Package 'GenoView'

July 15, 2015

Type Package

Title Condensed, overlapped plotting of genomic data tracks

Version 1.3.0

Date 2014-12-29

Author Sharon Lee, Dennis Wang

Maintainer Sharon Lee <s274lee@uwaterloo.ca>

Description Superimposing input data over existing genomic references allows for fast, accurate visual comparisons. The GenoView package is a novel bioinformatics package which condenses genomic data tracks to offer a comprehensive view of genetic variants. Its main function is to display mutation data over exons and protein domains, which easily identifies potential genomic locations of interest.

biocViews Visualization

Depends R (>= 2.10), gridExtra, GenomicRanges

Imports ggbio, ggplot2, grid, biovizBase

Suggests TxDb.Hsapiens.UCSC.hg19.knownGene, PFAM.db, AnnotationDbi, gtable, gWidgets, gWidgetsRGtk2, RGtk2, RColorBrewer

License GPL-3

LazyData Yes

Collate 'Domain.R' 'GUI.R' 'Main.R' 'Text_interface.R'

NeedsCompilation no

R topics documented:

`GenoView-package` *Condensed, overlapped plotting of genomic data tracks*

Description

The GenoView package provides interactive visualization utilities which combine user specified and existing genomic data.

Details

Package:	GenoView
Type:	Package
Version:	1.0
Date:	2013-12-05
License:	GPL-3

The package currently focuses on visualizing mutation data with `mutExonPlot`. It provides a GUI and a text interface (see `mep-Interfaces`). Databases and variables which provide hg19 data to `mutExonPlot` are already built into `mepHuman`.

Author(s)

Sharon Lee, Dennis Wang

Maintainer: Sharon Lee <s274lee@uwaterloo.ca>

Examples

```
if (interactive()) {
  # Create a missense mutations dataset in TP53
  set.seed(1)
  locations = sample(7571720:7590863, size = 25)
  mut.df <- data.frame(chrom = "chr17",
                       start = locations,
                       end = locations,
                       strand = "-",
```

```
                        fill = 1:25)

  library(gWidgetsRGtk2)
  library(RGtk2)

  # hg19 GUI example
  mepHuman(dataFrame = mut.df, gui = TRUE)
}
```

---

colSel *Column name selection*

Description

Selects a column name by user input.

Usage

```
colSel(col.crit, dataFrame)
```

Arguments

col.crit	Description of type of data in the desired column.
dataFrame	Matrix or data.frame with column names.

Value

Character which is a column name in dataFrame

Author(s)

Sharon Lee

See Also

printOption

Examples

```
if(interactive()) {
  colSel("Gear", dataFrame = mtcars)
}
```

intSel *GRanges interval selection*

Description

Truncate the range of a GRanges object

Usage

```
intSel(gr, new.start = NULL, new.end = NULL)
```

Arguments

gr	GRanges object of length 1
new.start	New start location
new.end	New end location

Details

The start and end locations, input or user selected, create an interval which is smaller or equal to the original range of gr

Value

A Granges object of length 1

Author(s)

Sharon Lee

Examples

```
a <- GRanges(seqnames = "chr1",
                ranges = IRanges(start = 1000, end = 5000),
                strand = "-")
if (interactive()) {
  intSel(a)
}

b <- GRanges(seqnames = "chr2",
                ranges = IRanges(start = 10000, end = 50000),
                strand = "-")
intSel(gr = b, new.start = 12345, new.end = 49999)
```

makeGR *Make GRanges object from data.frame*

Description

Extracts GRanges data from relevant columns from a data.frame, performs seqlengths adjustments and complete case filtering

Usage

```
makeGR(dataFrame, chr.col, s.p.col, e.p.col, str.col,
id.col = NULL, plot.int, des.seqs, show.legend = FALSE, l.height = 0)
```

Arguments

dataFrame	Data.frame which contains at least Chromosome, Start Position, End Position, and Strand data. It may also contain a column which uniquely identifies each entry for legend plotting.
chr.col	Chromosome data column name
s.p.col	Start position data column name
e.p.col	End position data column name
str.col	Strand data column name
id.col	Fill data column name
plot.int	GRanges object of length 1 of the plotting interval in genomic coordinates
des.seqs	Desired seqlengths appropriate for dataFrame
show.legend	Determines whether legend information is kept based on boolean
l.height	Determines whether legend information is kept based on height allocation

Value

GRanges object containing filtered genomic data

Author(s)

