to apply to individual items in the chart.
groupStyle	Named list of styles by group, applied to the labels of all genes of that group (the group of GC_chart(), or the column given in style_by), e.g. list(tnsA = list(fill = "white")). Takes the same options as itemStyle; the itemStyle of a label takes precedence over the style of its group.
style_by	Column name or NULL; the column whose values groupStyle is matched against, e.g. the label column to style labels by their text, or "geneTrack" for the track of a gene with GC_cluster()(prevent_gene_overlap =), e.g. groupStyle = list("2" = list(...)) for the labels of the second track. Default is NULL: the group of GC_chart().
...	Additional arguments for further customization of the labels, e.g. adjustLabels: TRUE (default) turns labels that overlap another one, "hide" hides them instead, and FALSE leaves them as they are. A label that would turn out of the plot area, e.g. above its cluster or past the right end, is hidden rather than cut off. Hidden labels come back when there is room for them, e.g. when zoomed in with GC_zoom(). adjustLabels = "spread" puts the labels on one line below the lowest gene, moved apart where they would overlap, with a line from each label to its gene. leaderLine sets the distance from the lowest gene to the labels (length, in pixels, default 20) and the style of the lines, e.g. leaderLine = list(length = 30, stroke = "grey", strokeWidth = 1); leaderLine = FALSE spreads the labels without lines. spreadPosition = "top" puts the spread labels on one line above the highest gene instead (default "bottom"). The lines start at the side of the gene that faces its label, also when a label is dragged to the other side.

Value

A GC chart object with updated label settings for each specified cluster.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
```

```
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

# Add labels to all clusters
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
GC_labels()

# Bold labels for the genes of group A
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
  GC_labels(groupStyle = list(A = list(fontWeight = "bold")))

# Add labels and styling
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
GC_labels(
 label = "group",
 show = TRUE,
 x = 0,
 y = 50,
 dy = "-1.2em",
 dx = "0em",
 rotate = 0,
 adjustLabels = TRUE, # Rotate labels to prevent overlap; "hide" hides them
 fontSize = "12px",
 fontStyle = "italic",
 fill = "black",
 fontFamily = "sans-serif",
 textAnchor = "middle",
 cursor = "default",
)

# Alter style of a specific label
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
GC_labels(label = "group") %>%
GC_labels(
  cluster = 1,
  itemStyle = list(
    list(index = 0, fill = "red", fontSize = "14px", fontWeight = "bold")
  )
)
```

GC_legend *Set Legend for a Gene Chart*

Description

This function configures the legend for a gene chart. It allows toggling the legend's visibility, setting a background color, and assigning custom labels for the legend entries. The function can also handle additional customizations through various options.

Usage

```
GC_legend(
  GC_chart,
  group = NULL,
  show = TRUE,
  backgroundColor = NULL,
  order = list(),
  position = NULL,
  align = NULL,
  style = list(),
  legendOptions = list(),
  legendTextOptions = list(),
  corner_radius = NULL,
  groupStyle = NULL,
  ...
)
```

Arguments

GC_chart	The gene chart object to be modified.
group	Optional; character or NULL, specifies the groups to include in the legend. If NULL, groups are taken from the 'group' attribute of the 'GC_chart' object.
show	Logical, specifies whether to display the legend.
backgroundColor	String, the background color of the legend. Same as `style = list(backgroundColor = ...)`, which takes precedence. Default is NULL: as set by the theme, otherwise transparent.
order	Optional; list, specifies the order of the legend labels.
position	Character. Position of the legend, either "top" or "bottom". Default is NULL: the position of the theme, otherwise "bottom".
align	Character. Horizontal alignment of the legend items, "left", "center" or "right", like the title. Each row of a horizontal legend is aligned on its own. `x` moves the legend from this position. Default is NULL: the alignment of the theme, otherwise "left".
style	A list of CSS styles to be applied to the chart container. Each element of the list should be a valid CSS property-value pair. For example, list(backgroundColor = "white", border = "2px solid black"). Default is an empty list.
legendOptions	List, additional options for the legend.
legendTextOptions	List, additional text options for the legend.
corner_radius	Numeric or NULL, radius in pixels of the rounded corners of the legend markers, e.g. `3`. Default is NULL: square markers.
groupStyle	Named list of styles by group, applied to the text of the legend entry of that group, e.g. `list(tnsA = list(fontStyle = "italic"))` for italic gene names. Styles in `marker` go to its marker, e.g. `list(A = list(marker = list(stroke = "black")))`.

... Additional arguments to be passed to the legend configuration.

Value

Returns the modified gene chart object with the legend configured.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c('A', 'A', 'A', 'B', 'B')
)

# Customize legend
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
  GC_legend(
    position = "top", #bottom
    orientation = "horizontal", #vertical
    align = "center", #left / right
    x = 0,
    y = 0,
    adjustHeight = TRUE,
    backgroundColor = "#0000",
    order = list(),
    style = list(
      backgroundColor = "#0000"
      # Any other CSS styles
    ),
    legendOptions = list(
      cursor = "pointer",
      colorScheme = NULL,
      customColors = NULL # c("red", "green", "orange")
      # Additional styles
    ),
    legendTextOptions = list(
      cursor = "pointer",
      textAnchor = "start",
      dy = ".35em",
      fontSize = "12px",
      fontFamily = "sans-serif",
      fill = "black"
      # Additional styles
    )
  )
```

GC_links *Add Links to GC Chart*

Description

Add links generated by `get_links` to a `GC_chart` object. Links are added to the graph by their respective rowIDs.

Usage

```
GC_links(
  GC_chart,
  group = NULL,
  data = NULL,
  value1 = NULL,
  value2 = NULL,
  cluster = NULL,
  curve = NULL,
  measure = "identity",
  show_links = TRUE,
  label = TRUE,
  label_position = "gene",
  normal_color = NULL,
  inverted_color = NULL,
  low_color = NULL,
  vmin = NULL,
  vmax = NULL,
  use_group_colors = FALSE,
  color_bar = NULL,
  colorBarOptions = list(),
  linkWidth = NULL,
  linkStyle = list(),
  labelStyle = list(),
  ...
)
```

Arguments

`GC_chart`	Gene chart object.
`group`	The name of the column in the data to create value pairs from.
`data`	data.frame containing linking data.
`value1`	Optional vector of group values to generate links for.
`value2`	Optional vector of group values to generate links for.
`cluster`	Numeric or character vector or NULL; specifies which clusters.
`curve`	Logical; if `TRUE`, links are curved, otherwise straight. Default is NULL: as set by the theme, otherwise `TRUE`.

`measure` Character; specifies which measure to use for link color intensity. Should be "identity", "similarity", or "none".

`show_links` Logical; if `TRUE`, links are shown, otherwise hidden.

`label` Logical; if `TRUE`, shows measure labels on the links, otherwise hidden. When the genes also have labels from `GC_labels()`, the measure is shown below the gene name in the same label (or after it, when the label is rotated to avoid overlap).

`label_position` Where the measure labels go: `"gene"` (default), with the label of the gene as described for `label`, or `"link"`, in the middle of each link, white on dark links. A measure is only written where it fits within its link and clear of the others, so narrow links stay clean; zoomed in with `GC_zoom()`, links widen and more of them are written. With `"link"`, links from `data` have labels too.

`normal_color` Color for the links in their normal state. Default is NULL: the color of the theme, otherwise "#969696".

`inverted_color` Color for inverted links. Default is NULL: the color of the theme, otherwise "#d62728".

`low_color` Color that links fade to at a low measure: links run from nearly this color (10% of the link color, at `vmin`) to `normal_color` or `inverted_color` (at `vmax`). Default is NULL: the color of the theme, otherwise white.

`vmin, vmax` Numeric; the measures of the lightest and the full link color, also the ends of the color bar. Measures outside are drawn in the color of the nearest end. Default is NULL: as set by the theme, otherwise 0 and 100. Set `vmin` to the lowest measure to spread the colors over the data, e.g. `vmin = floor(min(alignment$identity))`.

`use_group_colors` Logical; if `TRUE`, color links by group.

`color_bar` Logical; if `TRUE`, the color bar is displayed. Default is NULL: as set by the theme, otherwise `TRUE`.

`colorBarOptions` List of options to customize the color bar appearance: `position`, where the bar goes: a corner of the plot area, `"bottomright"` (default), `"bottomleft"`, `"topright"` or `"topleft"`, or `"legend"` for an item of the legend (`GC_legend()`), lying down, in the font of the legend; without a legend it goes bottom right. `legendSide` puts it before the groups of the legend, `"left"` (default), or after them, `"right"`. `orientation`, `"vertical"` (default) or `"horizontal"`. `labels`, for a horizontal bar: `"inline"`, one line with the title, the lowest measure, the bar and the highest, or `"below"`, the title above the bar and ticks with their measures below it, as many as fit. Default is `"inline"`; in the legend it is always inline, as its rows are one line. The title of a horizontal bar is bold. With the labels below, the bar is about three lines tall, so in a corner it is more likely to cover genes. In the legend the bar is as tall as the legend markers, with a light grey outline when the theme leaves it without one. `x` and `y` move the bar in from its corner (default 0, and 24 pixels up from a bottom corner for a vertical bar); `width` and `height` set the size of the bar (default 10 by 60 pixels standing, 80 by 8 lying inline, 120 by 8 lying with the labels below); `title = FALSE` leaves out its title; `labelOptions`, `titleOptions` and `barOptions` style the labels, the title and the bar. The bar is drawn over the plot area, so in a corner it can cover genes or links: move it with `x` and `y`, or put it in the legend.

linkWidth	Numeric; specifies the width of the links. A value of 1 represents full width, and 0 represents no width.
linkStyle	A list of CSS styles to apply to the links.
labelStyle	A list of CSS styles specifically for the labels.
...	Additional arguments passed to the links.

Value

Modified GC_chart object with added links.

