

Package ‘param6’

February 15, 2022

Title A Fast and Lightweight R6 Parameter Interface

Description By making use of 'set6', alongside the S3 and R6 paradigms, this package provides a fast and lightweight R6 interface for parameters and parameter sets.

Version 0.2.4

License MIT + file LICENSE

URL <https://xoops.github.io/param6/>, <https://github.com/xoopsR/param6/>

BugReports <https://github.com/xoopsR/param6/issues>

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.1.1

Imports checkmate, data.table, dictionar6 (>= 0.1.2), set6 (>= 0.2.3), R6

Suggests testthat

Author Raphael Sonabend [aut, cre] (<<https://orcid.org/0000-0001-9225-4654>>)

Maintainer Raphael Sonabend <raphaelsonabend@gmail.com>

Repository CRAN

Date/Publication 2022-02-15 10:00:09 UTC

R topics documented:

param6-package	2
as.data.table.ParameterSet	2
as.ParameterSet	3
as.prm	3
c.ParameterSet	4
cnd	5
cpset	6
expect_equal_ps	7
length.ParameterSet	7

ParameterSet	7
prm	16
pset	18
rep.ParameterSet	19
support_dictionary	20
[.ParameterSet	20

Index	21
--------------	-----------

param6-package	<i>param6: A Fast and Lightweight R6 Parameter Interface</i>
----------------	--

Description

By making use of 'set6', alongside the S3 and R6 paradigms, this package provides a fast and lightweight R6 interface for parameters and parameter sets.

Author(s)

Maintainer: Raphael Sonabend <raphaelsonabend@gmail.com> ([ORCID](#))

See Also

Useful links:

- <https://xoopsR.github.io/param6/>
- <https://github.com/xoopsR/param6/>
- Report bugs at <https://github.com/xoopsR/param6/issues>

as.data.table.ParameterSet	<i>Coerce a ParameterSet to a data.table</i>
----------------------------	--

Description

Coercion from [ParameterSet](#) to `data.table::data.table`. Dependencies, transformations, and tag properties are all lost in coercion.

Usage

```
## S3 method for class 'ParameterSet'
as.data.table(x, sort = TRUE, ...)
```

Arguments

x	(ParameterSet)
sort	(logical(1)) If TRUE(default) sorts the ParameterSet alphabetically by id.
...	(ANY) Other arguments, currently unused.

as.ParameterSet	<i>Coercions to ParameterSet</i>
-----------------	----------------------------------

Description

Coercions to ParameterSet

Usage

```
as.ParameterSet(x, ...)

## S3 method for class 'data.table'
as.ParameterSet(x, ...)

## S3 method for class 'prm'
as.ParameterSet(x, ...)

## S3 method for class 'list'
as.ParameterSet(x, ...)
```

Arguments

x	(ANY) Object to coerce.
...	(ANY) Other arguments passed to ParameterSet , such as tag_properties.

as.prm	<i>Coercion Methods to prm</i>
--------	--------------------------------

Description

Methods for coercing various objects to a [prm](#).

Usage

```
as.prm(x, ...)
```

```
## S3 method for class 'ParameterSet'
```

```
as.prm(x, ...)
```

```
## S3 method for class 'data.table'
```

```
as.prm(x, ...)
```

Arguments

x	(ANY) Object to coerce.
...	(ANY) Other arguments, currently unused.

c.ParameterSet	<i>Concatenate Unique ParameterSet Objects</i>
----------------	--

Description

Concatenate multiple [ParameterSet](#) objects with unique ids and tags into a single [ParameterSet](#).

Usage

```
## S3 method for class 'ParameterSet'
```

```
c(..., pss = list(...))
```

Arguments

...	(ParameterSets) ParameterSet objects to concatenate.
pss	(list()) Alternatively pass a list of ParameterSet objects.

Details

Concatenates ids, tags, tag properties and dependencies. Assumes ids and tags are unique; trafos are combined into a list.

cnd *Create a ParameterSet Condition*

Description

Function to create a condition for [ParameterSet](#) dependencies for use in the \$deps public method.

Usage

```
cnd(type, value = NULL, id = NULL, error = NULL)
```

Arguments

type	(character(1)) The condition type determines the type of dependency to create, options are given in details.
value	(ANY) If id is NULL then value should be a value in the support of the parameter that the condition is testing, that will be passed to the condition determined by type. Can be left NULL if testing if increasing/decreasing.
id	(character(1)) If value is NULL then id should be the same as the id that the condition is testing, and the condition then takes the currently set value of the id in its argument. Can be left NULL if testing if increasing/decreasing.
error	(character(1)) Optional error message to be displayed on fail.

