Package: regutools (via r-universe)

October 31, 2026

Title regutools: an R package for data extraction from RegulonDB

Version 1.24.0

Date 2021-11-15

Description RegulonDB has collected, harmonized and centralized data from hundreds of experiments for nearly two decades and is considered a point of reference for transcriptional regulation in Escherichia coli K12. Here, we present the regutools R package to facilitate programmatic access to RegulonDB data in computational biology. regutools provides researchers with the possibility of writing reproducible workflows with automated queries to RegulonDB. The regutools package serves as a bridge between RegulonDB data and the Bioconductor ecosystem by reusing the data structures and statistical methods powered by other Bioconductor packages. We demonstrate the integration of regutools with Bioconductor by analyzing transcription factor DNA binding sites and transcriptional regulatory networks from RegulonDB. We anticipate that regutools will serve as a useful building block in our progress to further our understanding of gene regulatory networks.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.0)

Imports AnnotationDbi, AnnotationHub, Biostrings, DBI, GenomicRanges, Gviz, IRanges, RCy3, RSQLite, S4Vectors, methods, stats, utils, BiocFileCache

LazyData true

RoxygenNote 7.1.1

Suggests BiocStyle, knitr, RefManageR, rmarkdown, sessioninfo, testthat (>= 2.1.0), covr

URL https://github.com/ComunidadBioInfo/regutools

BugReports https://support.bioconductor.org/t/regutools

biocViews GeneRegulation, GeneExpression, SystemsBiology, Network,NetworkInference,Visualization, Transcription

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/pak/sysreqs cmake make libbz2-dev libicu-dev libjpeg-dev liblzma-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3 xz-utils libzmq3-dev zlib1g-dev

Repository Bioconductor 3.23

Date/Publication 2026-04-28 12:52:39 UTC

RemoteUrl https://github.com/bioc/regutools

RemoteRef RELEASE_3_23

RemoteSha 14d3139fd5e340d3d8dd922809d716c0a2fe4edc

Contents

build_condition . . . 3
connect_database . . . 4
convert_to_biostrings . . . 5
convert_to_granges . . . 6
existing_intervals . . . 7
existing_partial_match . . . 8
get_binding_sites . . . 9
get_dataset . . . 10
get_dna_objects . . . 11
get_gene_regulators . . . 13
get_gene_synonyms . . . 14
get_regulatory_network . . . 15
get_regulatory_summary . . . 16
guess_id . . . 18
list_attributes . . . 19
list_datasets . . . 20
non_existing_intervals . . . 21
plot_dna_objects . . . 22
regulondb . . . 23
regulondb-class . . . 24
regulondb_result-class . . . 24
show . . . 25

Index **26**

build_condition *Construct logical condition to query database*

Description

Given a list of filters, this function builds a logical condition to query database. The output is used in `get_dataset()`.

Usage

```
build_condition(regulondb, dataset, filters, operator, interval, partialmatch)
```

Arguments

regulondb	A `regulondb()` object.
dataset	dataset of interest
filters	List of filters to be used. The names should correspond to the attribute and the values correspond to the condition for selection.
operator	A string indicating if all the filters (AND) or some of them (OR) should be met
interval	the filters with values considered as interval
partialmatch	name of the condition(s) with a string pattern for full or partial match in the query

Value

A `character(1)` with the sql logical condition to query the dataset .

Author(s)

Carmina Barberena Jonás, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chávez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Build the condition for ara
build_condition(
```

```
    e_coli_regulondb,
    dataset = "GENE",
    filters = list(
        name = c("ara"),
        strand = c("forward"),
        posright = c("2000", "40000")
    ),
    operator = "AND",
    interval = "posright",
    partialmatch = "name"
)
```

---

connect_database *Connect to the regulondb database*

Description

This function downloads the RegulonDB SQLite database file prior to making a connection to it. It will cache the database file such that subsequent calls will run faster. This function requires an active internet connection.

Usage

```
connect_database(
  ah = AnnotationHub::AnnotationHub(),
  bfc = BiocFileCache::BiocFileCache()
)
```

Arguments

ah	An AnnotationHub object AnnotationHub-class. Can be NULL if you want to force to use the backup download mechanism.
bfc	A BiocFileCache object BiocFileCache-class. Used when ah is not available.

Value

An SQLiteConnection-class connection to the RegulonDB database.

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Connect to the database without using AnnotationHub
regulondb_conn_noAH <- connect_database(ah = NULL)
```

convert_to_biostrings *Function to convert output of regulondb queries to Biostrings objects*

Description

This function converts, when possible, a regulon_result object into a Biostrings object.

