Package ‘rtracklayer’

April 23, 2016

Title R interface to genome browsers and their annotation tracks

Version 1.30.4

Author Michael Lawrence, Vince Carey, Robert Gentleman

Depends R (>= 2.10), methods, GenomicRanges (>= 1.21.20)

Imports XML (>= 1.98-0), BiocGenerics (>= 0.13.8), S4Vectors (>=
0.7.11), IRanges (>= 2.3.7), XVector (>= 0.9.4), GenomeInfoDb
(>= 1.3.14), Biostrings (>= 2.37.1), zlibbioc, RCurl (>=
1.4-2), Rsamtools (>= 1.17.8), GenomicAlignments (>= 1.5.4),
tools

Suggests BSgenome (>= 1.33.4), humanStemCell, microRNA (>= 1.1.1),
genefilter, limma, org.Hs.eg.db, hgu133plus2.db,
BSgenome.Hsapiens.UCSC.hg19, TxDb.Hsapiens.UCSC.hg19.knownGene

LinkingTo S4Vectors, IRanges, XVector

Description Extensible framework for interacting with multiple genome
browsers (currently UCSC built-in) and manipulating
annotation tracks in various formats (currently GFF, BED, bedGraph, BED15,
WIG, BigWig and 2bit built-in). The user may export/import tracks to/from the
supported browsers, as well as query and modify the browser state, such as the
current viewport.

Maintainer Michael Lawrence <`michafla@gene.com`>

License Artistic-2.0 + file LICENSE

Collate io.R web.R ranges.R trackDb.R browser.R ucsc.R readGFF.R gff.R
bed.R wig.R utils.R bigWig.R chain.R quickload.R twobit.R
fasta.R tabix.R bam.R trackTable.R index.R compression.R
test_rtracklayer_package.R ncbi.R

biocViews Annotation,Visualization,DataImport

NeedsCompilation yes

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.

`asBED` *Coerce to BED structure*

Description

Coerce the structure of an object to one following BED-like conventions, i.e., with columns for blocks and thick regions.

Usage

```
asBED(x, ...)
## S4 method for signature 'GRangesList'
asBED(x)
```

Arguments

`x`	Generally, a tabular object to structure as BED
`...`	Arguments to pass to methods

Details

The exact behavior depends on the class of `object`.

`GRangesList` This treats `object` as if it were a list of transcripts, i.e., each element contains the exons of a transcript. The `blockStarts` and `blockSizes` columns are derived from the ranges in each element. Also, add `name` column from `names(object)`.

Value

A `GRanges`, with the columns `name`, `blockStarts` and `blockSizes` added.

Author(s)

Michael Lawrence

Examples

```
## Not run:
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
exons <- exonsBy(TxDb_Hsapiens_UCSC_hg19_knownGene)
mcols(asBED(exons))

## End(Not run)
```

---

asGFF *Coerce to GFF structure*

Description

Coerce the structure of an object to one following GFF-like conventions, i.e., using the `Parent` GFF3 attribute to encode the hierarchical structure. This object is then suitable for export as GFF3.

Usage

```
asGFF(x, ...)
## S4 method for signature 'GRangesList'
asGFF(x, parentType = "mRNA", childType = "exon")
```

Arguments

`x`	Generally, a tabular object to structure as GFF(3)
`parentType`	The value to store in the `type` column for the top-level (e.g., transcript) ranges.
`childType`	The value to store in the `type` column for the child (e.g., exon) ranges.
`...`	Arguments to pass to methods

Value

For the `GRangesList` method: A `GRanges`, with the columns: `ID` (unique identifier), `Name` (from `names(x)`, and the names on each element of `x`, if any), `type` (as given by `parentType` and `childType`), and `Parent` (to relate each child range to its parent at the top-level).

Author(s)

Michael Lawrence

Examples

```
## Not run:
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
exons <- exonsBy(TxDb_Hsapiens_UCSC_hg19_knownGene)
mcols(asGFF(exons))

## End(Not run)
```

BamFile-methods *Export to BAM Files*

Description

Methods for import and export of GAlignments or GAlignmentPairs objects from and to BAM files, represented as BamFile objects.

Usage

```
## S4 method for signature 'BamFile,ANY,ANY'
import(con, format, text, use.names = FALSE,
                                   param = ScanBamParam(...), ...)
## S4 method for signature 'ANY,BamFile,ANY'
export(object, con, format, ...)
```

Arguments

object	The object to export, such as a GAlignments or GAlignmentPairs.
con	A path, URL, connection or BamFile object.
format	If not missing, should be "bam".
text	Not supported.
use.names	Whether to parse QNAME as the names on the result.
param	The ScanBamParam object governing the import.
...	Arguments that are passed to ScanBamParam if param is missing.

Details

BAM fields not formally present in the GAlignments[Pairs] object are extracted from the metadata columns, if present; otherwise, the missing value, "".""", is output. The file is sorted and indexed. This can be useful for subsetting BAM files, although filterBam may eventually become flexible enough to be the favored alternative.

Author(s)

Michael Lawrence

See Also

The `readGAlignments` and `readGAlignmentPairs` functions for reading BAM files.

Examples

```
     library(Rsamtools)
     ex1_file <- system.file("extdata", "ex1.bam", package="Rsamtools")
     gal <- import(ex1_file, param=ScanBamParam(what="flag"))
     gal.minus <- gal[strand(gal) == "-"]
## Not run:
     export(gal, BamFile("ex1-minus.bam"))

## End(Not run)
```

`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`).

`colorByStrand`: Object of class `"matrix"` with two columns, as from `col2rgb`. The two colors indicate the color for each strand (positive, negative).

`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.

`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.

BEDFile-class *BEDFile objects*

Description

These functions support the import and export of the UCSC BED format and its variants, including BEDGraph.

Usage

```
## S4 method for signature 'BEDFile,ANY,ANY'
import(con, format, text, trackLine = TRUE,
                   genome = NA, colnames = NULL,
                   which = NULL, seqinfo = NULL, extraCols = character())
import.bed(con, ...)
import.bed15(con, ...)
import.bedGraph(con,  ...)

## S4 method for signature 'ANY,BEDFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'GenomicRanges,BEDFile,ANY'
export(object, con, format,
                  append = FALSE, index = FALSE,
                  ignore.strand = FALSE, trackLine = NULL)
## S4 method for signature 'UCSCData,BEDFile,ANY'
export(object, con, format,
                   trackLine = TRUE, ...)
export.bed(object, con, ...)
export.bed15(object, con, ...)
## S4 method for signature 'GenomicRanges,BED15File,ANY'
export(object, con, format,
                   expNames = NULL, trackLine = NULL, ...)
export.bedGraph(object, con, ...)
```

Arguments

`con`	A path, URL, connection or `BEDFile` object. For the functions ending in `.bed`, `.bedGraph` and `.bed15`, the file format is indicated by the function name. For the base `export` and `import` functions, the format must be indicated another way. If `con` is a path, URL or connection, either the file extension or the `format` argument needs to be one of "bed", "bed15" or "bedGraph". Compressed files ("gz", "bz2" and "xz") are handled transparently.
`object`	The object to export, should be a `GRanges` or something coercible to a `GRanges`. If the object has a method for `asBED` (like `GRangesList`), it is called prior to coercion. This makes it possible to export a `GRangesList` or `TxDb` in a way that preserves the hierarchical structure. For exporting multiple tracks, in the UCSC track line metaformat, pass a `GenomicRangesList`, or something coercible to one.
`format`	If not missing, should be one of "bed", "bed15" or "bedGraph".
`text`	If `con` is missing, a character vector to use as the input
`trackLine`	For import, an imported track line will be stored in a `TrackLine` object, as part of the returned `UCSCData`. For the UCSCData method on export, whether to output the UCSC track line stored on the object, for the other export methods, the actual TrackLine object to export.
`genome`	The identifier of a genome, or `NA` if unknown. Typically, this is a UCSC identifier like "hg19". An attempt will be made to derive the `seqinfo` on the return value using either an installed BSgenome package or UCSC, if network access is available.
`colnames`	A character vector naming the columns to parse. These should name columns in the result, not those in the BED spec, so e.g. specify "thick", instead of "thickStart".
`which`	A range data structure like `RangesList` or `GRanges`. Only the intervals in the file overlapping the given ranges are returned. This is much more efficient when the file is indexed with the tabix utility.
`index`	If `TRUE`, automatically compress and index the output file with bgzf and tabix. Note that tabix indexing will sort the data by chromosome and start. Does not work when exporting a `RangedDataList` with multiple elements; tabix supports a single track in a file.
`ignore.strand`	Whether to output the strand when not required (by the existence of later fields).
`seqinfo`	If not `NULL`, the `Seqinfo` object to set on the result. If the `genome` argument is not `NA`, it must agree with `genome(seqinfo)`.
`extraCols`	A character vector in the same form as `colClasses` from `read.table`. It should indicate the name and class of each extra/special column to read from the BED