Sharon Lee

Examples

```
library(biovizBase)

# Create a missense mutations dataset in TP53
set.seed(1)
N = 25
locations = sample(7571720:7590863, size = N)
mut.df <- data.frame(chrom = "chr17",
```

```
                    start = locations,
                    end = locations,
                    str = "-",
                    fill = 1:N)

# Create a truncated plotting interval
plot.int = GRanges(seqnames = "chr17",
                    IRanges(start = 7576000, end = 7579000),
                    strand = "-")

# Obtain hg19 seqlengths data
data(hg19Ideogram, package = "biovizBase")
seqs.hg19 <- seqlengths(hg19Ideogram)

makeGR(dataFrame = mut.df, chr.col = "chrom", s.p.col = "start",
        e.p.col = "end", str.col = "str", id.col = "fill",
        plot.int = plot.int, show.legend = TRUE,
        des.seqs = seqs.hg19)
```

`makePFAMObjs` *Make PFAM mapping objects*

Description

Produces custom objects from PFAM.db, targets PFAM mapping in `mutExonPlot`

Usage

```
makePFAMObjs()
```

Value

`desc`	list of PFAM definitions with Accession numbers
`ids`	character vector of Associated identifierss with Accession numbers as names

Author(s)

Sharon Lee

Examples

```
makePFAMObjs()
```

mep-Interfaces *Interfaces for MutExonPlot*

Description

Package interfaces which create, display, and save Mutations-Over-Exons plots in succession.

Usage

```
mepGUI(m.data, tx.db, gene.loc, seq.lens, pfam.gr = NULL, pfam.desc = NULL,
pfam.ids = NULL, ...)
mepTxtInt(m.data, tx.db, gene.loc, seq.lens, pfam.gr = NULL, pfam.desc = NULL,
pfam.ids = NULL, ...)
```

Arguments

m.data	Mutations data.frame or GRanges object which contains at minimum Chromosome, Start Position, End Position, and Strand data. It may also contain a column which uniquely identifies each mutation for legend plotting.
tx.db	TranscriptDb object using the same reference genome as the contents of m.data
gene.loc	GRanges object containing gene data. It has HUGO gene symbols as names and has seqnames, ranges, and strand data for each entry.
seq.lens	Seqlengths for m.data, using the same reference genome as tx.db
pfam.gr	GRanges object of protein domain coordinates and Associated identifications
pfam.desc	list of PFAM definitions with Accession numbers
pfam.ids	character vector of Associated identifications with Accession numbers as names
...	Input objects from other databases, datasets, vectors describing plotting options. An example of possible input objects, using the human genome, are generated in mepHuman.

Details

The GUI window consists of two parts, a input selection menu and a notebook widget which displays m.data and the finished plots. Once selection is complete, all required inputs are processed and passed onto mutExonPlot. The GUI displays and saves many plots at once.

The text interface reads text entry from the user to complete input selection for plotting. The interface displays and saves one plot at a time.

Note

The common objects that are passed from mepHuman are: tx.db, genesymbol, seqs.hg19, pfam.gr, pfam.desc, pfam.ids, int.opt, disp.opt.

PFAM Protein domain data might not be available for all species, and was originally obtains from UCSC.

Author(s)

Sharon Lee

See Also

mepHuman, mutExonPlot

Examples

```
## Not run:
library(TxDb.Hsapiens.UCSC.hg19.knownGene)

# Create a missense mutations dataset in TP53
set.seed(1)
locations = sample(7575000:7580000, size = 25)
mut.df <- data.frame(chrom = "chr17",
                     start = locations,
                     end = locations,
                     strand = "-",
                     fill = letters[1:25])

# Generating sample data using TP53
tx.db <- TxDb.Hsapiens.UCSC.hg19.knownGene
seqs.hg19 <- seqlengths(tx.db)
data(genesymbol, package = "biovizBase")

# Sample GRanges data with domains for P53
sample <- GRanges(seqnames = "chr17",
                     IRanges(start = c(7576884, 7577499, 7579707),
                             end = c(7577102, 7579403, 7579899)),
                     strand = "-",
                     domain = c("P53_tetramer", "P53", "P53_TAD"))

# Create PFAM objects
objs <- makePFAMObjs()
desc <- objs$desc
ids <- objs$ids

int.opt <- c("Whole", "Custom")
disp.opt <- c("All", "Reduce")
dom.opt <- c("All", "Longest")

# GUI
mepGUI(m.data = mut.df, tx.db = tx.db, gene.loc = genesymbol,
        seq.lens = seqs.hg19, pfam.desc = desc, pfam.ids = ids,
        pfam.gr = sample, int.opt = int.opt, disp.opt = disp.opt,
        dom.opt = dom.opt)

# Text Interface
mepTxtInt(m.data = mut.df, tx.db = tx.db, gene.loc = genesymbol,
        seq.lens = seqs.hg19, pfam.desc = desc, pfam.ids = ids,
        pfam.gr = sample, int.opt = int.opt, disp.opt = disp.opt,
```

```
        dom.opt = dom.opt)

## End(Not run)
```