Usage

```
convert_to_biostrings(regulondb_result, seq_type = "DNA")
```

Arguments

regulondb_result	A regulon_result object.
seq_type	A character string with either DNA or protein, specyfing what

Value

A XStringSet object.

Author(s)

Alejandro Reyes

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Obtain all the information from the "GENE" dataset
convert_to_biostrings(get_dataset(e_coli_regulondb, dataset = "GENE"))
```

`convert_to_granges` *Function to convert output of regulondb queries to GenomicRanges objects*

Description

This function converts, when possible, a regulon_result object into a GRanges object.

Usage

```
convert_to_granges(regulondb_result)
```

Arguments

`regulondb_result`	A regulon_result object.

Value

A GRanges object.

Author(s)

Alejandro Reyes

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Obtain all the information from the "GENE" dataset
convert_to_granges(get_dataset(e_coli_regulondb, dataset = "GENE"))
```

existing_intervals *Constructs a particular logical condition to query database*

Description

Given a list of filters, this function builds a logical condition to query database using intervals. The output is used in `build_condition()`.

Usage

```
existing_intervals(filters, interval, operator, partialmatch)
```

Arguments

`filters`	List of filters to be used. The names should correspond to the attribute and the values correspond to the condition for selection.
`interval`	the filters with values considered as interval.
`operator`	A string indicading if all the filters (AND) or some of them (OR) should be met.
`partialmatch`	name of the condition(s) with a string pattern for full or partial match in the query.

Value

A `character(1)` with the sql logical condition to query the dataset.

Author(s)

Carmina Barberena Jonás, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chávez

Examples

```
## Build the SQL query for existing interval partial matches for ara
existing_intervals(
    filters = list(
        name = "ara",
        strand = "for",
        posright = c("2000", "40000")
    ),
    interval = c("posright"),
    operator = "AND",
    partialmatch = c("name", "strand")
)
```

existing_partial_match *Constructs a logical condition to query database*

Description

Given a list of filters, this function builds a logical condition to query database using intervals. The output is used in `existing_intervals()` and `non_existing_intervals()`.

Usage

```
existing_partial_match(filters, partialmatch, operator)
```

Arguments

`filters`	List of filters to be used. The names should correspond to the attribute and the values correspond to the condition for selection.
`partialmatch`	name of the condition(s) with a string pattern for full or partial match in the query.
`operator`	A string indicating if all the filters (AND) or some of them (OR) should be met.

Value

A `character(1)` with the sql logical condition to query the dataset.

Author(s)

Carmina Barberena Jonás, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández

Examples

```
## Build the SQL query for existing partial matches for ara
existing_partial_match(
    filters = list(
        name = c("ara"),
        strand = c("forward"),
        posright = c("2000", "40000")
    ),
    partialmatch = "name",
    operator = "AND"
)
```

get_binding_sites *Get the binding sites for a Transcription Factor (TF)*

Description

Retrieve the binding sites and genome location for a given transcription factor.

Usage

```
get_binding_sites(regulondb, transcription_factor, output_format = "GRanges")
```

Arguments

regulondb	A regulondb() object.
transcription_factor	name of the transcription factor.
output_format	The output object. Can be either a GRanges (default) or Biostrings.

Value

Either a GRanges object or a Biostrings object summarizing information about the binding sites of the transcription factors.