Examples

```
# Add links between all groups in each cluster
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 240, 250, 300, 340, 380, 420),
  end = c(40, 120, 160, 200, 210, 270, 330, 370, 410, 450),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5',
           'Gene 6', 'Gene 7', 'Gene 8', 'Gene 9', 'Gene 10'),
  group = c('A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'D'),
  identity = c(NA, NA, NA, 50, 40, 100, 60, 65, 20, NA),
  similarity = c(NA, NA, NA, 40, 30, 90, 50, 55, 10, NA),
  cluster = c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3)
)
GC_chart(genes_data,
         cluster = "cluster",
         height = "200px") %>%
  GC_links(
    group = "group",
    value1 = "A",
    value2 = "B",
    measure = "identity",
    label = FALSE) %>%
  GC_labels(label = "group")

# Add links between group A of cluster 1 and A and B of cluster 2
GC_chart(genes_data,
         cluster = "cluster",
         height = "200px") %>%
         GC_labels(label = "group") %>%
         GC_links(group = "group",
                  value1 = c("A", "A"),
                  value2 = c("B", "A"),
                  label = FALSE,
                  cluster = c(1,2))

# A horizontal color bar in the legend
GC_chart(genes_data, cluster = "cluster", group = "group",
         height = "220px") %>%
  GC_links(group = "group", measure = "identity", label = FALSE,
           colorBarOptions = list(position = "legend")) %>%
  GC_legend(position = "top")
```

```
# Style links and color bar
GC_chart(genes_data,
         cluster = "cluster",
         height = "200px"
) %>%
  GC_links(
    group = "group",
    data = NULL,
    curve = TRUE,
    measure = "identity",
    show_links = TRUE,
    label = TRUE,
    normal_color = "#1f77b4",
    inverted_color = "#d62728",
    use_group_colors = FALSE,
    color_bar = TRUE,
    colorBarOptions = list(
      x = 0,
      y = 24,
      width = 10,
      height = 60,
      labelOptions = list(
        fontSize = 8,
        xOffset = 2,
        yOffset = 0
        # Any other CSS style
      ),
      titleOptions = list(
        fontSize = 10,
        xOffset = 2,
        yOffset = 0
        # Any other CSS style
      ),
      barOptions = list(
        stroke = "#000",
        strokeWidth = 0.5,
        opacity = 1
        # Any other CSS style
      )
    ),
    linkWidth = 1,
    linkStyle = list(
      stroke = "black",
      strokeWidth = 0.5,
      fillOpacity = 0.4
      # Any other CSS style
    ),
    labelStyle = list(
      fontSize = "8px"
      # Any other CSS style
    )
  ) %>%
```

```
  GC_labels(label = "group", cluster = 1) %>%
  GC_clusterLabel()
```

`GC_normalize` *Normalize Gene Clusters in a Genomic Chart*

Description

This function normalizes the genomic coordinates of a set of genes within a cluster, ensuring that the genes are evenly spaced along the entire range of the cluster. The function allows for the option to preserve the original gene lengths and to introduce customized gaps between the genes.

Usage

```
GC_normalize(
  GC_chart,
  group = NULL,
  cluster = NULL,
  normalize = TRUE,
  preserve_gene_length = TRUE,
  gap = NULL,
  custom_gaps = list()
)
```

Arguments

`GC_chart`	A chart object containing genomic data and clustering information. The chart object must include a `series` component with data for each cluster.
`group`	Column name containing gene names to apply custom gaps to.
`cluster`	A vector of cluster names to be normalized. If `NULL`, all clusters in the `GC_chart` will be normalized.
`normalize`	A logical value indicating whether to normalize the gene positions within each cluster. Default is `TRUE`.
`preserve_gene_length`	A logical vector indicating whether to preserve the original gene lengths for each cluster. If a single value is provided, it is recycled for all clusters.
`gap`	A numeric vector specifying the proportion of the total length to be used as the gap between genes in each cluster. The value must be between 0 and 1. If a single value is provided, it is recycled for all clusters. If `NULL`, the gaps are calculated based on the actual spacing between the genes in the original data.
`custom_gaps`	A named list where each name corresponds to a gene name and each value is a numeric factor by which to adjust the gap after that gene.

Value

The modified `GC_chart` object with normalized gene coordinates for the specified clusters.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 10, 90, 130),
  end = c(40, 120, 160, 40, 120, 160),
  name = c('Gene 1', 'Gene 2', 'Gene 3','Gene 1', 'Gene 2', 'Gene 3'),
  group = c('A', 'B', 'C', 'A', 'B', 'C'),
  cluster =  c(1, 1, 1, 2, 2, 2)
)

GC_chart(genes_data, group = "group", cluster = "cluster", height = "150px") %>%
  GC_normalize(
    cluster = 2,
    preserve_gene_length = TRUE,
    gap = NULL
 )
```

GC_pattern *Fill Genes with Patterns*

Description

Fills the genes, exons and legend markers with a pattern per group, for example for black and white figures. Patterns can also be drawn on top of the gene colours to show each group with both a colour and a pattern.

Usage

```
GC_pattern(
  GC_chart,
  patterns = NULL,
  color = "black",
  background = "white",
  size = 6,
  strokeWidth = 1
)
```

Arguments

GC_chart	The gene chart object to be modified.
patterns	NULL, a vector of pattern names or a named list of pattern names by group. NULL gives each group a pattern in a fixed order. A vector is recycled over the groups. Groups left out of a named list get no pattern ("none"). Available patterns: "none", "solid", "horizontal", "vertical", "diagonal", "antidiagonal", "grid", "crosshatch", "dots", "zigzag", "checkerboard", "rings", "bricks", "plus", "wideStripes" and "inverseDots" (background-coloured dots on a solid tile).

color	The colour of the pattern lines and dots: a single colour, a vector of colours recycled over the groups, or a named list of colours by group. For example, color = "white" with background = "black" gives white patterns on black.
background	The colour behind the pattern: a single colour, a vector of colours recycled over the groups, or a named list of colours by group. NULL keeps the gene colours set by GC_color(). When background is not NULL, intron lines and UTR outlines take color.
size	Numeric, the width and height of one pattern tile in pixels.
strokeWidth	Numeric, the width of the pattern lines in pixels.

Value

Returns the gene chart object with updated pattern settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
)

# Black and white
GC_chart(genes_data, cluster = "cluster", group = "group", height = "150px") %>%
  GC_pattern()

# Chosen patterns on grey backgrounds
GC_chart(genes_data, cluster = "cluster", group = "group", height = "150px") %>%
  GC_pattern(
    patterns = list(A = "dots", B = "crosshatch", C = "none"),
    background = c("white", "grey80")
  )

# Patterns on top of the gene colours
GC_chart(genes_data, cluster = "cluster", group = "group", height = "150px") %>%
  GC_pattern(background = NULL)

# White patterns on black for group C
GC_chart(genes_data, cluster = "cluster", group = "group", height = "150px") %>%
  GC_pattern(
    patterns = list(A = "checkerboard", B = "rings", C = "bricks"),
    color = list(A = "black", B = "black", C = "white"),
    background = list(A = "white", B = "white", C = "black")
  )
```

GC_scale *Update Scale of a GC Chart Cluster*

Description

Modify the scale settings for specified clusters within a GC chart.

Usage

```
GC_scale(
  GC_chart,
  cluster = NULL,
  start = NULL,
  end = NULL,
  padding = NULL,
  hidden = FALSE,
  breaks = list(),
  tickValues = NULL,
  reverse = FALSE,
  axis_position = NULL,
  axis_type = "position",
  y = NULL,
  scale_breaks = FALSE,
  scale_break_threshold = 20,
  scale_break_padding = 1,
  ticksCount = NULL,
  ticksFormat = NULL,
  tickStyle = list(),
  textStyle = list(),
  lineStyle = list(),
  ...
)
```

Arguments

GC_chart	A GC chart object.
cluster	Numeric or character vector specifying clusters in the GC chart to update.
start	Numeric vector indicating the starting point for the scale. Default is NULL.
end	Numeric vector indicating the end point for the scale. Default is NULL.
padding	Numeric value or percentage string indicating the padding on either side of the scale. The value can be a number or a string in the format of '2%'. Default is NULL: as set by the theme, otherwise 2.
hidden	Logical flag indicating whether the axis is hidden. Default is FALSE.

breaks	The scale breaks: stretches of the scale that are left out, with break marks on the line. Either a list of breaks for all selected clusters, e.g. `list(list(start = 5486, end = 9336))`, or a data frame with `start` and `end` columns and, for breaks per cluster, a `cluster` column. Positions are those of the data, also with `axis_type = "range"`. Default is an empty list ().
tickValues	Numeric vector or NULL, custom tick values to be used at the top of the cluster. If NULL, the default tick values are used.
reverse	Logical flag indicating whether to reverse the scale for the corresponding cluster. Default is FALSE.
axis_position	Character string indicating the type of the axis ('top' or bottom'). Default is NULL: as set by the theme, otherwise 'bottom'.
axis_type	Character string indicating the type of the axis ('position' or 'range'). Default is 'position'.
y	Numeric value from 1 to 100 indicating the y-position of the x-axis. Default is NULL.
scale_breaks	Logical flag indicating if scale breaks should be employed. Default is FALSE.
scale_break_threshold	Numeric value indicating the threshold percentage for determining scale breaks. Default is 20.
scale_break_padding	The room left on either side of a scale break: a number, as a percentage of the region shown (without its breaks), or a string in base pairs, e.g. `"500bp"`, for the same room around every break. Default is 1.
ticksCount	Numeric value indicating the number of ticks on the scale. Default is NULL: as set by the theme, otherwise 10.
ticksFormat	Format for tick labels; depends on axis_type, defaulting to ",.0f" for 'position' or ".2s" for 'range' when NULL.
tickStyle	List specifying the style for the ticks.
textStyle	List specifying the style for the tick text.
lineStyle	List specifying the style for the axis line.
...	Additional arguments for scale settings.