	file. As BED does not encode column names, these are assumed to be the last columns in the file. This enables parsing of the various BEDX+Y formats.
`append`	If `TRUE`, and `con` points to a file path, the data is appended to the file. Obviously, if `con` is a connection, the data is always appended.
`expNames`	character vector of column names in `object` to export as sample columns in the BED15 file.
`...`	Arguments to pass down to methods to other methods. For import, the flow eventually reaches the `BEDFile` method on `import`. For export, the `RangedData`, `BEDFile` method on `export` is the sink. When `trackLine` is `TRUE` or the target format is BED15, the arguments are passed through `export.ucsc`, so track line parameters are supported.

Details

The BED format is a tab-separated table of intervals, with annotations like name, score and even sub-intervals for representing alignments and gene models. Official (UCSC) child formats currently include BED15 (adding a number matrix for e.g. expression data across multiple samples) and BEDGraph (a compressed means of storing a single score variable, e.g. coverage; overlapping features are not allowed). Many tools and organizations have extended the BED format with additional columns for particular use cases. These are not yet supported by rtracklayer, but a mechanism will be added soon. The advantage of BED is its balance between simplicity and expressiveness. It is also relatively scalable, because only the first three columns (chrom, start and end) are required. Thus, BED is best suited for representing simple features. For specialized cases, one is usually better off with another format. For example, genome-scale vectors belong in BigWig, alignments from high-throughput sequencing belong in BAM, and gene models are more richly expressed in GFF.

The following is the mapping of BED elements to a `GRanges` or `RangedData` object. NA values are allowed only where indicated. These appear as a "." in the file. Only the first three columns (chrom, start and strand) are required. The other columns can only be included if all previous columns (to the left) are included. Upon export, default values are used to automatically pad the table, if necessary.

chrom, start, end the `ranges` component.

name character vector (NA's allowed) in the `name` column; defaults to NA on export.

score numeric vector in the `score` column, accessible via the `score` accessor. Defaults to 0 on export. This is the only column present in BEDGraph (besides chrom, start and end), and it is required.

strand strand factor (NA's allowed) in the `strand` column, accessible via the `strand` accessor; defaults to NA on export.

thickStart, thickEnd `Ranges` object in a column named `thick`; defaults to the ranges of the feature on export.

itemRgb an integer matrix of color codes, as returned by `col2rgb`, or any valid input to `col2rgb`, in the `itemRgb` column; default is NA on export, which translates to black.

blockSizes, blockStarts, blockCounts `RangesList` object in a column named `blocks`; defaults to empty upon BED15 export.

These columns are present only in BED15:

expCount, expIds, expScores A column for each unique element in expIds, containing the corresponding values from expScores. When a value is not present for a feature, NA is substituted. NA values become -10000 in the file.

Value

A GRanges with the metadata columns described in the details.

BEDX+Y formats

To import one of the multitude of BEDX+Y formats, such as those used to distribute ENCODE data through UCSC (narrowPeaks, etc), specify the extraCols argument to indicate the expected names and classes of the special columns. We assume that the last length(extraCols) columns are special, and that the preceding columns adhere to the BED format.

BEDFile objects

The BEDFile class extends RTLFile and is a formal represention of a resource in the BED format. To cast a path, URL or connection to a BEDFile, pass it to the BEDFile constructor. Classes and constructors also exist for the subclasses BED15File and BEDGraphFile.

Author(s)

Michael Lawrence

References

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

Examples

```
  test_path <- system.file("tests", package = "rtracklayer")
  test_bed <- file.path(test_path, "test.bed")

  test <- import(test_bed)
  test

  test_bed_file <- BEDFile(test_bed)
  import(test_bed_file)

  test_bed_con <- file(test_bed)
  import(test_bed_con, format = "bed")
  close(test_bed_con)

  import(test_bed, trackLine = FALSE)
  import(test_bed, genome = "hg19")
  import(test_bed, colnames = c("name", "strand", "thick"))

  which <- RangesList(chr7 = as(test, "RangesList")[[1]][1:2])
  import(test_bed, which = which)

## Not run:
  test_bed_out <- file.path(tempdir(), "test.bed")
  export(test, test_bed_out)

  test_bed_out_file <- BEDFile(test_bed_out)
  export(test, test_bed_out_file)

  export(test, test_bed_out, name = "Alternative name")

  test_bed_gz <- paste(test_bed_out, ".gz", sep = "")
  export(test, test_bed_gz)

  export(test, test_bed_out, index = TRUE)
  export(test, test_bed_out, index = TRUE, trackLine = FALSE)

  bed_text <- export(test, format = "bed")
  test <- import(format = "bed", text = bed_text)

## End(Not run)
```

`BigWigFile-class` *BigWig Import and Export*

Description

These functions support the import and export of the UCSC BigWig format, a compressed, binary form of WIG/BEDGraph with a spatial index and precomputed summaries. These functions do not work on Windows.

Usage

```
## S4 method for signature 'BigWigFile,ANY,ANY'
import(con, format, text,
                    selection = BigWigSelection(which, ...),
                    which = con, asRle = FALSE,
                    as = c("GRanges", "RleList", "NumericList"), ...)
import.bw(con, ...)

## S4 method for signature 'ANY,BigWigFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'GenomicRanges,BigWigFile,ANY'
export(object, con, format,
                    dataFormat = c("auto", "variableStep", "fixedStep",
                      "bedGraph"), compress = TRUE)
export.bw(object, con, ...)
```

Arguments

`con`	A path, URL or `BigWigFile` object. Connections are not supported. For the functions ending in `.bw`, the file format is indicated by the function name. For

	the `export` and `import` methods, the format must be indicated another way. If `con` is a path, or URL, either the file extension or the `format` argument needs to be "bigWig" or "bw".
`object`	The object to export, should be an `RleList`, `IntegerList`, `NumericList`, `GRanges` or something coercible to a `GRanges`.
`format`	If not missing, should be "bigWig" or "bw" (case insensitive).
`text`	Not supported.
`as`	Specifies the class of the return object. Default is `GRanges`, which has one range per range in the file, and a score column holding the value for each range. For `NumericList`, one numeric vector is returned for each range in the `selection` argument. For `RleList`, there is one `Rle` per sequence, and that `Rle` spans the entire sequence. As of version 1.23.11 `import` no longer returns `RangedData`.
`asRle`	Deprecated. Use `as` instead. If `TRUE`, the BigWig file is assumed to contain contiguous ranges that define a run-length encoding of a vector (like coverage), and a `RleList` is returned.
`selection`	A `BigWigSelection` object indicating the ranges to load.
`which`	A range data structure coercible to `RangesList`, like a `GRanges`, or a `BigWigFile`. Only the intervals in the file overlapping the given ranges are returned. By default, the value is the `BigWigFile` itself. Its `Seqinfo` object is extracted and coerced to a `RangesList` that represents the entirety of the file.
`dataFormat`	Probably best left to "auto". Exists only for historical reasons.
`compress`	If `TRUE`, compress the data. No reason to change this.
`...`	Arguments to pass down to methods to other methods. For import, the flow eventually reaches the `BigWigFile` method on `import`. For export, the `RangedData`, `BigWigFile` method on `export` is the sink.

Value

A `GRanges` (default), `RangedData`, `RleList` or `NumericList`. `GRanges` and `RangedData` return ranges with non-zero score values in a `score` metadata column. The length of the `NumericList` is the same length as the `selection` argument (one list element per range). The return order in the `NumericList` matches the order of the `BigWigSelection` object.

`BigWigFile` objects

A `BigWigFile` object, an extension of `RTLFile` is a reference to a BigWig file. To cast a path, URL or connection to a `BigWigFile`, pass it to the `BigWigFile` constructor.

