rtracklayer

October 25, 2011

`Bed15TrackLine-class`
Class "Bed15TrackLine"

Description

A UCSC track line for graphical tracks.

Objects from the Class

Objects can be created by calls of the form `new("Bed15TrackLine", ...)` or parsed from a character vector track line with `as(text, "Bed15TrackLine")`.

Slots

`expStep`: A `"numeric"` scalar indicating the step size for the heatmap color gradient.

`expScale`: A positive `"numeric"` scalar indicating the range of the data to be `[-expScale, expScale]` for determining the heatmap color gradient.

`expNames`: A `"character"` vector naming the the experimental samples.

`name`: Object of class `"character"` specifying the name of the track.

`description`: Object of class `"character"` describing the track.

`visibility`: Object of class `"character"` indicating the default visible mode of the track, see `UCSCTrackModes`.

`color`: Object of class `"integer"` representing the track color (as from `col2rgb`).

`priority`: Object of class `"numeric"` specifying the rank of this track.

Extends

Class `"TrackLine"`, directly.

Methods

as(object, "character") Export line to its string representation.

Author(s)

Michael Lawrence

References

Official documentation: http://genomewiki.ucsc.edu/index.php/Microarray_track.

See Also

export.bed15 for exporting bed15 tracks.

BigWigSelection-class

Selection of ranges and columns

Description

A BigWigSelection represents a query against a BigWig file, see import.bw. It is simply a RangedSelection that requires its colnames parameter to be "score", if non-empty, as that is the only column supported by BigWig.

Constructor

BigWigSelection(ranges = GRanges(), colnames = "score"): Constructors a BigWigSelection with the given ranges and colnames. ranges can be either a RangesList, or a character identifying a genome (see GenomicSelection).

Coercion

as(from, "BigWigSelection"): Coerces from to a BigWigSelection object. Typically, from is a GRanges or a RangesList, the ranges of which become the ranges in the new BigWigSelection.

Author(s)

Michael Lawrence

Examples

```
rl <- IRanges::RangesList(chr1 = IRanges(c(1, 5), c(3, 6)))

BigWigSelection(rl)
as(rl, "BigWigSelection") # same as above

# do not select the 'score' column
BigWigSelection(rl, character())
```

BrowserViewList-class *Lists of BrowserView*

Description

A formal list of BrowserView objects. Extends and inherits all its methods from Vector. Usually generated by passing multiple ranges to the browserView function.

Constructor

BrowserViewList(...): Concatenates the BrowserView objects in ... into a new BrowserViewList. This is rarely called by the user.

Author(s)

Michael Lawrence

Chain-class *Chain objects*

Description

A Chain object represents a UCSC chain alignment, typically imported from a chain file, and is essentially a list of ChainBlock objects. Each ChainBlock has a corresponding chromosome (its name in the list) and is a run-length encoded alignment, mapping a set of intervals on that chromosome to intervals on the same or other chromosomes.

Accessor Methods

In the code snippets below, x and object are ChainBlock objects.

ranges(x): Get the Ranges object holding the starts and ends of the "from" ranges. Each range is a contiguous block of positions aligned without gaps to the other sequence.

offset(x): Integer offset from the "from" start to the "end" start (which could be in another chromosome).

score(x): The score for each mapping.

space(x): The space (chromosome) of the "to" range.

reversed(x): Whether the mapping inverts the region, i.e., the alignment is between different strands.

Import

import.chain(con, exclude = "_", ...): Imports a chain file named con as a Chain object, a list of ChainBlocks. Alignments for chromosomes matching the exclude pattern are not imported.

Note

A chain file essentially details many local alignments, so it is possible for the "from" ranges to map to overlapping regions in the other sequence. The "from" ranges are guaranteed to be disjoint (but do not necessarily cover the entire "from" sequence).

Author(s)

Michael Lawrence

See Also

`liftOver` for performing lift overs using a chain alignment

GRangesForUCSCGenome

GRanges for a Genome

Description

These functions assist in the creation of `GRanges` in the context of a genome.

Usage

```
GRangesForUCSCGenome(genome, chrom = NULL, ranges = NULL, ...)
GRangesForBSGenome(genome, chrom = NULL, ranges = NULL, ...)
```

Arguments

`genome`	A string identifying a genome, usually one assigned by UCSC, like "hg19".
`chrom`	A character vector of chromosome names, or `NULL`.
`ranges`	A `Ranges` object with the intervals.
`...`	Additional arguments to pass to the `GRanges` constructor.

Details

The genome ID is stored in the metadata of the ranges and is retrievable via the `genome` function. The sequence lengths are also properly initialized for the genome. This mitigates the possibility of accidentally storing intervals for the wrong genome.

`GRangesForUCSCGenome` obtains sequence information from the UCSC website, while `GRangesForBSGenome` looks for it in an installed `BSGenome` package. Using the latter is more efficient in the long-run, but requires downloading and installing a potentially large genome package, or creating one from scratch if it does not yet exist for the genome of interest.

Value

A `GRanges` object, with the appropriate `seqlengths` and `genome` ID.

Author(s)

Michael Lawrence

GenomicSelection *Genomic data selection*

Description

Convenience constructor of a RangedSelection object for selecting a data on a per-chromosome basis for a given genome.

Usage

```
GenomicSelection(genome, chrom = NULL, colnames = character(0))
```

Arguments

genome	A string identifying a genome. Should match the end of a BSgenome package name, e.g. "hg19".
chrom	Character vector naming chromosomes to select.
colnames	The column names to select from the dataset.

Value

A RangedSelection object, selecting entire chromosomes

Author(s)

Michael Lawrence

See Also

RangedSelection, BigWigSelection

Examples

```
# every chromosome from hg19
GenomicSelection("hg19")
# chr1 and 2 from hg19, with a score column
GenomicSelection("hg19", c("chr1", "chr2"), "score")
```

RangedData-methods *Data on a Genome*

Description

The rtracklayer package adds convenience methods on top of RangedData and GRanges to manipulate data on genomic ranges. For RangedData the spaces are now called chromosomes (but could still refer to some other type of sequence). Similarly the universe refers to the genome.

Accessors

In the code snippets below, x is a RangedData or GRanges object.

chrom(x), chrom(x) <- value: Gets or sets the chromosome names for x. The length of value should equal the length of x. This is an alias for names(x).

genome(x), genome(x) <- value: Gets or sets the genome (a single string or NULL) for the ranges in x; simple wrappers around universe and universe<-, respectively.

Constructor

GenomicData(ranges, ..., strand = NULL, chrom = NULL, genome = NULL, asRangedData = TRUE): If asRangedData is TRUE, constructs a RangedData instance with the given ranges and variables in ... (see the RangedData constructor). If asRangedData is FALSE, constructs a GRanges instance with the given ranges and variables in ....

If non-NULL, the strand argument specifies the strand of each range. It should be a character vector or factor of length equal to that of ranges. All values should be either −, +, * or NA. (The NA code for strand is only acceptable when asRangedData is TRUE.) To get the levels for strand, call levels(strand()).

chrom argument is analogous to space in the RangedData and seqnames in GRanges constructors.

The genome argument should be a scalar string and is treated as the RangedData universe. See the examples.

