rtracklayer

November 11, 2009

R topics documented:

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 `WigTrackLine` 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, "WigTrackLine") Convert this line to a WIG 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

WigTrackLine for Wiggle 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 = RangedDataList(),
  browser = "ucsc", range = range(tracks),
  view = TRUE, trackParams = list(), viewParams = list(), ...)
```

Arguments

`object`	A list of `RangedData` instances, e.g. a `RangedDataList` instance.
`browser`	The name of the genome browser.
`range`	The `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`.
`...`	Arguments corresponding to slots in `RangesList` that override those in `range`.

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 <- GenomicRanges("hg18", "chr22", 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 guaranteed 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`.

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

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

`activeView(object, ...)` Returns the `BrowserView` that is currently active in the session.

`range(x, ...)` Gets the `RangesList` 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.

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

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

`track(object, name = deparse(substitute(track)), view = TRUE, ...) <- value` Loads one or more `tracks` into the session and optionally open a `view` of the track.

`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.

`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`.

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

`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 `RangesList` 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 `RangesList` to display.
`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, GenomicRanges(20000, 50000, "chr2"))
  ## 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-tracks *Export tracks*

Description

These functions output RangedData instances in various formats.

Usage

```
export.gff(object, con, version = c("1", "2", "3"), source =
           "rtracklayer")
export.gff1(object, con, ...)
export.gff2(object, con, ...)
export.gff3(object, con, ...)
export.bed(object, con, wig = FALSE, color = NULL, ...)
export.wig(object, con,
           dataFormat = c("auto", "bed", "variableStep", "fixedStep"), ...)
export.ucsc(object, con, subformat = c("auto", "gff1", "wig", "bed"), ...)
```

Arguments

`object` The object to export, such as a RangedData. If a UCSCData, the track line information is output. In the case of `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.

`wig` Whether to output the WIG variant of BED lines, not to be used directly.

`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", "wig" is used for numeric data, else "bed".

`...` For `export.gff1`, `export.gff2` and `export.gff3`: arguments to pass to `export.gff`. For `export.bed` and `export.wig` : arguments to pass to methods. For `export.ucsc`: arguments to pass to `export.subformat` or to set on the slots of the TrackLine subclass corresponding to `subformat`.

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 with BED line format
export.ucsc(RangedDataList(track, track2), dummy, subformat = "wig", dataFormat = "bed"
```

`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")
```

`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.

`getSeq-methods` *Retrieving a genome sequence*

Description

Methods for retrieving the sequence of a `RangesList` from an object.

Methods

The following methods are defined by **rtracklayer** for `getSeq(object, range = range(object), ...)`.

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

See Also

`sequence<-` for storing sequences.

`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")
import.gff1(con, ...)
import.gff2(con, ...)
import.gff3(con, ...)
import.bed(con, wig = FALSE, trackLine = !wig, genome = "hg18", ...)
import.wig(con, genome = "hg18", ...)
import.ucsc(con, subformat = c("auto", "gff1", "wig", "bed"),
            drop = FALSE, ...)
```

Arguments

`con` The connection from which to receive the input.

`version` The version of GFF ("1", "2" or "3").

`genome` The genome to set on the imported track.

`wig` Whether the BED lines are expected to be WIG formatted (not for public use).

`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.

`...` For `import.gff1`, `import.gff2` and `import.gff3`: arguments to pass to `import.gff`. For `import.bed` and `import.wig`: arguments to pass to methods. For `import.ucsc`: arguments to pass on to `import.subformat`.

Value

An instance of `RangedData` or one of its subclasses, except for `import.ucsc`, which returns a `RangedDataList` instance, unless there is one track and the `drop` parameter is `TRUE`.

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

`import` for the high-level interface to these routines.

Examples

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

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

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")
```

`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`.

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)
```

RangedData-methods *Data on a Genome*

Description

The rtracklayer package adds convenience methods on top of RangedData manipulate data on genomic ranges. The spaces are now called chromosomes (but could still refer to some other type of sequence). The universe refers to the genome and there is formal treatment of an optional strand variable.

Accessors

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

chrom(x): Gets the chromosome names (a factor) over the ranges in 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.

strand(x): Gets the strand factor in x, with the standard levels: -, + and *, referring to the negative, positive and either/both strands, respectively. Any strand factor stored in the RangedData should have those levels. NA's are allowed; the value is all NA if no strand has been specified.

score(x): gets the column representing a "score" in x, as a vector. This is the column named score, or, if this does not exist, the first column, if it is numeric. Otherwise, NULL is returned.

score(x) <- value: sets the column named score to value, which should be a numeric vector of length equal to the number of rows.

