

# Package ‘BiocSet’

March 12, 2025

**Title** Representing Different Biological Sets

**Version** 1.21.0

**Description** BiocSet displays different biological sets in a triple tibble format. These three tibbles are `element`, `set`, and `elementset`. The user has the ability to activate one of these three tibbles to perform common functions from the dplyr package. Mapping functionality and accessing web references for elements/sets are also available in BiocSet.

**Depends** R (>= 3.6), dplyr

**Imports** methods, tibble, utils, rlang, plyr, S4Vectors, BiocIO, AnnotationDbi, KEGGREST, ontologyIndex, tidyr

**Suggests** GSEABase, airway, org.Hs.eg.db, DESeq2, limma, BiocFileCache, GO.db, testthat, knitr, rmarkdown, BiocStyle

**biocViews** GeneExpression, GO, KEGG, Software

**License** Artistic-2.0

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**VignetteBuilder** knitr

**git\_url** <https://git.bioconductor.org/packages/BiocSet>

**git\_branch** devel

**git\_last\_commit** 263c319

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.21

**Date/Publication** 2025-03-12

**Author** Kayla Morrell [aut, cre],  
Martin Morgan [aut],  
Kevin Rue-Albrecht [ctb],  
Lluís Revilla Sancho [ctb]

**Maintainer** Kayla Morrell <kayla.morre116@gmail.com>

## Contents

BiocSet . . . . .	2
BiocSet-methods . . . . .	4
coerce . . . . .	6
elementset_funs . . . . .	6
element_funs . . . . .	8
genesetcollection . . . . .	9
import . . . . .	10
intersect_single . . . . .	11
mapping_element . . . . .	11
mapping_set . . . . .	13
OBOSet . . . . .	14
obo_relations . . . . .	15
set_funs . . . . .	16
tblelement . . . . .	17
tblelementset . . . . .	18
tblset . . . . .	18
union_single . . . . .	19
url_ref . . . . .	19
<b>Index</b>	<b>21</b>

---

BiocSet

*BiocSet class*

---

### Description

character()

The BiocSet constructor, the show method, the slot accessors, and creating a BiocSet object from an element set tibble rather than character vector(s).

### Usage

```
BiocSet(..., metadata = list(), active = c("elementset", "element", "set"))
```

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

```
es_element(x)
```

```
## S4 method for signature 'BiocSet'
es_element(x)
```

```
es_set(x)
```

```
## S4 method for signature 'BiocSet'
es_set(x)
```

```

es_elementset(x)

## S4 method for signature 'BiocSet'
es_elementset(x)

BiocSet_from_elementset(elementset, element, set, metadata)

```

### Arguments

...	Named character() vectors of element sets, or a named list of character() vectors. Each character vector is an element set. The names of the character vectors are the names of the sets.
metadata	A list() with arbitrary content, describing the set.
active	A character(1) to indicate which tibble is active. The default is "elementset".
object	A BiocSet object.
x	A BiocSet object.
elementset	A tibble with element set information.
element	A tibble with element information.
set	A tibble with set information.

### Value

An S4 BiocSet object shown as a tripple tibble, where each slot is a tibble.

### Slots

element	The element tibble from 'tbl_elementset'
set	The set tibble from 'tbl_elementset'
elementset	The elementset tibble created from user input
active	A character(1), indicates which tibble is active
metadata	A list() with arbitrary elements describing the set

### Examples

```

BiocSet(set1 = letters, set2 = LETTERS)
lst <- list(set1 = letters, set2 = LETTERS)
BiocSet(lst)

set.seed(123)
element <-
  tibble(
    element = letters[1:10],
    v1 = sample(10),
    v2 = sample(10)
  )
set <-

```

```

    tibble(
      set = LETTERS[1:2],
      v1 = sample(2),
      v2 = sample(2)
    )
  elementset <-
    tibble(
      element = letters[1:10],
      set = sample(LETTERS[1:2], 10, TRUE)
    )
  BiocSet_from_elementset(elementset, element, set)

```

---

 BiocSet-methods

*BiocSet methods*


---

## Description

`es_activate`: which of the three tibbles in the `BiocSet` object should be activated and have the chosen functionality applied to it.