---

mepHuman *Main setup for Mutations Exon Plot*

Description

Creates all main variables using the human genome from databases and objects, passes variables and input data to interface function

Usage

```
mepHuman(m.data, gui = FALSE)
```

Arguments

m.data	Mutations data.frame or GRanges object which contains at minimum Chromosome, Start Position, End Position, and Strand data. It may also contain a column which uniquely identifies each mutation for legend plotting.
gui	Determines whether the GUI is used.

Details

The mepHuman function creates the following objects which are passed to mepGui or mepTxtInt:

tx.db UCSC hg19 known Gene database from TxDb.Hsapiens.UCSC.hg19.knownGene

genesymbol GRanges object describing human genes from genesymbol

seqs.hg19 seqlengths of hg19 object

pfam.gr, pfam.desc, pfam.ids objects containing PFAM domain information

int.opt, disp.opt character vectors which describe the plotting input choices

Value

A mutExonPlot using the hg19 platform

Author(s)

Sharon Lee

See Also

mutExonPlot, mep-Interfaces

Examples

```
if (interactive()) {
  # Create a missense mutations dataset in TP53
  set.seed(1)
  locations = sample(7571720:7590863, size = 25)
  mut.df <- data.frame(chrom = "chr17",
                       start = locations,
                       end = locations,
                       strand = "-",
                       fill = 1:25)

  library(gWidgetsRGtk2)
  library(RGtk2)

  # GUI example
  mepHuman(m.data = mut.df, gui = TRUE)
}

if (interactive()) {
  # Text interface example
  mepHuman(m.data = mut.df)
}
```

`mutExonPlot` *Mutations-Over-Exons Plot*

Description

Superimpose user specified mutation data on the exons of a gene, with domain and legend annotations

Usage

```
mutExonPlot(mut.gr, exon.int, plot.int = exon.int,
disp.track = 2, p.height = 1/4, d.height = 0, l.height = 1/2,
plt.title = "mutExonPlot", id.col = "Fill", tx.db = NULL, pfam.gr = NULL,
pfam.desc = NULL, pfam.ids = NULL, gr = NULL, ...)
```

Arguments

`mut.gr`	GRanges object containing mutation data, with fill values for legend if applicable
`exon.int`	GRanges object of length 1 of the genomic interval in which to count exons
`plot.int`	GRanges object of length 1 of the plotting interval in genomic coordinates
`disp.track`	Selects exon plotting track: 1 for full transcripts, 2 for reduced track
`p.height`	Numeric height allocation of Mutations-Over-Exons section in final grob
`d.height`	Numeric height allocation of domain annotation section in final grob

l.height	Numeric height allocation of legend section in final grob
plt.title	Plot title
id.col	Column containing identifiers in the values of mut.gr. It is only relevant in plotting a legend.
tx.db	TrancscriptDb object
pfam.gr	GRanges object of protein domain coordinates and Associated identifications
pfam.desc	list of PFAM definitions with Accession numbers
pfam.ids	character vector of Associated identifications with Accession numbers as names
gr	GRanges object containing genomic data (cds, utr, gap, intron, exon) over exon.int
...	Objects passed from the interface function if applicable

Value

Some elements may not be returned, depending on the chosen plot components.

mep	Mutations-Over-Exons final plot grob
plot	Mutations-Over-Exons component grob
domain	Domain component grob
legend	Legend component grob
show.domain	show.domain boolean
gr	GRanges object containing genomic data (cds, utr, gap, intron, exon) over exon.int
exon.int	GRanges object of length 1 which specifies the gene's genomic coordinates

If plotting fails, FALSE is returned.

Note

plot.int should be smaller than or equal to exon.int.

When the reduced track option and domain plotting are selected in disp.track and d.height, the longest protein domain in the plotting interval is added the mutations over exon ploting section, rather than its own domain section.

