

# Package ‘plyxp’

April 8, 2026

**Title** Data masks for SummarizedExperiment enabling dplyr-like manipulation

**Version** 1.5.0

**Description** The package provides `rlang` data masks for the SummarizedExperiment class. The enables the evaluation of unquoted expression in different contexts of the SummarizedExperiment object with optional access to other contexts. The goal for `plyxp` is for evaluation to feel like a data.frame object without ever needing to unwind to a rectangular data.frame.

**License** MIT + file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Imports** dplyr, purrr, rlang, SummarizedExperiment, tidyr, tidyselect, vctrs, tibble, pillar, cli, glue, S7, S4Vectors, utils, methods

**Depends** R (>= 4.5.0)

**Suggests** devtools, knitr, rmarkdown, testthat, airway, IRanges, here

**LazyData** true

**URL** <https://github.com/jtlandis/plyxp>,  
<https://jtlandis.github.io/plyxp>

**BugReports** <https://www.github.com/jtlandis/plyxp/issues>

**biocViews** Annotation, GenomeAnnotation, Transcriptomics

**Config/testthat/edition** 3

**Collate** 'as.data.frame.R' 'connect\_masks.R' 'context-utils.R' 'data.R'  
'dplyr-across.R' 'dplyr-arrange.R' 'dplyr-filter.R'  
'dplyr-group\_by.R' 'dplyr-mutate.R' 'dplyr-pull.R'  
'dplyr-select.R' 'dplyr-slice.R' 'dplyr-summarise.R'  
'group\_utils.R' 'grouped\_list.R' 'vctrs-S4-.R' 'vctrs-S4-rep.R'  
'vctrs-S4-unchop.R' 'vctrs-S4-chop.R' 'mask\_env\_top.R'  
'plyxp-R6-manager.R' 'plyxp-R6.R' 'plyxp-class.R'  
'plyxp-package.R' 'plyxp-quos.R' 'plyxp.R' 'plyxp\_cannot.R'

'print.R' 'reexports-SummarizedExperiment.R'  
 'reexports-dplyr.R' 'utils.R' 'vctrs-S4-group.R'  
 'vctrs-S4-recycle.R' 'vctrs-S4-slice.R' 'zzz-globals.R' 'zzz.R'

**git\_url** <https://git.bioconductor.org/packages/plyxp>  
**git\_branch** devel  
**git\_last\_commit** b0d04f2  
**git\_last\_commit\_date** 2025-10-29  
**Repository** Bioconductor 3.23  
**Date/Publication** 2026-04-07  
**Author** Justin Landis [aut, cre] (ORCID:  
 <<https://orcid.org/0000-0001-5501-4934>>),  
 Michael Love [aut] (ORCID: <<https://orcid.org/0000-0001-8401-0545>>)  
**Maintainer** Justin Landis <jtlandis314@gmail.com>

## Contents

plyxp-package . . . . .	3
arrange . . . . .	3
as.data.frame . . . . .	4
dot-pronouns . . . . .	5
filter . . . . .	6
group_by . . . . .	7
group_data . . . . .	8
group_split . . . . .	9
group_vars . . . . .	9
list_unchop . . . . .	10
mutate . . . . .	10
new_plyxp . . . . .	11
plyxp . . . . .	12
plyxp-context . . . . .	13
pull . . . . .	14
reexports . . . . .	15
se . . . . .	16
select . . . . .	19
se_simple . . . . .	20
slice . . . . .	21
summarize . . . . .	21
vctrs-vec_chop . . . . .	23
vctrs-vec_recycle . . . . .	24
vctrs-vec_rep . . . . .	24
vctrs_slice . . . . .	26
vectors . . . . .	27
vec_group_id . . . . .	28
vec_phantom . . . . .	29
vec_ptype2 . . . . .	30
vec_ptype_common . . . . .	31

---

plyxp-package	<i>plyxp: Data masks for SummarizedExperiment enabling dplyr-like manipulation</i>
---------------	--

---

### Description

The package provides ‘rlang’ data masks for the SummarizedExperiment class. This enables the evaluation of unquoted expressions in different contexts of the SummarizedExperiment object with optional access to other contexts. The goal for ‘plyxp’ is for evaluation to feel like a data.frame object without ever needing to unwind to a rectangular data.frame.

### Value

API for using S4 classes with rlang data masks

### Author(s)

**Maintainer:** Justin Landis <jtlandis314@gmail.com> ([ORCID](#))

Authors:

- Michael Love <michaelisaiahlove@gmail.com> ([ORCID](#))

### See Also

Useful links:

- <https://github.com/jtlandis/plyxp>
- <https://jtlandis.github.io/plyxp>
- Report bugs at <https://www.github.com/jtlandis/plyxp/issues>

---

arrange	<i>arrange rows or columns of PlySummarizedExperiment</i>
---------	---

---

### Description

arrange() orders either the rows or columns of a PlySummarizedExperiment object. Note, to guarantee a valid PlySummarizedExperiment is returned, arranging in the assays evaluation context is disabled.

Unlike other dplyr verbs, arrange() largely ignores grouping. The PlySummarizedExperiment method also provides the same functionality via the .by\_group argument.

### Usage

```
## S3 method for class 'PlySummarizedExperiment'
arrange(.data, ..., .by_group = FALSE)
```

**Arguments**

`.data` An object Inheriting from `PlySummarizedExperiment`, the wrapper class for `SummarizedExperiment` objects

`...` [<data-masking>](#) Variables, or functions of variables. Use `desc()` to sort a variable in descending order.

`.by_group` If TRUE, will sort first by grouping variable. Applies to grouped data frames only.