Details

This function should never be used outside of creating a condition for a dependency in a [ParameterSet](#). Currently the following conditions are supported based on the type argument, we refer to the parameter depended on as in the independent parameter, and the other as the dependent:

- "eq" - If value is not NULL then checks if the independent parameter equals value, otherwise checks if the independent and dependent parameter are equal.
- "neq" - If value is not NULL then checks if the independent parameter does not equal value, otherwise checks if the independent and dependent parameter are not equal.
- "gt"/"lt" - If value is not NULL then checks if the independent parameter is greater/less than value, otherwise checks if the independent parameter is greater/less than the dependent parameter.
- "geq"/"leq" - If value is not NULL then checks if the independent parameter is greater/less than or equal to value, otherwise checks if the independent parameter is greater/less than or equal to the dependent parameter.
- "any" - If value is not NULL then checks if the independent parameter equals any of value, otherwise checks if the independent parameter equals any of dependent parameter.

- "nany" - If value is not NULL then checks if the independent parameter does not equal any of value, otherwise checks if the independent parameter does not equal any of dependent parameter.
- "len" - If value is not NULL then checks if the length of the independent parameter equals value, otherwise checks if the independent and dependent parameter are the same length.
- "inc" - Checks if the parameter is increasing.
- "sinc" - Checks if the parameter is strictly increasing.
- "dec" - Checks if the parameter is decreasing.
- "sdec" - Checks if the parameter is strictly decreasing.

 cpset

Concatenate ParameterSet Objects

Description

Concatenate multiple [ParameterSet](#) objects into a single [ParameterSet](#).

Usage

```
cpset(..., pss = list(...), clone = TRUE)
```

Arguments

...	(ParameterSets) Named ParameterSet objects to concatenate.
pss	(named list()) Alternatively pass a named list of ParameterSet objects.
clone	(logical(1)) If TRUE (default) parameter sets are deep cloned before combination, useful to prevent original sets being prefixed.

Details

Concatenates ids, tags, tag properties and dependencies, but not transformations.

expect_equal_ps	<i>Check if two parameters are equal</i>
-----------------	--

Description

Primarily for internal use

Usage

```
expect_equal_ps(obj, expected)
```

Arguments

obj, expected [ParameterSet](#)

length.ParameterSet	<i>Length of a ParameterSet</i>
---------------------	---------------------------------

Description

Gets the number of parameters in the [ParameterSet](#).

Usage

```
## S3 method for class 'ParameterSet'
length(x)
```

Arguments

x ([ParameterSet](#))

ParameterSet	<i>Parameter Set</i>
--------------	----------------------

Description

ParameterSet objects store parameters ([prm](#) objects) and add internal validation checks and methods for:

- Getting and setting parameter values
- Transforming parameter values
- Providing dependencies of parameters on each other
- Tagging parameters, which may enable further properties
- Storing subsets of parameters under prefixes

Active bindings

tags None -> named_list()
Get tags from the parameter set.

ids None -> character()
Get ids from the parameter set.

length None -> integer(1)
Get the length of the parameter set as the number of parameters.

deps None -> [data.table::data.table](#) Get parameter dependencies, NULL if none.

supports None -> named_list()
Get supports from the parameter set.

tag_properties list() -> self / None -> list()
If x is missing then returns tag properties if any.
If x is not missing then used to tag properties. Currently properties can either be:
i) 'required' - parameters with this tag must have set (non-NULL) values; if a parameter is both 'required' and 'linked' then exactly one parameter in the 'linked' tag must be tagged;
ii) 'linked' - parameters with 'linked' tags are dependent on one another and only one can be set (non-NULL at a time);
iii) 'unique' - parameters with this tag must have no duplicated elements, therefore this tag only makes sense for vector parameters;
iv) 'immutable' - parameters with this tag cannot be updated after construction.

values list() -> self / None -> list()
If x is missing then returns the set (non-NULL) values without transformation or filtering; use `$get_values` for a more sophisticated getter of values.
If x is not missing then used to set values of parameters, which are first checked internally with the `$check` method before setting the new values.
See examples at end.

trafo function()|list() -> self / None -> function()|list()
If x is missing then returns a transformation function if previously set, a list of transformation functions, otherwise NULL.
If x is not missing then it should either be:

- a function with arguments x and self, which internally correspond to self being the ParameterSet the transformation is being added to, and `x <-self$values`.
- a list of functions like above

The transformation function is automatically called after a call to `self$get_values()` and is used to transform set values, it should therefore result in a list. If using `self$get_values()` within the transformation function, make sure to set `transform = FALSE` to prevent infinite recursion, see examples at end.