Author(s)

Sharon Lee

Examples

```
## Not run:
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(biovizBase)

# Determine the published genomic datasets
tx.db <- TxDb.Hsapiens.UCSC.hg19.knownGene
data(genesymbol, package = "biovizBase")
```

```
# Create a mutation dataset in TP53
exon.int <- genesymbol["TP53"]
set.seed(2)
N = 10
mut.gr <- GRanges(seqnames = "chr17",
                  IRanges(start = sample(7575000:7580000,
                          size = N, replace = TRUE),
                          width = 1),
                  strand = "-")

mutExonPlot(mut.gr = mut.gr, exon.int = exon.int, l.height = 0, tx.db = tx.db)

## End(Not run)
```

`notebookVis` *Visualize Grob objects in gnotebook widget*

Description

Create a new page in the gnotebook widget to display the plot with a unique label

Usage

```
notebookVis(nb, plt, type, suff)
```

Arguments

`nb`	Visible gnotebook widget
`plt`	Grob object to plot
`type`	Description of `plt` in tab label
`suff`	Incremental numeric to follow `type` in tab label

Value

`plt`	The same input `plt` which has been visualized
`plt.lab`	The tab label for the notebook page displaying `plt`

Author(s)

Sharon Lee

Examples

```
if (interactive()) {
library(grid)
library(gWidgets)

# Create a grob object
l <- linesGrob()

# Create the notebook widget
win <- gwindow()
nb <- gnotebook(cont = win)
notebookVis(nb = nb, plt = l, type = "Example", suff = 1)
}
```

---

pfam.df *Human protein domain dataset*

Description

Processed dataset containing PFAM protein domain locations and identifications

Usage

```
data(pfam.df)
```

Format

A data frame with 343252 observations on the following 5 variables.

`PFAMID` a factor with PFAM Associated identifications as levels

`chr` a factor with chromosomes as levels

`start` a numeric vector

`end` a numeric vector

`strand` a factor with levels `-` `+`

Source

UCSC Table Browser hg19 known to Pfam table (default file name "ucscGenePfam.txt"") retrieved 2014/12/29

References

Karolchik D, et al. The UCSC Table Browser data retrieval tool. Nucleic Acids Res. 2004;32:D493-D496.

Examples

```
data(pfam.df)
```

PFAMIDE *Translates domain transcripts' identifications to descriptions*

Description

Maps the domain values of a GRanges object from Associated identifications to PFAM definitions

Usage

```
PFAMIDE(transcripts, desc, ids)
```

Arguments

transcripts	GRanges object with domain column in its values containing PFAM Associated identifications as characters
desc	list of PFAM definitions with Accession numbers
ids	character vector of Associated identifications with Accession numbers as names

Value

GRanges object with mapped domain column in its values

Note

There are some Accession numbers from ids that do not map to those in desc. The original Associated identification is retained, rather than producing NA.

Author(s)

Sharon Lee

See Also

makePFAMObjs

Examples

```
## Not run:
# Sample data with domains for P53
sample <- GRanges(seqnames = "chr17",
                  IRanges(start = c(7576884, 7577499, 7579707),
                          end = c(7577102, 7579403, 7579899)),
                  strand = "-",
                  domain = c("P53_tetramer", "P53", "P53_TAD"))

# Create the mapping objects
pfam.objs <- makePFAMObjs()
```

```
PFAMIDE(transcripts = sample, desc = pfam.objs$desc, ids = pfam.objs$ids)

## End(Not run)
```

---

plotExonRect *Plot Rectangular Exon Components*

Description

Adds colored rectangular blocks, denoting exon components such as coding sequences or untranslated regions, to a ggplot. Provides filtering to highlight a subset of the input data in the plot.

Usage

```
plotExonRect(plt, component = c("cds", "utr"), comp.df,
col.name = "type", filt.by.comp = FALSE, size = c(0.4, 0.15),
subset.df, comp.col = "dark grey",
sub.col = RColorBrewer::brewer.pal(3, "Set1")[2],
aesCst)
```

Arguments

plt	A ggplot object
component	Exon component string, such as "cds", "utr"
comp.df	Exon components data.frame, contains a col.name column which contains exon component strings and columns for start, end, and stepping (starting y position in plot)
col.name	Column name used to filter comp.df before plotting
filt.by.comp	Determines if comp.df should be filtered for component contents before plotting
size	Half the height of the rectangular blocks
subset.df	Secondary data.frame, contains a col.name column which contains exon component strings and columns for start, end, and stepping (starting y position in plot). This data is plotted in a highlighting colour.
comp.col	Colour of data in code.df
sub.col	Highlighting colour of data in subset.df
aesCst	Custom aes function which can handle constants in variables