BigWig files are more complex than most track files, and there are a number of methods on `BigWigFile` for accessing the additional information:

`seqinfo(x)`: Gets the `Seqinfo` object indicating the lengths of the sequences for the intervals in the file. No circularity or genome information is available.

`summary(ranges = as(seqinfo(object), "GenomicRanges"), size        = 1L, type = c("mean", "min", "max`
Aggregates the intervals in the file that fall into `ranges`, which should be something coercible to `GRanges`. The aggregation essentially compresses each sequence to a length of `size`. The

algorithm is specified by type; available algorithms include the mean, min, max, coverage (percent sequence covered by at least one feature), and standard deviation. When a window contains no features, defaultValue is assumed. The result is an RleList, with an element for each element in ranges. The driving use case for this is visualization of coverage when the screen space is small compared to the viewed portion of the sequence. The operation is very fast, as it leverages cached multi-level summaries present in every BigWig file.

If a summary statistic is not available / cannot be computed for a given range a warning is thrown and the defaultValue NA_real_ is returned.

BigWigFileList objects

A BigWigFileList() provides a convenient way of managing a list of BigWigFile instances.

Author(s)

Michael Lawrence

See Also

wigToBigWig for converting a WIG file to BigWig.

Examples

```
if (.Platform$OS.type != "windows") {
  test_path <- system.file("tests", package = "rtracklayer")
  test_bw <- file.path(test_path, "test.bw")

  ## GRanges
  ## Returns ranges with non-zero scores.
  gr <- import(test_bw)
  gr

  which <- GRanges(c("chr2", "chr2"), IRanges(c(1, 300), c(400, 1000)))
  import(test_bw, which = which)

  ## RleList
  ## Scores returned as an RleList is equivalent to the coverage.
  ## Best option when 'which' or 'selection' contain many small ranges.
  mini <- narrow(unlist(tile(which, 50)), 2)
  rle <- import(test_bw, which = mini, as = "RleList")
  rle

  ## NumericList
  ## The 'which' is stored as metadata:
  track <- import(test_bw, which = which, as = "NumericList")
  metadata(track)

## Not run:
  test_bw_out <- file.path(tempdir(), "test_out.bw")
  export(test, test_bw_out)

## End(Not run)
```

```
  bwf <- BigWigFile(test_bw)
  track <- import(bwf)

  seqinfo(bwf)

  summary(bwf) # for each sequence, average all values into one
  summary(bwf, range(head(track))) # just average the first few features
  summary(bwf, size = seqlengths(bwf) / 10) # 10X reduction
  summary(bwf, type = "min") # min instead of mean
}
```

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"): Constructs a BigWigSelection with the given ranges and colnames. ranges can be either something coercible to a RangesList, a character identifying a genome (see GenomicSelection), or a BigWigFile, in which case the ranges are derived from the bounds of its sequences.

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::IRanges(c(1, 5), c(3, 6)))

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

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

`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` or `GenomicRanges`, with a value column of name “blocks” and of type `RangesList`. Such an object is returned by `import.bed` and `asBED`.
`...`	Additional arguments for methods

Value

A `GRangesList` with an element for each range in `x`. The original block ranges are relative to the start of the containing range, so the returned ranges are shifted to absolute coordinates. The seqname and strand are inherited from the containing range.

Author(s)

Michael Lawrence

See Also

`import.bed` for importing a track from BED, which can store block information; `asBED` for coercing a `GenomicRanges` into a BED-like structure that can be passed to this function.

`browseGenome` *Browse a genome*

Description

A generic function for launching a genome browser.

Usage

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

Arguments

`object`	A `GRanges` object, a `RangedData` object, or a list of those objects (e.g. a `GenomicRangesList` object).
`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. As a derivative of `TrackDb`, each session contains a set of loaded tracks. In addition, it has a set of views, in the form of `BrowserView` instances, on those tracks. 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, ...) <- value` 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), i`
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)
```

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

browserViews-methods *Getting the browser views*

Description

Methods for getting browser views.

Methods

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

object = "UCSCSession" Gets the instances of `BrowserView` in the session.

See Also

`browserView` for creating a browser view.

Examples

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

## End(Not run)
```

`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

A `Chain` object can be loaded from a UCSC chain format file simply by passing the path `import` function. If the file extension is not “chain”, then either pass “chain” to the `format` argument, or cast the path to a `ChainFile` object. The `import.chain` function is provided as a (slight) convenience. It is documented below, along with the extra `exclude` argument to the `import` method.

`import.chain(con, exclude = "_", ...)`: Imports a chain file named `con` as a `Chain` object, a list of `ChainBlock`s. 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

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))
```

FastaFile-class *FastaFile objects*

Description

These functions support the import and export of the Fasta sequence format, using the Biostrings package.

Usage

```
## S4 method for signature 'FastaFile,ANY,ANY'
import(con, format, text,
           type = c("DNA", "RNA", "AA", "B"), ...)

## S4 method for signature 'ANY,FastaFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'XStringSet,FastaFile,ANY'
export(object, con, format, ...)
```

Arguments

`con`	A path or `FastaFile` object. URLs and connections are not supported. If `con` is not a `FastaFile`, either the file extension or the `format` argument needs to be "fasta". Compressed files ("gz", "bz2" and "xz") are handled transparently.
`object`	The object to export, should be an `XStringSet` or something coercible to a `DNAStringSet`, like a character vector.
`format`	If not missing, should be "fasta".
`text`	If `con` is missing, a character vector to use as the input
`type`	Type of biological sequence.
`...`	Arguments to pass down to `writeXStringSet` (export) or the `readDNAStringSet` family of functions (import).

FastaFile objects

The `FastaFile` class extends `RTLFile` and is a formal represention of a resource in the Fasta format. To cast a path, URL or connection to a `FastaFile`, pass it to the `FastaFile` constructor.

Author(s)

Michael Lawrence

See Also

These functions are implemented by the Biostrings `writeXStringSet` (export) and the `readDNAStringSet` family of functions (import).

See export-methods in the **BSgenome** package for exporting a BSgenome object as a FASTA file.

`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.

`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")
```

GFFFile-class *GFFFile objects*

Description

These functions support the import and export of the GFF format, of which there are three versions and several flavors.

Usage

```
## S4 method for signature 'GFFFile,ANY,ANY'
import(con, format, text,
           version = c("", "1", "2", "3"),
           genome = NA, colnames = NULL, which = NULL,
           feature.type = NULL, sequenceRegionsAsSeqinfo = FALSE)
import.gff(con, ...)
import.gff1(con, ...)
import.gff2(con, ...)
import.gff3(con, ...)

## S4 method for signature 'ANY,GFFFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'GenomicRanges,GFFFile,ANY'
export(object, con, format,
                   version = c("1", "2", "3"),
                   source = "rtracklayer", append = FALSE, index = FALSE)
## S4 method for signature 'GenomicRangesList,GFFFile,ANY'
export(object, con, format, ...)
export.gff(object, con, ...)
export.gff1(object, con, ...)
export.gff2(object, con, ...)
export.gff3(object, con, ...)
```

Arguments

`con`	A path, URL, connection or `GFFFile` object. For the functions ending in `.gff`, `.gff1`, etc, the file format is indicated by the function name. For the base `export` and `import` functions, the format must be indicated another way. If `con` is a path, URL or connection, either the file extension or the `format` argument needs to be one of “gff”, “gff1” “gff2”, “gff3”, “gvf”, or “gtf”. Compressed files (“gz”, “bz2” and “xz”) are handled transparently.
`object`	The object to export, should be a `GRanges` or something coercible to a `GRanges`. If the object has a method for `asGFF`, it is called prior to coercion. This makes it possible to export a `GRangesList` or `TxDb` in a way that preserves the hierarchical structure. For exporting multiple tracks, in the UCSC track line metaformat, pass a `GenomicRangesList`, or something coercible to one.