It is generally safer to call the transformation with `$transform(self$values)` as this will first check to see if `$trafo` is a function or list. If the latter then each function in the list is applied, one after the other.

Methods**Public methods:**

- [ParameterSet\\$new\(\)](#)

- `ParameterSet$print()`
- `ParameterSet$get_values()`
- `ParameterSet$add_dep()`
- `ParameterSet$rep()`
- `ParameterSet$extract()`
- `ParameterSet$remove()`
- `ParameterSet$getParameterValue()`
- `ParameterSet$setParameterValue()`
- `ParameterSet$set_values()`
- `ParameterSet$parameters()`
- `ParameterSet$transform()`
- `ParameterSet$clone()`

Method `new()`: Constructs a `ParameterSet` object.

Usage:

```
ParameterSet$new(prms = list(), tag_properties = NULL)
```

Arguments:

`prms` (`list()`)

List of `prm` objects. Ids should be unique.

`tag_properties` (`list()`)

List of tag properties. Currently supported properties are: i) 'required' - parameters with this tag property must be non-NULL; ii) 'linked' - only one parameter in a linked tag group can be non-NULL and the others should be NULL, this only makes sense with an associated `trafo`; iii) 'unique' - parameters with this tag must have no duplicated elements, only makes sense for vector parameters; iv) 'immutable' - parameters with this tag cannot be updated after construction.

Examples:

```
prms <- list(
  prm("a", Set$new(1), 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
  prm("d", "reals", 2, tags = "t2")
)
ParameterSet$new(prms)
```

Method `print()`: Prints the `ParameterSet` after coercion with [as.data.table.ParameterSet](#).

Usage:

```
ParameterSet$print(sort = TRUE)
```

Arguments:

`sort` (`logical(1)`)

If `TRUE` (default) sorts the `ParameterSet` alphabetically by id.

Examples:

```
prms <- list(
  prm("a", Set$new(1), 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
```

```

  prm("d", "reals", 2, tags = "t2")
)
p <- ParameterSet$new(prms)
p$print()
print(p)
p

```

Method `get_values()`: Gets values from the `ParameterSet` with options to filter by specific IDs and tags, and also to transform the values.

Usage:

```

ParameterSet$get_values(
  id = NULL,
  tags = NULL,
  transform = TRUE,
  inc_null = TRUE,
  simplify = TRUE
)

```

Arguments:

`id` (`character()`)

If not `NULL` then returns values for given ids.

`tags` (`character()`)

If not `NULL` then returns values for given tags.

`transform` (`logical(1)`)

If `TRUE` (default) and `$trafo` is not `NULL` then runs the set transformation function before returning the values.

`inc_null` (`logical(1)`)

If `TRUE` (default) then returns values for all ids even if `NULL`.

`simplify` (`logical(1)`)

If `TRUE` (default) then unlists scalar values, otherwise always returns a list.

Examples:

```

prms <- list(
  prm("a", "reals", 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
  prm("d", "reals", tags = "t2")
)
p <- ParameterSet$new(prms)
p$trafo <- function(x, self) {
  x$a <- exp(x$a)
  x
}
p$get_values()
p$get_values(inc_null = FALSE)
p$get_values(id = "a")
p$get_values(tags = "t1")

```

Method `add_dep()`: Gets values from the `ParameterSet` with options to filter by specific IDs and tags, and also to transform the values.

Usage:

```
ParameterSet$add_dep(id, on, cnd)
```

Arguments:

```
id (character(1))
```

The dependent variable for the condition that depends on the given variable, on, being a particular value. Should be in self\$ids.

```
on (character(1))
```

The independent variable for the condition that is depended on by the given variable, id. Should be in self\$ids.

```
cnd (cnd(1))
```

The condition defined by `cnd` which determines how id depends on on.