Author(s)

José Alquicira Hernández, Jacques van Helden, Joselyn Chávez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Get the binding sites for AraC
get_binding_sites(e_coli_regulondb, transcription_factor = "AraC")
```

get_dataset *Extract data from RegulonDB*

Description

This function retrieves data from RegulonDB. Attributes from datasets can be selected and filtered.

Usage

```
get_dataset(
  regulondb,
  dataset = NULL,
  attributes = NULL,
  filters = NULL,
  and = TRUE,
  interval = NULL,
  partialmatch = NULL,
  output_format = "regulondb_result"
)
```

Arguments

regulondb	A regulondb() object.
dataset	Dataset of interest. Use the function list_datasets for an overview of valid datasets.
attributes	Vector of attributes to be retrieved.
filters	List of filters to be used. The names should correspond to the attribute and the values correspond to the condition for selection.
and	Logical argument. If FALSE, filters will be considered under the "OR" operator
interval	the filters whose values will be considered as interval
partialmatch	name of the condition(s) with a string pattern for full or partial match in the query
output_format	A string specifying the output format. Possible options are "regulondb_result", "GRanges", "DNAStringSet" or "BStringSet".

Value

By default, a regulon_results object. If specified in the parameter output_format, it can also return either a GRanges object or a Biostrings object.

Author(s)

Carmina Barberena Jonas, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chávez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Obtain all the information from the "GENE" dataset
get_dataset(e_coli_regulondb, dataset = "GENE")

## Get the attributes posright and name from the "GENE" dataset
get_dataset(e_coli_regulondb,
    dataset = "GENE",
    attributes = c("posright", "name")
)

## From "GENE" dataset, get the gene name, strand, posright, product name
## and id of all genes regulated with name like "ara", strand as "forward"
## with a position right between 2000 and 40000
get_dataset(
    e_coli_regulondb,
    dataset = "GENE",
    attributes = c("name", "strand", "posright", "product_name", "id"),
    filters = list(
        name = c("ara"),
        strand = c("forward"),
        posright = c("2000", "40000")
    ),
    and = TRUE,
    partialmatch = "name",
    interval = "posright"
)
```

---

get_dna_objects *Retrieve genomic elements from regulonDB*

Description

Retrieve genomic elements from regulonDB

Usage

```
get_dna_objects(
```

```
  regulondb,
  genome = "eschColi_K12",
  grange = GRanges("chr", IRanges(1, 5000)),
  elements = "gene"
)
```

Arguments

regulondb	A regulondb() object.
genome	A valid UCSC genome name.
grange	A GenomicRanges::GRanges-class() object indicating position left and right.
elements	A character vector specifying which annotation elements to plot. It can be any from: "-10 promoter box", "-35 promoter box", "gene", "promoter", "Regulatory Interaction", "sRNA interaction", or "terminator".

Value

GenomicRanges::GRanges-class() object with the elements found.

Author(s)

Joselyn Chavez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) {
      regulondb_conn <- connect_database()
  }

## Build the regulondb object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "chr",
        database_version = "1",
        genome_version = "1"
    )

## Get all genes from E. coli
get_dna_objects(e_coli_regulondb)

## Get genes providing Genomic Ranges
grange <- GenomicRanges::GRanges(
    "chr",
    IRanges::IRanges(5000, 10000)
)
get_dna_objects(e_coli_regulondb, grange)

## Get aditional elements within genomic positions
get_dna_objects(e_coli_regulondb,
```

```
    grange,
    elements = c("gene", "promoter")
)
```

`get_gene_regulators` *Get TFs or genes that regulate the genes of interest*

Description

Given a list of genes (name, bnumber or GI), get all transcription factors or genes that regulate them. The effect of regulators over the gene of interest can be positive (+), negative (-) or dual (+/-)

Usage

```
get_gene_regulators(regulondb, genes, format = "multirow", output.type = "TF")
```

Arguments

`regulondb`	A regulondb class.
`genes`	Vector of genes (name, bnumber or GI).
`format`	Output format: multirow, onerow, table
`output.type`	How regulators will be represented: "TF"/"GENE"

Value

A regulondb_result object.

Author(s)

Carmina Barberena Jonas, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chávez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Get Transcription factors that regulate araC in one row
get_gene_regulators(
    e_coli_regulondb,
```

```
    genes = c("araC"),
    output.type = "TF",
    format = "onerow"
)

## Get genes that regulate araC in table format
get_gene_regulators(
    e_coli_regulondb,
    genes = c("araC"),
    output.type = "GENE",
    format = "table"
)
```

`get_gene_synonyms` *Retrieve gene synonyms*

Description

Given a list of genes (id, name, bnumber or gi), get the gene synonyms (name, bnumber of gi).