**Value**

an object inheriting `PlySummarizedExperiment` class

**Examples**

```
# arrange within rows/cols contexts separately
arrange(
  se_simple,
  rows(direction),
  cols(dplyr::desc(condition))
)

# access assay data to compute arrangement
arrange(
  se_simple,
  rows(rowSums(.assays_asis$counts)),
  cols(colSums(.assays_asis$counts))
)

# assay context is disabled
arrange(se_simple, counts) |> try()

# convert to `data.frame` first
as.data.frame(se_simple) |>
  arrange(counts)
```

---

as.data.frame

*create data.frame*

---

**Description**

create data.frame

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'
as.data.frame(x, ...)
```

**Arguments**

x SummarizedExperiment object  
 ... unused arguments

**Value**

a data.frame object

**Examples**

```
as.data.frame(se_simple)
```

---

dot-pronouns

*contextual plyxp pronouns*

---

**Description**

plyxp utilizes its own version of `rlang::.data` pronouns. These may be used to gain access to other evaluation contexts for a managed set of data-masks.

Similar to `rlang::.data`, `plyxp::.assays` and other exported pronouns are exported to pass R CMD Checks. When using a `plyxp` within your package, import the associated pronoun from `plyxp` but only use the fully unqualified name, `.assays`, `.assays_asis`, etc.

**Usage**

```
.assays
.assays_asis
.rows
.rows_asis
.cols
.cols_asis
```

**Format**

```
An object of class NULL of length 0.
An object of class NULL of length 0.
An object of class NULL of length 0.
An object of class NULL of length 0.
An object of class NULL of length 0.
An object of class NULL of length 0.
```

**Value**

access to specific values behind the rlang pronoun

**Examples**

```
mutate(
  se_simple,
  # access via pronoun
  rows(sum = rowSums(.assays_asis$counts)),
  cols(sum = vapply(.assays$counts, sum, numeric(1)))
)
```

---

filter

*filter PlySummarizedExperiment*

---

**Description**

The `filter()` function is used to subset an object, returning the observations that satisfy your conditions. An observation must return TRUE for all conditions within a context to be retained. Note, to guarantee a valid `PlySummarizedExperiment` is returned, filtering in the assays evaluation context is disabled.

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'
filter(.data, ..., .preserve = FALSE)
```

**Arguments**

<code>.data</code>	An object Inheriting from <code>PlySummarizedExperiment</code> , the wrapper class for <code>SummarizedExperiment</code> objects
<code>...</code>	conditions to filter on. These must be wrapped in <code>cols()</code> and or <code>rows()</code>
<code>.preserve</code>	Relevant when the <code>.data</code> input is grouped. If <code>.preserve = FALSE</code> (the default), the grouping structure is recalculated based on the resulting data, i.e. the number of groups may change.

**Value**

an object inheriting `PlySummarizedExperiment` class

**Examples**

```

# example code
filter(
  se_simple,
  rows(length > 30),
  cols(condition == "drug")
)

filter(
  se_simple,
  rows(rowSums(.assays_asis$counts) > 40),
  cols(colSums(.assays_asis$counts) < 50)
)

# assay context is disabled
filter(
  se_simple,
  counts > 12
) |> try()

# convert to `data.frame` first
as.data.frame(se_simple) |>
  filter(counts > 12)

```

---

group\_by

*apply groups to PlySummarizedExperiment*


---

**Description**

create grouping variables about the rowData and colData of a PlySummarizedExperiment object. Unlike the data.frame method the resulting output class is left unchanged. Thus dplyr generics for PlySummarizedExperiment must check grouping information manually.

**Usage**

```

## S3 method for class 'PlySummarizedExperiment'
group_by(.data, ..., .add = FALSE)

## S3 method for class 'PlySummarizedExperiment'
ungroup(x, ...)

```

**Arguments**

`.data` An object Inheriting from PlySummarizedExperiment, the wrapper class for SummarizedExperiment objects

**S4 Compatibility:**

At the moment, grouping on S4 Vectors is not yet supported. This is due to plyxp using [vec\_group\_loc][vctrs::vec\_group\_loc] to form grouping information. plyxp will eventually develop a method to handle S4 Vectors.

... **contextual expressions** specifying which columns to ungroup. Omitting ... ungroups the entire object.

.add When FALSE, the default, group\_by() will override existing groups.

x An object Inheriting from PlySummarizedExperiment, the wrapper class for SummarizedExperiment objects

**Value**

PlySummarizedExperiment object

**Functions**

- ungroup(PlySummarizedExperiment): Ungroup a PlySummarizedExperiment object

**Examples**

```
group_by(se_simple, rows(direction), cols(condition))
```

---

group\_data

*get grouping data*

---

**Description**

retrieve grouping information from a SummarizedExperiment object. This is stored within the metadata() of the object.

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'
group_data(.data)
```

**Arguments**

.data An object Inheriting from PlySummarizedExperiment, the wrapper class for SummarizedExperiment objects

**Value**

list of groupings for an SummarizedExperiment

**Examples**

```
group_by(se_simple, rows(direction), cols(condition)) |> group_data()
```

---

group_split	<i>Split a PlySummarizedExperiment based on groups</i>
-------------	--

---

**Description**

Splits a grouped PlySummarizedExperiment based on groups. Note the elements of the return value are ungrouped PlySummarizedExperiment objects.

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'  
group_split(.tbl, ..., .keep = TRUE)
```

**Arguments**

.tbl	a PlySummarizedExperiment object
...	ignored if the .tbl is grouped, otherwise it is passed to <a href="#">group_by</a> .
.keep	logical indicating of grouping variables should be kept

**Value**

A list of PlySummarizedExperiment objects

**Examples**

```
gse <- group_by(se_simple, rows(direction), cols(condition))  
gse |> group_split()  
gse |> group_split(.keep = FALSE)
```

---

group_vars	<i>get PlySummarizedExperiment grouping Variables</i>
------------	---

---

**Description**

like in [dplyr::group\\_vars\(\)](#) will get character strings for groupings with the exception of the return value being a list for each grouped context

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'  
group_vars(x)
```

**Arguments**

x	PlySummarizedExperiment
---	-------------------------

**Value**

NULL or list containing names of grouping columns

**Examples**