`filter`: choose rows where conditions are true.

`select`: keep only the variables listed.

`mutate`: add new variable and preserve the existing variables.

`summarise`: usually used with `group_by()`, output will have one row for each group.

`arrange`: order rows by an expression involving its variables.

`.tbl_nongroup_vars`: returns only non-grouping variables.

`group_by`: converts an existing `tbl` into a grouped `tbl`.

`left_join`: returns all rows from `x`, and all columns from `x` and `y`. If no rows in `x` match with `y` there will be NAs in the new column. If there are multiple matches then all combinations are returned.

`as.list`: coerces argument into a list.

`union`: combines all rows from two `BiocSet` objects and removes duplicate records from the combined `BiocSet` object.

`intersect`: combines all rows from two `BiocSet` objects and returns rows that appear in both `BiocSet` objects.

## Usage

```
es_activate(.data, what)
```

```
## S3 method for class 'BiocSet'
filter(.data, ...)
```

```
## S3 method for class 'BiocSet'
select(.data, ...)
```

```
## S3 method for class 'BiocSet'  
mutate(.data, ...)  
  
## S3 method for class 'BiocSet'  
summarise(.data, ...)  
  
## S3 method for class 'BiocSet'  
arrange(.data, ...)  
  
.tbl_nongroup_vars.BiocSet(x)  
  
## S3 method for class 'BiocSet'  
group_by(.data, ..., add = FALSE)  
  
## S3 method for class 'BiocSet'  
left_join(x, y, by, copy, suffix, ...)  
  
## S3 method for class 'BiocSet'  
as.list(x, ...)  
  
## S3 method for class 'BiocSet'  
union(x, y, ...)  
  
## S3 method for class 'BiocSet'  
intersect(x, y, ...)
```

### Arguments

<code>.data</code>	The BiocSet object.
<code>what</code>	Which of the three tibbles from BiocSet to activate.
<code>...</code>	Additional arguments passed to function.
<code>x</code>	For <code>.tbl_nongroup_vars</code> (internal), a BiocSet object. For <code>union</code> and <code>intersect</code> the first BiocSet object to perform the operations on.
<code>add</code>	logical, whether to add to the existing groups.
<code>y</code>	For <code>left_join</code> , a tibble to join. For <code>union</code> and <code>intersect</code> the second BiocSet object used.
<code>by</code>	A character vector of variables to join by.
<code>copy</code>	logical, allows you to join tables across srcs.
<code>suffix</code>	Character vector of length 2, if there are non-joined duplicate variables in 'x' and 'y' these suffixes will be added to the output.

### Value

A BiocSet object.

**Examples**

```

es <- BiocSet(set1 = letters, set2 = LETTERS)
es_activate(es, element)

es %>% es_activate(element) %>% filter(element == "a")

es %>% select(element)

es %>% es_activate(set) %>% mutate(pval = rnorm(1:2))

es %>% es_activate(set) %>% summarise(n = n())

es %>% es_activate(element) %>% arrange(desc(element))

es %>% mutate(pval = rnorm(1:52)) %>% es_elementset() %>%
  BiocSet:::tbl_nongroup_vars()

es %>% group_by(element, set)

es <- BiocSet(set1 = letters[1:5], set2 = LETTERS[1:5])
tbl <- tibble(x = 1:10, y = c(letters[1:5], LETTERS[1:5]))
es %>% left_join(tbl, by = c(element = "y"))

library(org.Hs.eg.db)
es <- go_sets(org.Hs.eg.db, "ENSEMBL")
head(as.list(es))

es1 <- BiocSet(set1 = letters[c(1:4)], set2 = LETTERS[c(1:4)])
es2 <- BiocSet(set1 = letters[c(3:8)], set2 = LETTERS[c(3:8)])

dplyr::union(es1, es2)

dplyr::intersect(es1, es2)

```

---

coerce	<i>as("BiocSet", "list")</i>
--------	------------------------------

---

**Description**

```
as("BiocSet", "list")
```

---

elementset_funs	<i>Functions applied to elementsets in a BiocSet object</i>
-----------------	---

---

**Description**

All of the major methods applied to a BiocSet object can be explicitly applied to the elementset tibble. These functions bypass the need to use the `es_activate` function by indicating what function should be used on the elementset tibble.