If ranges is not a Ranges object, this function calls as(ranges, "RangedData") and returns the result if successful. As a special case, the "chrom" column in a data.frame-like object is renamed to "space", for convenience. Thus, one could pass a data.frame with columns "start", "end" and, optionally, "chrom".

Author(s)

Michael Lawrence and Patrick Aboyoun

Examples

```
range1 <- IRanges(start=c(1,2,3), end=c(5,2,8))

## just ranges ##
## RangedData instance
rd <- GenomicData(range1)
## GRanges instance
gr <- GenomicData(range1, asRangedData = FALSE)

## with a genome (universe) ##
## RangedData instance
rd <- GenomicData(range1, genome = "hg18")
genome(rd) ## "hg18"
## GRanges instance
gr <- GenomicData(range1, genome = "hg18", asRangedData = FALSE)
genome(gr) ## "hg18"

## with some data ##
filter <- c(1L, 0L, 1L)
score <- c(10L, 2L, NA)
strand <- factor(c("+", NA, "-"), levels = levels(strand()))
```

```
## RangedData instance
rd <- GenomicData(range1, score, genome = "hg18")
rd[["score"]]
strand(rd) ## all NA
rd <- GenomicData(range1, score, filt = filter, strand = strand)
rd[["filt"]]
strand(rd) ## equal to 'strand'
## GRanges instance
gr <- GenomicData(range1, score, genome = "hg18", asRangedData = FALSE)
values(gr)[["score"]]
strand(gr) ## all '*'
gr <- GenomicData(range1, score, filt = filter, strand = strand,
                  asRangedData = FALSE)
values(gr)[["filt"]]
strand(gr) ## equal to 'strand'

## multiple chromosomes ##
range2 <- IRanges(start=c(15,45,20,1), end=c(15,100,80,5))
ranges <- c(range1, range2)
score <- c(score, c(0L, 3L, NA, 22L))
chrom <- paste("chr", rep(c(1,2), c(length(range1), length(range2))), sep="")
## RangedData instance
rd <- GenomicData(ranges, score, chrom = chrom, genome = "hg18")
chrom(rd) # equal to 'chrom'
rd[["score"]] # unlists over the chromosomes
score(rd)
rd[1][["score"]] # equal to score[1:3]
## GRanges instance
gr <- GenomicData(ranges, score, chrom = chrom, genome = "hg18",
                  asRangedData = FALSE)
chrom(gr) # equal to 'chrom'
values(gr)[["score"]]
values(gr[chrom(gr) == "chr1"])[["score"]]

## coercion from data.frame ##
df <- as.data.frame(rd)
GenomicData(df)
GenomicData(df, asRangedData = FALSE)
```

`RangesList-methods` *Ranges on a Genome*

Description

Genomic coordinates are often specified in terms of a genome identifier, chromosome name, start position and end position. `RangedData` represents this with a `RangesList` instance, and the `rtracklayer` package adds convenience methods to `RangesList` for the manipulation of genomic ranges. The spaces (or names) of `RangesList` are the chromosome names. The `universe` slot indicates the genome, usually as given by UCSC (e.g. "hg18").

Accessors

In the code snippets below, `x` is a `RangesList` object.

`chrom(x)`, `chrom(x) <- value`: Gets or sets the chromosome names for `x`. This is an alias for `names(x)`.

`genome(x)`, `genome(x) <- value`: Gets or sets the genome (a single string or `NULL`) for the ranges in `x`; simple wrappers around `universe` and `universe<-`, respectively.

Constructor

`GenomicRanges(start, end, chrom = NULL, genome = NULL)`:

This function has been deprecated in favor of constructing a `GRanges` using `GRangesForUCSCGenome`.

Constructs a `RangesList` containing ranges specified by `start` and `end`, optionally split into elements based on `chrom`, a vector of chromosome identifiers (or `NULL` for no splitting). The `genome` argument should be a scalar string and is treated as the `RangesList` universe. See the examples.

Author(s)

Michael Lawrence

Examples

```
## Not run:
GenomicRanges(c(1,2,3), c(5,2,8))
GenomicRanges(c(1,2,3), c(5,2,8), c("chr1", "chr1", "chr2"))
GenomicRanges(c(1,2,3), c(5,2,8), genome = "hg18")

## End(Not run)
```

UCSCData-class *Class "UCSCData"*

Description

Each track in UCSC has an associated `TrackLine` that contains metadata on the track.

Slots

`trackLine`**:** Object of class `"TrackLine"` holding track metadata.

Methods

`export.bed(object, con, variant = c("base", "bedGraph", "bed15"), color, trackLi`
Exports the track and its track line (if `trackLine` is `TRUE`) to `con` in the Browser Extended Display (BED) format. The arguments in `...` are passed to `export.ucsc`.

`export.bed15(object, con, expNames = NULL, ...)` Exports the track and its track line (if `trackLine` is `TRUE`) to `con` in the Bed15 format. The data is taken from the columns named in `expNames`, which defaults to the `expNames` in the track line, if any, otherwise all column names. The arguments in `...` are passed to `export.ucsc`.

`export.gff(object)` Exports the track and its track line (as a comment) to `con` in the General Feature Format (GFF).

`export.ucsc(object, con, subformat, ...)` Exports the track and its track line to `con` in the UCSC meta-format.

`as(object, "UCSCData")` Constructs a `UCSCData` from a `RangedData` instance, by adding a default track line and ensuring that the sequence/chromosome names are compliant with UCSC conventions. If there is a numeric score, the track line type is either "bedGraph" or "wig", depending on the feature density. Otherwise, "bed" is chosen.

Author(s)

Michael Lawrence

See Also

`import` and `export` for reading and writing tracks to and from connections (files), respectively.

`UCSCSchema-class` *UCSC Schema*

Description

This is a preliminary class that describes a table in the UCSC database. The description includes the table name, corresponding genome, row count, and a textual description of the format. In the future, we could provide more table information, like the links and sample data frame. This is awaiting a use-case.

Accessor methods

In the code snippets below, `x`/`object` is a `UCSCSchema` object.

`genome(x)`: Get the genome for the table.

`tableName(x)`: Get the name of the table.

`nrow(x)`: Get the number of rows in the table.

`formatDescription(x)`: Get a textual description of the table format.

Author(s)

Michael Lawrence

Examples

```
## Not run:
session <- browserSession()
genome(session) <- "mm9"
query <- ucscTableQuery(session, "knownGene")
schema <- ucscSchema(query)
nrow(schema)

## End(Not run)
```

UCSCTableQuery-class
Querying UCSC Tables

Description

The UCSC genome browser is backed by a large database, which is exposed by the Table Browser web interface. Tracks are stored as tables, so this is also the mechanism for retrieving tracks. The UCSCTableQuery class represents a query against the Table Browser. Storing the query fields in a formal class facilitates incremental construction and adjustment of a query.