```
out <- group_by(se_simple, rows(direction))
group_vars(out)
```

---

list_unchop	<i>unchop a list of objects</i>
-------------	---------------------------------

---

**Description**

A generic version of `vctrs::list_unchop` meant to support S4 Vectors.

**Usage**

```
list_unchop(x, ptype = NULL, ..., indices = NULL)
```

**Arguments**

x	a list
ptype	the expected prototype of the output
...	unused arguments
indices	optional list of integer vectors whose size is equal to that of x. This maps the the final index of each element in the output.

**Value**

an object of type ptype or the common ptype of elements of x.

---

mutate	<i>Mutate a PlySummarizedExperiment object</i>
--------	--

---

**Description**

Mutate a `PlySummarizedExperiment` object under an data mask. Unlike a few other `dplyr` implementations, all contextual evaluations of `mutate()` for `SummarizedExperiment` are valid.

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'
mutate(.data, ...)
```

**Arguments**

.data            An object Inheriting from PlySummarizedExperiment, the wrapper class for SummarizedExperiment objects

...              expressions to evaluate

**Value**

an object inheriting PlySummarizedExperiment class

**Examples**

```
mutate(se_simple,
  counts_1 = counts + 1,
  logp_counts = log(counts_1),
  # access assays context with ".assays" pronoun,
  # note that assays are sliced into a list to
  # fit dimensions of cols context
  cols(sum = purrr::map_dbl(.assays$counts, sum)),
  # access assays context "asis" with the same pronoun
  # but with a "_asis" suffix.
  rows(sum = rowSums(.assays_asis$counts))
)
```

---

new\_plyxp

*SummarizedExperiment Shell Object*

---

**Description**

A container object for the SummarizedExperiment class.

This S4 class is implemented to bring unique dplyr syntax to the SummarizedExperiment object without clashing with the tidySummarizedExperiment package. As such, this is a simple wrapper that contains one slot, which holds a SummarizedExperiment object.

**Usage**

```
new_plyxp(se)
```

```
PlySummarizedExperiment(se)
```

**Arguments**

se              SummarizedExperiment object

**Value**

PlySummarizedExperiment object

**Slots**

se contains the underlying SummarizedExperiment class.

**Examples**

```
se <- SummarizedExperiment(
  assays = list(counts = matrix(1:6, nrow = 3)),
  colData = S4Vectors::DataFrame(condition = c("A", "B"))
)
new_plyxp(se = se)
# or
PlySummarizedExperiment(se = se)
```

---

plyxp

*Modify SummarizedExperiment Object*

---

**Description**

Modify the underlying SummarizedExperiment object with a function.

**Usage**

```
plyxp(.data, .f, ..., .caller = caller_env())
```

```
plyxp_on(.data, .f, ..., .on, .caller = caller_env())
```

**Arguments**

.data	a PlySummarizedExperiment object
.f	within plyxp(): a function that returns a SummarizedExperiment object. within plyxp_on(): .f should return a value compatible with .on(se)<-
...	additional arguments passed to .f
.caller	environment in which plyxp should signal an error if one occurs.
.on	a symbol matching an accessor and setter function for the SummarizedExperiment Class.

**Value**

a PlySummarizedExperiment object

## Functions

- `plyxp_on()`: pass a function to the result of an accessor of the SummarizedExperiment Class. This function is a wrapper for the expression:

```
plyxp::plyxp(.data, function(se, ...) {
  .f <- rlang::as_function(.f)
  obj <- .on(se)
  obj <- .f(se, ...)
  .on(se) <- obj
  se
}, ...)
```

where `.on` is the symbol for the accessor function into a SummarizedExperiment Class. Note: the setter variant must exist in the environment that `plyxp_on()` is called. All other arguments are diffused as quosures and will be evaluated in the environment they were quoted.

## Examples

```
plyxp(se_simple, function(x) x)
plyxp_on(se_simple,
  .f = lapply, # function to call on `.on` args,
  .on = rowData, # data `.f` will be used on
  paste, "foo"
) # arguments for `.f`
```

---

plyxp-context

*plyxp contexts*

---

## Description

Contextual user-facing helper function for dplyr verbs with SummarizedExperiment objects. These functions are intended to be used as the top level call to any dplyr verbs ... argument, similar to that of `across()/if_any()/if_all()`.

Specifies that the following expressions should be evaluated within the `colData` context.

Specifies that the following expressions should be evaluated within the `rowData` context.

Specify a single expression to evaluate in another context

Specify a single expression to evaluate in another context

Specify a single expression to evaluate in another context

## Usage

```
cols(...)
```

```
rows(...)
```

```
col_ctx(x, asis = FALSE)
```

```
row_ctx(x, asis = FALSE)

assay_ctx(x, asis = FALSE)
```

### Arguments

`x, ...` expressions to evaluate within its associated context

`asis` `asis = FALSE` (the default) will indicate using active bindings that attempt to coerce the underlying data into a format that is appropriate for the current context. Indicating `TRUE` will instead bind the underlying data as is.

### Value

function called for its side-effects

### Examples

```
# cols
mutate(se_simple,
  cols(is_drug = condition == "drug"),
  # bind a different context
  effect = col_ctx(counts + (is_drug * rbinom(n(), 20, .3)))
)
```

---

<code>pull</code>	<i>extract data from object</i>
-------------------	---------------------------------

---

### Description

similar to `dplyr::pull.data.frame` except allows to extract objects from different contexts.