`format`	If not missing, should be one of “gff”, “gff1” “gff2”, “gff3”, “gvf”, or “gtf”.
`version`	If the format is given as “gff”, i.e., it does not specify a version, then this should indicate the GFF version as one of “” (for import only, from the `gff-version` directive in the file or “1” if none), “1”, “2” or “3”.
`text`	If `con` is missing, a character vector to use as the input.
`genome`	The identifier of a genome, or `NA` if unknown. Typically, this is a UCSC identifier like “hg19”. An attempt will be made to derive the `seqinfo` on the return value using either an installed BSgenome package or UCSC, if network access is available.
`colnames`	A character vector naming the columns to parse. These should name either fixed fields, like `source` or `type`, or, for GFF2 and GFF3, any attribute.
`which`	A range data structure like `RangesList` or `GRanges`. Only the intervals in the file overlapping the given ranges are returned. This is much more efficient when the file is indexed with the tabix utility.
`feature.type`	`NULL` (the default) or a character vector of valid feature types. If not `NULL`, then only the features of the specified type(s) are imported.
`sequenceRegionsAsSeqinfo`	If `TRUE`, attempt to infer the `Seqinfo` (`seqlevels` and `seqlengths`) from the “##sequence-region” directives as specified by GFF3.
`source`	The value for the source column in GFF. This is typically the name of the package or algorithm that generated the feature.
`index`	If `TRUE`, automatically compress and index the output file with bgzf and tabix. Note that tabix indexing will sort the data by chromosome and start. Does not work when exporting a `RangedDataList` with multiple elements; tabix supports a single track in a file.
`append`	If `TRUE`, and `con` points to a file path, the data is appended to the file. Obviously, if `con` is a connection, the data is always appended.
`...`	Arguments to pass down to methods to other methods. For import, the flow eventually reaches the `GFFFile` method on `import`. For export, the `RangedData`, `GFFFile` method on `export` is the sink. When `trackLine` is `TRUE` or the target format is BED15, the arguments are passed through `export.ucsc`, so track line parameters are supported.

Details

The Generic Feature Format (GFF) format is a tab-separated table of intervals. There are three different versions of GFF, and they all have the same number of columns. In GFF1, the last column is a grouping factor, whereas in the later versions the last column holds application-specific attributes, with some conventions defined for those commonly used. This attribute support facilitates specifying extensions to the format. These include GTF (Gene Transfer Format, an extension of GFF2) and GVF (Genome Variation Format, an extension of GFF3). The rtracklayer package recognizes the “gtf” and “gvf” extensions and parses the extra attributes into columns of the result; however, it does not perform any extension-specific processing. Both GFF1 and GFF2 have been proclaimed obsolete; however, the UCSC Genome Browser only supports GFF1 (and GTF), and GFF2 is still in broad use.

GFF is distinguished from the simpler BED format by its flexible attribute support and its hierarchical structure, as specified by the `group` column in GFF1 (only one level of grouping) and the `Parent` attribute in GFF3. GFF2 does not specify a convention for representing hierarchies, although its GTF extension provides this for gene structures. The combination of support for hierarchical data and arbitrary descriptive attributes makes GFF(3) the preferred format for representing gene models.

Although GFF features a `score` column, large quantitative data belong in a format like BigWig and alignments from high-throughput experiments belong in BAM. For variants, the VCF format (supported by the VariantAnnotation package) seems to be more widely adopted than the GVF extension.

A note on the UCSC track line metaformat: track lines are a means for passing hints to visualization tools like the UCSC Genome Browser and the Integrated Genome Browser (IGB), and they allow multiple tracks to be concatenated in the same file. Since GFF is not a UCSC format, it is not common to annotate GFF data with track lines, but rtracklayer still supports it. To export or import GFF data in the track line format, call `export.ucsc` or `import.ucsc`.

The following is the mapping of GFF elements to a `GRanges` or `RangedData` object. NA values are allowed only where indicated. These appear as a "." in the file. GFF requires that all columns are included, so `export` generates defaults for missing columns.

seqid, start, end the `ranges` component.

source character vector in the `source` column; defaults to "rtracklayer" on export.

type character vector in the `type` column; defaults to "sequence_feature" in the output, i.e., SO:0000110.

score numeric vector (NA's allowed) in the `score` column, accessible via the `score` accessor; defaults to `NA` upon export.

strand strand factor (NA's allowed) in the `strand` column, accessible via the `strand` accessor; defaults to `NA` upon export.

phase integer vector, either 0, 1 or 2 (NA's allowed); defaults to `NA` upon export.

group a factor (GFF1 only); defaults to the `seqid` (e.g., chromosome) on export.

In GFF versions 2 and 3, attributes map to arbitrary columns in the result. In GFF3, some attributes (`Parent`, `Alias`, `Note`, `DBxref` and `Ontology_term`) can have multiple, comma-separated values; these columns are thus always `CharacterList` objects.

Value

A `GRanges` with the metadata columns described in the details.

GFFFile objects

The `GFFFile` class extends `RTLFile` and is a formal represention of a resource in the GFF format. To cast a path, URL or connection to a `GFFFile`, pass it to the `GFFFile` constructor. The `GFF1File`, `GFF2File`, `GFF3File`, `GVFFile` and `GTFFile` classes all extend `GFFFile` and indicate a particular version of the format.

It has the following utility methods:

`genome`: Gets the genome identifier from the "genome-build" header directive.

Author(s)

Michael Lawrence

References

GFF1, GFF2 http://www.sanger.ac.uk/resources/software/gff/spec.html

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

GVF http://www.sequenceontology.org/resources/gvf.html

GTF http://mblab.wustl.edu/GTF22.html

Examples

```
  test_path <- system.file("tests", package = "rtracklayer")
  test_gff3 <- file.path(test_path, "genes.gff3")

  ## basic import
  test <- import(test_gff3)
  test

  ## import.gff functions
  import.gff(test_gff3)
  import.gff3(test_gff3)

  ## GFFFile derivatives
  test_gff_file <- GFF3File(test_gff3)
  import(test_gff_file)
  test_gff_file <- GFFFile(test_gff3)
  import(test_gff_file)
  test_gff_file <- GFFFile(test_gff3, version = "3")
  import(test_gff_file)

  ## from connection
  test_gff_con <- file(test_gff3)
  test <- import(test_gff_con, format = "gff")
  close(test_gff_con)

  ## various arguments
  import(test_gff3, genome = "hg19")
  import(test_gff3, colnames = character())
  import(test_gff3, colnames = c("type", "geneName"))

  ## 'which'
  which <- RangesList(chr10 = IRanges(90000, 93000))
  import(test_gff3, which = which)

## Not run:
  ## 'append'
  test_gff3_out <- file.path(tempdir(), "genes.gff3")

  export(test[seqnames(test) == "chr10"], test_gff3_out)
  export(test[seqnames(test) == "chr12"], test_gff3_out, append = TRUE)
  import(test_gff3_out)

  ## 'index'
  export(test, test_gff3_out, index = TRUE)
  test_bed_gz <- paste(test_gff3_out, ".gz", sep = "")
  import(test_bed_gz, which = which)

  ## RangedDataList
  rdl <-
    RangedDataList(new("UCSCData", test[1],
                       trackLine = new("BasicTrackLine", name = "chr10")),
                   new("UCSCData", test[2],
                       trackLine = new("BasicTrackLine", name = "chr12")))
  names(rdl) <- names(test)
  export(rdl, test_gff3_out)
  import.ucsc(test_gff3_out)

## End(Not run)
```

GRangesForUCSCGenome *GRanges for a Genome*

Description

These functions assist in the creation of Seqinfo or GRanges for a genome.