Details

There are five supported fields for a table query:

session The UCSCSession instance from the tables are retrieved. Although all sessions are based on the same database, the set of user-uploaded tracks, which are represented as tables, is not the same, in general.

trackName The name of a track from which to retrieve a table. Each track can have multiple tables. Many times there is a primary table that is used to display the track, while the other tables are supplemental. Sometimes, tracks are displayed by aggregating multiple tables.

tableName The name of the specific table to retrieve. May be NULL, in which case the behavior depends on how the query is executed, see below.

range A genome identifier, a GRanges or a RangesList indicating the portion of the table to retrieve, in genome coordinates. Simply specifying the genome string is the easiest way to download data for the entire genome, and GRangesForUCSCGenome facilitates downloading data for e.g. an entire chromosome.

names Names/accessions of the desired features

A common workflow for querying the UCSC database is to create an instance of UCSCTableQuery using the ucscTableQuery constructor, invoke tableNames to list the available tables for a track, and finally to retrieve the desired table either as a data.frame via getTable or as a RangedData track via track. See the examples.

The reason for a formal query class is to facilitate multiple queries when the differences between the queries are small. For example, one might want to query multiple tables within the track and/or same genomic region, or query the same table for multiple regions. The UCSCTableQuery instance can be incrementally adjusted for each new query. Some caching is also performed, which enhances performance.

Constructor

ucscTableQuery(x, track, range = genome(x), table = NULL, names = NULL):
Creates a UCSCTableQuery with the UCSCSession given as x and the track name given by the single string track. range should be a genome string identifier, a GRanges instance or RangesList instance, and it effectively defaults to genome(x). If the genome is missing, it is taken from the session. The table name is given by table, which may be a single string or NULL. Feature names, such as gene identifiers, may be passed via names as a character vector.

Executing Queries

Below, `object` is a `UCSCTableQuery` instance.

- `track(object, asRangedData = TRUE)`: Retrieves the indicated table as a track, i.e. a `RangedData` instance. Note that not all tables are available as tracks. Pass `asRangedData = FALSE` to obtain a `GRanges` object.
- `getTable(object)`: Retrieves the indicated table as a `data.frame`. Note that not all tables are output in parseable form.
- `tableNames(object)`: Gets the names of the tables available for the session, track and range specified by the query.

Accessor methods

In the code snippets below, x/`object` is a `UCSCTableQuery` object.

- `browserSession(object)`, `browserSession(object) <- value`: Get or set the `UCSCSession` to query.
- `trackName(x)`, `trackName(x) <- value`: Get or set the single string indicating the track containing the table of interest.
- `trackNames(x)` List the names of the tracks available for retrieval for the assigned genome.
- `tableName(x)`, `tableName(x) <- value`: Get or set the single string indicating the name of the table to retrieve. May be `NULL`, in which case the table is automatically determined.
- `range(x)`, `range(x) <- value`: Get or set the `GRanges` indicating the portion of the table to retrieve in genomic coordinates. Any missing information, such as the genome identifier, is filled in using `range(browserSession(x))`. It is also possible to set the genome identifier string or a `RangesList`.
- `names(x)`, `names(x) <- value`: Get or set the names of the features to retrieve. If `NULL`, this filter is disabled.
- `ucscSchema(x)`: Get the `UCSCSchema` object describing the selected table.

Author(s)

Michael Lawrence

Examples

```
## Not run:
session <- browserSession()
genome(session) <- "mm9"
trackNames(session) ## list the track names
## choose the Conservation track for a portion of mm9 chr1
query <- ucscTableQuery(session, "Conservation",
                        GRangesForUCSCGenome("mm9", "chr12",
                                             IRanges(57795963, 57815592))))
## list the table names
tableNames(query)
## get the phastCons30way track
tableName(query) <- "phastCons30way"
## retrieve the track data
track(query)
## get a data.frame summarizing the multiple alignment
tableName(query) <- "multiz30waySummary"
```

```
getTable(query)

genome(session) <- "hg18"
query <- ucscTableQuery(session, "snp129",
                        names = c("rs10003974", "rs10087355", "rs10075230"))
ucscSchema(query)
getTable(query)

## End(Not run)
```

---

activeView-methods *Accessing the active view*

Description

Get the active view.

Methods

The following methods are defined by **rtracklayer**.

object = "BrowserSession" `activeView(object)`: Gets the active `BrowserView` from a browser session.

`activeView(object) <- value`: Sets the active `BrowserView` in a browser session.

BasicTrackLine-class *Class "BasicTrackLine"*

Description

The type of UCSC track line used to annotate most types of tracks (every type except Wiggle).

Objects from the Class

Objects can be created by calls of the form `new("BasicTrackLine", ...)` or parsed from a character vector track line with `as(text, "BasicTrackLine")` or converted from a `GraphTrackLine` using `as(wig, "BasicTrackLine")`.

Slots

`itemRgb`: Object of class `"logical"` indicating whether each feature in a track uploaded as BED should be drawn in its specified color.

`useScore`: Object of class `"logical"` indicating whether the data value should be mapped to color.

`group`: Object of class `"character"` naming a group to which this track should belong.

`db`: Object of class `"character"` indicating the associated genome assembly.

`offset`: Object of class `"numeric"`, a number added to all positions in the track.

`url`: Object of class `"character"` referring to additional information about this track.

`htmlUrl`: Object of class `"character"` referring to an HTML page to be displayed with this track.

`name`: Object of class `"character"` specifying the name of the track.

`description`: Object of class `"character"` describing the track.

`visibility`: Object of class `"character"` indicating the default visible mode of the track, see `UCSCTrackModes`.

`color`: Object of class `"integer"` representing the track color (as from `col2rgb`).

`priority`: Object of class `"numeric"` specifying the rank of the track.

Extends

Class `"TrackLine"`, directly.

Methods

as(object, "character") Export line to its string representation.

as(object, `"GraphTrackLine"`) Convert this line to a graph track line, using defaults for slots not held in common.

Author(s)

Michael Lawrence

References

`http://genome.ucsc.edu/goldenPath/help/customTrack.html#TRACK` for the official documentation.

See Also

`GraphTrackLine` for Wiggle/bedGraph tracks.

---

blocks-methods *Get blocks/exons*

Description

Obtains the block ranges (subranges, usually exons) from an object, such as a `RangedData` imported from a BED file.

Usage

```
blocks(x, ...)
```

Arguments

`x`	The instance from which to obtain the block/exon information. Currently must be a `RangedData`, presumably imported with `import.bed`.
`...`	Additional arguments for methods

Details

For the RangedData method, there must be two columns in x: blockStarts and blockSizes, each field of which should be a comma-separated list of block starts and widths, respectively. This comes from the BED specification.

Author(s)

Michael Lawrence

See Also

import.bed for importing a track from BED, which can store block information.

---

browseGenome *Browse a genome*

Description

A generic function for launching a genome browser.

Usage

```
browseGenome(object, ...)
## S4 method for signature 'RangedDataORRangedDataList'
browseGenome(object,
  browser = "UCSC", range = base::range(object),
  view = TRUE, trackParams = list(), viewParams = list(),
  name = "customTrack", ...)
```

Arguments

object	A list of RangedData instances, e.g. a RangedDataList instance.
browser	The name of the genome browser.
range	A genome identifier or a GRanges or RangesList to display in the initial view.
view	Whether to open a view.
trackParams	Named list of parameters to pass to track<-.
viewParams	Named list of parameters to pass to browserView.
name	The name for the track. Ignored if object is a RangedDataList, in which case the names are taken from the list names.
...	Arguments passed to browserSession.