Usage

```
get_gene_synonyms(
  regulondb,
  genes,
  from = "name",
  to = c("id", "name", "bnumber", "gi")
)
```

Arguments

`regulondb`	A `regulondb()` object.
`genes`	Character vector of gene identifiers (id, name, bnumber or gi).
`from`	A `character()` specifying one of: id, name, bnumber of gi
`to`	A `character()` specifying one or more of: id, name, bnumber of gi

Value

A regulondb_result object.

Author(s)

Jesús Emiliano Sotelo Fonseca

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Lists all available identifiers for "araC"
get_gene_synonyms(e_coli_regulondb, "araC", from = "name")

## Retrieve only the ID
get_gene_synonyms(e_coli_regulondb, "araC", from = "name", to = "id")

## Use an ID to retrieve the synonyms
get_gene_synonyms(e_coli_regulondb, "ECK120000998", from = "id")
```

---

get_regulatory_network

Return complete regulatory network.

Description

This function retrieves all the regulation networks in regulonDB between TF-TF, GENE-GENE or TF-GENE depending on the parameter 'type'.

Usage

```
get_regulatory_network(
  regulondb,
  regulator = NULL,
  type = "TF-GENE",
  cytograph = FALSE
)
```

Arguments

regulondb	A regulondb() object.
regulator	Name of TF or gene that acts as regulator. If NULL, the function retrieves all existent networks in the regulonDB.
type	"TF-GENE", "TF-TF", "GENE-GENE"
cytograph	If TRUE, displays network in Cytoscape. This option requires previous instalation and launch of Cytoscape.

Value

A regulondb_result object.

Author(s)

Carmina Barberena Jonas, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chávez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Retrieve regulation of 'araC'
get_regulatory_network(e_coli_regulondb,
    regulator = "AraC",
    type = "TF-GENE"
)

## Retrieve all GENE-GENE networks
get_regulatory_network(e_coli_regulondb, type = "GENE-GENE")

## Retrieve TF-GENE network of AraC and display in Cytoscape
## Note that Cytospace needs to be open for this to work
cytoscape_present <- try(RCy3::cytoscapePing(), silent = TRUE)
if (!is(cytoscape_present, "try-error")) {
    get_regulatory_network(
        e_coli_regulondb,
        regulator = "AraC",
        type = "TF-GENE",
        cytograph = TRUE
    )
}
```

`get_regulatory_summary` *Return summary of gene regulation.*

Description

This function takes the output of `get_gene_regulators()` with format multirow, onerow or table, or a vector with genes and retrieves information about the TFs and their regulated genes

Usage

```
get_regulatory_summary(regulondb, gene_regulators)
```

Arguments

regulondb	A regulondb() object.
gene_regulators	Result from get_gene_regulators() or vector of genes

Value

A data frame with the following columns:

- The name or gene of TF
- Regulated Genes per TF
- Percent of regulated genes per TF
- positive, negative or dual regulation
- Name(s) of regulated genes

Author(s)

Carmina Barberena Jonas, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chávez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Get the araC regulators
araC_regulation <-
    get_gene_regulators(
        e_coli_regulondb,
        genes = c("araC"),
        format = "multirow",
        output.type = "TF"
    )

## Summarize the araC regulation
get_regulatory_summary(e_coli_regulondb, araC_regulation)

## Retrieve summary of genes 'araC' and 'modB'
```

```
get_regulatory_summary(e_coli_regulondb,
    gene_regulators = c("araC", "modB")
)

## Obtain the summary for 'ECK120000050' and 'modB'
get_regulatory_summary(e_coli_regulondb,
    gene_regulators = c("ECK120000050", "modB")
)
```

`guess_id` *Guess gene id type*

Description

Given a gene identifier, return the most likely gene_id type.

Usage

```
guess_id(gene, regulondb)
```

Arguments

`gene`	Character vector of gene identifiers (id, name, bnumber or gi).
`regulondb`	A `regulondb()` object.

Value

A `character(1)` vector with the name column guessed value.

Author(s)

Jesús Emiliano Sotelo Fonseca

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## Lists all available identifiers for "araC"
## Guess name
```

```
guess_id("araC", e_coli_regulondb)

## Guess id
guess_id("ECK120000050", e_coli_regulondb)

## Guess bnumber
guess_id("b0064", e_coli_regulondb)
```

`list_attributes` *List attributes/fields from a dataset/table*

Description

List all attributes and their description of a dataset from RegulonDB. The result of this function may be used as parameter 'values' in `list_attributes()` function.