Usage

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

SeqinfoForUCSCGenome(genome)
SeqinfoForBSGenome(genome)
```

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

For the `GRangesFor*` functions, a `GRanges` object, with the appropriate `seqlengths` and `genome` ID.

The `SeqinfoFor*` functions return a `Seqinfo` for the indicated genome.

Author(s)

Michael Lawrence

`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.

`alwaysZero`**:** Object of class `"logical"` indicating whether to fix the lower limit of the Y axis at zero.

`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.

`io` *Import and export*

Description

The functions `import` and `export` load and save objects from and to particular file formats. The rtracklayer package implements support for a number of annotation and sequence formats.

Usage

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

Arguments

`object`	The object to export.
`con`	The connection from which data is loaded or to which data is saved. 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 a `RTLFile` derivative, the data is loaded from or saved to the underlying resource. 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. This argument is unnecessary when `con` is a derivative of `RTLFile`.
`text`	If `con` is missing, this can be a character vector directly providing the string data to import.
`...`	Parameters to pass to the format-specific method.

Details

The rtracklayer package supports a number of file formats for representing annotated genomic intervals. These are each represented as a subclass of `RTLFile`. Below, we list the major supported formats, with some advice for when a particular file format is appropriate:

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 and is the most flexible format, in that a feature may have any number of attributes (in version 2 and above). Version 3 (GFF3) is the preferred version. Its specification lays out conventions for representing various types of data, including gene models, for which it is the format of choice. For variants, rtracklayer has rudimentary support for an extention of GFF3 called GVF. UCSC supports GFF1, but it needs to be encapsulated in the UCSC metaformat, i.e. `export.ucsc(subformat = "gff1")`. The BED format is typically preferred over GFF for interaction with UCSC. GFF files can be indexed with the tabix utility for fast range-based queries via rtracklayer and Rsamtools.

BED The Browser Extended Display format is for displaying qualitative tracks in a genome browser, in particular UCSC. It finds a good balance between simplicity and expressiveness. It is much simpler than GFF and yet can still represent multi-exon gene structures. It is somewhat limited by its lack of the attribute support of GFF. To circumvent this, many tools and organizations have extended BED with additional columns. These are not officially valid BED files, and as such rtracklayer does not yet support them (this will be addressed soon). The rtracklayer package does support two official extensions of BED: Bed15 and bedGraph, see below. BED files can be indexed with the tabix utility for fast range-based queries via rtracklayer and Rsamtools.

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 in UCSC as a compact heatmap. Few other tools support this format. With 15 columns per feature, this format is probably too verbose for e.g. ChIP-seq coverage (use multiple BigWig tracks instead).

BEDGRAPH A variant of BED that represents a score column more compactly than BED and especially BED15, although only one sample is supported. The data is displayed in UCSC as a bar or line graph. For large data (the typical case), `BigWig` is preferred.

WIG The Wiggle format is meant for storing dense numerical data, such as window-based GC and conservation scores. The data is displayed in UCSC as a bar or line graph. The WIG format only works for intervals with a uniform width. For non-uniform widths, consider `bedGraph`. For large data, consider `BigWig`.

BIGWIG The BigWig format is a binary version of both `bedGraph` and `WIG` (which are now somewhat obsolete). A BigWig file contains a spatial index for fast range-based queries and also embeds summary statistics of the scores at several zoom levels. Thus, it is ideal for visualization of and parallel computing on genome-scale vectors, like the coverage from a high-throughput sequencing experiment.

In summary, for the typical use case of combining gene models with experimental data, GFF is preferred for gene models and `BigWig` is preferred for quantitative score vectors. Note that the Rsamtools package provides support for the `BAM` file format (for representing read alignments), among others. Based on this, the rtracklayer package provides an `export` method for writing `GAlignments` and `GappedReads` objects as `BAM`. For variants, consider VCF, supported by the VariantAnnotation package.

There is also support for reading and writing biological sequences, including the UCSC `TwoBit` format for compactly storing a genome sequence along with a mask. The files are binary, so they are efficiently queried for particular ranges. A similar format is `FA`, supported by Rsamtools.

Value

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

Author(s)

Michael Lawrence

See Also

Format-specific options for the popular formats: GFF, BED, BED15, BEDGRAPH, WIG, BIGWIG

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")
```

`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 `GRangesList` object. Each element contains the ranges mapped from the corresponding element in the input (may be one-to-many).

Author(s)

Michael Lawrence

References

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

Examples

```
## Not run:
chain <- import.chain("hg19ToHg18.over.chain")
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
tx_hg19 <- transcripts(TxDb.Hsapiens.UCSC.hg19.knownGene)
tx_hg18 <- liftOver(tx_hg19, chain)

## End(Not run)
```

`Quickload-class` *Quickload Access*

Description

The `Quickload` class represents a Quickload data source, essentially directory layout separating tracks and sequences by genome, along with a few metadata files. This interface abstracts those details and provides access to a Quickload at any URL supported by R (HTTP, FTP, and local files). This is an easy way to make data accessible to the Integrated Genome Browser (IGB).

Constructor

`Quickload(uri = "quickload", create = FALSE)`: Constructs a new `Quickload` object, representing a repository at `uri`. If `create` is `TRUE`, and `uri` is writeable (i.e., local), the repository is created if it does not already exist. If it does exist, then a message is emitted to indicate that the repository was not recreated.

Accessor Methods

In the code snippets below, `x` represents a `Quickload` object.

`x$genome`, `x[["genome"]]`: Get the `QuickloadGenome` object for the genome named `genome`. This is where all the data is stored.

`length(x)`: number of genomes in the repository

`uri(x)`: Get the URI pointing to the Quickload repository.

`genome(x)`: Get the identifiers of the genomes present in the repository.

Author(s)

Michael Lawrence

Examples

```
ql <- Quickload(system.file("tests", "quickload", package = "rtracklayer"))
uri(ql)
genome(ql)
ql$T_species_Oct_2011
```

QuickloadGenome-class *Quickload Genome Access*

Description

A Quickload data source is a collection of tracks and sequences, separated by genome. This class, `QuickloadGenome` provides direct access to the data for one particular genome.

Constructor

`QuickloadGenome(quickload, genome, create = FALSE,       seqinfo = seqinfo(genome),       title = toS`
: Constructs a new `QuickloadGenome` object, representing `genome` in the repository `quickload` (a URI string or a `Quickload` object).

  The `genome` argument can be an ID corresponding to a genome (potentially) in `quickload` or an installed `BSgenome` package. It can also be any instance of a class which has methods for `organism` and `releaseDate`. A good example is `BSgenome` or any other derivative of `GenomeDescription`. Those items are necessary for constructing the canonical Quickload genome string (G_Species_Month_Year).

  If `create` is `TRUE`, and the genome does not already exist, the genome will be created, using `seqinfo` for the sequence lengths and `title` for the display name of the genome in a UI. Creation only works if the repository is local and writeable. Reasonable defaults are used for `seqinfo` and `title` when the necessary methods are available (and they are for `BSgenome`).

Accessor Methods

In the code snippets below, `x` and `object` represent a `Quickload` object.

`seqinfo(x), seqinfo(x) <- value`: Gets or sets the `Seqinfo` object indicating the lengths of the sequences in the genome. No circularity information or genome identifier is stored.

`quickload(x)`: Get the Quickload object that contains this genome.

`uri(x)`: Get the uri pointing to the genome directory in the Quickload repository

`genome(x)`: Get the name of the genome, e.g. “H_sapiens_Feb_2009”.

`releaseDate(x)`: Get the release portion of the genome name, e.g., “Feb_2009”.

`organism(object)`: Get the organism portion of the genome name, e.g., “H sapiens”.

Data Access

`length(x)`: number of datasets

`names(x), trackNames(x)`: names of the datasets

`mcols(x)`: merged metadata on the datasets

`track(x, name), x$name`: get the track called `name` `track(x, name, format = bestFileFormat(value), ...) <- value`, `x$name <- value`: store the track `value` under `name`. Note that track storing is only supported for local repositories, i.e., those with a `file://` URI scheme.

Currently, supported `value` types include a `GenomicRanges`, `GRangesList`, or a file resource (copied to the repository). The file resource may be represented as a path, URL, `RTLFile` or `RsamtoolsFile`. If not a file name, `value` is written in `format`. For generic interval data, this means a BigWig file (if there is a numeric "score" column) or a BED file otherwise. An `RleList` (e.g., coverage) is output as BigWig. For `UCSCData` values, the format is chosen according to the type of track line. For `RsamtoolsFile` objects, the file and its index are copied.

The arguments in `...` become attributes in the XML metadata. The "description" attribute is standard and is a blurb for describing the track in a UI. For the rest, the interpretation is up to the client. IGB supports an ever-growing list; please see its documentation.

`referenceSequence(x)`: Get the reference sequence, as a `DNAStringSet`.

`referenceSequence(x) <- value`: Set the reference sequence, as a `DNAStringSet`. It is written as a 2bit file. This only works on local repositories.