**Usage**

```
filter_elementset(.data, ...)  
select_elementset(.data, ...)  
mutate_elementset(.data, ...)  
summarise_elementset(.data, ...)  
arrange_elementset(.data, ...)  
left_join_elementset(.data, ...)  
tibble_from_elementset(.data)  
data.frame_from_elementset(.data)
```

**Arguments**

```
.data      A BiocSet object.  
...        Additional arguments passed to the function.
```

**Value**

A BiocSet object.  
For `tibble_from_elementset`, a tibble.  
For `data.frame_from_elementset`, a data.frame.

**Examples**

```
es <- BiocSet(set1 = letters, set2 = LETTERS)  
filter_elementset(es, element == "a" | element == "A")  
  
es %>% select_elementset(element)  
  
es %>% mutate_elementset(pval = rnorm(1:52))  
  
es %>% summarise_elementset(n = n())  
  
es %>% arrange_elementset(desc(element))  
  
tbl <- tibble(x = 5:6, y = c("set1", "set2"))  
es %>% left_join_elementset(tbl, by = c(set = "y"))  
  
tibble_from_elementset(es)  
  
data.frame_from_elementset(es)
```

**Description**

All of the major methods applied to a BiocSet object can be explicitly applied to the element tibble. These functions bypass the need to use the `es_activate` function by indicating what function should be used on the element tibble.

**Usage**

```
filter_element(.data, ...)  
select_element(.data, ...)  
mutate_element(.data, ...)  
summarise_element(.data, ...)  
arrange_element(.data, ...)  
left_join_element(.data, ...)  
tibble_from_element(.data, how = unlist)  
data.frame_from_element(.data, how = unlist)
```

**Arguments**

<code>.data</code>	A BiocSet object.
<code>...</code>	Additional arguments passed to the function.
<code>how</code>	Multiple entries will become a list.

**Value**

A BiocSet object.  
For `tibble_from_element`, a tibble.  
For `data.frame_from_element`, a data.frame.

**Examples**

```
es <- BiocSet(set1 = letters, set2 = LETTERS)  
filter_element(es, element == "a")  
  
es %>% select_element(element)
```



```
es %>% mutate_element(pval = rnorm(1:52))

es %>% summarise_element(n = n())

es %>% arrange_element(desc(element))

tbl <- tibble(x = 1:5, y = letters[1:5])
es <- BiocSet(set1 = letters[c(1,3,5)], set2 = letters[c(2,4)])
left_join_element(es, tbl, by = c(element = "y"))

tibble_from_element(es)

data.frame_from_element(es)
```

---

genesetcollection	<i>GeneSetCollection</i>
-------------------	--------------------------

---

## Description

The following functions deal with converting a `BiocSet` object into a `GeneSetCollection` object, or vice versa.

## Usage

```
GeneSetCollection_from_BiocSet(biocset)
```

```
BiocSet_from_GeneSetCollection(gsc)
```

## Arguments

<code>biocset</code>	The <code>BiocSet</code> object that will become a <code>GeneSetCollection</code> object.
<code>gsc</code>	The <code>GeneSetCollection</code> that will become a <code>BiocSet</code> object.

## Value

For `'GeneSetCollection_from_BiocSet()'`, a `GeneSetCollection`.

For `'BiocSet_from_GeneSetCollection()'`, a `BiocSet` object.

## Examples

```
biocset <- BiocSet(set1 = letters, set2 = LETTERS)
gsc <- GeneSetCollection_from_BiocSet(biocset)
gsc

BiocSet_from_GeneSetCollection(gsc)
```

---

import	<i>Importing/exporting</i>
--------	----------------------------

---

## Description

Importing/exporting and formatting of element sets as a BiocSet object.