Usage

```
list_attributes(regulondb, dataset)
```

Arguments

`regulondb`	A `regulondb()` object.
`dataset`	Dataset of interest. The name should correspond to a table of the database.

Value

A character vector with the field names.

Author(s)

Carmina Barberena Jonás, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández, Joselyn Chavez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## List the transcription factor attributes
list_attributes(e_coli_regulondb, "TF")
```

```
## List the operon attributes
list_attributes(e_coli_regulondb, "OPERON")
```

`list_datasets` *List available datasets in RegulonDB database*

Description

This function returns a vector of all available tables from a regulondb class.

Usage

```
list_datasets(regulondb)
```

Arguments

`regulondb`	A regulondb class.

Value

A `character()` with the names of the available datasets.

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build the regulon db object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "E.coli",
        database_version = "1",
        genome_version = "1"
    )

## List the available datasets
list_datasets(e_coli_regulondb)
```

non_existing_intervals *Constructs a logical condition to query database*

Description

Given a list of filters, this function builds a logical condition to query database using intervals. The output is used in `build_condition()`.

Usage

```
non_existing_intervals(filters, interval, operator, partialmatch)
```

Arguments

`filters`	List of filters to be used. The names should correspond to the attribute and the values correspond to the condition for selection.
`interval`	the filters whose values will be considered as interval
`operator`	A string indicating if all the filters (AND) or some of them (OR) should be met.
`partialmatch`	name of the condition(s) with a string pattern for full or partial match in the query.

Value

A `character(1)` with the sql logical condition to query the dataset.

Author(s)

Carmina Barberena Jonás, Jesús Emiliano Sotelo Fonseca, José Alquicira Hernández

Examples

```
## Build the SQL query for finidng non-existing intervals for the gene ara
non_existing_intervals(
    filters = list(name = "ara", strand = "for"),
    interval = NULL,
    operator = "AND",
    partialmatch = c("name", "strand")
)
```

`plot_dna_objects` *Plot annotation elements within genomic region*

Description

Plot annotation elements within genomic region

Usage

```
plot_dna_objects(
  regulondb,
  genome = "eschColi_K12",
  grange = GRanges("chr", IRanges(1, 5000)),
  elements = "gene"
)
```

Arguments

`regulondb`	A `regulondb()` object.
`genome`	A valid UCSC genome name.
`grange`	A `GenomicRanges::GRanges-class()` object indicating position left and right.
`elements`	A character vector specifying which annotation elements to plot. It can be any from: `"-10 promoter box"`, `"-35 promoter box"`, `"gene"`, `"promoter"`, `"Regulatory Interaction"`, `"sRNA interaction"`, or `"terminator"`.

Value

A plot with genomic elements found within a genome region, including genes and regulators.

Author(s)

Joselyn Chavez

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) {
      regulondb_conn <- connect_database()
  }

## Build the regulondb object
e_coli_regulondb <-
    regulondb(
        database_conn = regulondb_conn,
        organism = "chr",
        database_version = "1",
        genome_version = "1"
    )
```

```
## Plot some genes from E. coli using default parameters
plot_dna_objects(e_coli_regulondb)

## Plot genes providing Genomic Ranges
grange <- GenomicRanges::GRanges(
    "chr",
    IRanges::IRanges(5000, 10000)
)
plot_dna_objects(e_coli_regulondb, grange)

## Plot aditional elements within genomic positions
plot_dna_objects(e_coli_regulondb,
    grange,
    elements = c("gene", "promoter")
)
```

`regulondb` *Constructor function of a regulondb class*

Description

The build_regulondb function is a constructor function of a regulondb class.

Usage