Value

Updated GC chart with new scale settings.

Examples

```
genes_data <- data.frame(
  start = c(100, 1000, 2000),
  end = c(150, 1500, 2500),
  name = c('Gene 4', 'Gene 5', 'Gene 6'),
  group = c('B', 'A', 'C'),
  cluster = c(2, 2, 2)
)
#Example usage with all custom options
```

```
GC_chart(genes_data, cluster = "cluster", group = "group", height = "200px") %>%
  GC_scale(
    start = 1,
    end = 2600,
    padding = 2,
    hidden = FALSE,
    reverse = FALSE,
    axis_position = "bottom",
    # breaks = list(
    #  list(start = 160, end = 900),
    #  list(start = 1600, end = 1900)
    # ),
    # tickValues = c(1, 2600),
    scale_breaks = TRUE,
    scale_break_threshold = 20,
    scale_break_padding = 1,
    ticksCount = 10,
    ticksFormat = ",.0f",
    y = NULL,
    tickStyle =
      list(
        stroke = "grey",
        strokeWidth = 1,
        lineLength = 6
        # Any other CSS styles
      ),
    textStyle =
      list(
        fill = "black",
        fontSize = "10px",
        fontFamily = "Arial",
        cursor = "default"
        # Any other CSS styles
      ),
    lineStyle = list(
      stroke = "grey",
      strokeWidth = 1
      # Any other CSS styles
    )
  ) %>%
  GC_legend(FALSE)
```

GC_scaleBar *Update Scale Bar of a GC Chart Cluster*

Description

Modify the scale bar settings for specified clusters within a GC chart.

Usage

```
GC_scaleBar(
  GC_chart,
  show = TRUE,
  cluster = NULL,
  scaleBarLineStyle = list(),
  scaleBarTickStyle = list(),
  labelStyle = list(),
  ...
)
```

Arguments

GC_chart	A GC chart object.
show	Logical. Whether to show the scale bar.
cluster	Numeric or character vector. Clusters in the GC chart to update.
scaleBarLineStyle	List of style options for the scale bar line.
scaleBarTickStyle	List of style options for the scale bar ticks.
labelStyle	List of style options for the scale bar label.
...	Additional arguments for scale bar settings.

Value

Updated GC chart with new scale bar settings.

Examples

```
genes_data <- data.frame(
 start = c(1000, 9000, 13000, 17000, 21000),
 end = c(4000, 12000, 16000, 20000, 24000),
 name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
 group = c('A', 'B', 'B', 'A', 'C'),
 cluster = c(1, 1, 2, 2, 2)
)

# Set scale bar for individual clusters
GC_chart(genes_data, cluster ="cluster", group = "group") %>%
GC_scaleBar(cluster = 1, title = "1 kb", scaleBarUnit = 1000) %>%
GC_scaleBar(cluster = 2, title = "2 kb", scaleBarUnit = 2000)

# Style scale bar
GC_chart(genes_data, cluster ="cluster", group = "group", height = "400px") %>%
  GC_scaleBar(
    title = "1kb",
    scaleBarUnit = 1000,
    cluster = NULL,
    x = 0,
```

```
    y = 50,
    labelStyle = list(
      labelPosition =  "left",
      fontSize = "10px",
      fontFamily = "sans-serif",
      fill = "red", # Text color
      cursor = "default"
      # Any other CSS style for the label
    ),
    textPadding = -2,
    scaleBarLineStyle = list(
      stroke = "black",
      strokeWidth = 1
      # Any other CSS style for the line
    ),
    scaleBarTickStyle = list(
      stroke = "black",
      strokeWidth = 1
      # Any other CSS style for the tick
    )
  )
```

GC_sequence *Update Sequence Display of a GC Chart Cluster*

Description

Modify the sequence display and break markers of specified clusters within a GC chart.

Usage

```
GC_sequence(
  GC_chart,
  show = TRUE,
  cluster = NULL,
  y = NULL,
  sequenceStyle = list(),
  markerStyle = list(),
  ...
)
```

Arguments

GC_chart	A GC chart object.
show	Logical, whether to display the sequence (default is TRUE).
cluster	Numeric or character vector specifying clusters to update.

y	Vertical position of the sequence line. Default is NULL: as set by the theme, otherwise 50.
sequenceStyle	A list of styling options for the sequence line.
markerStyle	A list of styling options for the sequence break markers, e.g. markerHeight, tiltAmount and gap (distance in pixels between each marker and the break), plus any CSS style.
...	Additional customization arguments for sequence display.

Details

This function allows customization of the sequence line and break markers in a GC chart. It offers options to adjust the sequence line (sequenceStyle) and break markers (markerStyle). The y parameter can be used to set the vertical position of the sequence.

Value

An updated GC chart with modified sequence display settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

# Basic usage
GC_chart(genes_data, cluster ="cluster", group = "group", height = "200px") %>%
GC_labels("name") %>%
GC_sequence(show = TRUE, y = 50, cluster = NULL)

# Customize sequence and marker styles
GC_chart(genes_data, cluster="cluster", group = "group", height = "200px") %>%
  GC_scale(hidden = TRUE, scale_breaks = TRUE) %>%
  GC_sequence(
    start = NULL,
    end = NULL,
    sequenceStyle = list(
      stroke = "blue",
      strokeWidth = 1
      # Any other CSS style
    ),
    markerStyle = list(
      stroke = "blue",
      strokeWidth = 1,
      gap = 3,
      tiltAmount = 5
      # Any other CSS style
    )
```

```
) %>%
GC_legend(FALSE)
```

GC_theme *Set the Theme of a GC Chart*

Description

A theme sets the default look of every part of a chart: fonts, text and line colours, the background, the gene colours and the options of each component. Options set with the other GC_ functions (e.g. GC_labels(fontSize =)) always take precedence over the theme, whatever the order of the calls.

Usage

```
GC_theme(
  GC_chart,
  theme = NULL,
  textScale = NULL,
  fontFamily = NULL,
  textColor = NULL,
  lineColor = NULL,
  background = NULL,
  palette = NULL,
  pattern = NULL,
  ...
)
```

Arguments

GC_chart	The GC chart object to be modified.
theme	The theme to start from: the name of a built-in theme ("default", "classic", "dark", "vintage", "infographic", "publication" or "bw", see geneviewer_theme()) or a list of theme settings. "bw" is black and white: genes, exons and legend markers are filled with a pattern per group. Default is NULL: change the current theme of the chart.
textScale	Text size, as a factor of the default font sizes: e.g. 1.5 makes all text one and a half times as large. Font sizes set in the theme are scaled too; those set with the other GC_ functions are not. The title and its default height grow with the text.
fontFamily	Font family of all text, e.g. "Arial".
textColor	Colour of all text.
lineColor	Colour of the sequence line, the axis, the coordinates, the scale bar and the lines from spread labels to their genes (GC_labels(adjustLabels = "spread")).
background	Background colour of the chart, also the fill of the UTRs of transcripts.

palette — Gene colours: the name of a d3 colour scheme (e.g. "schemeSet2") or a geneviewer palette (e.g. "okabeIto", see geneviewer_palette()), or a vector of colours, used in the order of the groups. Colours set with GC_color() or the colorScheme or customColors of GC_genes() take precedence.

pattern — Patterns of the genes, exons and legend markers: a list with the arguments of GC_pattern() (patterns, color, background, size, strokeWidth), e.g. list(patterns = c("dots", "grid")). Set with the theme, they are merged with the patterns of the current theme; GC_pattern() takes precedence. FALSE removes the patterns of the theme.

... — Options per component, as lists named after the component (see Details), e.g. labels = list(fontSize = "10px").

Details

Call GC_theme() with a theme to start from that theme, or without one to change the current theme. The global settings (fontFamily, textColor, lineColor, background, palette, pattern) are applied to all components; options set for a component, in the theme or in ..., take precedence over them. textScale scales all text of the chart: the default font sizes and those set in the theme, not those set with the other GC_ functions.

The options of each component are those of its GC_ function, by their argument name (e.g. normal_color) or their name in the browser (e.g. normalColor). Components and their functions:

Component	Function	Example
chart	GC_chart()	list(style = list(padding = "10px"))
grid	GC_grid()	list(margin = list(left = "80px"))
cluster	GC_cluster()	list(separate_strands = TRUE)
title	GC_title()	list(height = 40, titleFont = list(fontSize = "18px"))
clusterTitle	GC_clusterTitle()	list(titleFont = list(fill = "grey"))
clusterFooter	GC_clusterFooter()	list(align = "center")
clusterLabel	GC_clusterLabel()	list(width = "120px", fontWeight = "normal")
sequence	GC_sequence()	list(sequenceStyle = list(stroke = "black"))
genes	GC_genes()	list(marker = "rbox", stroke = "none")
transcript	GC_transcript()	list(styleExons = list(stroke = "black"))
labels	GC_labels()	list(fontSize = "10px", leaderLine = list(stroke = "grey"))
coordinates	GC_coordinates()	list(textStyle = list(fontSize = "8px"))
scale	GC_scale()	list(axis_position = "top", ticksCount = 5)
scaleBar	GC_scaleBar()	list(labelStyle = list(fontSize = "12px"))
legend	GC_legend()	list(position = "top")
links	GC_links()	list(normal_color = "grey", curve = FALSE)
tooltip	GC_tooltip()	list(backgroundColor = "black", color = "white")
trackMouse	GC_trackMouse()	list(cursor = "cell")
toolbar	GC_toolbar()	list(color = "#9e9e9e", hoverColor = "black")
annotations	GC_annotation()	list(text = list(style = list(fontSize = "14px")))

annotations takes a list per annotation type: text, line, arrow, textMarker, symbol, rectangle, promoter, terminator and arc.

A theme only sets how things look: it does not add components to the chart (e.g. a scale needs GC_scale()) or change the data.

The "publication" theme follows common journal figure guidelines: Arial or Helvetica, 7 pt text when the chart is used at its final print size, black text and keylines, visible axis ticks, no patterns or shadows, and the colour-blind-safe Wong palette. Gene labels are italic, as gene symbols are in journals; use GC_labels(fontStyle = "normal") for protein names or other labels. Journals often also require axis labels with units in parentheses and bold panel labels; these are content-specific and must be added by the author. Export as SVG to retain editable vector text and artwork, then check that fonts are embedded when converting to the PDF or EPS supplied to the journal.

Value

The GC chart object with the theme.

See Also

geneviewer_theme() to see the settings of a built-in theme.

Examples

