Package 'Bioc.gff'

October 31, 2025

Title Read and write GFF and GTF files

Version 1.0.0

Date 2025-09-24

Description Parse GFF and GTF files using C++ classes. The package also provides utilities to read and write GFF3 files. The GFF (General Feature Format) format is a tab-delimited file format for describing genes and other features of DNA, RNA, and protein sequences. GFF files are often used to describe the features of genomes.

Depends R (>= 4.5.0)

Imports BiocBaseUtils, BiocGenerics, BiocIO, curl, GenomicRanges, IRanges, methods, Rsamtools, S4Vectors, Seqinfo, stats, utils, XVector

Suggests BiocFileCache, BiocStyle, GenomicFeatures, GenomeInfoDbData, knitr, httr2, rmarkdown, rvest, tinytest, txdbmaker, TxDb.Hsapiens.UCSC.hg19.knownGene

VignetteBuilder knitr

LinkingTo S4Vectors, XVector, IRanges

License Artistic-2.0

biocViews Software, Infrastructure, DataImport

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://github.com/Bioconductor/Bioc.gff

BugReports https://github.com/Bioconductor/Bioc.gff/issues

Collate 'Bioc.gff-package.R' 'GFF-coercion.R' 'readGFF.R' 'GFFFile-class.R' 'index.R' 'metadataFromNCBI.R' 'utilities.R'

git_url https://git.bioconductor.org/packages/Bioc.gff

git_branch RELEASE_3_22

git_last_commit 5bbebde

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2025-10-31

Author Michael Lawrence [aut],
Hervé Pagès [aut],
Marcel Ramos [ctb],
Bioconductor Package Maintainer [cre]

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

Contents

`Bioc.gff-package` *Bioc.gff: Read and write GFF and GTF files*

Description

Parse GFF and GTF files using C++ classes. The package also provides utilities to read and write GFF3 files. The GFF (General Feature Format) format is a tab-delimited file format for describing genes and other features of DNA, RNA, and protein sequences. GFF files are often used to describe the features of genomes.

Author(s)

Maintainer: Bioconductor Package Maintainer <maintainer@bioconductor.org>

Authors:

- Michael Lawrence
- Hervé Pagès

Other contributors:

- Marcel Ramos [contributor]

See Also

Useful links:

- https://github.com/Bioconductor/Bioc.gff
- Report bugs at https://github.com/Bioconductor/Bioc.gff/issues

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)
...	Arguments to pass to methods
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.

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).

Methods (by class)

- asGFF(GRangesList): Coerce to GFF GRanges structure

Author(s)

Michael Lawrence

Examples

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(GenomicFeatures)
exons <- exonsBy(TxDb.Hsapiens.UCSC.hg19.knownGene)
mcols(asGFF(exons))
```

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

```
GFFFile(resource, version = c("", "1", "2", "3"))

export.gff(object, con, ...)

## S4 method for signature 'ANY'
export.gff(object, con, ...)

## S4 method for signature 'ANY,GFFFile,ANY'
export(object, con, format, ...)

## S4 method for signature 'CompressedGRangesList,GFFFile,ANY'
export(object, con, format, ...)

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

## S4 method for signature 'SimpleGRangesList,GFFFile,ANY'
export(object, con, format, ...)

export.gff1(object, con, ...)

## S4 method for signature 'ANY'
export.gff1(object, con, ...)

export.gff2(object, con, ...)

## S4 method for signature 'ANY'
export.gff2(object, con, ...)

export.gff3(object, con, ...)

## S4 method for signature 'ANY'
export.gff3(object, con, ...)
```

```
## 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.gff1(con, ...)

## S4 method for signature 'ANY'
import.gff1(con, ...)

import.gff2(con, ...)

## S4 method for signature 'ANY'
import.gff2(con, ...)

import.gff3(con, ...)