### Usage

```
## S3 method for class 'PlySummarizedExperiment'
pull(.data, var = -1, name = NULL, ...)
```

### Arguments

`.data` An object Inheriting from `PlySummarizedExperiment`, the wrapper class for `SummarizedExperiment` objects

`var` A variable as specified by `dplyr::pull`

`name` ignored argument. Due to the range of data types a `PlySummarizedExperiment` this argument is not supported

`...` unused argument

**Value**

an element from either the assays, rowData, or colData of a SummarizedExperiment object

**Examples**

```
# last element of default context (assays)
pull(se_simple, var = -1)
# first element of rows context
pull(se_simple, var = rows(1))
# element from col context by literal variable name
pull(se_simple, var = cols(condition))

# use `pull()` to return contextual info
mutate(se_simple, rows(counts = .assays$counts)) |>
  # get last stored element
  pull(rows(-1))
```

---

reexports

*Objects exported from other packages*


---

**Description**

These objects are imported from other packages. Follow the links below to see their documentation.

**S4Vectors** [metadata](#), [metadata<-](#)

**SummarizedExperiment** [assay](#), [assay<-](#), [assays](#), [assays<-](#), [colData](#), [colData<-](#), [rowData](#), [rowData<-](#), [SummarizedExperiment](#)

**Value**

exported functions available from plyxp

**See Also**

[arrange\(\)](#) [mutate\(\)](#) [filter\(\)](#) [summarize\(\)](#) [select\(\)](#) [pull\(\)](#) [group\\_by\(\)](#) [group\\_data\(\)](#) [group\\_vars\(\)](#)  
[ungroup\(\)](#) [group\\_split\(\)](#)

[PlySummarizedExperiment-methods](#)

**Examples**

```
arrange(se_simple, rows(direction)) |>
  mutate(logp_counts = log1p(counts)) |>
  filter(cols(condition == "drug"))

assays(se_simple)
rowData(se_simple)
colData(se_simple)
```

---

se

*PlySummarizedExperiment Methods*

---

### Description

Methods from SummarizedExperiment package re-implemented for PlySummarizedExperiment.

### Usage

```
se(x)

## S4 method for signature 'PlySummarizedExperiment'
se(x)

se(x) <- value

## S4 replacement method for signature 'PlySummarizedExperiment'
se(x) <- value

## S4 method for signature 'SummarizedExperiment'
se(x)

## S4 replacement method for signature 'SummarizedExperiment'
se(x) <- value

## S4 method for signature 'PlySummarizedExperiment'
assays(x, withDimnames = TRUE, ...)

## S4 replacement method for signature 'PlySummarizedExperiment,list'
assays(x, withDimnames = TRUE, ...) <- value

## S4 replacement method for signature 'PlySummarizedExperiment,SimpleList'
assays(x, withDimnames = TRUE, ...) <- value

## S4 method for signature 'PlySummarizedExperiment,missing'
assay(x, i, withDimnames = TRUE, ...)

## S4 method for signature 'PlySummarizedExperiment,numeric'
assay(x, i, withDimnames = TRUE, ...)

## S4 method for signature 'PlySummarizedExperiment,character'
assay(x, i, withDimnames = TRUE, ...)

## S4 replacement method for signature 'PlySummarizedExperiment,missing'
assay(x, i, withDimnames = TRUE, ...) <- value

## S4 replacement method for signature 'PlySummarizedExperiment,numeric'
```

```
assay(x, i, withDimnames = TRUE, ...) <- value

## S4 replacement method for signature 'PlySummarizedExperiment,character'
assay(x, i, withDimnames = TRUE, ...) <- value

## S4 method for signature 'PlySummarizedExperiment'
rowData(x, use.names = TRUE, ...)

## S4 replacement method for signature 'PlySummarizedExperiment'
rowData(x, ...) <- value

## S4 method for signature 'PlySummarizedExperiment'
colData(x, ...)

## S4 replacement method for signature 'PlySummarizedExperiment,DataFrame'
colData(x, ...) <- value

## S4 replacement method for signature 'PlySummarizedExperiment,NULL'
colData(x, ...) <- value

## S4 method for signature 'PlySummarizedExperiment'
metadata(x, ...)

## S4 replacement method for signature 'PlySummarizedExperiment'
metadata(x, ...) <- value

## S4 method for signature 'PlySummarizedExperiment'
rownames(x)

## S4 method for signature 'PlySummarizedExperiment'
colnames(x)

## S4 method for signature 'PlySummarizedExperiment'
nrow(x)

## S4 method for signature 'PlySummarizedExperiment'
ncol(x)

## S4 method for signature 'PlySummarizedExperiment'
dimnames(x)

## S4 replacement method for signature 'PlySummarizedExperiment,list'
dimnames(x) <- value

## S4 replacement method for signature 'PlySummarizedExperiment,NULL'
dimnames(x) <- value
```

## Arguments

<code>x</code>	PlySummarizedExperiment object
<code>value</code>	replacement value
<code>withDimnames</code>	logical
<code>...</code>	additional arguments
<code>i</code>	character or numeric index
<code>use.names</code>	logical

## Value

Replacement functions return a PlySummarizedExperiment object. Other functions will return the same object as the method from SummarizedExperiment.

## Functions

- `se(PlySummarizedExperiment)`: get the `se` slot of the PlySummarizedExperiment object
- `se(x) <- value`: set the `se` slot of the PlySummarizedExperiment object
- `se(PlySummarizedExperiment) <- value`: set the `se` slot of the PlySummarizedExperiment object
- `se(SummarizedExperiment)`: get the SummarizedExperiment object
- `se(SummarizedExperiment) <- value`: get the SummarizedExperiment object
- `assays(PlySummarizedExperiment)`: get the assays of the PlySummarizedExperiment object
- `assays(x = PlySummarizedExperiment) <- value`: set the assays of the PlySummarizedExperiment object
- `assays(x = PlySummarizedExperiment) <- value`: set the assays of the PlySummarizedExperiment object
- `assay(x = PlySummarizedExperiment, i = missing)`: get the first assay of the PlySummarizedExperiment object
- `assay(x = PlySummarizedExperiment, i = numeric)`: get assay from a PlySummarizedExperiment object
- `assay(x = PlySummarizedExperiment, i = character)`: get assay from a PlySummarizedExperiment object
- `assay(x = PlySummarizedExperiment, i = missing) <- value`: set assay in a PlySummarizedExperiment object
- `assay(x = PlySummarizedExperiment, i = numeric) <- value`: set assay in a PlySummarizedExperiment object
- `assay(x = PlySummarizedExperiment, i = character) <- value`: set assay in a PlySummarizedExperiment object
- `rowData(PlySummarizedExperiment)`: get `rowData` in a PlySummarizedExperiment object
- `rowData(PlySummarizedExperiment) <- value`: set `rowData` in a PlySummarizedExperiment object
- `colData(PlySummarizedExperiment)`: get `colData` in a PlySummarizedExperiment object
- `colData(x = PlySummarizedExperiment) <- value`: set `colData` in a PlySummarizedExperiment object