```
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 2, 2, 2)
)

chart <- GC_chart(genes_data, cluster = "cluster", group = "group",
                  height = "200px") %>%
  GC_labels("name") %>%
  GC_clusterLabel()

# A built-in theme
chart %>% GC_theme("dark")

# A built-in theme with other fonts and colours
chart %>% GC_theme("classic", fontFamily = "Georgia",
                   palette = c("#1b9e77", "#d95f02", "#7570b3"))

# Black and white with patterns, and other patterns
chart %>% GC_theme("bw")
chart %>% GC_theme("bw", pattern = list(patterns = c("dots", "crosshatch", "none")))

# Larger text, e.g. for a slide or a figure that is printed small
chart %>% GC_theme("classic", textScale = 1.4)

# Change one component; options of the GC_ functions still take precedence
chart %>%
  GC_theme("classic", labels = list(fontSize = "14px", fill = "grey")) %>%
  GC_labels("name", fontSize = "10px")
```

GC_title *Add Title to a GC_chart*

Description

Modify the cluster title and subtitle of specified clusters within a GC chart and adjust the display settings.

Usage

```
GC_title(
  GC_chart,
  title = NULL,
  subtitle = NULL,
  style = list(),
  subtitleFont = list(),
  titleFont = list(),
  show = TRUE,
  height = NULL,
  align = NULL,
  spacing = NULL,
  ...
)
```

Arguments

GC_chart	A GC chart object.
title	Character vector. Titles to set for the clusters.
subtitle	Character vector. Subtitles to set for the clusters.
style	A list of CSS styles to be applied to the chart container. Each element of the list should be a valid CSS property-value pair. For example, list(backgroundColor = "white", border = "2px solid black"). Default is an empty list.
subtitleFont	List. Settings for the subtitle font.
titleFont	List. Settings for the title font.
show	Logical. Whether to display the title. Default is TRUE.
height	Character. Height for the title (e.g., "50px"). Default is NULL: the height of the theme, otherwise 50 pixels with a subtitle and 40 without.
align	Character. Horizontal alignment of the title and subtitle, "left", "center" or "right". x moves them from this position. Default is NULL: the alignment of the theme, otherwise "center".
spacing	Numeric. Pixels between the title and the subtitle. Default is NULL: as set by the theme, otherwise 20.
...	Additional customization arguments for title and subtitle.

Value

Updated GC chart with new title settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
)

# Basic usage
GC_chart(genes_data, cluster ="cluster", group = "group", height = "400px") %>%
GC_labels("name") %>%
GC_title(
  title = "Cluster 1 Data",
  subtitle = "Detailed View",
  show = TRUE
)

# Customizing title style
GC_chart(genes_data, cluster = "cluster", group = "group", height = "500px") %>%
  GC_labels("name") %>%
  GC_title(
    title = "Cluster 1 Data",
    subtitle = "Detailed View",
    show = TRUE,
    height = "50px",
    cluster = 1,
    x = 0,
    y = 25, # height / 2
    align = "center",
    spacing = 20,
    style = list(
      backgroundColor = "#0000"
      # Any other CSS styles
    ),
    titleFont = list(
      fontSize = "16px",
      fontStyle = "normal",
      fontWeight = "bold",
      textDecoration = "normal",
      fontFamily = "sans-serif",
      cursor = "default",
      fill = "black"
      # Any other CSS styles
    ),
    subtitleFont = list(
      fontSize = "14px",
      fontStyle = "normal",
```

```
    fontWeight = "bold",
    textDecoration = "normal",
    fontFamily = "sans-serif",
    cursor = "default",
    fill = "black"
    # Any other CSS styles
  )
)
```

GC_toolbar *Set the Toolbar of a GC Chart*

Description

The toolbar holds the chart's own controls, in a corner of the chart: undo, redo and reset of moves, with `GC_events(drag = TRUE)`, and the save button of `GC_chart(save_button = TRUE)`. `GC_toolbar()` sets when and where it shows, which buttons it has, its colours and size, and the icons of its buttons. Colours and size can also be set for all charts of a theme, with `GC_theme(toolbar = list(...))`; `GC_toolbar()` takes precedence.

Usage

```
GC_toolbar(
  GC_chart,
  show = NULL,
  position = NULL,
  buttons = NULL,
  size = NULL,
  color = NULL,
  hoverColor = NULL,
  tooltipBackground = NULL,
  tooltipColor = NULL,
  tooltipBorder = NULL,
  tooltipRadius = NULL,
  tooltipShadow = NULL,
  fontFamily = NULL,
  icons = NULL
)
```

Arguments

GC_chart	A GC chart object.
show	When the toolbar shows: `"hover"` (default) while the mouse is over the chart or one of its buttons has keyboard focus, `"always"` (e.g. on touch screens), or `"never"`. Keyboard shortcuts work also when it does not show.
position	The corner of the chart: `"topright"` (default), `"topleft"`, `"bottomright"` or `"bottomleft"`.

buttons — The buttons, of "undo", "redo", "reset", "zoom" and "save" (default: all). Undo, redo and reset show with GC_events(drag = TRUE), the zoom button (reset zoom) with GC_zoom() while zoomed in, the save button unless GC_chart(save_button = FALSE).

size — Size of the icons in pixels (default 20).

color — Colour of the icons (default "#D3D3D3").

hoverColor — Colour of an icon under the mouse (default "#555555").

tooltipBackground — Background colour of the tooltips and of the menu of the save button (default "#333333").

tooltipColor — Text colour of the tooltips and the menu (default "white").

tooltipBorder — CSS border of the tooltips and the menu, e.g. "1px solid #b7b9be" (default none).

tooltipRadius — Corner radius of the tooltips and the menu: pixels, or a CSS length (default 3).

tooltipShadow — CSS box shadow of the tooltips and the menu, e.g. "1px 2px 10px rgba(0, 0, 0, 0.2)" (default none).

fontFamily — Font of the tooltips and the menu (default Arial).

icons — Icons of the buttons: a named list or character vector, with names of buttons, e.g. list(reset = "arrows-to-circle", save = "camera"). Buttons that are not named keep their icon.

Details

Icons are Font Awesome icons, by name, e.g. "arrow-rotate-left" for a solid icon, or by their classes, e.g. "fa-regular fa-floppy-disk" for another style.

Value

The GC chart object with the toolbar settings.

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90),
  end = c(40, 80, 120),
  name = c("Gene 1", "Gene 2", "Gene 3")
)
GC_chart(genes_data, height = "150px") %>%
  GC_labels("name") %>%
  GC_events(select = TRUE, drag = TRUE) %>%
  GC_toolbar(show = "always", position = "topleft", size = 16,
             color = "#9e9e9e", hoverColor = "black",
             icons = list(reset = "arrows-to-circle", save = "camera"))
```

GC_tooltip *Set Tooltip for a Gene Chart*

Description

This function configures the tooltip for a gene chart.

Usage

```
GC_tooltip(
  GC_chart,
  formatter = "<b>Start:</b> {start}<br><b>End:</b> {end}",
  show = TRUE,
  cluster = NULL,
  ...
)
```

Arguments

GC_chart	The gene chart object to be modified.
formatter	A character string defining the HTML content of the tooltip. It can include placeholders like {start} and {end} which will be replaced by actual data values. The default value shows start and end data.
show	Logical, whether to display the tooltip or not.
cluster	Optional; used to specify which clusters in the chart should have tooltips.
...	Additional arguments that can be used to further customize the tooltip.

Value

Returns the gene chart object with the tooltip configured.

Examples

```
# Set tooltip
genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5'),
  group = c('A', 'B', 'B', 'A', 'C')
)

# Add tooltips to the gene chart
GC_chart(genes_data, group = "group", height = "200px") %>%
GC_tooltip(formatter = " <b>Start:</b> {start}<br><b>end:</b> {end}")
```

GC_trackMouse *Track Mouse Movement in a GC_chart*

Description

This function enables or disables mouse tracking on specified clusters within a GC chart. When enabled, the x and y coordinates of the mouse are displayed which can be used to place annotations.

Usage

```
GC_trackMouse(GC_chart, show = TRUE, cluster = NULL, ...)
```

Arguments

GC_chart	A GC chart object to which the annotations will be added.
show	Logical, specifies whether to track the mouse or not.
cluster	Numeric or character vector specifying the clusters to which annotations should be added.
...	CSS styles of the coordinate tooltip, such as backgroundColor, fontSize or color, and cursor for the cursor over the cluster (default "crosshair").

Value

Updated GC chart object with mouse tracking settings.

See Also

GC_annotation

Examples

```
genes_data <- data.frame(
  start = c(10, 50, 90, 130, 170, 210),
  end = c(40, 80, 120, 160, 200, 240),
  name = c('Gene 1', 'Gene 2', 'Gene 3', 'Gene 4', 'Gene 5', 'Gene 6'),
  group = c('A', 'A', 'B', 'B', 'A', 'C'),
  cluster = c(1, 1, 1, 2, 2, 2)
)

# Enable mouse tracking
GC_chart(genes_data, cluster = "cluster", group = "group", height = "220px") %>%
GC_trackMouse()
```

GC_transcript *Modify Transcript Characteristics within a Chart*

Description

This function updates the chart with specific characteristics for transcripts based on the given parameters. It can show/hide transcripts, apply a color scheme, assign custom colors, filter by cluster, and accept additional options.

Usage