Value

Returns a BrowserSession.

Author(s)

Michael Lawrence

See Also

BrowserSession and BrowserView, the two main classes for interfacing with genome browsers.

Examples

```
## Not run:
## open UCSC genome browser:
browseGenome()
## to view a specific range:
range <- GRangesForUCSCGenome("hg18", "chr22", IRanges(20000, 50000))
browseGenome(range = range)
## a slightly larger range:
browseGenome(range = range, end = 75000)
## with a track:
track <- import(system.file("tests", "v1.gff", package = "rtracklayer"))
browseGenome(RangedDataList(track))

## End(Not run)
```

BrowserSession-class

Class "BrowserSession"

Description

An object representing a genome browser session. Each session corresponds to a set of loaded tracks and a set of BrowserView instances. Note that this is a virtual class; a concrete implementation is provided by each backend driver.

Objects from the Class

A virtual Class: No objects may be created from it. See browserSession for obtaining an instance of an implementation for a particular genome browser.

Methods

This specifies the API implemented by each browser backend. Note that a backend is not required to support all operations, and that each backend often has additional parameters for each of the methods. See the backend-specific documentation for more details. The only built-in backend is UCSCSession.

If a method is denoted as *virtual*, it must be implemented by the backend to support the corresponding feature. Otherwise, the fallback behavior is described.

virtual `browserView(object, range = range(object), track = trackNames(object), ...)` Constructs a BrowserView of `range` for this session.

virtual `browserViews(object, ...)` Gets the BrowserView instances belonging to this session.

`activeView(object, ...)` Returns the BrowserView that is currently active in the session. Fallback calls `browserViews` and queries each view with `activeView`.

`range(x, ...)` Gets the GRanges representing the range of the genome currently displayed by the browser (i.e. the range shown by the active view) or a default value (possibly `NULL`) if no views exist.

virtual `getSeq(object, range = range(object), ...)` gets a genomic sequence of `range` from this session.

virtual `sequence(object, ...) <- value` Loads a sequence into the session.

virtual `track(object, name = deparse(substitute(track)), view = TRUE, ...) <- valu` Loads one or more `tracks` into the session and optionally open a `view` of the track. The default implementation will coerce `value` to `RangedData`, so the backend should implement at least a method for `RangedData`.

`x[[i]] <- value` Loads the track `value` into session `x`, under the name `i`. Shortcut to above.

`x$name <- value` Loads the track `value` into session `x`, under the name `name`. Shortcut to above.

virtual `track(object, ...)` Gets a track from a session as a `RangedData`.

`x[[i]]` Gets the track named `i` from session `x`. A shortcut to `track`.

`x$name` Gets the track named `name` from session `x`. A shortcut to `track`.

virtual `trackNames(object, ...)` Gets the names of the tracks stored in this session.

virtual `genome(x),genome(x) <- value` Gets or sets the genome identifier (e.g. "hg18") for the session.

virtual `close(con, ...)` Close this session.

`show(object, ...)` Output a textual description of this session.

Author(s)

Michael Lawrence

See Also

`browserSession` for obtaining implementations of this class for a particular genome browser.

`browserSession-methods`

Get a genome browser session

Description

Methods for getting browser sessions.

Methods

The following methods are defined by **rtracklayer**.

object = "character" `browserSession(object, ...)`: Creates a `BrowserSession` from a genome browser identifier. The identifier corresponds to the prefix of the session class name (e.g. "UCSC" in "UCSCSession"). The arguments in ... are passed to the initialization function of the class.

object = "browserView" Gets the `BrowserSession` for the view.

object = "missing" Calls `browserSession("ucsc", ...)`.

BrowserView-class *Class "BrowserView"*

Description

An object representing a genome browser view of a particular segment of a genome.

Objects from the Class

A virtual Class: No objects may be created from it directly. See `browserView` for obtaining an instance of an implementation for a particular genome browser.

Slots

`session`: Object of class `"BrowserSession"` the browser session to which this view belongs.

Methods

This specifies the API implemented by each browser backend. Note that a backend is not guaranteed to support all operations. See the backend-specific documentation for more details. The only built-in backend is `UCSCView`.

`browserSession(object)` Obtains the `BrowserSession` to which this view belongs.

`close(object)` Close this view.

`range(object)` Obtains the `GRanges` displayed by this view.

`trackNames(object)` Gets the names of the visible tracks in the view.

`trackNames(object) <- value` Sets the visible tracks by their names.

`show(object)` Outputs a textual description of this view.

`visible(object)` Get a named logical vector indicating whether each track is visible.

`visible(object) <- value` Set a logical vector indicating the visibility of each track, with the same names and in the same order as that returned by `visible(object)`.

Author(s)

Michael Lawrence

See Also

`browserView` for obtaining instances of this class.

`browserView-methods` *Getting browser views*

Description

Methods for creating and getting browser views.

Usage

```
browserView(object, range, track, ...)
```

Arguments

`object`	The object from which to get the views.
`range`	The `GRanges` or `RangesList` to display. If there are multiple elements, a view is created for each element and a `BrowserViewList` is returned.
`track`	List of track names to make visible in the view.
`...`	Arguments to pass to methods

Methods

The following methods are defined by **rtracklayer**.

object = "UCSCSession" `browserView(object, range = range(object), track = trackNames(object), imagewidth = 800, ...)`: Creates a `BrowserView` of `range` with visible tracks specified by `track`. The `imagewidth` parameter specifies the width of the track image in pixels. `track` may be an instance of `UCSCTrackModes`. Arguments in `...` are passed to `ucscTrackModes` to create the `UCSCTrackModes` instance that will override modes indicated by the `track` parameter.

Examples

```
## Not run:
  session <- browserSession()
  browserView(session,
              GRangesForUCSCGenome("hg19", "chr2", IRanges(20000, 50000)))
  ## only view "knownGene" track
  browserView(session, track = "knownGene")

## End(Not run)
```

browserViews-methods *Getting the browser views*

Description

Methods for getting browser views.

Methods

The following methods are defined by **rtracklayer**.

Gets the instances of `BrowserView` in the session.

See Also

object = "UCSCSession" `browserView` for creating a browser view.

Examples

```
## Not run:
session <- browseGenome()
browserViews(session)

## End(Not run)
```

cpneTrack *CPNE1 SNP track*

Description

A `RangedData` object (created by the `GGtools` package) with features from a subset of the SNPs on chromosome 20 from 60 HapMap founders in the CEU cohort. Each SNP has an associated data value indicating its association with the expression of the CPNE1 gene according to a Cochran-Armitage 1df test. The top 5000 scoring SNPs were selected for the track.

Usage

```
data(cpneTrack)
```

Format

Each feature (row) is a SNP. The association test scores are accessible via `score`.

Source

Vince Carey and the `GGtools` package.

Examples

```
data(cpneTrack)
plot(start(cpneTrack), score(cpneTrack))
```

`export` *Export objects to connections*

Description

Exports (serializes) an object in a given format to a given connection.