Author(s)

Michael Lawrence

Examples

```
tests_dir <- system.file("tests", package = "rtracklayer")
ql <- Quickload(file.path(tests_dir, "quickload"))
qlg <- QuickloadGenome(ql, "T_species_Oct_2011")
seqinfo(qlg)
organism(qlg)
releaseDate(qlg)
names(qlg)
mcols(qlg)
if (.Platform$OS.type != "windows") { # temporary
qlg$bedData
}

## Not run:
## populating the test repository
ql <- Quickload(file.path(tests_dir, "quickload"), create = TRUE)
reference_seq <- import(file.path(tests_dir, "test.2bit"))
names(reference_seq) <- "test"
qlg <- QuickloadGenome(ql, "T_species_Oct_2011", create = TRUE,
                       seqinfo = seqinfo(reference_seq))
referenceSequence(qlg) <- reference_seq
test_bed <- import(file.path(tests_dir, "test.bed"))
names(test_bed) <- "test"
qlg$bedData <- test_bed
test_bedGraph <- import(file.path(tests_dir, "test.bedGraph"))
names(test_bedGraph) <- "test"
start(test_bedGraph) <- seq(1, 90, 10)
width(test_bedGraph) <- 10
```

```
track(qlg, "bedGraphData", format = "bw") <- test_bedGraph

## End(Not run)
```

RangedData-methods *Data on a Genome*

Description

The rtracklayer package adds convenience methods on top of RangedData and GenomicRanges 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 GenomicRanges 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)`.

`score(x)`: Gets the "score" column from the element metadata of a GenomicRanges or GRangesList. Many track formats have a score column, so this is often used during export. The IRanges package defines a method for RangedData. The ANY fallback for this method simply returns NULL.

Constructor

`GenomicData(ranges, ..., strand = NULL, chrom = NULL, genome = NULL)`: Constructs a GRanges instance with the given ranges and variables in ... (see the GRanges constructor).

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 *. To get the levels for strand, call levels(strand()).

chrom argument is analogous to seqnames in the GRanges and space in RangedData 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::RangesList(chr1 = IRanges::IRanges(c(1,2,3), c(5,2,8)))

## just ranges ##
## GRanges instance
gr <- GenomicData(range1)

## with a genome (universe) ##
## GRanges instance
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()))
## GRanges instance
gr <- GenomicData(range1[[1]], score, chrom = "chr1", genome = "hg18")
mcols(gr)[["score"]]
strand(gr) ## all '*'
gr <- GenomicData(range1[[1]], score, filt = filter, strand = strand,
                  chrom = "chr1")
mcols(gr)[["filt"]]
strand(gr) ## equal to 'strand'
## coercion from data.frame ##
df <- as.data.frame(gr)
```

`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)`.

Author(s)

Michael Lawrence

readGFF *Reads a file in GFF format*

Description

Reads a file in GFF format and creates a data frame or DataFrame object from it.

Usage

```
readGFF(filepath, version=0,
        columns=NULL, tags=NULL, filter=NULL, nrows=-1,
        raw_data=FALSE)

GFFcolnames(GFF1=FALSE)
```

Arguments

`filepath`	A single string containing the path or URL to the file to read. Alternatively can be a connection.
`version`	`readGFF` should do a pretty descent job at detecting the GFF version. Use this argument *only* if it doesn't or if you want to force it to parse and import the file as if its 9-th column was in a different format than what it really is (e.g. specify `version=1` on a GTF or GFF3 file to interpret its 9-th column as the `"group"` column of a GFF1 file). Supported versions are 1, 2, and 3.
`columns`	The standard GFF columns to load. All of them are loaded by default.
`tags`	The tags to load. All of them are loaded by default.
`filter`	
`nrows`	`-1` or the maximum number of rows to read in (after filtering).
`raw_data`	
`GFF1`	

Author(s)

H. Pages

See Also

- `import` for importing a GFF file as a GRanges object.
- `makeGRangesFromDataFrame` in the **GenomicRanges** package for making a GRanges object from a data frame or DataFrame object.
- `makeTxDbFromGFF` in the **GenomicFeatures** package for importing a GFF file as a TxDb object.
- The DataFrame class in the **S4Vectors** package.

Examples

```
## Standard GFF columns.
GFFcolnames()
GFFcolnames(GFF1=TRUE)  # "group" instead of "attributes"

tests_dir <- system.file("tests", package="rtracklayer")
test_gff3 <- file.path(tests_dir, "genes.gff3")

## Load everything.
df0 <- readGFF(test_gff3)
head(df0)

## Load some tags only (in addition to the standard GFF columns).
my_tags <- c("ID", "Parent", "Name", "Dbxref", "geneID")
df1 <- readGFF(test_gff3, tags=my_tags)
head(df1)

## Load no tags (in that case, the "attributes" standard column
## is loaded).
df2 <- readGFF(test_gff3, tags=character(0))
head(df2)

## Load some standard GFF columns only (in addition to all tags).
my_columns <- c("seqid", "start", "end", "strand", "type")
df3 <- readGFF(test_gff3, columns=my_columns)
df3
table(df3$seqid, df3$type)
makeGRangesFromDataFrame(df3, keep.extra.columns=TRUE)

## Combine use of 'columns' and 'tags' arguments.
readGFF(test_gff3, columns=my_columns, tags=c("ID", "Parent", "Name"))
readGFF(test_gff3, columns=my_columns, tags=character(0))

## Use the 'filter' argument to load only features of type "gene"
## or "mRNA" located on chr10.
my_filter <- list(type=c("gene", "mRNA"), seqid="chr10")
readGFF(test_gff3, filter=my_filter)
readGFF(test_gff3, columns=my_columns, tags=character(0), filter=my_filter)
```

`RTLFile-class` *RTLFile objects*

Description

A `RTLFile` object is the base class for classes representing files accessible with rtracklayer. It wraps a resource (either a path, URL or connection).

Accessor Methods

In the code snippets below, `x` represents a `RTLFile` object.

`path(x)`: Gets the path, as a `character` vector, to the resource represented by the `RTLFile` object, if possible.

`resource(x)`: Gets the low-level resource, either a character vector (a path or URL) or a connection.

Related functions

`FileForFormat(path, format = file_ext(path))`: Determines the file type of `path` and returns a high-level file object such as BamFile, BEDFile, BigWigFile etc..

Author(s)

Michael Lawrence

See Also

Implementing classes include: `BigWigFile`, `TwoBitFile`, `BEDFile`, `GFFFile`, and `WIGFile`.

`sequence<--methods` *Load a sequence*

Description

Methods for loading sequences.

Methods

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

`TabixFile-methods` *TabixFile Import/Export*

Description

These methods support the import and export of `Rsamtools:TabixFile`TabixFile objects. These are generally useful when working with tabix-indexed files that have a non-standard format (i.e., not BED nor GFF), as well as exporting an object with arbitrary columns (like a GRanges) to an indexed, tab-separated file. This relies on the tabix header, which indicates the columns in the file that correspond to the chromosome, start and end. The BED and GFF parsers handle tabix transparently.

Usage

```
## S4 method for signature 'TabixFile,character,ANY'
import(con, format, text,
                   which = if (is.na(genome)) NULL
                           else as(seqinfoForGenome(genome), "GenomicRanges"),
                   genome = NA, header = TRUE, ...)
## S4 method for signature 'TabixFile,missing,ANY'
import(con, format, text, ...)
exportToTabix(object, con, ...)
```

Arguments

`con`	For `import`, a `TabixFile` object; for `exportToTabix`, a string naming the destination file.
`object`	The object to export. It is coerced to a `data.frame`, written to a tab-separated file, and indexed with tabix for efficient range-based retrieval of the data using `import`.
`format`	If any known format, like "bed" or "gff" (or one of their variants), then the appropriate parser is applied. If any other value, then the tabix header is consulted for the format. By default, this is taken from the file extension.
`text`	Ignored.
`which`	A range data structure coercible to `RangesList`, like a `GRanges`. Only the intervals in the file overlapping the given ranges are returned. The default is to use the range over the entire genome given by `genome`, if specified.
`genome`	The identifier of a genome, or `NA` if unknown. Typically, this is a UCSC identifier like "hg19". An attempt will be made to derive the `seqinfo` on the return value using either an installed BSgenome package or UCSC, if network access is available.
`header`	If `TRUE`, then the header in the indexed file, which might include a track line, is sent to the parser. Otherwise, the initial lines are skipped, according to the `skip` field in the tabix index header.
`...`	Extra arguments to pass to the underlying import routine, which for non-standard formats is `read.table` or `write.table`.

Value

For `import`, a `GRanges` or `RangedData`, depending on arguments.

For `exportToTabix`, a `TabixFile` object that is directly passable to `import`.

Author(s)

Michael Lawrence

References

http://samtools.sourceforge.net/tabix.shtml

See Also

`scanTabix` and friends

`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::IRanges(start, end),
                strand = strand, chrom = chrom))
```