```
GC_transcript(
  GC_chart,
  transcript = NULL,
  type = NULL,
  strand = NULL,
  group = NULL,
  selection = NULL,
  show = TRUE,
  colorScheme = NULL,
  customColors = NULL,
  styleExons = list(),
  styleIntrons = list(),
  styleUTRs = list(),
  itemStyleExons = list(),
  itemStyleIntrons = list(),
  itemStyleUTRs = list(),
  groupStyleExons = NULL,
  groupStyleIntrons = NULL,
  groupStyleUTRs = NULL,
  style_by = NULL,
  labelOptions = list(),
  ...
)
```

Arguments

GC_chart	The chart object to be modified.
transcript	Optional column name used for clustering transcript data. Default is NULL.
type	Column name identifying the feature type ("UTR", "exon"). Default is NULL.
strand	Optional column name indicating strand orientation. Acceptable values include 1, 'forward', 'sense', or '+' to represent the forward strand, and -1, 0, 'reverse', 'antisense', "complement" or '-' to represent the reverse strand. Default is NULL, meaning strand information is not used.
group	Optional column name used for transcript grouping to influence color aesthetics.

selection	Numeric or character, the specific transcript to filter transcripts by.
show	Logical, whether to show the transcripts or not.
colorScheme	Character or NULL, the name of the color scheme to use: a d3 scheme (e.g. "schemeSet2") or a geneviewer palette (e.g. "okabeIto", see geneviewer_palette()). When no selection is made, the legend uses the same colours. NULL keeps the colours set by GC_color().
customColors	List or NULL, custom colors to apply to the genes: a vector of colours, recycled over the groups, or a named list of colours by group. Groups left out of a named list keep the colours of the colour scheme. When no selection is made, the legend uses the same colours.
styleExons	List, styles to apply to exons in the chart.
styleIntrons	List, styles to apply to introns in the chart.
styleUTRs	List, styles to apply to UTRs in the chart.
itemStyleExons	List, a list of styles to apply to individual exons in the chart.
itemStyleIntrons	List, a list of styles to apply to individual introns in the chart.
itemStyleUTRs	List, a list of styles to apply to individual UTRs in the chart.
groupStyleExons, groupStyleIntrons, groupStyleUTRs	Named lists of styles by group, applied to all exons, introns or UTRs of that group (of group, or of the column given in style_by), e.g. groupStyleExons = list(tx1 = list(fill = "red")). They take the same options as the matching itemStyle, which takes precedence.
style_by	Column name or NULL; the column whose values the group styles are matched against. Default is NULL: group.
labelOptions	List, options for styling labels such as font size, color, and position.
...	Additional arguments to be passed to the gene options.

Details

The first call of GC_transcript() makes the chart again with the transcripts as clusters. Settings of the whole chart made before it (e.g. GC_title(), GC_legend(), GC_theme() and the margins of GC_grid()) are kept. Settings per cluster (e.g. GC_labels(), GC_clusterTitle()) are for the old clusters: set them after GC_transcript().

Value

Returns the modified gene chart object.

Examples

```
transcript_data <- data.frame(
  transcript = c("transcript1", "transcript1", "transcript1", "transcript1",
                 "transcript2", "transcript2", "transcript2"),
  type = c("5_utr", "exon", "exon", "3_utr",
           "5_utr", "exon", "3_utr"),
  start = c(1, 101, 201, 301,
```

```
            1, 101, 301),
  end = c(50, 150, 250, 350,
          50, 150, 350),
  strand = rep("forward", 7)
)

# All default transcript settings
GC_chart(
  transcript_data,
  start = "start",
  end = "end",
  height = "200px"
) %>%
  GC_transcript(
    transcript = "transcript",
    strand = "strand",
    type = "type",
    group = NULL,
    show = TRUE,
    selection = NULL,
    colorScheme = NULL,
    customColors = NULL,
    styleExons = list(
      show = TRUE,
      strokeWidth = 0,
      cursor = "default",
      marker = "box",
      markerSize = "medium",
      arrowheadWidth = NULL,
      arrowheadHeight = NULL,
      markerHeight = NULL,
      cornerRadius = NULL
      # Any other CSS style
    ),
    styleIntrons = list(
      show = TRUE,
      strokeWidth = 1,
      fill = "none",
      cursor = "default",
      marker = "intron",
      markerSize = "medium",
      arrowheadWidth = NULL,
      arrowheadHeight = NULL,
      markerHeight = NULL,
      cornerRadius = NULL
      # Any other CSS style
    ),
    styleUTRs = list(
      show = TRUE,
      fontSize = "10px",
      fontStyle = "normal",
      fontFamily = "sans-serif",
      cursor = "default",
```

```
      color = "black",
      fill = "#FFF",
      strokeWidth = 1,
      marker = "box",
      markerSize = "medium",
      arrowheadWidth = NULL,
      arrowheadHeight = NULL,
      markerHeight = NULL,
      cornerRadius = NULL
      # Any other CSS style
    ),
    labelOptions = list(
      show = TRUE,
      xOffset = 2,
      yOffset = 0,
      fontSize = "12px",
      fontStyle = "normal",
      fontWeight = "normal",
      fontFamily = "sans-serif",
      cursor = "default",
      color = "black"
    ),
    itemStyleExons = list(),
    itemStyleIntrons = list(),
    itemStyleUTRs = list()
  )
# Change the appearance of a specific intron
GC_chart(transcript_data,
         start = "start",
         end = "end",
         height = "200px"
) %>%
  GC_transcript(
    transcript = "transcript",
    type = "type",
    selection = 2,
    itemStyleExons = list(list(index = 0, fill = "red")
    )
  )
```

GC_zoom *Add a Zoom Slider to a GC Chart*

Description

GC_zoom() adds a slider below the clusters to zoom in on part of them. Drag a handle to change the window, drag the window to move it, click the track to move the window there, and double-click the slider or use the reset zoom button of the toolbar (GC_toolbar()) to show everything again. With the keyboard, the arrow keys move a handle or the window by 1% of the slider (Shift: 10%), Home and End to the ends.

Usage

```
GC_zoom(
  GC_chart,
  show = TRUE,
  range = NULL,
  mode = NULL,
  min = NULL,
  update = NULL,
  height = NULL,
  overview = NULL,
  color = NULL,
  style = NULL,
  windowStyle = NULL,
  handleStyle = NULL
)
```

Arguments

GC_chart	A GC chart object.
show	Logical, whether to show the slider. FALSE removes it.
range	The window shown first, as c(start, end) in the units of mode. NULL (default) shows everything. A Shiny app can keep the window the user picked with range = input$<outputId>_zoom$range (see GC_events()(zoom = TRUE)). The chart keeps the window the user picked when it is drawn again with the same data, the same range and mode and the same GC_events()(render_id =); a new render_id marks a new load, which starts at range again.
mode	How the window applies to the clusters: "relative" (default) zooms every cluster to the same share of its own range, e.g. range = c(0.25, 0.5) for its second quarter, which suits clusters of different genomes. "absolute" shows the same base pairs in every cluster, e.g. range = c(12000, 18000), and the slider spans the range of all clusters.
min	The narrowest window, in the units of mode (default 1% of the slider).
update	"live" (default) draws the chart while the slider moves, "release" when it is let go, for large charts.
height	Height of the slider in pixels (default 24). The clusters make room for it.
overview	Logical, whether the slider shows the genes of each cluster, one row per cluster, in the colours of their group (default TRUE).
color	One colour for the genes of the overview instead of their group colours.
style, windowStyle, handleStyle	Named lists of CSS styles of the slider track, the window and the two handles, e.g. style = list(backgroundColor = "#eeeeee").

Details

The chart is drawn again for each window: genes, labels, links, coordinates and annotations follow the window, and parts outside it are cut off at the edges of the plot area. The axis shows the ticks

of the window. The window of GC_scale()(start, end) is the whole range the slider covers. Text moved by dragging (GC_events()(drag = TRUE)) keeps its offset. Saved images leave out the slider and the room it takes.

In Shiny, GC_events()(zoom = TRUE) reports the window, and GC_zoom_to() sets it from the server.

Value

The GC chart object with the zoom slider.

See Also

GC_zoom_to(), GC_events(), GC_scale()

Examples

```
GC_chart(ophA_clusters, cluster = "cluster", group = "class",
         height = "300px") %>%
  GC_labels("name") %>%
  GC_zoom(range = c(0.2, 0.6))

# The same base pairs in every cluster
GC_chart(ophA_clusters, cluster = "cluster", group = "class",
         height = "300px") %>%
  GC_zoom(mode = "absolute", update = "release", height = 30,
          handleStyle = list(backgroundColor = "#d9d9d9"))
```

GC_zoom_to *Set the Zoom of a GC Chart in Shiny*

Description

Moves the zoom slider of GC_zoom() in a running Shiny app, without drawing the chart again from R.

Usage

```
GC_zoom_to(outputId, range = NULL, session = shiny::getDefaultReactiveDomain())
```

Arguments

outputId	The output ID of the chart, as in GC_chartOutput().
range	The window, as c(start, end) in the units of the mode of GC_zoom(): shares of the clusters ("relative") or base pairs ("absolute"). NULL shows everything.
session	The Shiny session.

Value

Nothing; called for its effect.

See Also

GC_zoom()

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    actionButton("first", "First half"),
    actionButton("all", "Everything"),
    GC_chartOutput("chart", height = "300px")
  )
  server <- function(input, output, session) {
    output$chart <- renderGC_chart(
      GC_chart(ophA_clusters, cluster = "cluster", group = "class") %>%
        GC_zoom() %>%
        GC_events(zoom = TRUE)
    )
    observeEvent(input$first, GC_zoom_to("chart", c(0, 0.5)))
    observeEvent(input$all, GC_zoom_to("chart", NULL))
    observeEvent(input$chart_zoom, str(input$chart_zoom))
  }
  shinyApp(ui, server)
}
```

---

genbank_to_fasta *Convert GenBank to FASTA Format*

Description

This function reads a GenBank file, extracts the sequence and writes it to a new file in FASTA format. It parses the DEFINITION and VERSION for the header and sequences from the ORIGIN section.