Usage

```
export(object, con, format, ...)
```

Arguments

`object`	The object to export.
`con`	The connection to which the object is exported. If this is a character vector, it is assumed to be a filename and a corresponding file connection is created and then closed after exporting the object. If missing, the function will return the output as a character vector, rather than writing to a connection.
`format`	The format of the output. If missing and `con` is a filename, the format is derived from the file extension.
`...`	Parameters to pass to the format-specific export routine.

Details

This function delegates to another function, depending on the specified format. The name of the delegate is of the form `export.format` where `format` is specified by the `format` argument.

Value

If `con` is missing, a character vector containing the string output. Otherwise, nothing is returned.

Author(s)

Michael Lawrence

See Also

`import` for the reverse

Examples

```
  track <- import(system.file("tests", "v1.gff", package = "rtracklayer"))
  ## Not run: export(track, "my.gff", version = "3")
  ## equivalently,
  ## Not run: export(track, "my.gff3")
  ## or
  ## Not run:
  con <- file("my.gff3")
  export(track, con, "gff3")
  close(con)

## End(Not run)
```

```
## or as a string
export(track, format = "gff3")
```

---

export-tracks *Export tracks*

Description

These functions output RangedData instances in various formats.

Usage

```
export.gff(object, con, version = c("1", "2", "3"), source =
           "rtracklayer", append = FALSE, ...)
export.gff1(object, con, ...)
export.gff2(object, con, ...)
export.gff3(object, con, ...)
export.bed(object, con, variant = c("base", "bedGraph", "bed15"),
           color = NULL, append = FALSE, ...)
export.bed15(object, con, expNames = NULL, ...)
export.bedGraph(object, con, ...)
export.wig(object, con,
           dataFormat = c("auto", "variableStep", "fixedStep"), ...)
export.ucsc(object, con, subformat = c("auto", "gff1", "wig", "bed",
           "bed15", "bedGraph"), append = FALSE, ...)
## not yet supported on Windows
export.bw(object, con,
          dataFormat = c("auto", "variableStep", "fixedStep", "bedGraph"),
          seqlengths = NULL, compress = TRUE, ...)
```

Arguments

object	The object to export, such as a RangedData, or anything coercible to a RangedData. If a UCSCData, the track line information is output. In the case of export.bed15, export.bedGraph, export.wig, and export.ucsc, a RangedDataList object with possibly multiple tracks is supported.
con	The connection to which the object is exported.
version	The GFF version, either "1", "2" or "3" (default is "1").
source	The source of the GFF information, for GFF.
variant	Which variant of BED lines to output, not for the user.
color	Recycled vector of colors, as interpreted by col2rgb for BED features. If NULL, the color column in the featureData is used, if any.
dataFormat	The format of the data lines for WIG tracks, see references. The "auto" format uses the most efficient format possible.
subformat	The format of the tracks within the UCSC container. If "auto", the type is determined from the trackline. If object is not a UCSCData, this essentially means "wig" or "bedGraph" (depending on the density) if there is a numeric score, else "bed".

expNames	Names of the columns in object that hold the experimental data. Defaults to all column names, unless object is a UCSCData, in which case the expNames field is taken from the track line, if it exists.
seqlengths	The lengths of each sequence in object. If NULL, the chromosome lengths are retrieved for the genome specified on object, if possible.
append	Logical, whether to append the output to the connection
compress	Logical, indicating whether to compress the bigWig output
...	For export.gff1, export.gff2 and export.gff3: arguments to pass to export.gff. For export.bed: arguments to pass to methods. For export.bed15, export.bedGraph and export.wig: arguments to pass to export.ucsc. For export.ucsc: arguments to pass to export.subformat or to set on the slots of the TrackLine subclass corresponding to subformat.

Details

The following is some advice for choosing a file format.

GFF The General Feature Format is meant to represent any set of genomic features, with application-specific columns represented as "attributes". There are three principal versions (1, 2, and 3). This is a good format for interoperating with other genomic tools. UCSC supports GFF1, but it needs to be encapsulated in the UCSC metaformat, i.e. `export.ucsc(subformat = "gff1")`.

BED The Browser Extended Display format is for displaying tracks in a genome browser, in particular UCSC. There are many options to control the appearance of the track, see GraphTrackLine. To output a track line when object is not a UCSCData, call `export.ucsc(subformat = "bed")`.

BED15 An extension of BED with 15 columns, Bed15 is meant to represent data from microarray experiments. Multiple samples/columns are supported, and the data is displayed as a compact heatmap. With 15 columns per feature, this format is probably too verbose for e.g. ChIP-seq coverage (use multiple WIG tracks instead).

BEDGRAPH A variant of BED that represents experimental data more compactly than BED and especially BED15, although only one sample is supported. The data is displayed as a bar or line graph. For dense data, WIG is preferred.

WIG The Wiggle format is meant for storing dense numerical data, such as the coverage from a ChIP-seq experiment. The data is displayed as a bar or line graph.

In summary, BED is usually best for displaying qualitative features or sparse quantiative features (like ChIP-seq peaks), while WIG is usually best for displaying dense data like coverage.

In general, columns in the RangedData are mapped to the column in the track format of the same name. For example, a column named "itemRgb" will be mapped to the corresponding column in BED-formatted output, while it is ignored for other formats. Missing values are mapped between NA in R and the format-specific missing value indicator, usually ".". The following describes how the RangedData object is mapped to each track format. Default values for columns are given in parentheses.

GFF Maps columns named "source" ("rtracklayer"), "feature" ("sequence"), "score" ("."), "strand" ("."), "frame" ("."), and (version 1 only) "group" (seqname). In GFF versions 2 and 3, extra columns are mapped to attributes.

BED Maps columns named "name" ("."), "score" ("."), "strand" ("."), "thickStart" (start), "thickEnd" (end), "itemRgb" ("0,0,0"), "blockSizes", and "blockStarts". Note that the BED field "blockCounts" is derived automatically. The intervals specified by "thickStart", "thickEnd"

and "blockStarts" are 0-based, half-open as in BED. Note that this is different from the chromosome start/end stored in the `Ranges` object (1-based, closed). The "itemRgb" column should be specified in a format understood by `col2rgb`.

BED15 In addition to the behavior for `BED` above, encodes columns named by the `expNames` parameter into the fields "expCount", "expIds" and "expScores".

BEDGRAPH The "score" column is used for the quantitative values.

WIG The "score" column is used for the quantitative values.

The graph formats do not encode a strand. Thus, when targeting the UCSC format, if a track contains features from multiple strands, one track will be output for each strand. The string "m", "p" or "NA" is appended to the base track name for the minus, plus and NA/* strand, respectively.

Value

If `con` is missing, a character vector containing the string output, otherwise nothing.

Author(s)

Michael Lawrence

References

GFF1 and GFF2 http://www.sanger.ac.uk/Software/formats/GFF

GFF3 http://www.sequenceontology.org/gff3.shtml

BED http://genome.ucsc.edu/goldenPath/help/customTrack.html#BED

WIG http://genome.ucsc.edu/goldenPath/help/wiggle.html

UCSC http://genome.ucsc.edu/goldenPath/help/customTrack.html

See Also

See `export` for the high-level interface to these functions.

