

Package: osp.snapshots (via r-universe)

September 28, 2026

Type Package

Title Import and Manipulate Open Systems Pharmacology Project
Snapshots

Version 1.1.0

Description Provides a convenient R interface for working with PKSIM project snapshots exported as JSON files. You provide the snapshot file, and 'osp.snapshots' handles the complex nested structure, allowing you to easily import, navigate, modify, and export snapshot data. Building blocks can be converted to data frames for integration with the 'ospsuite' R framework and broader R ecosystem.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL <https://esqlabs.github.io/osp.snapshots/>

Depends R (>= 4.1.0)

Imports R6, jsonlite, cli, fs, lifecycle, ospsuite (>= 12.2.0), purrr,
tibble, dplyr, glue, lubridate, stringr

Suggests devtools, ggplot2, gt, knitr, quarto, rlang, rmarkdown,
testthat (>= 3.1.6), tidyr, withr

VignetteBuilder knitr

Config/testthat/edition 3

Additional_repositories <https://open-systems-pharmacology.r-universe.dev>

Config/Needs/website quarto

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Repository <https://esqlabs.r-universe.dev>

Date/Publication 2026-07-30 14:28:35 UTC

RemoteUrl <https://github.com/esqLABS/osp.snapshots>

RemoteRef v1.1.0

RemoteSha 72ea69e02819a2ad011fd594bcdde80912e214c6

Contents

add_compound	4
add_event	5
add_expression_profile	6
add_formulation	7
add_individual	8
add_observed_data	9
add_observer_set	9
add_population	10
add_protocol	11
add_simulation	12
AdvancedParameter	14
age	15
as_tibbles	16
CalculationMethods	17
Compound	19
CompoundProcessSelection	21
convert_ospuite_time_to_duration	22
convert_ospuite_time_unit_to_lubridate	23
create_compound	23
create_compound_process_selection	27
create_descriptor_condition	28
create_event	29
create_event_selection	30
create_expression_profile	30
create_formula_reference	32
create_formulation	33
create_formulation_selection	37
create_individual	38
create_molecule_list	40
create_observed_data	41
create_observer	42
create_observer_set	44
create_observer_set_selection	45
create_output_interval	45
create_output_mapping	46
create_output_schema	47
create_parameter	48
create_population	50
create_process	52
create_protocol	54
create_protocol_selection	56
create_schema	57
create_schema_item	59
create_snapshot	61
create_solver_settings	62
error	63

Event	64
EventSelection	65
export_snapshot	66
ExpressionProfile	66
Formulation	68
FormulationSelection	70
fraction_unbound	71
gestational_age	72
get_compounds_dfs	73
get_default_alternative	73
get_events_dfs	74
get_expression_profiles_dfs	75
get_formulations_dfs	75
get_individuals_dfs	76
get_observed_data_dfs	77
get_observer_sets_dfs	77
get_populations_dfs	78
get_protocols_dfs	79
height	79
Individual	80
intestinal_permeability	82
lipophilicity	83
load_snapshot	84
loadDataSetFromSnapshot	85
LocalizedParameter	86
Observer	88
ObserverSet	90
ObserverSetSelection	91
OriginData	92
osp_models	94
OutputInterval	94
OutputMapping	95
OutputSchema	96
Parameter	97
permeability	98
Population	99
print.biliary_clearance	101
print.compound_additional_parameters	101
print.compound_processes	102
print.hepatic_clearance	102
print.induction	103
print.inhibition	103
print.metabolizing_enzymes	104
print.parameter_collection	104
print.physicochemical_property	105
print.protein_binding_partners	105
print.renal_clearance	106
print.snapshot_collection	106

print.transporter_proteins	107
Process	107
Protocol	109
ProtocolSelection	111
range	112
Range-class	113
remove_compound	114
remove_event	115
remove_expression_profile	115
remove_formulation	116
remove_individual	117
remove_observed_data	118
remove_observer_set	118
remove_population	119
remove_protocol	120
remove_simulation	120
Schema	121
SchemaItem	122
Simulation	123
Snapshot	125
solubility	134
SolverSettings	136
time	137
validate_gender	138
validate_population	138
validate_snapshot	139
validate_species	139
validate_unit	140
values	140
weight	141

Index 142

add_compound	<i>Add one or more compounds to a snapshot</i>
--------------	--

Description

Add one or more [Compound](#) objects to a [Snapshot](#).