Examples:

```
# not run as errors
\dontrun{
# Dependency on specific value
prms <- list(
  prm("a", "reals", NULL),
  prm("b", "reals", 1)
)
p <- ParameterSet$new(prms)
p$add_dep("a", "b", cnd("eq", 2))
# 'a' can only be set if 'b' equals 2
p$values$a <- 1
p$values <- list(a = 1, b = 2)

# Dependency on variable value
prms <- list(
  prm("a", "reals", NULL),
  prm("b", "reals", 1)
)
p <- ParameterSet$new(prms)
p$add_dep("a", "b", cnd("eq", id = "b"))
# 'a' can only be set if it equals 'b'
p$values$a <- 2
p$values <- list(a = 2, b = 2)
}
```

Method `rep()`: Replicate the ParameterSet with identical parameters. In order to avoid duplicated parameter ids, every id in the ParameterSet is given a prefix in the format `prefix__id`. In addition, linked tags are also given the same prefix to prevent incorrectly linking parameters.

The primary use-case of this method is to treat the ParameterSet as a collection of identical ParameterSet objects.

Note that this mutates the ParameterSet, if you want to instead create a new object then use [rep.ParameterSet](#) instead (or copy and deep clone) first.

Usage:

```
ParameterSet$rep(times, prefix)
```

Arguments:

times (integer(1))

Numer of times to replicate the ParameterSet.

prefix (character(1)|character(length(times)))

The prefix to add to ids and linked tags. If length 1 then is internally coerced to `paste0(prefix, seq(times))`, otherwise the length should be equal to times.

Method `extract()`: Creates a new ParameterSet by extracting the given parameters.

Usage:

```
ParameterSet$extract(id = NULL, tags = NULL, prefix = NULL)
```

Arguments:

id (character())

If not NULL then specifies the parameters by id to extract. Should be NULL if prefix is not NULL.

tags (character())

If not NULL then specifies the parameters by tag to extract. Should be NULL if prefix is not NULL.

prefix (character())

If not NULL then extracts parameters according to their prefix and additionally removes the prefix from the id. A prefix is determined as the string before "__" in an id.

Examples:

```
# extract by id
prms <- list(
  prm("a", "reals", NULL),
  prm("b", "reals", 1)
)
p <- ParameterSet$new(prms)
p$extract("a")
# equivalently
p["a"]

# extract by prefix
prms <- list(
  prm("Pre1__par1", Set$new(1), 1, tags = "t1"),
  prm("Pre1__par2", "reals", 3, tags = "t2"),
  prm("Pre2__par1", Set$new(1), 1, tags = "t1"),
  prm("Pre2__par2", "reals", 3, tags = "t2")
)
p <- ParameterSet$new(prms)
p$extract(tags = "t1")
p$extract(prefix = "Pre1")
# equivalently
p[prefix = "Pre1"]
```

Method `remove()`: Removes the given parameters from the set.

Usage:

```
ParameterSet$remove(id = NULL, prefix = NULL)
```

Arguments:

id (character())

If not NULL then specifies the parameters by id to extract. Should be NULL if prefix is not NULL.

prefix (character())

If not NULL then extracts parameters according to their prefix and additionally removes the prefix from the id. A prefix is determined as the string before "__" in an id.

Method `getParameterValue()`: Deprecated method added for distr6 compatibility. Use `$values/$get_values()` in the future. Will be removed in 0.3.0.

Usage:

```
ParameterSet$getParameterValue(id, ...)
```

Arguments:

id Parameter id

... Unused

Method `setParameterValue()`: Deprecated method added for distr6 compatibility. Use `$set_values` in the future. Will be removed in 0.3.0.

Usage:

```
ParameterSet$setParameterValue(..., lst = list(...))
```

Arguments:

... Parameter ids

lst List of parameter ids

Method `set_values()`: Convenience function for setting multiple parameters without changing or accidentally removing others.

Usage:

```
ParameterSet$set_values(..., lst = list(...))
```

Arguments:

... Parameter ids

lst List of parameter ids

Method `parameters()`: Deprecated method added for distr6 compatibility. Use `$print/as.data.table()` in the future. Will be removed in 0.3.0.

Usage:

```
ParameterSet$parameters(...)
```

Arguments:

... Unused

Method `transform()`: Applies the internal transformation function. If no function has been passed to `$trafo` then `x` is returned unchanged. If `$trafo` is a function then `x` is passed directly to this. If `$trafo` is a list then `x` is evaluated and passed down the list iteratively.