## Usage

```
## S4 method for signature 'GMTFile,ANY,ANY'
import(con, format, text, ...)

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

## S4 method for signature 'OBOFile,ANY,ANY'
import(con, format, text, ...)

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

## Arguments

con	For import, the file name or URL the element set is loaded from. For export, the file name or URL the element set is written to.
format	For import, the format of the input. For export, the format of the output.
text	If con is missing this is a character vector directly providing the element set that should be imported.
...	Parameters to pass to the format-specific method
object	For 'export()', the object to be exported.

## Value

For 'import()', a BiocSet object

For 'export()', a GMTFile object representing the location where the BiocSet object was written to

## Examples

```
gmtFile <- system.file(package = "BiocSet", "extdata",
  "hallmark.gene.symbol.gmt")
tbl <- import(gmtFile)

tbl2 <- BiocSet(set1 = letters, set2 = LETTERS)
fl <- tempfile(fileext = ".gmt")
gmt <- export(tbl2, fl)
```

```

oboFile <- system.file(package = "BiocSet", "extdata", "sample_go.obo")
tst_obo <- import(oboFile)
fl <- system.file("extdata", "sample_go.obo", package = "BiocSet")
tbl <- import(fl)
new_fl <- tempfile(fileext = ".obo")
obo <- export(tbl, new_fl)

```

---

intersect_single	<i>Intersect on a single BiocSet object</i>
------------------	---

---

### Description

This function performs an intersection within a single BiocSet object.

### Usage

```
intersect_single(x, ...)
```

### Arguments

x	A BiocSet object.
...	Additional arguments passed to function.

### Value

A BiocSet object with a single set 'intersect' and intersected elements from x.

### Examples

```

es1 <- BiocSet(set1 = letters[c(1:10)], set2 = letters[c(4:20)])
intersect_single(es1)

```

---

mapping_element	<i>Functions for mapping elements in the element tibble to different id types</i>
-----------------	---

---

### Description

Functions for dealing with unique mapping and multiple mapping. `map_add_element` will add the mapping as a new column instead of overwriting the current one used for the mapping.

**Usage**

```
map_unique(es, org, from, to)

map_multiple(
  es,
  org,
  from,
  to,
  multi = c("list", "filter", "asNA", "CharacterList")
)

map_add_element(es, org, from, add)
```

**Arguments**

<code>es</code>	The BiocSet object to map the elements on.
<code>org</code>	The AnnotationDbi object to identify keys/mappings from.
<code>from</code>	A character to indicate which identifier to map from.
<code>to</code>	A character to indicate which identifier to map to.
<code>multi</code>	How should multiple values be returned? Options include: <ul style="list-style-type: none"> <li>• <code>list</code>: This will just return a list object to the end user.</li> <li>• <code>filter</code>: This will remove all elements that contain multiple matches and will therefore return a shorter vector than what came in whenever some of the keys match more than one value.</li> <li>• <code>asNA</code>: This will return an NA value whenever there are multiple matches.</li> <li>• <code>CharacterList</code>: This just returns a SimpleCharacterList object.</li> <li>• <code>FUN</code>: A function can be supplied to the 'multiVals' argument for custom behaviors.</li> </ul>
<code>add</code>	The id to add to the BiocSet object.

**Value**

For `map_unique`, a BiocSet object with unique elements.

For `map_multiple`, a BiocSet object with multiple mappings for certain elements.

For `map_add_element`, a BiocSet object with a new column in the element tibble with the mapping of the new id type.

**Examples**

```
library(org.Hs.eg.db)
es <- BiocSet(set1 = c("C5", "GANC"), set2 = c("AFM", "CGB1", "ADAM32"))
map_unique(es, org.Hs.eg.db, "SYMBOL", "ENTREZID")

map_multiple(es, org.Hs.eg.db, "SYMBOL", "ENSEMBLTRANS", "asNA")

map <- map_add_element(es, org.Hs.eg.db, "SYMBOL", "ENTREZID")
es %>% mutate_element(entrez = map)
```

## Description

Functions for creating BiocSet objects from GO sets and KEGG sets, and creating a new set mapping from a current BiocSet object. `map_add_set` will add the mapping as a new column instead of overwriting the current one used for the mapping.

## Usage