Usage

```
add_compound(snapshot, compound)
```

Arguments

snapshot	A Snapshot object.
compound	A Compound object created with create_compound() , or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam") |>
  add_compound(create_compound(name = "Drug X"))

snapshot <- load_snapshot("Midazolam") |>
  add_compound(list(
    create_compound(name = "Drug X"),
    create_compound(name = "Drug Y")
  ))

## End(Not run)
```

add_event

Add one or more events to a snapshot

Description

Add one or more [Event](#) objects to a [Snapshot](#).

Usage

```
add_event(snapshot, event)
```

Arguments

snapshot A [Snapshot](#) object.

event An [Event](#) object created with [create_event\(\)](#), or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
evt <- create_event(
  name = "Breakfast",
  template = "Meal: Standard (Human)"
)
snapshot <- load_snapshot("Midazolam") |>
  add_event(evt)

snapshot <- load_snapshot("Midazolam") |>
  add_event(list(evt, evt))
```

```
## End(Not run)
```

```
add_expression_profile
```

Add one or more expression profiles to a snapshot

Description

Add one or more [ExpressionProfile](#) objects to a [Snapshot](#).

Usage

```
add_expression_profile(snapshot, expression_profile)
```

Arguments

snapshot	A Snapshot object
expression_profile	An ExpressionProfile object, or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Create a new expression profile
profile_data <- list(
  Type = "Enzyme",
  Species = "Human",
  Molecule = "CYP3A4",
  Category = "Healthy",
  Parameters = list()
)
profile <- ExpressionProfile$new(profile_data)

# Add a single expression profile
snapshot <- add_expression_profile(snapshot, profile)

# Add several at once
snapshot <- add_expression_profile(snapshot, list(profile, profile))

## End(Not run)
```

add_formulation	<i>Add one or more formulations to a snapshot</i>
-----------------	---

Description

Add one or more [Formulation](#) objects to a [Snapshot](#).

Usage

```
add_formulation(snapshot, formulation)
```

Arguments

snapshot	A Snapshot object
formulation	A Formulation object created with create_formulation() , or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Add a single formulation
form <- create_formulation(name = "Tablet", type = "Weibull")
snapshot <- add_formulation(snapshot, form)

# Add several at once
forms <- list(
  create_formulation("Tablet", type = "Weibull"),
  create_formulation("Oral solution", type = "First Order")
)
snapshot <- add_formulation(snapshot, forms)

## End(Not run)
```

add_individual	<i>Add one or more individuals to a snapshot</i>
----------------	--

Description

Add one or more [Individual](#) objects to a [Snapshot](#).

Usage

```
add_individual(snapshot, individual)
```

Arguments

snapshot	A Snapshot object
individual	An Individual object created with create_individual() , or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Add a single individual
ind <- create_individual(name = "New Patient", age = 35, weight = 70)
snapshot <- add_individual(snapshot, ind)

# Add several at once
patients <- list(
  create_individual("Patient_A", age = 25),
  create_individual("Patient_B", age = 45)
)
snapshot <- add_individual(snapshot, patients)

## End(Not run)
```

add_observed_data	<i>Add one or more observed-data entries to a snapshot</i>
-------------------	--

Description

Add one or more `osp suite::DataSet` (observed data) objects to a [Snapshot](#).

Usage

```
add_observed_data(snapshot, observed_data)
```

Arguments

snapshot	A Snapshot object.
observed_data	A <code>DataSet</code> object, typically created with create_observed_data() or loadDataSetFromSnapshot() , or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
dataset <- create_observed_data(
  name = "Study A",
  time = c(0, 1, 2),
  values = c(0, 5, 8),
  value_dimension = "Concentration (mass)"
)
snapshot <- load_snapshot("Midazolam") |>
  add_observed_data(dataset)

snapshot <- load_snapshot("Midazolam") |>
  add_observed_data(list(dataset, dataset))

## End(Not run)
```

add_observer_set	<i>Add one or more observer sets to a snapshot</i>
------------------	--

Description

Add one or more `ObserverSet` building blocks to a `Snapshot`. The exported function is the canonical, pipeable surface for the mutation; it validates the snapshot before delegating to the underlying R6 method.