Value

ggplot object with additional geom_rect() layers

Author(s)

Sharon Lee

Examples

```
library(ggplot2)

set.seed(3)
N = 30
# Create sample base exon components
starts = sample(1:10000, size = N, replace = TRUE)
sample.df <- data.frame(start = starts,
                        end = starts + sample(100:500, size = N),
                        stepping = 1,
                        type = sample(c("cds", "utr", "gap"),
                                      size = N, replace = TRUE))

# Create sample highlighted exon components
N = 5
starts = sample(1:10000, size = N, replace = TRUE)
sample.sub <- data.frame(start = starts,
                        end = starts + sample(100:500, size = N),
                        stepping = 1,
                        type = sample(c("cds", "utr", "gap"),
                                      size = N, replace = TRUE))

# Using an aes function which evaluates arguments locally without parsing
aesCst <- function(...) {
    structure(list(...),  class = "uneval")
}

step1 <-ggplot() +
        geom_abline(intercept = 1, slope = 0,
                    aes(xmin = -1, xmax = 1, ymin = 0.5, ymax = 1.5),
                    color = "dark grey")

step2 <- plotExonRect(plt = step1, component = "cds", comp.df = sample.df,
                        filt.by.comp = TRUE, size = 0.4, subset.df = sample.sub,
                        aesCst = aesCst)
plotExonRect(plt = step2, component = "utr", comp.df = sample.df,
                        filt.by.comp = TRUE, size = 0.15, subset.df = sample.sub,
                        aesCst = aesCst)
```

`printOption` *Print numbered options*

Description

Prints a numbered list as a guide for selecting an option or an index.

Usage

```
printOption(opt.vec, inst = TRUE, desc = "an option",
match.opt = FALSE, match.ind = FALSE)
```

Arguments

opt.vec	A vector of printable objects, preferably character, which fits the descriptor desc.
inst	Controls instruction printing using desc.
desc	A char which indicates the type of object in opt.vec.
match.opt	Chooses the selected element from opt.vec.
match.ind	Chooses the index of the selected element from opt.vec.

Value

The selected object from opt.vec if match.opt = TRUE. The integer index if match.ind = TRUE.

Author(s)

Sharon Lee

Examples

```
colours <- c("Red", "Blue", "Green", "Yellow")
printOption(colours, desc = "a colour") ## no return value

if (interactive()){
printOption(colours, desc = "a colour", match.ind = TRUE) # returns integer

printOption(colours, desc = "a colour", match.opt = TRUE) # returns character
}
```

---

sample.mut.df *Sample hg19 TP53 missense mutations dataset*

Description

Data.frame containing hg19 human genome TP53 missense mutations, to be used in examples and vignette

Usage

```
data(sample.mut.df)
```

Format

A data frame with 10 observations on the following 7 variables.

ids a factor with sample ID

seqnames a factor with chromosome locations

start a numeric, integer vector

`end` a numeric, integer vector

`width` a numeric, integer vector

`strand` a factor with levels - +

`midpoint` a numeric vector

Examples

```
## Not run:
data(sample.mut.df)

## End(Not run)
```

`updatePFAM` *Process raw UCSC PFAM table*

Description

Truncates and maps the contents of the raw hg19 UCSC PFAM table

Usage

```
updatePFAM(raw.file = "ucscGenePfam.txt", search.dir = getwd(),
file.name = "", ID2DE = FALSE)
```

Arguments

`raw.file`	Filename of raw PFAM table
`search.dir`	Directory in which the raw.file should be located
`file.name`	Filename of the processed table
`ID2DE`	Controls mapping from PFAM identification for Accession number to its definition

Details

The protein domain table can be obtained by downloading from the hg19 annotation database of UCSC (see default `raw.file`), or extracting data using the UCSC Table Browser Tool.

Value

Produces a truncated tab-delimited text file named `file.name` in `search.dir`.

Returns the processed file name if a file was created, FALSE otherwise.

Author(s)

Sharon Lee

References

Karolchik D, et al. The UCSC Table Browser data retrieval tool. Nucleic Acids Res. 2004;32:D493-D496.

Examples

```
updatePFAM()
```

Index