Usage:

```
ParameterSet$transform(x = self$values)
```

Arguments:

x (named list(1))
List of values to transform.

Returns: named list(1)

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
ParameterSet$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
library(set6)

## $value examples
p <- ParameterSet$new(list(prm(id = "a", support = Reals$new())))
p$values$a <- 2
p$values

## $trafo examples
p <- ParameterSet$new(list(prm(id = "a", 2, support = Reals$new())))
p$trafo

# simple transformation
p$get_values()
p$trafo <- function(x, self) {
  x$a <- exp(x$a)
  x
}
p$get_values()

# more complex transformation on tags
p <- ParameterSet$new(
  list(prm(id = "a", 2, support = Reals$new(), tags = "t1"),
       prm(id = "b", 3, support = Reals$new(), tags = "t1"),
       prm(id = "d", 4, support = Reals$new()))
)
# make sure `transform = FALSE` to prevent infinite recursion
p$trafo <- function(x, self) {
  out <- lapply(self$get_values(tags = "t1", transform = FALSE),
               function(.x) 2^.x)
  out <- c(out, list(d = x$d))
  out
}
p$get_values()

## -----
```

```

## Method `ParameterSet$new`
## -----

prms <- list(
  prm("a", Set$new(1), 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
  prm("d", "reals", 2, tags = "t2")
)
ParameterSet$new(prms)

## -----
## Method `ParameterSet$print`
## -----

prms <- list(
  prm("a", Set$new(1), 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
  prm("d", "reals", 2, tags = "t2")
)
p <- ParameterSet$new(prms)
p$print()
print(p)
p

## -----
## Method `ParameterSet$get_values`
## -----

prms <- list(
  prm("a", "reals", 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
  prm("d", "reals", tags = "t2")
)
p <- ParameterSet$new(prms)
p$trafo <- function(x, self) {
  x$a <- exp(x$a)
  x
}
p$get_values()
p$get_values(inc_null = FALSE)
p$get_values(id = "a")
p$get_values(tags = "t1")

## -----
## Method `ParameterSet$add_dep`
## -----

# not run as errors
## Not run:
# Dependency on specific value
prms <- list(
  prm("a", "reals", NULL),
  prm("b", "reals", 1)
)

```

```

)
p <- ParameterSet$new(prms)
p$add_dep("a", "b", cnd("eq", 2))
# 'a' can only be set if 'b' equals 2
p$values$a <- 1
p$values <- list(a = 1, b = 2)

# Dependency on variable value
prms <- list(
  prm("a", "reals", NULL),
  prm("b", "reals", 1)
)
p <- ParameterSet$new(prms)
p$add_dep("a", "b", cnd("eq", id = "b"))
# 'a' can only be set if it equals 'b'
p$values$a <- 2
p$values <- list(a = 2, b = 2)

## End(Not run)

## -----
## Method `ParameterSet$extract`
## -----

# extract by id
prms <- list(
  prm("a", "reals", NULL),
  prm("b", "reals", 1)
)
p <- ParameterSet$new(prms)
p$extract("a")
# equivalently
p["a"]

# extract by prefix
prms <- list(
  prm("Pre1__par1", Set$new(1), 1, tags = "t1"),
  prm("Pre1__par2", "reals", 3, tags = "t2"),
  prm("Pre2__par1", Set$new(1), 1, tags = "t1"),
  prm("Pre2__par2", "reals", 3, tags = "t2")
)
p <- ParameterSet$new(prms)
p$extract(tags = "t1")
p$extract(prefix = "Pre1")
# equivalently
p[prefix = "Pre1"]

```


Description

The `prm` class is required for [ParameterSet](#) objects, it allows specifying a parameter as a named set and optionally setting values and tags.

Usage

```
prm(id, support, value = NULL, tags = NULL, .check = TRUE)
```

Arguments

<code>id</code>	(character(1)) Parameter identifier.
<code>support</code>	([set6::Set] character(1)) Either a set object from set6 or a character representing the set if it is already present in the support_dictionary . If a <code>set6::Set</code> is provided then the set and its string representation are added automatically to support_dictionary in order to provide fast internal checks. Common sets (such as the reals, naturals, etc.) are already provided in support_dictionary .
<code>value</code>	ANY Optional to assign the parameter, will internally be checked that it lies within the given support.
<code>tags</code>	(character()) An optional character vector of tags to apply to the parameter. On their own tags offer little extra benefit, however they can be assigned properties when creating ParameterSet objects that enable them to be more powerful.
<code>.check</code>	For internal use only.

Examples