`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(track)),`
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)
```

`TrackDb-class` *Track Databases*

Description

The `TrackDb` class is an abstraction around a database of tracks. Implementations include `BrowserSession` derivatives and `QuickloadGenome`. Here, a track is defined as an interval dataset.

Accessor Methods

Every implementation should support these methods:

`length(x)`: number of tracks

`names(x)`, `trackNames(x)`: names of the tracks

`mcols(x)`: merged metadata on the tracks

`track(x, name)`, `x$name`, `x[[name]]`: get the track called `name`

`track(x, name) <- value`, `x$name <- value`, `x[[name]] <- value`: store the track `value` under `name`. Different implementations will support different types for `value`. Generally, an interval data structure like `GenomicRanges`.

Author(s)

Michael Lawrence

`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).

`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.

TwoBitFile-class *2bit Files*

Description

These functions support the import and export of the UCSC 2bit compressed sequence format. The main advantage is speed of subsequence retrieval, as it only loads the sequence in the requested intervals. Compared to the FA format supported by Rsamtools, 2bit offers the additional feature of masking and also has better support in Java (and thus most genome browsers). The supporting `TwoBitFile` class is a reference to a TwoBit file.

Usage

```
## S4 method for signature 'TwoBitFile,ANY,ANY'
import(con, format, text,
           which = as(seqinfo(con), "GenomicRanges"), ...)
## S4 method for signature 'TwoBitFile'
getSeq(x, which = as(seqinfo(con), "GenomicRanges"))
import.2bit(con, ...)

## S4 method for signature 'ANY,TwoBitFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'DNAStringSet,TwoBitFile,ANY'
export(object, con, format)
## S4 method for signature 'DNAStringSet,character,ANY'
export(object, con, format, ...)
export.2bit(object, con, ...)
```

Arguments

`con`	A path, URL or `TwoBitFile` object. Connections are not supported. For the functions ending in `.2bit`, the file format is indicated by the function name. For the `export` and `import` methods, the format must be indicated another way. If `con` is a path, or URL, either the file extension or the `format` argument needs to be “twoBit” or “2bit”.
`object,x`	The object to export, either a `DNAStringSet` or something coercible to a `DNAStringSet`, like a character vector.
`format`	If not missing, should be “twoBit” or “2bit” (case insensitive).
`text`	Not supported.
`which`	A range data structure coercible to `RangesList`, like a `GRanges`, or a `TwoBitFile`. Only the intervals in the file overlapping the given ranges are returned. By default, the value is the `TwoBitFile` itself. Its `Seqinfo` object is extracted and coerced to a `RangesList` that represents the entirety of the file.
`...`	Arguments to pass down to methods to other methods. For import, the flow eventually reaches the `TwoBitFile` method on `import`. For export, the `TwoBitFile` methods on `export` are the sink.

Value

For import, a `DNAStringSet`.

`TwoBitFile` objects

A `TwoBitFile` object, an extension of `RTLFile` is a reference to a TwoBit file. To cast a path, URL or connection to a `TwoBitFile`, pass it to the `TwoBitFile` constructor.

A TwoBit file embeds the sequence information, which can be retrieved with the following:

`seqinfo(x)`: Gets the `Seqinfo` object indicating the lengths of the sequences for the intervals in the file. No circularity or genome information is available.

Author(s)

Michael Lawrence

See Also

export-methods in the **BSgenome** package for exporting a BSgenome object as a twoBit file.

Examples

```
  test_path <- system.file("tests", package = "rtracklayer")
  test_2bit <- file.path(test_path, "test.2bit")

  test <- import(test_2bit)
  test

  test_2bit_file <- TwoBitFile(test_2bit)
  import(test_2bit_file) # the whole file

  which_range <- IRanges(c(10, 40), c(30, 42))
  which <- GRanges(names(test), which_range)
  import(test_2bit, which = which)

  seqinfo(test_2bit_file)

## Not run:
  test_2bit_out <- file.path(tempdir(), "test_out.2bit")
  export(test, test_2bit_out)

  ## just a character vector
  test_char <- as.character(test)
  export(test_char, test_2bit_out)

## 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, trackLine = TRUE, ...)` 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.

`UCSCFile-class` *UCSCFile objects*

Description

These functions support the import and export of tracks emucscded within the UCSC track line metaformat, whereby multiple tracks may be concatenated within a single file, along with metadata mostly oriented towards visualization. Any `UCSCData` or `RangedDataList` object is automatically exported in this format, if the targeted format is known to be compatible. The BED and WIG import methods check for a track line, and delegate to these functions if one is found. Thus, calling this API directly is only necessary when importing embedded GFF (rare), or when one wants to create the track line during the export process.

Usage

```
## S4 method for signature 'UCSCFile,ANY,ANY'
import(con, format, text,
                   subformat = "auto", drop = FALSE,
                   genome = NA, ...)
import.ucsc(con, ...)

## S4 method for signature 'ANY,UCSCFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'GenomicRanges,UCSCFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'GenomicRangesList,UCSCFile,ANY'
export(object, con, format,
                   append = FALSE, index = FALSE, ...)
## S4 method for signature 'UCSCData,UCSCFile,ANY'
export(object, con, format,
                   subformat = "auto", append = FALSE, index = FALSE, ...)
export.ucsc(object, con, ...)
```

Arguments

`con`	A path, URL, connection or `UCSCFile` object. For the functions ending in `.ucsc`, the file format is indicated by the function name. For the base `export` and `import` functions, “ucsc” must be passed as the `format` argument.
`object`	The object to export, should be a `GRanges` or something coercible to a `GRanges`. For exporting multiple tracks pass a `GenomicRangesList`, or something coercible to one.
`format`	If not missing, should be “ucsc”.
`text`	If con is missing, a character vector to use as the input
`subformat`	The file format to use for the actual features, between the track lines. Must be a text-based format that is compatible with track lines (most are). If an `RTLFile` subclass other than `UCSCFile` is passed as `con` to `import.ucsc` or `export.ucsc`,

	the subformat is assumed to be the corresponding format of `con`. Otherwise it defaults to "auto". The following describes the logic of the "auto" mode. For import, the subformat is taken as the `type` field in the track line. If none, the file extension is consulted. For export, if `object` is a `UCSCData`, the subformat is taken as the `type` in its track line, if present. Otherwise, the subformat is chosen based on whether `object` contains a "score" column. If there is a score, the target is either `BEDGraph` or `WIG`, depending on the structure of the ranges. Otherwise, `BED` is the target.
`genome`	The identifier of a genome, or `NA` if unknown. Typically, this is a UCSC identifier like "hg19". An attempt will be made to derive the `seqinfo` on the return value using either an installed BSgenome package or UCSC, if network access is available. This defaults to the `db` BED track line parameter, if any.
`drop`	If `TRUE`, and there is only one track in the file, return the track object directly, rather than embedding it in a list.
`append`	If `TRUE`, and `con` points to a file path, the data is appended to the file. Obviously, if `con` is a connection, the data is always appended.
`index`	If `TRUE`, automatically compress and index the output file with bgzf and tabix. Note that tabix indexing will sort the data by chromosome and start. Does not work when exporting a `RangedDataList` with multiple elements; tabix supports a single track in a file.
`...`	Should either specify track line parameters or arguments to pass down to the import and export routine for the subformat.

Details

The UCSC track line permits the storage of multiple tracks in a single file by separating them with a so-called "track line", a line belonging with the word "track" and containing various `key=value` pairs encoding metadata, most related to visualization. The standard fields in a track depend on the type of track being annotated. See `TrackLine` and its derivatives for how these lines are represented in R. The class `UCSCData` is an extension of `GRanges` with a formal slot for a `TrackLine`. Each `GRanges` in the returned `GenomicRangesList` has the track line stored in its metadata, under the `trackLine` key.

For each track object to be exported, if the object is not a `UCSCData`, and there is no `trackLine` element in the metadata, then a new track line needs to be generated. This happens through the coercion of `object` to `UCSCData`. The track line is initialized to have the appropriate `type` parameter for the subformat, and the required `name` parameter is taken from the name of the track in the input list (if any). Otherwise, the default is simply "R Track". The `db` parameter (specific to BED track lines) is taken as `genome(object)` if not `NA`. Additional arguments passed to the export routines override parameters in the provided track line.