```
regulondb(database_conn, organism, genome_version, database_version)
```

Arguments

`database_conn`	A SQLiteConnection-class connection to the RegulonDB database made with `connect_database()`.
`organism`	A character vector with the name of the organism of the database.
`genome_version`	A character vector with the version of the genome build.
`database_version`	A character vector with the version of regulondb build.

Value

A regulondb object.

Examples

```
## Connect to the RegulonDB database if necessary
if (!exists("regulondb_conn")) regulondb_conn <- connect_database()

## Build a regulondb object
e_coli_regulondb <-
```

```
regulondb(
    database_conn = regulondb_conn,
    organism = "E.coli",
    database_version = "1",
    genome_version = "1"
)
```

`regulondb-class` *The regulondb class*

Description

The regulondb class is an extension of the SQLiteConnection, which as the name suggests, consists of an SQLite connection to a database with the table design of the RegulonDb database. In addition to the slots defined in the SQLiteConnection object, the regulondb class also contains additional slots to store information about database versions, organism information and genome build versions.

Slots

`organism` A character vector with the name of the organism of the database.

`genome_version` A character vector with the version of the genome build.

`database_version` A character vector with the version of regulondb build.

`regulondb_result-class`
The regulondb_results class

Description

The regulondb class is an extension of the DataFrame class, with additional slots that host information of the database used to obtain these results.

Slots

`organism` A character string with the name of the organism of the database.

`genome_version` A character string with the version of the genome build.

`database_version` A character string with the version of regulondb build.

`dataset` A character string with the name of the table used for the query in get_dataset().

show *Methods for regulondb objects*

Description

Methods for regulondb objects

Usage

```
## S4 method for signature 'regulondb'
show(object)
```

Arguments

object	A regulondb object

Value

A regulondb object.

Index

∗ **TF,**
get_regulatory_network, 15
∗ **TFs,**
get_gene_regulators, 13
∗ **attributes**
list_attributes, 19
∗ **bnumber,**
get_gene_synonyms, 14
guess_id, 18
∗ **database**
list_datasets, 20
∗ **datasets**
list_datasets, 20
∗ **data**
list_attributes, 19
list_datasets, 20
∗ **geneid,**
get_gene_synonyms, 14
guess_id, 18
∗ **gi,**
get_gene_synonyms, 14
guess_id, 18
∗ **networks,**
get_gene_regulators, 13
get_regulatory_network, 15
get_regulatory_summary, 16
∗ **regulation**
get_gene_regulators, 13
get_regulatory_network, 15
get_regulatory_summary, 16
∗ **retrieval,**
get_gene_regulators, 13
get_regulatory_network, 15
get_regulatory_summary, 16
list_attributes, 19
∗ **retrieval**
list_datasets, 20
∗ **summary,**
get_regulatory_summary, 16
∗ **synonyms**
get_gene_synonyms, 14
guess_id, 18

AnnotationHub-class, *4*

BiocFileCache-class, *4*
build_condition, 3
build_condition(), *7*, *21*
build_regulondb *(*regulondb*)*, 23

connect_database, 4
connect_database(), *23*
convert_to_biostrings, 5
convert_to_granges, 6

existing_intervals, 7
existing_intervals(), *8*
existing_partial_match, 8

get_binding_sites, 9
get_dataset, 10
get_dataset(), *3*
get_dna_objects, 11
get_gene_regulators, 13
get_gene_regulators(), *16, 17*
get_gene_synonyms, 14
get_regulatory_network, 15
get_regulatory_summary, 16
GRanges, *6*
guess_id, 18

list_attributes, 19
list_attributes(), *19*
list_datasets, 20

non_existing_intervals, 21
non_existing_intervals(), *8*

plot_dna_objects, 22

regulondb, 23, *23*, *25*

regulondb(), *3*, *9, 10*, *12*, *14, 15*, *17–19*, *22*
regulondb-class, 24
regulondb_result, *13, 14*, *16*
regulondb_result-class, 24

show, 25
show,regulondb-method *(*show*)*, 25
SQLiteConnection-class, *4*, *23*

XStringSet, *5*