Usage

```
genbank_to_fasta(path, output_dir = tempdir())
```

Arguments

path	Path to the GenBank file.
output_dir	Directory for the output FASTA file, which gets the base name of the input file with a .fasta extension. Defaults to tempdir(). Use NULL to save it next to the input file.

Value

The path of the FASTA file, invisibly.

Examples

```
# Path to the example GenBank file in the package
gbk_file <- system.file("extdata", "BGC0000001.gbk", package = "geneviewer")

# Convert the GenBank file to FASTA format in a temporary folder
genbank_to_fasta(gbk_file, output_dir = tempdir())
```

---

geneviewer-shiny *Shiny bindings for geneviewer*

Description

Output and render functions for using geneviewer within Shiny applications and interactive Rmd documents.

Usage

```
GC_chartOutput(outputId, width = "100%", height = "400px")

renderGC_chart(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	Output variable to read from.
width	Width of the output widget, must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
height	Height of the output widget, must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a GC chart.
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Details

In Shiny, a fixed pixel width set by GC_chart() (or GC_grid()) takes precedence over the output width. Put the output in a container with overflow-x: auto to scroll a chart wider than the available space. Responsive charts keep the width set by GC_chartOutput(), including when switching back from a fixed width. Standalone widgets keep their usual htmlwidgets sizing. Height behavior is unchanged.

Value

GC_chart widget that can be placed in the UI.

See Also

GC_chart()

Examples

```
if (interactive()) {
  library(shiny)
  library(geneviewer)

  ui <- fluidPage(
    titlePanel("Omphalotin Gene Cluster Visualization"),
    mainPanel(
      GC_chartOutput("gcChart", width = "100%", height = "500px")
    )
  )

  server <- function(input, output) {
    output$gcChart <- renderGC_chart({
      GC_chart(
        ophA_clusters, # Ensure 'ophA_clusters' data is defined or available
        cluster = "cluster",
        group = "class"
      ) %>%
      GC_title(title = c("<i>O. olearius</i>", "<i>D. bispora</i>")) %>%
      GC_labels("name") %>%
      GC_legend(position = "bottom") %>%
      GC_scaleBar() %>%
      GC_clusterLabel(title = "ophA")
    })
  }

  shinyApp(ui = ui, server = server)
}
```

---

geneviewer_palette *Colours of a geneviewer Palette*

Description

Besides the d3 colour schemes (e.g. "schemeSet2"), the gene colours can be set by the name of a geneviewer palette, in GC_theme() (palette) and in the colorScheme of GC_genes(), GC_transcript() and GC_color():

Usage

```
geneviewer_palette(name = NULL)
```

Arguments

name	Name of the palette. Default is NULL: the names of all palettes.

Details

- "okabeIto": 8 colours that stay distinct with colour blindness (Okabe & Ito, also known as the Wong palette), the palette of the "publication" theme.
- "classic": 9 clear colours, the palette of the "classic" theme.
- "vivid": 9 bright colours for dark backgrounds.
- "dark": 11 soft colours for dark backgrounds, the palette of the "dark" theme.
- "vintage": 10 muted colours, the palette of the "vintage" theme.
- "infographic": 15 strong colours, the palette of the "infographic" theme.
- "greys": 6 greys, from dark to light.
- "blues": 7 blues, from dark to light.

Value

The colours of the palette, or the names of all palettes.

Examples

```
geneviewer_palette("okabeIto")
geneviewer_palette()

genes_data <- data.frame(
  start = c(10, 90, 130, 170, 210),
  end = c(40, 120, 160, 200, 240),
  group = c('A', 'B', 'B', 'A', 'C')
)
GC_chart(genes_data, group = "group", height = "100px") %>%
  GC_genes(colorScheme = "okabeIto")
```

---

geneviewer_theme *Settings of a Built-in Theme*

Description

Returns the settings of a built-in theme of GC_theme(), to see them or to start a theme of your own.

Usage

```
geneviewer_theme(name = "default")
```

Arguments

name	Name of the theme: "default", "classic", "dark", "vintage", "infographic", "publication" or "bw".

Value

A list of theme settings, to use as the theme of GC_theme().

Examples

```
geneviewer_theme("dark")

# A theme of your own, based on "classic"
my_theme <- utils::modifyList(geneviewer_theme("classic"),
                              list(textColor = "navy"))
```

---

get_introns *Calculate Intron Positions Based on Exons*

Description

This function takes a dataset containing transcript information and calculates the start and end positions of introns based on the positions of exons. It generates a combined dataframe including both the original exons and the calculated introns. Overlapping, nested, and adjacent exon/UTR intervals are treated as a continuous covered region; only positive gaps become introns.

Usage

```
get_introns(data)
```

Arguments

data	A dataframe containing start and end positions of transcripts. Optional columns are type (exon or UTR), transcript (containing unique transcript IDs), and strand (indicating the direction, forward or reverse, of each transcript).

Value

A dataframe with the original exons and the calculated introns, sorted by transcript and start position. Each row includes the start, end, type (exon, UTR or intron), transcript, and strand for each segment.

Examples

```
exons <- data.frame(
  transcript = "tx1",
  type = "exon",
  start = c(100, 400, 900),
  end = c(250, 600, 1100),
  strand = "forward"
)
get_introns(exons)
```

---

hs_dystrophin_transcripts *Human Dystrophin Transcripts Data*

Description

This dataset contains the Exon positions on human Dystrophin transcripts 'Dp427p2', 'Dp260-2', 'Dp140', 'Dp116', and 'Dp71'.

Usage

```
hs_dystrophin_transcripts
```

Format

A data frame with 202 rows and 5 columns:

transcript Transcript names. A character vector representing the names of the transcripts.

type Transcript type.

start Start positions. An integer vector showing the starting positions of each Exon.

end End positions. An integer vector showing the ending positions of each Exon.

length Transcript lengths. An integer vector indicating the length of each transcript.

---

human_hox_genes *Human HOX Gene Cluster Data*

Description

This dataset represents the positions and clusters of various human HOX genes. HOX genes are a group of related genes that control the body plan of an embryo along the head-tail axis. The dataset includes genes from HOXA, HOXB, HOXC, and HOXD clusters.

Usage

```
human_hox_genes
```

Format

A data frame with 39 rows and 6 columns:

symbol HGNC symbols for the HOX genes. A character vector.

chromosome_name Chromosome number where each gene is located. A numeric vector.

start Start positions of the genes on the chromosome. A numeric vector.

end End positions of the genes on the chromosome. A numeric vector.

strand -1 for reverse and 1 for forward.

cluster Cluster identification as a character vector, specifying the HOX gene cluster (HOXA, HOXB, HOXC, HOXD).

name Simplified names for the HOX genes. A character vector.

`mitochondrial_gene_order`

Human and Locust Mitochondrial Gene Order

Description

The 13 protein-coding genes and two ribosomal RNAs of each mitochondrial genome compared in Figure 3 of Jameson et al. (2003). Rows follow circular gene order, starting at COX1, with the locust first. tRNAs are excluded. Coordinates remain those of the original RefSeq records, before rotation. Blocks reproduce the colours of the published comparison; they do not imply a reconstructed sequence of evolutionary events.

Usage

```
mitochondrial_gene_order
```

Format

A data frame with 30 rows and 9 columns:

cluster Species name.

accession Versioned RefSeq accession.

genome_length Complete mitochondrial genome length in base pairs.

gene Gene name; RNS and RNL denote the small and large rRNAs.

product Gene product from the RefSeq annotation.

block Gene block coloured in the published comparison.

start, end Original 1-based inclusive genomic coordinates.

strand Either "forward" or "complement".

Source

RefSeq NC_001712.1 and NC_012920.1, read with `read_gbk()`.

References

Jameson D, Gibson AP, Hudelot C, Higgs PG. OGRe: a relational database for comparative analysis of mitochondrial genomes. Nucleic Acids Res. 2003;31:202-206. doi:10.1093/nar/gkg077.

Examples

```
# Rotate each circular genome to start at COX1
cox1 <- with(subset(mitochondrial_gene_order, gene == "COX1"),
             setNames(start, cluster))
genes <- within(mitochondrial_gene_order, {
  plot_start <- (start - cox1[cluster]) %% genome_length + 1
  plot_end <- plot_start + end - start
})
GC_chart(genes, start = "plot_start", end = "plot_end", cluster = "cluster",
         group = "block", strand = "strand") %>%
  GC_labels("gene") %>%
  GC_clusterLabel()
```

---

mummer_alignment *Perform Sequence Alignment Using MUMmer*

Description

This function orchestrates the alignment of sequences in a specified directory using MUMmer, a tool for aligning large DNA or protein sequences. It can handle GenBank and FASTA file formats. and performs checks to ensure necessary files are present.

Usage

```
mummer_alignment(
  path,
  cluster = NULL,
  maptype = "many-to-many",
  seqtype = "protein",
  mummer_options = "",
  filter_options = "",
  remove_files = TRUE,
  output_dir = tempdir()
)
```

Arguments

path	The directory containing the sequence files.
cluster	Optional vector of cluster names to consider for alignment. If NULL, clusters are inferred from file names. The order of names determines the alignment sequence.

maptype	The type of mapping to perform; "many-to-many" or "one-to-one". "many-to-many" allows for multiple matches between clusters, "one-to-one" restricts alignments to unique matches between a pair.
seqtype	The type of sequences, either "protein" or "nucleotide".
mummer_options	Additional command line options for MUMmer. To see all available options, you can run `nucmer --help` or `promer --help` in the terminal depending on whether you are aligning nucleotide or protein sequences.
filter_options	Additional options for filtering MUMmer results. To view all filtering options, run `delta-filter --help` in the terminal.
remove_files	Logical indicating whether to remove intermediate files generated during the process, defaults to TRUE.
output_dir	Parent directory for a unique working subdirectory; defaults to `tempdir()`. With `remove_files = FALSE`, copied inputs and alignment files are retained in that subdirectory.

Value

A data frame combining all alignment results, or NULL if errors occur during processing.

References

Kurtz S, Phillippy A, Delcher AL, Smoot M, Shumway M, Antonescu C, Salzberg SL (2004). Versatile and open software for comparing large genomes. Genome Biology, 5(R12).

Examples

```
## Not run:
# Requires the external MUMmer programs (nucmer or promer, delta-filter and
# show-coords) on the PATH.