**Examples**

```
assays(se_simple)
rowData(se_simple)
colData(se_simple)
```

---

select	<i>select assays, rowData, and colData names</i>
--------	--

---

**Description**

Select one or more values from each context. By default omitting an expression for a context is the same as selecting NOTHING from that context.

The `<tidy-select>` implementation within `plyxp` is almost similar to `dplyr` except when used within the `across()` function. When used from `across()`, the data provided to `eval_select` is a zero length slice of the data. This was an intentional choice to prevent the evaluation of potentially expensive chopping operations for `S4Vectors`. This means that predicate function from `where()` will NOT be able to query the original data.

**Usage**

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

**Arguments**

<code>.data</code>	An object Inheriting from <code>PlySummarizedExperiment</code> , the wrapper class for <code>SummarizedExperiment</code> objects
<code>...</code>	<code>&lt;tidy-select&gt;</code> one or more selection expressions. Supports wrapping expressions within the <code>&lt;plyxp-contexts&gt;</code> .

**Value**

an object inheriting `PlySummarizedExperiment` class

**Examples**

```
# only keep assays, other contexts are dropped
select(se_simple, everything())

# only keep rowData, other contexts are dropped
select(se_simple, rows(everything()))

select(se_simple, rows(where(is.numeric)))

# Note on `where()` clause, all data is available within select
select(se_simple, rows(where(~ any(grepl("-", .x)))))
```

```

# within an `across()`, only a zero-length slice available, so the
# `where()` predicate cannot access the data
mutate(
  se_simple,
  rows(
    across(
      where(~ any(grepl("-", .x))),
      ~ sprintf("%s foo", .x)
    )
  )
)
# here is an acceptable usage of the `where()` predicate
mutate(
  se_simple,
  rows(
    across(
      where(is.character),
      ~ sprintf("%s foo", .x)
    )
  )
)

```

---

se\_simple

*Plyxp Simple Example Summarized Experiment*


---

## Description

A small data SummarizedExperiment Object of 20 observations, 5 rows and 4 columns.

## Usage

```
se_simple
```

## Format

```

se_simple:
assays counts sampled data points between 1:20
      logcounts log transform of counts
rowData/.features gene fake gene name
      length fake gene length
      direction fake strand
colData/.samples sample fake sample name
      condition control or drug treatment

```

## Value

a SummarizedExperiment object

**Examples**

```
SummarizedExperiment::assays(se_simple)
SummarizedExperiment::rowData(se_simple)
SummarizedExperiment::colData(se_simple)
```

---

slice	<i>Slice a PlySummarizedExperiment</i>
-------	--

---

**Description**

slice() selects rows and/or columns by position.

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'
slice(.data, ..., .preserve = FALSE)
```

**Arguments**

.data	a PlySummarizedExperiment object
...	expressions that resolve to integer values to slice, .data by. Note that only the rows() and cols() contexts are available here.
.preserve	Logical value. When FALSE, the default, group combinations are recomputed based on the resulting data. When TRUE, group combinations are retained, despite any groups being empty.

**Examples**

```
gse <- group_by(se_simple, rows(direction), cols(condition))
sgse <- slice(gse, rows(which(direction == "-")))
group_data(gse)
group_data(sgse)
sgse2 <- slice(gse, rows(which(direction == "-")), .preserve = TRUE)
group_data(sgse2)
```

---

summarize	<i>Summarize PlySummarizedExperiment</i>
-----------	--

---

**Description**

Summarize PlySummarizedExperiment

**Usage**

```
## S3 method for class 'PlySummarizedExperiment'
summarize(.data, ..., .retain = c("auto", "ungrouped", "none"))

## S3 method for class 'PlySummarizedExperiment'
summarise(.data, ..., .retain = c("auto", "ungrouped", "none"))
```

**Arguments**

<code>.data</code>	An object Inheriting from <code>PlySummarizedExperiment</code> , the wrapper class for <code>SummarizedExperiment</code> objects
<code>...</code>	expressions to summarize the object
<code>.retain</code>	This argument controls how <code>rowData()</code> or <code>colData()</code> is retained after summarizing. When "auto" (the default), <code>.retain</code> behavior depends on the groupings of <code>.data</code> . When exactly one dimension is grouped, "auto" behaves like "ungrouped-dim", and "none" otherwise. When "ungrouped-dim", the ungrouped dimension's data are retained in the resulting <code>SummarizedExperiment</code> object and scalar outputs are recycled to the length of the ungrouped dimension. When "none", all outputs are expected to be scalar and only computed values are retained in <code>rowData()</code> and <code>colData()</code>

**Value**

an object inheriting `PlySummarizedExperiment` class

**Examples**

```
# outputs in assay context may be either
# length 1, or the length of the ungrouped
# dimension while .retain = "auto"/"ungrouped-dim"
se_simple |>
  group_by(rows(direction)) |>
  summarise(
    col_sums = colSums(counts),
    sample = sample(1:20, 1L)
  )

# .retain = "none" will drop ungrouped dimensions and
# outputs of assay context should be length 1.
se_simple |>
  group_by(rows(direction)) |>
  summarise(
    col_sums = list(colSums(counts)),
    .retain = "none"
  )

