Package 'plyxp'

July 10, 2025

Title Data masks for SummarizedExperiment enabling dplyr-like manipulation

Version 1.2.0

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 7.3.2

VignetteBuilder knitr

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

Depends R (>= 4.4.0)

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

git_url https://git.bioconductor.org/packages/plyxp

git_branch RELEASE_3_21

git_last_commit 054beec

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-07-09

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` *plyxp: Data masks for SummarizedExperiment enabling dplyr-like manipulation*

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.

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 *arrange rows or columns of PlySummarizedExperiment*

Description

`arrange()` orders either the rows or columns of a `PlySummarizedExperiment` object. Note, to guarentee 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

`.data`	An object Inheriting from `PlySummarizedExperiment`, the wrapper class for `SummarizedExperiment` objects
`...`	conditions to filter on. These must be wrapped in `cols()` and or `rows()`
`.preserve`	Relevant when the .data input is grouped. If .preserve = FALSE (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_vars *get PlySummarizedExperiment grouping Variables*

Description

like in `dplyr::group_vars()` will get character strings for groupings with the exection 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)
```

`mutate` *Mutate a PlySummarizedExperiment object*

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))))
```

---

pull *extract data from object*

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 SummarizedExperiment, assay, assay<-, assays, assays<-, colData, colData<-, rowData, rowData<-

Value

exported functions available from plyxp

See Also

arrange() mutate() filter() summarize() select() pull() group_by() group_data() group_vars() ungroup()

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 '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
```

Arguments

x	PlySummarizedExperiment object
value	replacement value
withDimnames	logical
...	additional arguments
i	character or numeric index
use.names	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
- `assays(PlySummarizedExperiment)`: get the assays o 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` *select assays, rowData, and colData names*

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 `accross()`, the data provided to eval_select is a zero length slice of the data. This was an intentional choice to prevent the evaluation of potentionally 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

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

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 avialble, 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)
```

---

summarize *Summarize PlySummarizedExperiment*

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

.data	An object Inheriting from PlySummarizedExperiment, the wrapper class for SummarizedExperiment objects
...	expressions to summarize the object
.retain	This argument controls how rowData() or colData() is retained after summarizing. When "auto" (the default), .retain behavior depends on the groupings of .data. 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 SummarizedExperiment 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 rowData() and colData()

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_recycle` *Recycle a vector*

Description

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

Usage

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

Arguments

`x`	A vector to recycle.
`size`	Desired output size.
`...`	Depending on the function used: • For `vec_recycle_common()`, vectors to recycle.

- For `vec_recycle()`, these dots should be empty.

`x_arg` Argument name for `x`. These are used in error messages to inform the user about which argument has an incompatible size.

`call` The execution environment of a currently running function, e.g. `caller_env()`. The function will be mentioned in error messages as the source of the error. See the `call` argument of `abort()` 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 `S4Vectors`.

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

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

`vec_phantom` *Printing within tibble with S4 objects*

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 pillar_shaft

Value

plyxp_pillar_format -> formatted version of your S4 vector vec_phantom -> integer vector with arbitrary object in phatomData 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")) {
  ilist <- IRanges::IntegerList(list(c(1L,2L,3L),c(5L,6L)))
  phantom <- vec_phantom(ilist)
  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)
```

Index