# Basic alignment with default options
mummer_alignment(
  path = "/path/to/sequences",
  maptype = "many-to-many",
  seqtype = "protein"
)

# Alignment with specific MUMmer options
mummer_alignment(
  path = "/path/to/sequences",
  maptype = "one-to-one",
  seqtype = "protein",
  mummer_options = "--maxgap=500 --mincluster=100",
  filter_options = "-i 90"
)

## End(Not run)
```

ophA_clusters *ophA Gene Cluster from Omphalotus olearius*

Description

This dataset represents the ophA gene cluster from Omphalotus olearius and Dendrothele bispora, involved in the production of omphalotin, a cyclic N-methylated peptide.

Usage

```
ophA_clusters
```

Format

A data frame with 17 rows and 5 columns:

name Gene names. A character vector.

start Start positions of the genes. An integer vector.

end End positions of the genes. An integer vector.

class Classifications of the genes. A character vector indicating the type of protein or function associated with each gene.

cluster Cluster identification as a character vector, specifying whether the gene belongs to the ophA or dbophA gene cluster.

References

van der Velden NS et al. Autocatalytic backbone N-methylation in a family of ribosomal peptide natural products. Nat Chem Biol. 2017 Aug;13(8):833-835. doi: 10.1038/nchembio.2393. Epub 2017 Jun 5. PMID: 28581484.

pQEB1_alignment *Nucleotide alignment of the pQEB1 plasmids*

Description

The aligned regions between neighbouring plasmids of pQEB1_plasmids, in the order pQEB1_inv3_Tn5403, pQEB1, pQEB1_inv1, pQEB1_inv1_inv2 and pQEB1_inv1_arsTn, from a one-to-one nucleotide alignment with MUMmer (nucmer, see `mummer_alignment()`). Regions shorter than 500 bp are left out. Regions with `start2` larger than `end2` are inverted.

Usage

```
pQEB1_alignment
```

Format

A data frame with 14 rows and 9 columns:

cluster1, cluster2 Names of the two plasmids. Character vectors.

start1, end1 Start and end of the region in `cluster1`. Integer vectors.

start2, end2 Start and end of the region in `cluster2`. Integer vectors.

length1, length2 Length of the region in each plasmid. Integer vectors.

identity Percentage identity of the region. A numeric vector.

Source

Made by `data-raw/pQEB1_plasmids.R`.

`pQEB1_plasmids` *Genes of the blaKPC-2 plasmid pQEB1 and four of its variants*

Description

The genes of five IncN plasmids of the pQEB1 lineage, which carries the carbapenemase gene blaKPC-2, from a hospital outbreak of *Enterobacter cloacae* and *Klebsiella oxytoca*. The variants differ from pQEB1 by inversions and transposon insertions. Made by `data-raw/pQEB1_plasmids.R` from the GenBank files, together with pQEB1_alignment.

Usage

```
pQEB1_plasmids
```

Format

A data frame with 401 rows and 10 columns:

cluster Name of the plasmid. A character vector.

host Species and strain the plasmid was found in. A character vector.

accession GenBank accession of the plasmid. A character vector.

locus_tag Locus tag of the gene. A character vector.

gene Gene name, or "" when the gene has none. A character vector.

product Description of the gene product. A character vector.

category Function of the gene from its name and product, e.g. "Carbapenemase (KPC)", "Mobile element" or "Conjugation". A character vector.

start Start position of the gene. A numeric vector.

end End position of the gene. A numeric vector.

strand Strand of the gene, "forward" or "complement". A character vector.

Source

GenBank accessions NZ_PQ162285, NZ_PQ162289, NZ_PQ162291, NZ_PQ162292 and NZ_PQ162293.

References

Moran RA, Behruznia M, Holden E, Garvey MI, McNally A. pQEB1: a hospital outbreak plasmid lineage carrying blaKPC-2. Microb Genom. 2024;10(9).

`protein_blast` *Find Homologous Proteins Between Clusters by Pairwise Alignment*

Description

Compares the proteins of a query cluster with those of the other clusters. It generates all protein combinations between the query cluster and each other cluster, aligns each pair with a global (Needleman-Wunsch) pairwise alignment from `pwalign::pairwiseAlignment()`, calculates sequence identity and similarity, and keeps the best hit of each gene above a minimum identity threshold.

Usage

```
protein_blast(
  data,
  query,
  id = "protein_id",
  start = "start",
  end = "end",
  cluster = "cluster",
  genes = NULL,
  identity = 30,
  parallel = TRUE
)
```

Arguments

`data`	A dataframe or a character vector specifying the path to .gbk files. When a character vector is provided, it is interpreted as file paths to .gbk files which are then read and processed. The dataframe must contain columns for unique protein identifiers, cluster identifiers, protein sequences, and the start and end positions of each gene.
`query`	The name of the query cluster the other clusters are compared with.
`id`	The name of the column that contains the gene identifiers. Defaults to "protein_id".
`start`	The name of the column specifying the start positions of genes. Defaults to "start".

end	The name of the column specifying the end positions of genes. Defaults to "end".
cluster	The name of the column specifying the cluster names. Defaults to "cluster".
genes	An optional vector of gene identifiers to include in the analysis. Defaults to NULL.
identity	Minimum percentage identity of the global alignment for a hit to be kept. Defaults to 30.
parallel	Deprecated and ignored. The alignments are now fast enough without parallel processing. Kept so that existing code keeps working.

Details

Despite its name, the function does not run BLAST: there is no seed search, local alignment, substitution matrix or E-value. Identity and similarity are percentages of the full length of the global alignment, gaps included, so proteins of very different length, or that share only one domain, get lower values than they would get from BLASTP. Set `identity` with this in mind.

Value

A modified version of the input `data` dataframe, including additional columns `BlastP` (the protein of the query cluster that is the best hit; the name is kept for compatibility), `identity`, `similarity` and `score` (the synteny score of the cluster).

Note

This function relies on the Biostrings and pwalign packages from Bioconductor for sequence alignment. Ensure these packages are installed.

References

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48(3), 443-453. doi:10.1016/00222836(70)900574

Examples

```
if (requireNamespace("Biostrings", quietly = TRUE) &&
    requireNamespace("pwalign", quietly = TRUE)) {
  genes <- data.frame(
    protein_id = c("A1", "A2", "B1", "B2"),
    cluster = c("A", "A", "B", "B"),
    start = c(1, 200, 1, 200),
    end = c(150, 350, 150, 350),
    translation = c("MKTAYIAKQRQISFVKSHFSRQ", "MSDNELLKRLGEAVLAHPTTG",
                    "MKTAYIAKQRQISFVKSHFSRQ", "MLPGGRTSWVQEHCDKAYNIF")
  )
  protein_blast(genes, query = "A", identity = 30)
}
```

read_bed *Read BED Files*

Description

This function reads BED files from a specified directory or file path and combines them into a single data frame. BED files use 0-based coordinate starts, while this function transforms the coordinates to 1-based during import. `chromStart` and `thickStart` are increased by 1; end positions and `blockSizes` are kept as they are. `blockStarts` become 1-based positions relative to `chromStart` (the first block starts at 1), as in `rtracklayer::import.bed()`, so the start of a block is `chromStart + blockStart - 1`.

Usage

```
read_bed(path)
```

Arguments

`path`	A character string specifying the directory containing BED files or the file path to a single BED file.

Details

This function can read multiple BED files from a directory or a single BED file from a specified path. It adds a 'filename' column with the name of the file, and combines the data frames from all files into one.

Value

A data frame combining data from the BED files.

Examples

```
bed_file <- tempfile(fileext = ".bed")
writeLines(c(
  "chr1\t100\t500\tgeneA\t0\t+",
  "chr1\t700\t1200\tgeneB\t0\t-"
), bed_file)

# Read a single BED file
bed_data <- read_bed(bed_file)

# Read all BED files in a directory
bed_data <- read_bed(dirname(bed_file))

unlink(bed_file)
```

read_fasta *Read Protein Sequences from FASTA Files*

Description

This function reads protein sequences from the specified FASTA file or all FASTA files within a directory. It specifically looks for metadata in the FASTA headers with key-value pairs separated by an equals sign =. For example, from the header '>protein1 [gene=scnD] [protein=ScnD]', it extracts 'gene' as the key and 'scnD' as its value, and similarly for other key-value pairs.

Usage

```
read_fasta(fasta_path, sequence = TRUE, keys = NULL, file_extension = "fasta")
```

Arguments

fasta_path	Path to the FASTA file or directory containing FASTA files.
sequence	Logical; if TRUE, the protein sequences are included in the returned data frame.
keys	An optional vector of strings representing specific keys within the fasta header to retain in the final data frame. If NULL (the default), all keys within the specified feature are included.
file_extension	Extension of the FASTA files to be read from the directory (default is 'fasta').

Details

The Biostrings package is required to run this function.

Value

A data frame with columns for each piece of information extracted from the FASTA headers.

Examples

```
if (requireNamespace("Biostrings", quietly = TRUE)) {
  fasta_file <- system.file("extdata", "BGC0000108.fasta", package = "geneviewer")