```
go_sets(org, from, go = c("GO", "GOID"), evidence = NULL, ontology = NULL)
```

```
kegg_sets(species)
```

```
map_set(.data, from, to)
```

```
map_add_set(.data, org, from, add)
```

## Arguments

<code>org</code>	The AnnotationDbi object to identify keys/mappings from.
<code>from</code>	A character to indicate which identifier to map from.
<code>go</code>	A character to indicate the column name for the GO ids. Default is "GO".
<code>evidence</code>	A character to indicate the evidence codes for GO associations with a gene of interest. Default is all possible evidence codes.
<code>ontology</code>	A character to indicate which Gene Ontology to use. Default is BP, CC, and MF.
<code>species</code>	Which species the pathways are from.
<code>.data</code>	The BiocSet object that contains the set tibble being mapped.
<code>to</code>	A character to indicate which identifier to map to.
<code>add</code>	The id to add to the BiocSet object.

## Value

For `go_sets`, a BiocSet object with GO ids as the set ids.

For `kegg_sets`, a BiocSet object with Entrez IDs reported as elements (default from KEGGREST) and KEGG pathways as sets.

For `map_set`, a BiocSet object with the mapped set present in the set tibble.

For `map_add_set`, a BiocSet object with a new column in the set tibble with the mapping of the new id type.

**Examples**

```

library(org.Hs.eg.db)
go <- go_sets(org.Hs.eg.db, "ENSEMBL")

kegg_sets("hsa")

es <- BiocSet(set1 = letters, set2 = LETTERS)
es %>% map_set("set1", "foo")

library(GO.db)
map <- map_add_set(go, GO.db, "GOID", "DEFINITION")
go %>% mutate_set(definition = map)

```

---

OBOSet

*OBOSet class*


---

**Description**

A class representing the 'OBO' file format as a BiocSet.

**Usage**

```
OBOSet(elementset, element, set, metadata)
```

**Arguments**

elementset	A tibble with element set information.
element	A tibble with element information.
set	A tibble with set information.
metadata	A tibble with key-value pairs describing OBO file format header data

**Value**

An S4 OBOSet object. OBO sets conform to the 'obo' file format, with OBO 'Term' entries corresponding to elements. Parent / child relationships (e.g., 'is\_a') are summarized as 'parents', 'ancestors', and 'children' character list columns of 'set'.

**Examples**

```

OBOSet()
oboFile <- system.file(package = "BiocSet", "extdata", "sample_go.obo")
import(oboFile)

```

---

`obo_relations`*Functions to display relationships of an OBOSet object*

---

**Description**

These functions will display the relationships (children, parents, or ancestors) for either the elements or the sets of an OBOSet object.

**Usage**`oboset_element_children(oboset)``oboset_element_parents(oboset)``oboset_element_ancestors(oboset)``oboset_set_children(oboset)``oboset_set_parents(oboset)``oboset_set_ancestors(oboset)`**Arguments**`oboset`            The OBOSet of interest.**Value**

A 2 column tibble.

**Examples**

```
oboFile <- system.file("extdata", "sample_go.obo", package = "BiocSet")
obo <- import(oboFile)
oboset_element_children(obo)

oboset_element_parents(obo)

oboset_element_ancestors(obo)

oboset_set_children(obo)

oboset_set_parents(obo)

oboset_set_ancestors(obo)
```

---

`set_funs`*Functions applied to sets in a BiocSet object*

---

**Description**

All of the major methods applied to a BiocSet object can be explicitly applied to the set tibble. These functions bypass the need to use the `es_activate` function by indicating what function should be used on the element tibble.

**Usage**

```
filter_set(.data, ...)
select_set(.data, ...)
mutate_set(.data, ...)
summarise_set(.data, ...)
arrange_set(.data, ...)
left_join_set(.data, ...)
tibble_from_set(.data, how = unlist)
data.frame_from_set(.data, how = unlist)
```

**Arguments**

<code>.data</code>	A BiocSet object.
<code>...</code>	Additional argument passed to the function.
<code>how</code>	Multiple entries will become a list.

**Value**

A BiocSet object.  
For `tibble_from_set`, a tibble.  
For `data.frame_from_set`, a data.frame.

**Examples**