# using an `across()` function will help
# nest ungrouped dimensions
se_simple |>
  group_by(rows(direction)) |>
```

```

summarise(
  col_sums = list(colSums(counts)),
  cols(across(everything(), list)),
  .retain = "none"
)

```

---

vctrs-vec\_chop      *chop a vector*

---

## Description

A re-export of `vctrs::vec_chop` as an S7 generic function to allow S4Vectors.

## Usage

```
vec_chop(x, ..., indices = NULL)
```

## Arguments

<code>x</code>	A vector
<code>...</code>	These dots are for future extensions and must be empty.
<code>indices</code>	For <code>vec_chop()</code> , a list of positive integer vectors to slice <code>x</code> with, or <code>NULL</code> . Can't be used if <code>sizes</code> is already specified. If both <code>indices</code> and <code>sizes</code> are <code>NULL</code> , <code>x</code> is split into its individual elements, equivalent to using an <code>indices</code> of <code>as.list(vec_seq_along(x))</code> . For <code>list_unchop()</code> , a list of positive integer vectors specifying the locations to place elements of <code>x</code> in. Each element of <code>x</code> is recycled to the size of the corresponding index vector. The size of <code>indices</code> must match the size of <code>x</code> . If <code>NULL</code> , <code>x</code> is combined in the order it is provided in, which is equivalent to using <code>vec_c()</code> .

## Value

a S3 or S4 vector

## Examples

```

vec_chop(1L)
vec_chop(S4Vectors::Rle(c(rep(1, 3), rep(4, 5))), indices = list(c(2, 3, 4), c(1, 5:8)))

```

---

vctrs-vec\_recycle      *Recycle a vector*

---

### Description

A re-export of `vctrs::vec_recycle` as an *S7* generic function to allow *S4*Vectors.

### Usage

```
vec_recycle(x, size, ..., x_arg = "", call = caller_env())
```

### Arguments

<code>x</code>	A vector to recycle.
<code>size</code>	Desired output size.
<code>...</code>	Depending on the function used: <ul style="list-style-type: none"> <li>• For <code>vec_recycle_common()</code>, vectors to recycle.</li> <li>• For <code>vec_recycle()</code>, these dots should be empty.</li> </ul>
<code>x_arg</code>	Argument name for <code>x</code> . These are used in error messages to inform the user about which argument has an incompatible size.
<code>call</code>	The execution environment of a currently running function, e.g. <code>caller_env()</code> . The function will be mentioned in error messages as the source of the error. See the <code>call</code> argument of <code>abort()</code> for more information.

### Value

a *S3* or *S4* vector

### Examples

```
vec_recycle(1L, size = 5L)
vec_recycle(S4Vectors::Rle(1L), size = 5L)
```

---

vctrs-vec\_rep      *replicate a vector*

---

### Description

A re-export of `vctrs::vec_rep` and `vctrs::vec_rep_each` as an *S7* generic function to allow *S4*Vectors.

**Usage**

```
vec_rep(
  x,
  times,
  ...,
  error_call = caller_env(),
  x_arg = "x",
  times_arg = "times"
)

vec_rep_each(
  x,
  times,
  ...,
  error_call = caller_env(),
  x_arg = "x",
  times_arg = "times"
)
```

**Arguments**

<code>x</code>	A vector.
<code>times</code>	For <code>vec_rep()</code> , a single integer for the number of times to repeat the entire vector. For <code>vec_rep_each()</code> , an integer vector of the number of times to repeat each element of <code>x</code> . <code>times</code> will be <a href="#">recycled</a> to the size of <code>x</code> .
<code>...</code>	These dots are for future extensions and must be empty.
<code>error_call</code>	The execution environment of a currently running function, e.g. <code>caller_env()</code> . The function will be mentioned in error messages as the source of the error. See the <code>call</code> argument of <a href="#">abort()</a> for more information.
<code>x_arg, times_arg</code>	Argument names for errors.

**Value**

a new S3 or S4 vector replicated by specified times

**Examples**

```
vec_rep(1:2, times = 5)
vec_rep(S4Vectors::Rle(1:2), times = 5)

vec_rep_each(1:2, times = 5)
vec_rep_each(S4Vectors::Rle(1:2), times = 5)
```

---

vctrs\_slice

*Get observations of a vector*


---

## Description

This extends `vctrs::vec_slice` to `S4Vectors::Vector` class by masking `vec_slice` with `S7::new_generic`. Atomic vectors and other base S3 classes (`list`, `data.frame`, `factor`, `Dat`, `POSIXct`) will dispatch to the `vctrs::vec_slice` method as normal. Dispatch support on the `S4Vectors::Vector` and `S4Vectors::DataFrame` classes provides a unified framework for working with base R vectors and `S4Vectors`.

### `S4Vectors::Vector` Implementation:

This method will naively call the `[]` method for any S4 class that inherits from the `S4Vectors::Vector` class. This may not be a very efficient way to slice up an S4 class, but will work.

With this implementation, the `x@mc01` data is expected to be retained after a call to `plyxp::vec_slice(x, i)`.

### `S4Vectors::DataFrame` Implementation:

The `DataFrame` implementation works similar to how `vctrs::vec_slice` works on a `data.frame` object. What is being sliced is the rows of `x@listData`. To maintain the size stability of the `DataFrame` object, we change `@nrows` to the appropriate value, and perform a recursive call if `@elementMetadata` is not `NULL`.

### Performance:

Depending on the size and complexity of your S4 Vector object, you may find the standard subset operation is extremely slow. For example, consider a `SummarizedExperiment` whose `rowData` contains a `CompressedGRangesList` object assigned to the name "exons" and whose length is 250,000 and underlying `@unlistData` is length 1,600,000. Performing a `by .features` grouping operation and attempting to evaluate the exons within the row context would force the `CompressedGRangesList` object to be chopped element-wise.

Unfortunately, there is a massive performance hit in attempting to construct 250,000 `GRanges`. Unless you do not mind waiting over an hour for each `dplyr` verb in which exons gets evaluated, doing so is not recommended.

The `plyxp` package is planning to export a new generic named `plyxp_s4_proxy_vec()`. This attempts to reconstruct certain standard `S4Vectors::Vectors` as standard vectors or tibbles. The equivalent exons object would require much more memory use, but at the advantage of only taking several seconds to construct. When you are done, you can attempt to restore the original S4 Vector with `plyxp_restore_s4_proxy()`.

In development, `plyxp_s4_proxy_vec()` is faster to work with because there are less checks on the object validity and all `@elementMetadata` and `@metadata` are dropped from the objects.

## Usage