  # Read sequences from a single FASTA file
  sequences_df <- read_fasta(fasta_file)

  # Read all FASTA files in a directory, without the sequences
  sequences_df <- read_fasta(dirname(fasta_file), sequence = FALSE)
}
```

read_gbk *Read Data from GenBank Files*

Description

This function reads data from a single GenBank file or directory with GenBank files. It allows selective extraction of information by specifying sections and features.

Usage

```
read_gbk(path, sections = NULL, features = NULL, origin = TRUE)
```

Arguments

path	A string representing the file path to the target GenBank (.gbk) file or directory.
sections	An optional vector of strings representing the names of specific sections within the GenBank file to extract (e.g., "LOCUS", "DEFINITION", "ACCESSION", "VERSION"). If NULL (the default), the function extracts all available sections.
features	An optional vector of strings indicating specific feature types to extract from the FEATURES section of the GenBank file (e.g., "CDS", "gene", "mRNA"). If NULL (the default), the function extracts all feature types present in the FEATURES section.
origin	A boolean flag; when set to TRUE (the default), the origin sequence data is included in the output.

Value

A list containing the contents of the specified sections and features of the GenBank file. Each section and feature is returned as a separate list element.

Examples

```
# Path to example GenBank file in the package
genbank_file <- system.file(
  "extdata",
  "BGC0000001.gbk",
  package = "geneviewer"
  )

# Read all data from the example GenBank file
gbk_data <- read_gbk(genbank_file)

# Read only specific sections from the example GenBank file
gbk_data <- read_gbk(genbank_file, sections = c("LOCUS", "DEFINITION"))

# Read specific features from the FEATURES section of the example GenBank file
gbk_data <- read_gbk(genbank_file, features = c("gene", "CDS"))
```

```
# Read data without the origin sequence
gbk_data <- read_gbk(genbank_file, origin = FALSE)
```

`read_gff` *Read GFF Files*

Description

This function reads GFF files from a specified directory or file path and combines them into a single data frame.

Usage

```
read_gff(path, fields = NULL)
```

Arguments

`path`	A character string specifying the directory containing GFF files or the file path to a single GFF file.
`fields`	An optional vector of character strings specifying the fields to extract from the GFF files.

Details

This function can read multiple GFF files from a directory or a single GFF file. It processes each file, extracts the specified fields (if provided), adds a 'name' column with the filename, and combines the data frames from all files into one.

Value

A data frame containing the combined data from the GFF files.

Examples

```
gff_file <- system.file("extdata", "HQ113105.1.gff3", package = "geneviewer")

# Read a single GFF file
gff_data <- read_gff(gff_file)

# Read GFF files from a directory
gff_data <- read_gff(dirname(gff_file))

# Read specific fields
gff_data <- read_gff(
  gff_file,
  fields = c("seqid", "type", "start", "end", "strand", "Name")
)
```

`tn7_crispr_elements` *Genes of Tn7 and of Tn7-like transposons carrying a CRISPR-Cas system*

Description

The genes of Tn7, of Tn6022 and of fourteen Tn7-like transposons that have picked up a minimal CRISPR-Cas system, which has lost the genes needed to acquire spacers and to cut the target and is thought to guide the transposon to its insertion site instead. Each element is drawn from its right end, next to the gene it inserted beside, so that `tnsA` is at the left. The elements, the genomes they sit in and the insertion sites are those of Table S1 and Fig. 3 of the reference below, and the rows are named after the TnsA tree node that figure labels them with. Each region starts at the insertion site and ends at the left end of the element or after the cas genes, as in that figure. Only the first part of each element is covered, up to the end of the cas genes; the longer elements carry further cargo beyond it. Made by `data-raw/tn7_crispr_elements.R`.

Usage

```
tn7_crispr_elements
```

Format

A data frame with 307 rows and 13 columns:

cluster Name of the element, as in Fig. 3 of the reference. A character vector.

strain Species and strain the element was found in. A character vector.

strain_short The strain with its genus abbreviated, e.g. "V. cholerae VC35", for short labels. A character vector.

accession Accession of the sequence the region comes from. A character vector.

site Gene the element has inserted next to, e.g. "glmS", "yciA" or "SRP-RNA". A character vector.

locus_tag Locus tag of the gene. A character vector.

gene Gene name, or "" when the gene has none; "R" or "L" for the right or left end of the element, and "CRISPR" for a CRISPR array. A character vector.

label The name Fig. 3 of the reference writes on the gene, or "" when it names none: the transposition and cas genes, the insertion sites in short (e.g. "glmS" for "attTn7 (glmS)") and the resistance genes and integrase of Tn7. A character vector.

product Description of the gene product. A character vector.

category Role of the gene, e.g. "tnsA", "tniQ / tnsD", "cas8f / cas5f", "CRISPR array", "insertion site", "right end" or "left end". A character vector.

start Start position of the gene in the region. A numeric vector.

end End position of the gene in the region. A numeric vector.

strand Strand of the gene, "forward" or "complement". A character vector.

Details

Tn7 itself carries `tnsE` and no cas genes. It is taken from the chromosome of *Acinetobacter baumannii* UC20804, where it sits at its insertion site `attTn7`, next to `glmS`; this Tn7 is 99.9% identical over its whole length to the Tn7 of GenBank DQ176450. Tn6022 is the other element without cas genes; it sits inside `yifB` (`comM`), which the annotation splits in two, and only the fragment at its right end is kept here. Its ends are the window between those two fragments and are not verified to the base. In element #2757 `tnsA` is fused to `tnsB`: the annotated gene is longer than the `tnsB` of Tn7 by about the size of a TnsA, and that extra length at its start is given as a separate row, with gene name "tnsA (fused)" and label "tnsA". The boundary between the two is an estimate.

Besides the genes, each element has a row for its right end (gene `R`, category "right end") and its left end (gene `L`, category "left end"). Both are single positions, with `start` equal to `end`. The left end of most elements lies far beyond the last gene covered here (up to about 100 kb), at its true position. Genes that run on past the left end are cut off there.

Source

GenBank accession CP076807, RefSeq accession NZ_KB849325 and the RefSeq genomes of Table S1 of the reference, read with `read_gbk()`. The CRISPR arrays that RefSeq does not annotate were found from the repeat consensus of the type I-F arrays these elements carry.

References

Peters JE, Makarova KS, Shmakov S, Koonin EV. Recruitment of CRISPR-Cas systems by Tn7-like transposons. Proc Natl Acad Sci USA. 2017;114(35):E7358-E7366.

Index

∗ **datasets**
  BRCA1_splice_variants, 3
  coronavirus_genomes, 4
  erythromycin_BlastP, 6
  erythromycin_cluster, 7
  hs_dystrophin_transcripts, 69
  human_hox_genes, 69
  mitochondrial_gene_order, 70
  ophA_clusters, 73
  pQEB1_alignment, 73
  pQEB1_plasmids, 74
  tn7_crispr_elements, 81

BRCA1_splice_variants, 3

coronavirus_genomes, 4

erythromycin_BlastP, 6
erythromycin_cluster, 7

gbk_features_to_df, 7
GC_align, 8
GC_annotation, 9, *57*
GC_annotation(), *50*
GC_chart, 11
GC_chart(), *22*, *24*, *32*, *50*, *54, 55*, *65, 66*
GC_chartOutput *(*geneviewer-shiny*)*, 65
GC_chartOutput(), *63*
GC_cluster, 12
GC_cluster(), *29*, *32*, *50*
GC_clusterFooter, 14
GC_clusterFooter(), *50*
GC_clusterLabel, 15
GC_clusterLabel(), *50*
GC_clusterTitle, 18
GC_clusterTitle(), *24*, *50*, *59*
GC_color, 20
GC_color(), *50*, *66*
GC_coordinates, 21
GC_coordinates(), *50*
GC_element_style, 23
GC_element_style(), *25*, *27*
GC_events, 25
GC_events(), *54, 55*, *62, 63*
GC_genes, 28
GC_genes(), *13*, *50*, *66*
GC_grid, 30
GC_grid(), *16*, *50*, *59*, *65*
GC_labels, 31
GC_labels(), *13*, *37*, *49, 50*, *59*
GC_legend, 33
GC_legend(), *37*, *50*, *59*
GC_links, 36
GC_links(), *4*, *50*
GC_normalize, 40
GC_pattern, 41
GC_pattern(), *50*
GC_scale, 43
GC_scale(), *8*, *50, 51*, *63*
GC_scaleBar, 45
GC_scaleBar(), *50*
GC_sequence, 47
GC_sequence(), *50*
GC_theme, 49
GC_theme(), *54*, *59*, *66–68*
GC_title, 52
GC_title(), *50*, *59*
GC_toolbar, 54
GC_toolbar(), *50*, *61*
GC_tooltip, 56
GC_tooltip(), *50*
GC_trackMouse, *10*, 57
GC_trackMouse(), *50*
GC_transcript, 58
GC_transcript(), *50*, *66*
GC_zoom, 61
GC_zoom(), *25, 26*, *32*, *37*, *55*, *63, 64*
GC_zoom_to, 63
GC_zoom_to(), *26*, *63*

genbank_to_fasta, 64
geneviewer-shiny, 65
geneviewer_palette, 66
geneviewer_palette(), *20*, *29*, *50*, *59*
geneviewer_theme, 67
geneviewer_theme(), *49*, *51*
get_introns, 68

hs_dystrophin_transcripts, 69
human_hox_genes, 69

mitochondrial_gene_order, 70
mummer_alignment, 71
mummer_alignment(), *73*

ophA_clusters, 73

pQEB1_alignment, 73, *74*
pQEB1_plasmids, *73*, 74
protein_blast, 75
protein_blast(), *5*, *6*
pwalign::pairwiseAlignment(), *75*

read_bed, 77
read_fasta, 78
read_gbk, 79
read_gbk(), *5*, *70*, *82*
read_gff, 80
renderGC_chart *(*geneviewer-shiny*)*, 65

tn7_crispr_elements, 81