```
library(set6)

# Constructing a prm with a Set support
prm(
  id = "a",
  support = Reals$new(),
  value = 1
)

# Constructing a prm with a support already in the dictionary
prm(
  id = "a",
  support = "reals",
  value = 1
)

# Adding tags
prm(
  id = "a",
  support = "reals",
```

```

value = 1,
tags = c("tag1", "tag2")
)

```

pset

Convenience Function for Constructing a ParameterSet

Description

See [ParameterSet](#) for full details.

Usage

```
pset(..., prms = list(...), tag_properties = NULL, deps = NULL, trafo = NULL)
```

Arguments

...	(prm) prm objects.
prms	(list()) List of prm objects.
tag_properties	(list()) List of tag properties. Currently supported properties are: i) 'required' - parameters with this tag property must be non-NULL; ii) 'linked' - only one parameter in a linked tag group can be non-NULL and the others should be NULL, this only makes sense with an associated trafo; iii) 'unique' - parameters with this tag must have no duplicated elements, only makes sense for vector parameters; iv) 'immutable' - parameters with this tag cannot be updated after construction.
deps	(list()) List of lists where each element is passed to \$add_dep. See examples.
trafo	(function()) Passed to \$trafo. See examples.

Examples

```

library(set6)

# simple example
prms <- list(
  prm("a", Set$new(1), 1, tags = "t1"),
  prm("b", "reals", 1.5, tags = "t1"),
  prm("d", "reals", 2, tags = "t2")
)
p <- pset(prms = prms)

# with properties, deps, trafo
p <- pset(

```

```

prm("a", Set$new(1), 1, tags = "t1"),
prm("b", "reals", 1.5, tags = "t1"),
prm("d", "reals", 2, tags = "t2"),
tag_properties = list(required = "t2"),
deps = list(
  list(id = "a", on = "b", cond = cnd("eq", 1.5))
),
trafo = function(x, self) return(x)
)

```

rep.ParameterSet	<i>Replicate a ParameterSet</i>
------------------	---------------------------------

Description

In contrast to the `$rep` method in [ParameterSet](#), this method deep clones the [ParameterSet](#) and returns a new object.

Usage

```

## S3 method for class 'ParameterSet'
rep(x, times, prefix, ...)

```

Arguments

x	(ParameterSet)
times	(<code>integer(1)</code>) Numer of times to replicate the ParameterSet .
prefix	(<code>character(1) character(length(times))</code>) The prefix to add to ids and linked tags. If length 1 then is internally coerced to <code>paste0(prefix, seq(times))</code> , otherwise the length should be equal to <code>times</code> .
...	(ANY) Other arguments, currently unused.

Details

In order to avoid duplicated parameter ids, every id in the [ParameterSet](#) is given a prefix in the format `prefix__id`. In addition, linked tags are also given the same prefix to prevent incorrectly linking parameters.

The primary use-case of this method is to treat the [ParameterSet](#) as a collection of identical [ParameterSet](#) objects.

support_dictionary	<i>Support Dictionary</i>
--------------------	---------------------------

Description

[dictionary6::Dictionary](#) for parameter supports

Details

See [dictionary6::Dictionary](#) for full details of how to add other [set6::Set](#) objects as supports to this dictionary.

Examples

```
support_dictionary$keys
support_dictionary$items
```

[.ParameterSet	<i>Extract a sub-ParameterSet by Parameters</i>
----------------	---

Description

Creates a new [ParameterSet](#) by extracting the given parameters. S3 method for the `$extract` public method.

Usage

```
## S3 method for class 'ParameterSet'
object[...]
```

Arguments

object	(ParameterSet)
...	(ANY) Passed to ParameterSet \$extract

Index

[.ParameterSet, 20

as.data.table.ParameterSet, 2, 9
as.ParameterSet, 3
as.prm, 3

c.ParameterSet, 4
cnd, 5, 11
cpset, 6

data.table::data.table, 2, 8
dictionary6::Dictionary, 20

expect_equal_ps, 7

length.ParameterSet, 7

param6 (param6-package), 2
param6-package, 2
ParameterSet, 2–7, 7, 17–20
prm, 3, 7, 9, 16, 18
pset, 18

rep.ParameterSet, 11, 19

set6::Set, 17, 20
support_dictionary, 17, 20