If the subformat is either WIG or BEDGraph, and the features are stranded, a separate track will be output in the file for each strand. Neither of those formats encodes the strand and disallow overlapping features (which might occur upon destranding).

Value

A `GenomicRangesList` unless `drop` is `TRUE` and there is only a single track in the file. In that case, the first and only object is extracted from the list and returned. The structure of that object depends on the format of the data. The `GenomicRangesList` contains `GRanges` objects with a `trackLine` element in their metadata, whereas the `RangedDataList` contains `UCSCData` objects.

UCSCFile objects

The `UCSCFile` class extends `RTLFile` and is a formal represention of a resource in the UCSC format. To cast a path, URL or connection to a `UCSCFile`, pass it to the `UCSCFile` constructor.

Author(s)

Michael Lawrence

References

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

`ucscGenomes` *Get available genomes on UCSC*

Description

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

Usage

```
ucscGenomes(organism=FALSE)
```

Arguments

`organism`	A `logical(1)` indicating whether scientific name should be appended.

Details

For populating the organism column, the web url `http://genome.ucsc.edu/cgi-bin` is scraped for every assembly version to get the scientific name.

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
`organism`	The scientific name of the species (e.g. "Homo sapiens")

Author(s)

Michael Lawrence

See Also

`UCSCSession` for details on specifying the genome.

Examples

```
ucscGenomes()
```

`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)
```

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)** Gets the `GRanges` last displayed in this session.

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

`seqinfo` Gets the `Seqinfo` object with the lengths of the chromosomes in the currenet genome. No circularity information is available.

`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`. The `value` may be either a range object (like a `GRanges`) or a file object (like a `BEDFile`).

`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.

`getTable(object, name, range = base::range(object), table = NULL)`: Retrieves the table indicated by the track `name` and `table` name, over `range`, as a `data.frame`. See `getTable`.

`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 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)`: Retrieves the indicated table as a track, i.e. a `GRanges` object. 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, and that UCSC will truncate responses if they exceed certain limits (usually around 100,000 records). The safest (and most efficient) bet for large queries is to download the file via FTP and query it locally.

`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)  # a GRanges object
## 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)
```

`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` 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(),` 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.

`WIGFile-class` *WIG Import and Export*

Description

These functions support the import and export of the UCSC WIG (Wiggle) format.

Usage

```
## S4 method for signature 'WIGFile,ANY,ANY'
import(con, format, text, genome = NA,
                    trackLine = TRUE, which = NULL, seqinfo = NULL, ...)
import.wig(con, ...)

## S4 method for signature 'ANY,WIGFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'GenomicRanges,WIGFile,ANY'
export(object, con, format,
                   dataFormat = c("auto", "variableStep", "fixedStep"),
                   writer = .wigWriter, append = FALSE, ...)
## S4 method for signature 'GenomicRangesList,WIGFile,ANY'
export(object, con, format, ...)
## S4 method for signature 'UCSCData,WIGFile,ANY'
export(object, con, format,
                   trackLine = TRUE, ...)
export.wig(object, con, ...)
```

Arguments

`con`	A path, URL, connection or `WIGFile` object. For the functions ending in `.wig`, the file format is indicated by the function name. For the base `export` and `import` functions, the format must be indicated another way. If con is a path, URL or connection, either the file extension or the `format` argument needs to be "wig". Compressed files ("gz", "bz2" and "xz") are handled transparently.
`object`	The object to export, should be a `GRanges` or something coercible to a `GRanges`. For exporting multiple tracks, in the UCSC track line metaformat, pass a `GenomicRangesList`, or something coercible to one.
`format`	If not missing, should be "wig".
`text`	If con is missing, a character vector to use as the input
`trackLine`	Whether to parse/output a UCSC track line. An imported track line will be stored in a `TrackLine` object, as part of the returned `UCSCData`.
`genome`	The identifier of a genome, or `NA` if unknown. Typically, this is a UCSC identifier like "hg19". An attempt will be made to derive the `seqinfo` on the return value using either an installed BSgenome package or UCSC, if network access is available.
`seqinfo`	If not `NULL`, the `Seqinfo` object to set on the result. If the `genome` argument is not `NA`, it must agree with `genome(seqinfo)`.
`which`	A range data structure like `RangesList` or `GRanges`. Only the intervals in the file overlapping the given ranges are returned. This is inefficient; use BigWig for efficient spatial queries.
`append`	If `TRUE`, and con points to a file path, the data is appended to the file. Obviously, if con is a connection, the data is always appended.
`dataFormat`	Probably best left to "auto". Exists only for historical reasons.

`writer`	Function for writing out the blocks; for internal use only.
`...`	Arguments to pass down to methods to other methods. For import, the flow eventually reaches the `WIGFile` method on `import`. For export, the `RangedData`, `WIGFile` method on `export` is the sink. When `trackLine` is `TRUE`, the arguments are passed through `export.ucsc`, so track line parameters are supported.

Details

The WIG format is a text-based format for efficiently representing a dense genome-scale score vector. It encodes, for each feature, a range and score. Features from the same sequence (chromosome) are grouped together into a block, with a single block header line indicating the chromosome. There are two block formats: fixed step and variable step. For fixed step, the number of positions (or step) between intervals is the same across an entire block. For variable step, the start position is specified for each feature. For both fixed and variable step, the span (or width) is specified in the header and thus must be the same across all features. This requirement of uniform width dramatically limits the applicability of WIG. For scored features of variable width, consider BEDGraph or BigWig, which is generally preferred over both WIG and BEDGraph. To efficiently convert an existing WIG or BEDGraph file to BigWig, call `wigToBigWig`. Neither WIG, BEDGraph nor BigWig allow overlapping features.

Value

A `GRanges` with the score values in the `score` metadata column, which is accessible via the `score` function.

WIGFile objects

The `WIGFile` class extends `RTLFile` and is a formal represention of a resource in the WIG format. To cast a path, URL or connection to a `WIGFile`, pass it to the `WIGFile` constructor.

Author(s)

Michael Lawrence

References

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

Examples

```
test_path <- system.file("tests", package = "rtracklayer")
test_wig <- file.path(test_path, "step.wig")

## basic import calls
test <- import(test_wig)
test
import.wig(test_wig)
test_wig_file <- WIGFile(test_wig)
import(test_wig_file)
test_wig_con <- file(test_wig)
```

```
    import(test_wig_con, format = "wig")
    close(test_wig_con)
    test_wig_con <- file(test_wig)
    import(WIGFile(test_wig_con))
    close(test_wig_con)

    ## various options
    import(test_wig, genome = "hg19")
    import(test_wig, trackLine = FALSE)
    which <- as(test[3:4,], "RangesList")
    import(test_wig, which = which)

  ## Not run:
    ## basic export calls
    test_wig_out <- file.path(tempdir(), "test.wig")
    export(test, test_wig_out)
    export.wig(test, test_wig_out)
    test_foo_out <- file.path(tempdir(), "test.foo")
    export(test, test_foo_out, format = "wig")
    test_wig_out_file <- WIGFile(test_wig_out)
    export(test, test_wig_out_file)

    ## appending
    test2 <- test
    metadata(test2)$trackLine <- initialize(metadata(test)$trackLine,
                                            name = "test2")
    export(test2, test_wig_out_file, append = TRUE)

    ## passing track line parameters
    export(test, test_wig_out, name = "test2")

    ## no track line
    export(test, test_wig_out, trackLine = FALSE)

    ## gzip
    test_wig_gz <- paste(test_wig_out, ".gz", sep = "")
    export(test, test_wig_gz)

  ## End(Not run)
```

`wigToBigWig` *Convert WIG to BigWig*

Description

This function calls the Kent C library to efficiently convert a WIG file to a BigWig file, without loading the entire file into memory. This solves the problem where simple tools write out text WIG files, instead of more efficiently accessed binary BigWig files.

Usage

```
wigToBigWig(x, seqinfo,
            dest = paste(file_path_sans_ext(x, TRUE), "bw", sep = "."))
```

Arguments

`x`	Path or URL to the WIG file. Connections are not supported.
`seqinfo`	`Seqinfo` object, describing the genome of the data. All BigWig files must have this defined.
`dest`	The path to which to write the BigWig file. Defaults to `x` with the extension changed to “bw”.

Author(s)

Michael Lawrence

See Also

`BigWig` import and export support

Index