## S4 method for signature 'ANY'
import.gff3(con, ...)

## S4 method for signature 'GFFFile'
genome(x)
```

Arguments

resource	character(1) or connection A low-level resource typically a path, URL, or connection.
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".
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.
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.
...	Arguments to pass down to methods to other methods. For import, the flow eventually reaches the GFFFile method on import. When trackLine is TRUE

	or the target format is BED15, the arguments are passed through `export.ucsc`, so track line parameters are supported.
`format`	If not missing, should be one of "gff", "gff1" "gff2", "gff3", "gvf", or "gtf".
`source`	The value for the source column in GFF. This is typically the name of the package or algorithm that generated the feature.
`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. Tabix supports a single track in a file.
`text`	If `con` is missing, a character vector to use as the input.
`genome`	The identifier of a genome, or a `Seqinfo`, 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 `GRanges` or other range-based object supported by `findOverlaps`. 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.
`x`	A `GFFFile` object.

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 `Bioc.gff` 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 `Bioc.gff` 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` 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 "Bioc.gff" 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.

Functions

- `export.gff()`:
- `export.gff(ANY)`:
- `export(object = ANY, con = GFFFile, format = ANY)`:
- `export(object = CompressedGRangesList, con = GFFFile, format = ANY)`:
- `export(object = GenomicRanges, con = GFFFile, format = ANY)`:
- `export(object = SimpleGRangesList, con = GFFFile, format = ANY)`:
- `export.gff1()`:
- `export.gff1(ANY)`:
- `export.gff2()`:
- `export.gff2(ANY)`:
- `export.gff3()`:
- `export.gff3(ANY)`:
- `import(con = GFFFile, format = ANY, text = ANY)`:
- `import.gff1()`:
- `import.gff1(ANY)`:
- `import.gff2()`:
- `import.gff2(ANY)`:
- `import.gff3()`:
- `import.gff3(ANY)`:
- `genome(GFFFile)`: Gets the genome identifier from the "genome-build" header directive.

GFFFile objects

The GFFFile class extends BiocFile and is a formal representation 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.

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_gff3 <- system.file(
    "extdata", "genes.gff3", package = "Bioc.gff", mustWork = TRUE
)

## 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")

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

## 'which'
library(GenomicRanges)
which <- GRanges("chr10:90000-93000")
import(test_gff3, which = which)

## '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, ".bgz", sep = "")
import(test_bed_gz, which = which)

## cleanup
file.remove(
    test_gff3_out, test_bed_gz, paste(test_bed_gz, "tbi", sep = ".")
)
```

`metadataFromNCBI` *Obtain metadata from NCBI*

Description

These helper functions obtain both the Taxonomy ID and the Organism name from the NCBI Taxonomy Browser. They are a modern re-write of the old functions in `rtracklayer`. They use `httr2` and `rvest` to parse the HTML content.

Usage

```
isNCBISpeciesURL(url)

metadataFromNCBI(url)

parseOrganismFromNCBI(html)

parseTaxonomyIDFromNCBI(html, url)
```

Arguments

`url`	A URL to the NCBI Taxonomy Browser, typically obtained from a GFF file with the `## species` line.

Value

- metadataFromNCBI: A `list` with two elements: `Taxonomy ID` and `Organism`.
- parseOrganismFromNCBI: A `character` with the Organism name.
- isNCBISpeciesURL: A `logical` indicating if the URL is from the NCBI Taxonomy Browser.
- parseTaxonomyIDFromNCBI: A `character` with the Taxonomy ID.

Examples

```
isNCBISpeciesURL(.NCBI_TAX_URL)

metadataFromNCBI(
    paste0(.NCBI_TAX_URL, "?mode=Info&id=9606")
)
metadataFromNCBI(
    paste0(.NCBI_TAX_URL, "?id=3702")
)
metadataFromNCBI(
    paste0(.NCBI_TAX_URL, "?name=drosophila+melanogaster")
)
metadataFromNCBI(
    paste0(.NCBI_TAX_URL, "?name=drosophila+miranda")
)
```

---

readGFF *Reads a file in GFF format*

Description

Reads a file in GFF format and creates a data frame or S4Vectors::DataFrame() object from it. This is a lower-level function that should not be called by the end user. Users are recommended to use the import() function on the GFFFile or file path.

Usage

```
GFFcolnames(GFF1 = FALSE)

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

Arguments

GFF1	logical(1) Use "group" instead of "attributes" for the 9th column name. Default is FALSE.
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	named list() Specify to load only desired features, e.g., list(type = c("gene", "mRNA"), seqid = "chr10").
nrows	-1 or the maximum number of rows to read in (after filtering).
raw_data	logical(1) If TRUE, numeric columns (e.g. "start" or "score") are loaded as character vectors and as-is i.e. how they are found in the file.

Value

A DataFrame with columns corresponding to those in the GFF.

Author(s)

H. Pagès

See Also

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

Examples

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

test_gff3 <- system.file(
    "extdata", "genes.gff3", package="Bioc.gff", mustWork=TRUE
)

## 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)

library(GenomicRanges)
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)
```

Index