```
es <- BiocSet(set1 = letters, set2 = LETTERS)
filter_set(es, set == "set1")

es %>% select_set(set)
```



```
es %>% mutate_set(pval = rnorm(1:2))

es %>% summarise_set(n = n())

es %>% arrange_set(desc(set))

tbl <- tibble(x = 10:11, y = c("set1", "set2"))
es <- BiocSet(set1 = letters[c(1,3,5)], set2 = letters[c(2,4)])
left_join_set(es, tbl, by = c(set = "y"))

tibble_from_set(es)

data.frame_from_set(es)
```

---

tbl\_element

*Element representation as an S3 class tibble*

---

## Description

Element representation as an S3 class tibble

## Usage

```
.tbl_element(tbl_elementset)
```

## Arguments

tbl\_elementset An S3 elementset tibble.

## Value

An S3 element object in a tibble representation.

## Examples

```
tbl <- BiocSet:::tbl_elementset(set1 = letters, set2 = LETTERS)
BiocSet:::tbl_element(tbl)
```

---

tblelementset	<i>Element set representation as an S3 class tibble</i>
---------------	---

---

**Description**

Element set representation as an S3 class tibble

**Usage**

```
.tbl_elementset(...)  
.is_tbl_elementset(x)
```

**Arguments**

...	For <code>tbl_elementset</code> , named character vectors of gene sets. Each character vector is a element set. The name of the character vector is the name of the element set.
x	A tibble that may or may not be an elementset

**Value**

For `tbl_elementset`, an S3 elementset object in a tibble representation.

For `.is_tbl_elementset` (internal), a logical indicating if the tibble is an elementset.

**Examples**

```
es <- BiocSet:::.tbl_elementset(set1 = letters, set2 = LETTERS)  
BiocSet:::.is_tbl_elementset(es)
```

---

tblset	<i>Set representation as an S3 class tibble</i>
--------	---

---

**Description**

Set representation as an S3 class tibble

**Usage**

```
.tbl_set(tbl_elementset)
```

**Arguments**

`tbl_elementset` An S3 elementset tibble.

**Value**

An S3 set object in a tibble representation.

**Examples**

```
tbl <- BiocSet:::.tbl_elementset(set1 = letters, set2 = LETTERS)
BiocSet:::.tbl_set(tbl)
```

---

union_single	<i>Union on a single BiocSet object</i>
--------------	---

---

**Description**

This function performs a union within a single BiocSet object.

**Usage**

```
union_single(x, ...)
```

**Arguments**

x	A BiocSet object.
...	Additional arguments passed to function.

**Value**

For union\_single, a BiocSet object with a single set union and unioned elements from x.

**Examples**

```
es3 <- BiocSet(set1 = letters[c(1:10)], set2 = letters[c(4:20)])
union_single(es3)
```

---

url_ref	<i>Functions to access reference urls for different identifiers</i>
---------	---

---

**Description**

Functions to access reference urls for different identifiers

**Usage**

```
url_ref_element(es)
```

```
url_ref_set(es)
```

```
url_ref(es)
```

**Arguments**

es                    A BiocSet object that the reference urls should be added to.

**Value**

For url\_ref\_element, a BiocSet object with the url column added to the element tibble.

For url\_ref\_set, a BiocSet object with the url column added to the set tibble.

For url\_ref, a BiocSet object with the url column added to both the element and set tibbles.

**Examples**