```
vec_slice(x, i, ...)
```

**Arguments**

`x` A vector

`i` An integer, character or logical vector specifying the locations or names of the observations to get/set. Specify TRUE to index all elements (as in `x[[]]`), or NULL, FALSE or `integer()` to index none (as in `x[NULL]`).

... These dots are for future extensions and must be empty.

**Value**

a new S3 or S4 vector subsetted by `i`

**Examples**

```
vec_slice(1:10, i = 5)
vec_slice(S4Vectors::Rle(rep(1:3, each = 3)), i = 5)
```

---

vectors

*S7 classes for vctrs and S4 Vectors*


---

**Description**

A set of S7 classes and Class unions that help establish S7 method dispatch. These classes were made to re-export several vctrs functions such that internals for plyxp were consistent with room for optimization.

**Usage**

```
class_vctrs
class_s4_vctrs
class_DF
```

**Format**

An object of class `S7_union` of length 1.

An object of class `classRepresentation` of length 1.

An object of class `classRepresentation` of length 1.

**Value**

S7 class union or base class

**See Also**

[vec\\_rep\(\)](#), [vec\\_recycle\(\)](#), [vec\\_slice\(\)](#)

## Examples

```
# used for defining methods on S7 generics

S7::method(vec_slice, class_s4_vctrs)
```

---

vec_group_id	<i>Create Group Indices</i>
--------------	-----------------------------

---

## Description

A alternative to `vctrs::vec_group_id` and `vctrs::vec_group_loc` as an S4 generic function to allow S4Vectors.

## Usage

```
vec_group_id(x, ...)

vec_group_loc(x, ...)
```

## Arguments

x	a vctrs vector or an S4Vector
...	unsued arguments

## Value

either a tibble or DataFrame object

## Examples

```
data <- S4Vectors::DataFrame(
  letter = sample(letters, 500, TRUE),
  LETTER = sample(letters, 500, TRUE)
)

vec_group_id(data)
vec_group_loc(data)
```

## Description

plyxp uses [pillar](#) for its printing. If you want to change how your S4 object is printed within plyxp's print method, consider writing a method for this function.

To print S4 objects in a tibble, plyxp hacks a custom integer vector built from [vctrs](#) where the S4 object lives in an attribute named "phantomData". You can create your own S4 phantom vector with `vec_phantom()`. This function is not used outside of printing for plyxp

The default method for formatting a `vec_phantom()` is to call [showAsCell\(\)](#).

## Usage

```
vec_phantom(x)
```

```
plyxp_pillar_format(x, ...)
```

```
show_tidy(x, ...)
```

```
use_show_tidy()
```

```
use_show_default()
```

## Arguments

x	The S4 object
...	other arguments passed from <a href="#">pillar_shaft</a>

## Value

`plyxp_pillar_format` -> formatted version of your S4 vector `vec_phantom` -> integer vector with arbitrary object in `phantomData` attribute.

## tidy printing

By default, plyxp will not affect the show method for `SummarizedExperiment` objects. The `PlySummarizedExperiment` object will always use the tibble abstraction method. If you want to use tibble abstraction, you may use `use_show_tidy()` to enable or `use_show_default()` #' to disable this feature. These functions are called for their side effects, #' modifying the global option "show\_SummarizedExperiment\_as\_tibble\_abstraction".

To show an object as the tibble abstraction regardless of the set option, use the S3 generic `show_tidy(...)`.

**Examples**

```

if (require("IRanges")) {
  ildist <- IRanges::IntegerList(list(c(1L, 2L, 3L), c(5L, 6L)))
  phantom <- vec_phantom(ildist)
  pillar::pillar_shaft(phantom)

  plyxp_pillar_format.CompressedIntegerList <- function(x) {
    sprintf("Int: [%i]", lengths(x))
  }
  print(pillar::pillar_shaft(phantom))
  rm(plyxp_pillar_format.CompressedIntegerList)
}

# default printing for PlySummarizedExperiment object
se_simple
# default printing for SummarizedExperiment object
se <- se(se_simple)
se
# use `plyxp` tibble abstraction
use_show_tidy()
se
# restore default print
use_show_default()
se
# explicitly using tibble abstraction
show_tidy(se)

```

---

vec\_ptype2

*Create a common prototype of two vectors*


---

**Description**

given two objects, `vec_ptype2()` finds a common prototype

**Usage**

```
vec_ptype2(x, y, ...)
```

**Arguments**

x	first object
y	second object
...	unused arguments

**Value**

an object of size 0.

---

vec_ptype_common	<i>find the common ptype</i>
------------------	------------------------------

---

**Description**

find the common ptype

**Usage**