Examples

```
dummy <- file() # dummy file connection for demo
track <- import(system.file("tests", "bed.wig", package = "rtracklayer"))
## output a track as GFF2
export.gff(track, dummy, version = "2")
## equivalently
export.gff2(track, dummy)
## output as WIG string in variableStep format
wig <- export.wig(track, dummy, dataFormat = "variableStep")
## output multiple tracks in UCSC meta-format
track2 <- import(system.file("tests", "v1.gff", package = "rtracklayer"))
## output to WIG
library(IRanges)  # for the RangedDataList() constructor
export.ucsc(RangedDataList(track, track2), dummy, subformat = "wig")
```

genomeBrowsers *Get available genome browsers*

Description

Gets the identifiers of the loaded genome browser drivers.

Usage

```
genomeBrowsers(where = topenv(parent.frame()))
```

Arguments

where	The environment in which to search for drivers.

Details

This searches the specified environment for classes that extend BrowserSession. The prefix of the class name, e.g. "ucsc" in "UCSCSession", is returned for each driver.

Value

A character vector of driver identifiers.

Author(s)

Michael Lawrence

See Also

browseGenome and browserSession that create browserSession implementations given an identifier returned from this function.

import *Importing objects*

Description

Imports an object from a connection according to a specified format.

Usage

```
import(con, format, text, ...)
```

Arguments

con	The connection through which the data is received. If this is a character vector, it is assumed to be a filename.
format	The format in which to expect the input. If omitted and con is a filename, the format is taken from the file extension.
text	If con is missing, this can be a character vector directly providing the string data to import.
...	Arguments to pass to the format-specific import routines.

Details

This function delegates to a format-specific function named according to the scheme import.format where format is specified by the format parameter.

Value

The object parsed from the connection or text.

Author(s)

Michael Lawrence

See Also

export to do the reverse.

Examples

```
track <- import(system.file("tests", "bed.wig", package = "rtracklayer"))
track <- import(system.file("tests", "v1.gff", package = "rtracklayer"), version = "1")
# or
track <- import(system.file("tests", "v1.gff", package = "rtracklayer"), "gff1")
```

---

import.gff *Importing tracks*

Description

These are the functions for importing RangedData instances from connections or text.

Usage

```
import.gff(con, version = c("1", "2", "3"), genome = "hg18",
           asRangedData = TRUE, colnames = NULL)
import.gff1(con, ...)
import.gff2(con, ...)
import.gff3(con, ...)
import.bed(con, variant = c("base", "bedGraph", "bed15"),
           trackLine = TRUE, genome = "hg18",
           asRangedData = TRUE, colnames = NULL, ...)
import.bed15(con, genome = "hg18", asRangedData = TRUE, ...)
```

```
import.bedGraph(con, genome = "hg18", asRangedData = TRUE, ...)
import.wig(con, genome = "hg18", asRangedData = TRUE, ...)
import.ucsc(con,
            subformat = c("auto", "gff1", "wig", "bed", "bed15", "bedGraph"),
            drop = FALSE, asRangedData = TRUE, ...)
## not yet supported on Windows
import.bw(con, selection = BigWigSelection(...), ...)
```

Arguments

con	The connection, filename or URL from which to receive the input.
version	The version of GFF ("1", "2" or "3").
genome	The genome to set on the imported track.
asRangedData	A logical value. If TRUE, a RangedData object is returned. If FALSE, a GRanges object is returned.
variant	Variant of BED lines, not for the user.
trackLine	Whether the BED data has a track line (it normally does though track lines are not mandatory).
subformat	The expected subformat of the UCSC data. If "auto", automatic detection of the subformat is attempted.
drop	If TRUE and there is only one track in the UCSC data, return the track instead of a list.
selection	A RangedSelection object indicating the intervals to retrieve from a bigWig file. Note that this retrieval is very efficient, due to the indexing of the bigWig format.
colnames	Character vector indicating which columns (excluding the required sequence name, start and end) should be imported. If NULL, all columns are imported. This allows some significant optimizations, especially when it is not necessary to import GFF attributes attributes, which are expensive to parse.
...	For import.gff1, import.gff2 and import.gff3: arguments to pass to import.gff. For import.ucsc: arguments to pass on to import.subformat. For the others, arguments to pass to methods.

Value

For all but import.ucsc, an instance of RangedData (or one of its subclasses) or GRanges if asRangedData is TRUE or FALSE respectively.

For import.ucsc when drop is FALSE, an instance of RangedDataList or GRangesList if asRangedData is TRUE or FALSE respectively.

Author(s)

Michael Lawrence and Patrick Aboyoun

References

GFF1 and GFF2 http://www.sanger.ac.uk/Software/formats/GFF

GFF3 http://www.sequenceontology.org/gff3.shtml

BED http://genome.ucsc.edu/goldenPath/help/customTrack.html#BED

WIG http://genome.ucsc.edu/goldenPath/help/wiggle.html

UCSC http://genome.ucsc.edu/goldenPath/help/customTrack.html

See Also

import for the high-level interface to these routines.

Examples

```
  # import a GFF V2 file
  gffRD <- import.gff(system.file("tests", "v2.gff", package = "rtracklayer"),
                      version = "2")
  gffGR <- import.gff(system.file("tests", "v2.gff", package = "rtracklayer"),
                      version = "2", asRangedData = FALSE)
  # or
  gffRD <- import.gff2(system.file("tests", "v2.gff", package = "rtracklayer"))
  gffGR <- import.gff2(system.file("tests", "v2.gff", package = "rtracklayer"),
                       asRangedData = FALSE)

  # import a WIG file
  wigRD <- import.wig(system.file("tests", "bed.wig", package = "rtracklayer"))
  wigGR <- import.wig(system.file("tests", "bed.wig", package = "rtracklayer"),
                      asRangedData = FALSE)
  # or
  wigRD <- import.ucsc(system.file("tests", "bed.wig", package = "rtracklayer"),
                       subformat = "wig", drop = TRUE)
  wigGR <- import.ucsc(system.file("tests", "bed.wig", package = "rtracklayer"),
                       subformat = "wig", drop = TRUE, asRangedData = FALSE)

  # bigWig
## Not run:
  bw <- import(system.file("tests", "test.bw", package = "rtracklayer"),
               ranges = GenomicRanges::GRanges("chr19", IRanges(1, 6e7)))

## End(Not run)
```

---

sequence<-methods *Load a sequence*

Description

Methods for loading sequences.

Methods

No methods are defined by **rtracklayer** for the `sequence(object, ...) <- value` generic.

track<-methods *Laying tracks*

Description

Methods for loading RangedData instances (tracks) into genome browsers.

Usage

```
## S4 replacement method for signature 'BrowserSession,RangedData'
track(object, name = deparse(substitute(track)), view = FALSE, ...) <- value
```

Arguments

object	A BrowserSession into which the track is loaded.
value	The track(s) to load.
name	The name(s) of the track(s) being loaded.
view	Whether to create a view of the track after loading it.
...	Arguments to pass on to methods.

Methods

The following methods are defined by **rtracklayer**. A browser session implementation must implement a method for either RangedData or RangedDataList. The base browserSession class will delegate appropriately.

object = "BrowserSession", value = "RangedData" Load this track into the session.

object = "BrowserSession", value = "RangedDataList" Load all tracks into the session.

object = "UCSCSession", value = "RangedDataList" track(object, name = deparse(substitute(t
view = FALSE, format = "gff", ...) <- value: Load the tracks into the session using the specified format. The arguments in ... are passed on to export.ucsc, so they could be slots in a TrackLine subclass or parameters to pass on to the export function for format.