```
es <- BiocSet("GO:0000002" = c("TP53", "TNF"), "GO:0000003" = c("IL6"))
url_ref_element(es)

url_ref_set(es)

url_ref(es)
```

# Index

## \* internal

- BiocSet-methods, 4
- tblelement, 17
- tblelementset, 18
- tblset, 18
- .is\_tbl\_elementset (tblelementset), 18
- .tbl\_element (tblelement), 17
- .tbl\_elementset (tblelementset), 18
- .tbl\_nongroup\_vars.BiocSet (BiocSet-methods), 4
- .tbl\_set (tblset), 18
- arrange.BiocSet (BiocSet-methods), 4
- arrange\_element (element\_funs), 8
- arrange\_elementset (elementset\_funs), 6
- arrange\_set (set\_funs), 16
- as.list.BiocSet (BiocSet-methods), 4
- BiocSet, 2
- BiocSet-class (BiocSet), 2
- BiocSet-methods, 4
- BiocSet\_from\_elementset (BiocSet), 2
- BiocSet\_from\_GeneSetCollection (genesetcollection), 9
- coerce, 6
- coerce, BiocSet, list-method (coerce), 6
- data.frame\_from\_element (element\_funs), 8
- data.frame\_from\_elementset (elementset\_funs), 6
- data.frame\_from\_set (set\_funs), 16
- element\_funs, 8
- elementset\_funs, 6
- es\_activate (BiocSet-methods), 4
- es\_element (BiocSet), 2
- es\_element, BiocSet-method (BiocSet), 2
- es\_elementset (BiocSet), 2
- es\_elementset, BiocSet-method (BiocSet), 2
- es\_set (BiocSet), 2
- es\_set, BiocSet-method (BiocSet), 2
- export, BiocSet, GMTFile, ANY-method (import), 10
- export, BiocSet, OBOFile, ANY-method (import), 10
- filter.BiocSet (BiocSet-methods), 4
- filter\_element (element\_funs), 8
- filter\_elementset (elementset\_funs), 6
- filter\_set (set\_funs), 16
- genesetcollection, 9
- GeneSetCollection\_from\_BiocSet (genesetcollection), 9
- go\_sets (mapping\_set), 13
- group\_by.BiocSet (BiocSet-methods), 4
- import, 10
- import, GMTFile, ANY, ANY-method (import), 10
- import, OBOFile, ANY, ANY-method (import), 10
- intersect.BiocSet (BiocSet-methods), 4
- intersect\_single, 11
- kegg\_sets (mapping\_set), 13
- left\_join.BiocSet (BiocSet-methods), 4
- left\_join\_element (element\_funs), 8
- left\_join\_elementset (elementset\_funs), 6
- left\_join\_set (set\_funs), 16
- map\_add\_element (mapping\_element), 11
- map\_add\_set (mapping\_set), 13
- map\_multiple (mapping\_element), 11
- map\_set (mapping\_set), 13
- map\_unique (mapping\_element), 11

mapping\_element, 11  
mapping\_set, 13  
mutate.BiocSet (BiocSet-methods), 4  
mutate\_element (element\_funs), 8  
mutate\_elementset (elementset\_funs), 6  
mutate\_set (set\_funs), 16

obo\_relations, 15  
OBOSet, 14  
OBOSet-class (OBOSet), 14  
oboset\_element\_ancestors  
    (obo\_relations), 15  
oboset\_element\_children  
    (obo\_relations), 15  
oboset\_element\_parents (obo\_relations),  
    15  
oboset\_set\_ancestors (obo\_relations), 15  
oboset\_set\_children (obo\_relations), 15  
oboset\_set\_parents (obo\_relations), 15

select.BiocSet (BiocSet-methods), 4  
select\_element (element\_funs), 8  
select\_elementset (elementset\_funs), 6  
select\_set (set\_funs), 16  
set\_funs, 16  
show, BiocSet-method (BiocSet), 2  
summarise.BiocSet (BiocSet-methods), 4  
summarise\_element (element\_funs), 8  
summarise\_elementset (elementset\_funs),  
    6  
summarise\_set (set\_funs), 16

tblelement, 17  
tblelementset, 18  
tblset, 18  
tibble\_from\_element (element\_funs), 8  
tibble\_from\_elementset  
    (elementset\_funs), 6  
tibble\_from\_set (set\_funs), 16

union.BiocSet (BiocSet-methods), 4  
union\_single, 19  
url\_ref, 19  
url\_ref\_element (url\_ref), 19  
url\_ref\_set (url\_ref), 19