```
vec_ptype_common(..., .ptype = NULL)
```

**Arguments**

...	<dynamic-dots> a collection of objects
.ptype	the expected prototype

**Value**

an object of size 0.

# Index

- \* **datasets**
  - dot-pronouns, [5](#)
  - se\_simple, [20](#)
  - vectors, [27](#)
- \* **internal**
  - plyxp-package, [3](#)
  - reexports, [15](#)
  - .assays (dot-pronouns), [5](#)
  - .assays\_asis (dot-pronouns), [5](#)
  - .cols (dot-pronouns), [5](#)
  - .cols\_asis (dot-pronouns), [5](#)
  - .rows (dot-pronouns), [5](#)
  - .rows\_asis (dot-pronouns), [5](#)
- abort(), [24](#), [25](#)
- arrange, [3](#)
- arrange(), [15](#)
- as.data.frame, [4](#)
- assay, [15](#)
- assay (reexports), [15](#)
- assay, PlySummarizedExperiment, character-method  
(se), [16](#)
- assay, PlySummarizedExperiment, missing-method  
(se), [16](#)
- assay, PlySummarizedExperiment, numeric-method  
(se), [16](#)
- assay<- (reexports), [15](#)
- assay<-, PlySummarizedExperiment, character-method  
(se), [16](#)
- assay<-, PlySummarizedExperiment, missing-method  
(se), [16](#)
- assay<-, PlySummarizedExperiment, numeric-method  
(se), [16](#)
- assay\_ctx (plyxp-context), [13](#)
- assays, [15](#)
- assays (reexports), [15](#)
- assays, PlySummarizedExperiment-method  
(se), [16](#)
- assays<- (reexports), [15](#)
- assays<-, PlySummarizedExperiment, list-method  
(se), [16](#)
- assays<-, PlySummarizedExperiment, SimpleList-method  
(se), [16](#)
- class\_DF (vectors), [27](#)
- class\_s4\_vctrs (vectors), [27](#)
- class\_vctrs (vectors), [27](#)
- col\_ctx (plyxp-context), [13](#)
- colData, [15](#)
- colData (reexports), [15](#)
- colData, PlySummarizedExperiment-method  
(se), [16](#)
- colData<- (reexports), [15](#)
- colData<-, PlySummarizedExperiment, DataFrame-method  
(se), [16](#)
- colData<-, PlySummarizedExperiment, NULL-method  
(se), [16](#)
- colnames, PlySummarizedExperiment-method  
(se), [16](#)
- cols (plyxp-context), [13](#)
- contextual expressions, [8](#)
- desc(), [4](#)
- dimnames, PlySummarizedExperiment-method  
(se), [16](#)
- dimnames<-, PlySummarizedExperiment, list-method  
(se), [16](#)
- dimnames<-, PlySummarizedExperiment, NULL-method  
(se), [16](#)
- dot-pronouns, [5](#)
- dplyr::group\_vars(), [9](#)
- dplyr::pull, [14](#)
- eval\_select, [19](#)
- filter, [6](#)
- filter(), [15](#)
- group\_by, [7](#), [9](#)
- group\_by(), [15](#)

- group\_data, 8
- group\_data(), 15
- group\_split, 9
- group\_split(), 15
- group\_vars, 9
- group\_vars(), 15
- list\_unchop, 10
- metadata, 15
- metadata (reexports), 15
- metadata, PlySummarizedExperiment-method (se), 16
- metadata<- (reexports), 15
- metadata<-, PlySummarizedExperiment-method (se), 16
- mutate, 10
- mutate(), 15
- ncol, PlySummarizedExperiment-method (se), 16
- new\_plyxp, 11
- nrow, PlySummarizedExperiment-method (se), 16
- pillar, 29
- pillar\_shaft, 29
- PlySummarizedExperiment (new\_plyxp), 11
- PlySummarizedExperiment-class (new\_plyxp), 11
- PlySummarizedExperiment-methods, 15
- PlySummarizedExperiment-methods (se), 16
- plyxp, 12
- plyxp-context, 13
- plyxp-package, 3
- plyxp-printing (vec\_phantom), 29
- plyxp\_on (plyxp), 12
- plyxp\_pillar\_format (vec\_phantom), 29
- pull, 14
- pull(), 15
- recycled, 25
- reexports, 15
- row\_ctx (plyxp-context), 13
- rowData, 15
- rowData (reexports), 15
- rowData, PlySummarizedExperiment-method (se), 16
- rowData<- (reexports), 15
- rowData<-, PlySummarizedExperiment-method (se), 16
- rownames, PlySummarizedExperiment-method (se), 16
- rows (plyxp-context), 13
- se, 16
- se, PlySummarizedExperiment-method (se), 16
- se, SummarizedExperiment-method (se), 16
- se<- (se), 16
- se<-, PlySummarizedExperiment-method (se), 16
- se<-, SummarizedExperiment-method (se), 16
- se\_simple, 20
- select, 19
- select(), 15
- show\_tidy (vec\_phantom), 29
- showAsCell(), 29
- slice, 21
- summarise (summarize), 21
- summarize, 21
- summarize(), 15
- SummarizedExperiment, 15
- SummarizedExperiment (reexports), 15
- ungroup (group\_data), 8
- ungroup(), 15
- ungroup.PlySummarizedExperiment (group\_by), 7
- use\_show\_default (vec\_phantom), 29
- use\_show\_tidy (vec\_phantom), 29
- vctrs, 29
- vctrs-vec\_chop, 23
- vctrs-vec\_recycle, 24
- vctrs-vec\_rep, 24
- vctrs::list\_unchop, 10
- vctrs::vec\_chop, 23
- vctrs::vec\_group\_id, 28
- vctrs::vec\_group\_loc, 28
- vctrs::vec\_recycle, 24
- vctrs::vec\_rep, 24
- vctrs::vec\_rep\_each, 24
- vctrs\_slice, 26
- vec\_c(), 23
- vec\_chop (vctrs-vec\_chop), 23
- vec\_group\_id, 28

`vec_group_loc` (`vec_group_id`), 28  
`vec_phantom`, 29  
`vec_ptype2`, 30  
`vec_ptype_common`, 31  
`vec_recycle` (`vctrs-vec_recycle`), 24  
`vec_recycle()`, 27  
`vec_rep` (`vctrs-vec_rep`), 24  
`vec_rep()`, 27  
`vec_rep_each` (`vctrs-vec_rep`), 24  
`vec_slice` (`vctrs_slice`), 26  
`vec_slice()`, 27  
vectors, 27  
`where()`, 19