See Also

track for getting a track from a session.

Examples

```
## Not run:
  session <- browserSession()
  track <- import(system.file("tests", "v1.gff", package = "rtracklayer"))
  track(session, "My Track") <- track

## End(Not run)
```

liftOver — *Lift intervals between genome builds*

Description

A reimplementation of the UCSC liftover tool for lifting features from one genome build to another. In our preliminary tests, it is significantly faster than the command line tool. Like the UCSC tool, a chain file is required input.

Usage

```
liftOver(x, chain, ...)
```

Arguments

x	The intervals to lift-over, usually a GRanges.
chain	A Chain object, usually imported with import.chain.
...	Arguments for methods.

Value

A GRanges object, with intervals mapped through the chain.

Author(s)

Michael Lawrence

References

http://genome.ucsc.edu/cgi-bin/hgLiftOver

targets — *microRNA target sites*

Description

A data frame of human microRNA target sites retrieved from MiRBase. This is a subset of the hsTargets data frame in the microRNA package. See the rtracklayer vignette for more details.

Usage

```
data(targets)
```

Format

A data frame with 2981 observations on the following 6 variables.

`name` The miRBase ID of the microRNA.

`target` The Ensembl ID of the targeted transcript.

`chrom` The name of the chromosome for target site.

`start` Target start position.

`end` Target stop position.

`strand` The strand of the target site, `"+"`, or `"-"`.

Source

The `microRNA` package, dataset `hsTargets`. Originally MiRBase (http://microrna.sanger.ac.uk/).

Examples

```
data(targets)
targetTrack <- with(targets,
    GenomicData(IRanges(start, end),
                strand = strand, chrom = chrom))
```

---

tracks-methods *Accessing track names*

Description

Methods for getting and setting track names.

Methods

The following methods are defined by **rtracklayer** for **getting** track names via the generic `trackNames(object, ...)`.

Get the tracks loaded in the session.

object = "UCSCSession" Get the visible tracks according to the modes (all tracks not set to "hide").

object = "UCSCTrackModes" Get the visible tracks according to the modes (all tracks not set to "hide").

object = "UCSCView" Get the visible tracks in the view.

The following methods are defined by **rtracklayer** for **setting** track names via the generic `trackNames(object) <- value`.

object = "UCSCTrackModes" Sets the tracks that should be visible in the modes. All specified tracks with mode "hide" in `object` are set to mode "full". Any tracks in `object` that are not specified in the value are set to "hide". No other modes are changed.

object = "UCSCView" Sets the visible tracks in the view. This opens a new web browser with only the specified tracks visible.

ucscGenomes *Get available genomes on UCSC*

Description

Get a `data.frame` describing the available UCSC genomes.

Usage

```
ucscGenomes()
```

Value

A `data.frame` with the following columns:

`db`	UCSC DB identifier (e.g. "hg18")
`species`	The name of the species (e.g. "Human")
`date`	The date the genome was built
`name`	The official name of the genome build

Author(s)

Michael Lawrence

See Also

`UCSCSession` for details on specifying the genome.

Examples

```
ucscGenomes()
```

UCSCSession-class *Class "UCSCSession"*

Description

An implementation of `BrowserSession` for the UCSC genome browser.

Objects from the Class

Objects can be created by calls of the form `browserSession("ucsc", url = "http://genome.ucsc.edu/cgi-bin", ...)`. The arguments in `...` correspond to libcurl options, see `getCurlHandle`. Setting these options may be useful e.g. for getting past a proxy.

Slots

`url`: Object of class `"character"` holding the base URL of the UCSC browser.

`hguid`: Object of class `"numeric"` holding the user identification code.

`views`: Object of class `"environment"` containing a list stored under the name "instances". The list holds the instances of `BrowserView` for this session.

Extends

Class "BrowserSession", directly.

Methods

`browserView(object, range = range(object), track = trackNames(object), ...)` Creates a `BrowserView` of `range` with visible tracks specified by `track`. `track` may be an instance of `UCSCTrackModes`. Arguments in `...` should match parameters to a `ucscTrackModes` method for creating a `UCSCTrackModes` instance that will be merged with and override modes indicated by the `track` parameter.

`browserViews`**(object)** Gets the `BrowserView` instances for this session.

`range`**(x, asRangedData = TRUE)** Gets the `GRanges` last displayed in this session. Set `asRangedData` to `FALSE` to obtain a `GRanges` object.

`genome(x)` Gets the genome identifier of the session, i.e. `genome(range(x))`.

`range(x) <- value` Sets `value`, usually a `GRanges` object or `RangesList`, as the range of session `x`. Note that this setting only lasts until a view is created or manipulated. This mechanism is useful, for example, when treating the UCSC browser as a database, rather than a genome viewer.

`genome(x) <- value` Sets the genome identifier on the range of session `x`.

`getSeq(object, range, track = "Assembly")` Gets the sequence in `range` and `track`.

`track(object, name = names(track), format = "auto", ...) <- value` Loads a track, stored under `name` and formatted as `format`. The "auto" format resolves to "bed" for qualitative data. For quantitative data, i.e., data with a numeric `score` column, "wig" or "bedGraph" is chosen, depending on how well the data compresses into wig. The arguments in `...` are passed on to `export.ucsc`, so they could be slots in a `TrackLine` subclass (and thus specify visual attributes like color) or parameters to pass on to the export function for `format`.

`track(object, name, range = range(object), table = NULL)` Retrieves a `RangedData` with features in `range` from track named `name`. Some built-in tracks have multiple series, each stored in a separate database table. A specific table may be retrieved by passing its name in the `table` parameter. See `tableNames` for a way to list the available tables.

`trackNames`**(object)** Gets the names of the tracks stored in the session.

`ucscTrackModes`**(object)** Gets the default view modes for the tracks in the session.

Author(s)

Michael Lawrence

See Also

`browserSession` for creating instances of this class.

TrackLine-class *Class "TrackLine"*

Description

An object representing a "track line" in the UCSC format. There are two concrete types of track lines: BasicTrackLine (used for most types of tracks) and GraphTrackLine (used for graphical tracks). This class only declares the common elements between the two.

Objects from the Class

Objects can be created by calls of the form `new("TrackLine", ...)` or parsed from a character vector track line with `as(text, "TrackLine")`. But note that UCSC only understands one of the subclasses mentioned above.

Slots

`name`**:** Object of class `"character"` specifying the name of the track.

`description`**:** Object of class `"character"` describing the track.

`visibility`**:** Object of class `"character"` indicating the default visible mode of the track, see UCSCTrackModes.

`color`**:** Object of class `"integer"` representing the track color (as from col2rgb).

`priority`**:** Object of class `"numeric"` specifying the rank of this track.

Methods

as(object, "character") Export line to its string representation.

Author(s)

Michael Lawrence

References

http://genome.ucsc.edu/goldenPath/help/customTrack.html#TRACK for the official documentation.

See Also

BasicTrackLine (used for most types of tracks) and GraphTrackLine (used for Wiggle/bedGraph tracks).