Constructor

GenomicData(ranges, ..., strand = NULL, chrom = NULL, genome = NULL): Constructs a RangedData instance with the given ranges and variables in ... (see the RangedData constructor). If non-NULL, strand 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. To get these levels, call levels(strand()). chrom is analogous to splitter in RangedData; if non-NULL it should be coercible to a factor indicating how the ranges, variables and strand should be split up across the chromosomes. The genome argument should be a scalar string and is treated as the RangedData universe. See the examples.

Author(s)

Michael Lawrence

Examples

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

## just ranges
gr <- GenomicData(range1)

## with a genome (universe)
gr <- GenomicData(range1, genome = "hg18")
genome(gr) ## "hg18"

## with some data
filter <- c(1L, 0L, 1L)
score <- c(10L, 2L, NA)
strand <- factor(c("+", NA, "-"), levels = levels(strand()))
gr <- GenomicData(range1, score, genome = "hg18")
gr[["score"]]
strand(gr) ## all NA
gr <- GenomicData(range1, score, filt = filter, strand = strand)
gr[["filt"]]
strand(gr) ## equal to 'strand'

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="")

gr <- GenomicData(ranges, score, chrom = chrom, genome = "hg18")
chrom(gr) # equal to 'chrom'
gr[["score"]] # unlists over the chromosomes
score(gr)
gr[1][["score"]] # equal to score[1:3]
```

`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. This information can be represented by 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)`: Gets the chromosome names (a factor) over the ranges in `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)`: 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

```
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")
```

`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 = "UCSCSessionobject = "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.

`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.

`genome`: Object of class `"character"` identifying the genome of this track, e.g. "hg18".

`assayData`: Object of class `"AssayData"` holding the experimental measurements under `dataVals`.

`phenoData`: Object of class `"AnnotatedDataFrame"` holding the experimental design matrix.

`featureData`: Object of class `"AnnotatedDataFrame"` holding feature information, generally including columns `chrom` (chromosome), `start` (start position), `end` (end position) and `strand` (strand on the DNA: "+", "-", or `NA`).

`experimentData`: Object of class `"MIAME"` holding experimental metadata.

`annotation`: Object of class `"character"` referring to the annotation dataset.

`.__classVersion__`: Object of class `"Versions"` holding version information.

Methods

- **`export.bed(object, con, wig = FALSE, trackLine = !wig)`** Exports the track and its track line (if `trackLine` is `TRUE`) to `con` in the Browser Extended Display (BED) format.
- **`export.gff(object)`** Exports the track and its track line (as a comment) to `con` in the General Feature Format (GFF).
- **`export.ucsc(object)`** 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.

Author(s)

Michael Lawrence

See Also

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

`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 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 override slots in `range` or else match parameters to a `ucscTrackModes` method for creating a `UCSCTrackModes` instance that will override modes indicated by the `track` parameter.

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

range(x) Gets the `RangesList` last displayed in this session.

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

range(x) <- value Sets `value`, a `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), view = TRUE, format = "gff", ...) <- value Loads a track, stored under `name` and formatted as `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`.

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.

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 four 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 `RangesList` indicating the portion of the table to retrieve, in genome coordinates. The `genome` indicated by the `RangesList` also determines which tracks are available and must always be non-`NULL`. If the `RangesList` is empty, the table is downloaded for the entire genome.

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 = GenomicRanges(), table = NULL)`: Creates a `UCSCTableQuery` with the `UCSCSession` given as `x` and the track name given by the single string `track`. `range` should be a `RangesList` instance, and it effectively defaults to `range(x)`. Any missing information in `range`, often the genome identifier, is filled in from `range(x)`. The table name is given by `table`, which may be a single string or `NULL`.

Executing Queries

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

`track(object)`: Retrieves the indicated table as a track, i.e. a `RangedData` instance. Note that not all tables are available as tracks.

`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.

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 RangesList 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)).

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",
                        GenomicRanges(57795963, 57815592, "chr12"))
## 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)
## End(Not run)
```

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 WigTrackLine (used for Wiggle 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 WigTrackLine (used for Wiggle 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`.

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. Class "characterORMIAME", 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 names given in the `names` attribute. Note that `object` can be a `UCSCTrackModes` instance. The other parameters are character vectors naming 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

Objects are created by calling browserView(session, ...) where session is a UCSCSession.

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 RangesList displayed by this view.

range(object) <- value Sets the 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.

`WigTrackLine-class` *Class "WigTrackLine"*

Description

A UCSC track line for Wiggle tracks.

Objects from the Class

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

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.

`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` for exporting tracks in the Wiggle format.

Index