UCSCTrackModes-class
Class "UCSCTrackModes"

Description

A vector of view modes ("hide", "dense", "full", "pack", "squish") for each track in a UCSC view.

Objects from the Class

Objects may be created by calls of the form `ucscTrackModes(object = character(), hide = character(), dense = character(), pack = character(), squish = character(), full = character())`, where `object` should be a character vector of mode names (with its `names` attribute specifying the corresponding track names). The other parameters should contain track names that override the modes in `object`. Later parameters override earlier ones, so, for example, if a track is named in `hide` and `full`, it is shown in the full view mode.

Slots

`.Data`: Object of class `"character"` holding the modes ("hide", "dense", "full", "pack", "squish"), with its `names` attribute holding corresponding track names.

`labels`: Object of class `"character"` holding labels (human-readable names) corresponding to each track/mode.

Extends

Class `"character"`, from data part. Class `"vector"`, by class "character", distance 2.

Methods

`trackNames(object)` Gets the names of the visible tracks (those that do not have mode "hide").

`trackNames(object) <- value` Sets the names of the visible tracks. Any tracks named in `value` are set to "full" if the are currently set to "hide" in this object. Any tracks not in `value` are set to "hide". All other modes are preserved.

`object[i]` Gets the track mode of the tracks indexed by `i`, which can be any type of index supported by character vector subsetting. If `i` is a character vector, it indexes first by the internal track IDs (the `names` on `.Data`) and then by the user-level track names (the `labels` slot).

`object[i] <- value` Sets the track modes indexed by `i` (in the same way as in `object[i]` above) to those specified in `value`.

Author(s)

Michael Lawrence

See Also

`UCSCView` on which track view modes may be set.

`ucscTrackModes-methods`
Accessing UCSC track modes

Description

Generics for getting and setting UCSC track visibility modes ("hide", "dense", "full", "pack", "squish").

Methods

The following methods are defined by **rtracklayer** for **getting** the track modes through the generic `ucscTrackModes(object, ...)`.

`function(object, hide = character(), dense = character(), pack = character(), squish = character(), full = character())` Creates an instance of `UCSCTrackModes` from `object`, a character vector of mode names, with the corresponding track ids given in the `names` attribute. Note that `object` can be a `UCSCTrackModes` instance, as `UCSCTrackModes` extends `character`. The other parameters are character vectors identifying the tracks for each mode and overriding the modes specified by `object`.

object = "character", object = "missing" The same interface as above, except `object` defaults to an empty character vector.

object = "UCSCView" Gets modes for tracks in the view.

object = "UCSCSession" Gets default modes for the tracks in the session. These are the modes that will be used as the default for a newly created view.

The following methods are defined by **rtracklayer** for **setting** the track modes through the generic `ucscTrackModes(object) <- value`.

object = "UCSCView", value = "UCSCTrackModes" Sets the modes for the tracks in the view.

object = "UCSCView", value = "character" Sets the modes from a character vector of mode names, with the corresponding track names given in the `names` attribute.

See Also

`trackNames` and `trackNames<-` for just getting or setting which tracks are visible (not of mode "hide").

Examples

```
# Tracks "foo" and "bar" are fully shown, "baz" is hidden
modes <- ucscTrackModes(full = c("foo", "bar"), hide = "baz")
# Update the modes to hide track "bar"
modes2 <- ucscTrackModes(modes, hide = "bar")
```

UCSCView-class *Class "UCSCView"*

Description

An object representing a view of a genome in the UCSC browser.

Objects from the Class

Calling `browserView(session, range = range(object), track = trackNames(object), ...)` creates `BrowserView` of `range` with visible tracks specified by `track`. `track` may be an instance of `UCSCTrackModes`. Arguments in `...` should match parameters to a `ucscTrackModes` method for creating a `UCSCTrackModes` instance that will be merged with and override modes indicated by the `track` parameter.

Slots

`hgsid`**:** Object of class `"numeric"`, which identifies this view to UCSC.

`session`**:** Object of class `"BrowserSession"` to which this view belongs.

Extends

Class `"BrowserView"`, directly.

Methods

`activeView(object)` Obtains a logical indicating whether this view is the active view.

`range(object)` Obtains the `GRanges` displayed by this view.

`range(object) <- value` Sets the `GRanges` or `RangesList` displayed by this view.

`trackNames(object)` Gets the names of the visible tracks in this view.

`trackNames(object) <- value` Sets the visible tracks by name.

`visible(object)` Get a named logical vector indicating whether each track is visible.

`visible(object) <- value` Set a logical vector indicating the visibility of each track, in the same order as returned by `visible(object)`.

`ucscTrackModes(object)` Obtains the `UCSCTrackModes` for this view.

`ucscTrackModes(object) <- value` Sets the `UCSCTrackModes` for this view. The `value` may be either a `UCSCTrackModes` instance or a character vector that will be coerced by a call to `ucscTrackModes`.

Author(s)

Michael Lawrence

See Also

`browserView` for creating instances of this class.

GraphTrackLine-class
Class "GraphTrackLine"

Description

A UCSC track line for graphical tracks.

Objects from the Class

Objects can be created by calls of the form `new("GraphTrackLine", ...)` or parsed from a character vector track line with `as(text, "GraphTrackLine")` or converted from a `BasicTrackLine` using `as(basic, "GraphTrackLine")`.

Slots

`altColor`: Object of class `"integer"` giving an alternate color, as from `col2rgb`.

`autoScale`: Object of class `"logical"` indicating whether to automatically scale to min/max of the data.

`gridDefault`: Object of class `"logical"` indicating whether a grid should be drawn.

`maxHeightPixels`: Object of class `"numeric"` of length three (max, default, min), giving the allowable range for the vertical height of the graph.

`graphType`: Object of class `"character"`, specifying the graph type, either "bar" or "points".

`viewLimits`: Object of class `"numeric"` and of length two specifying the data range (min, max) shown in the graph.

`yLineMark`: Object of class `"numeric"` giving the position of a horizontal line.

`yLineOnOff`: Object of class `"logical"` indicating whether the `yLineMark` should be visible.

`windowingFunction`: Object of class `"character"`, one of "maximum", "mean", "minimum", for removing points when the graph shrinks.

`smoothingWindow`: Object of class `"numeric"` giving the window size of a smoother to pass over the graph.

`type`: Scalar `"character"` indicating the type of the track, either `"wig"` or `"bedGraph"`.

`name`: Object of class `"character"` specifying the name of the track.

`description`: Object of class `"character"` describing the track.

`visibility`: Object of class `"character"` indicating the default visible mode of the track, see `UCSCTrackModes`.

`color`: Object of class `"integer"` representing the track color (as from `col2rgb`).

`priority`: Object of class `"numeric"` specifying the rank of this track.

Extends

Class `"TrackLine"`, directly.

Methods

as(object, "character") Export line to its string representation.

as(object, "BasicTrackLine") Convert this line to a basic UCSC track line, using defaults for slots not held in common.

Author(s)

Michael Lawrence

References

Official documentation: http://genome.ucsc.edu/goldenPath/help/wiggle.html.

See Also

export.wig, export.bedGraph for exporting graphical tracks.

Index