Usage

```
add_observer_set(snapshot, observer_set)
```

Arguments

snapshot A Snapshot object.
 observer_set An ObserverSet object, or a list of such objects.

Value

The updated Snapshot object, invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam")

observer_set <- ObserverSet$new(list(
  Name = "BrainPlasmaConcentration",
  Observers = list()
))

snapshot |> add_observer_set(observer_set)

snapshot |> add_observer_set(list(observer_set, observer_set))

## End(Not run)
```

add_population	<i>Add one or more populations to a snapshot</i>
----------------	--

Description

Add one or more [Population](#) objects to a [Snapshot](#).

Usage

```
add_population(snapshot, population)
```

Arguments

snapshot A [Snapshot](#) object.
 population A [Population](#) object created with [create_population\(\)](#), or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
pop <- create_population(name = "Adults", number_of_individuals = 100)
snapshot <- load_snapshot("Midazolam") |>
  add_population(pop)

snapshot <- load_snapshot("Midazolam") |>
  add_population(list(
    create_population(name = "Adults", number_of_individuals = 100),
    create_population(name = "Children", number_of_individuals = 50)
  ))

## End(Not run)
```

add_protocol	<i>Add one or more protocols to a snapshot</i>
--------------	--

Description

Add one or more [Protocol](#) objects to a [Snapshot](#).

Usage

```
add_protocol(snapshot, protocol)
```

Arguments

snapshot	A Snapshot object.
protocol	A Protocol object created with create_protocol() , or a list of such objects.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
prot <- create_protocol(
  name = "Single oral dose",
  application_type = "Oral",
  dosing_interval = "Single"
)
snapshot <- load_snapshot("Midazolam") |>
  add_protocol(prot)

snapshot <- load_snapshot("Midazolam") |>
  add_protocol(list(prot, prot))

## End(Not run)
```

add_simulation	<i>Build a simulation from a snapshot and attach it</i>
----------------	---

Description

Build a [Simulation](#) from named arguments and attach it to a [Snapshot](#), or attach a pre-built [Simulation](#).

A simulation is almost pure references into the snapshot, so the snapshot is its entry point. In build mode, with the snapshot in hand, `add_simulation()` resolves compound references and derives sensible defaults (calculation methods, formulation key, alternatives) from the referenced building blocks before attaching. Supply name plus exactly one of `individual` / `population`, and configure each compound inline through `compounds`, selecting a specific alternative by friendly property name and label, or a multi-slot protocol's formulations, where the derived defaults are not enough. References to building blocks not yet in the snapshot trigger one informational warning per simulation; the add proceeds either way.

Usage

```
add_simulation(
    snapshot,
    simulation = NULL,
    name = NULL,
    model = "4Comp",
    individual = NULL,
    population = NULL,
    compounds = NULL,
    events = NULL,
    observer_sets = NULL,
    observed_data_names = NULL,
    solver = NULL,
    output_schema = NULL,
    output_selections = NULL,
    output_mappings = NULL,
    parameters = NULL,
    advanced_parameters = NULL,
    description = NULL,
    allow_aging = NULL
)
```

Arguments

<code>snapshot</code>	A Snapshot object.
<code>simulation</code>	A pre-built Simulation object, or a list of such objects. Leave <code>NULL</code> to build a simulation from the named arguments.
<code>name</code>	Character. Simulation name (required in build mode).
<code>model</code>	Character. PK-Sim model name (defaults to "4Comp").

individual	Character. Name of the individual building block. Mutually exclusive with population.
population	Character. Name of the population building block. Mutually exclusive with individual.
compounds	List of inline compound-config lists (<code>list(name =, protocol =, formulation =, processes =, calculation_methods =, alternatives =)</code>). Each entry is resolved against the snapshot. <code>alternatives</code> is a named character vector (or named list of length-one strings) mapping a friendly property name (<code>lipophilicity</code> , <code>fraction_unbound</code> , <code>solubility</code> , <code>intestinal_permeability</code> , <code>permeability</code>) to the alternative label to select on that compound, for example <code>alternatives = c(solubility = "FaSSIF")</code> ; it overrides the derived default for the named groups only, every other group is still defaulted from the compound. <code>formulation</code> accepts a single string (bound to the protocol's inferred first slot key, unchanged) or a named character vector (or named list of length-one strings) mapping application-slot key to formulation name for a multi-slot protocol, for example <code>formulation = c(Formulation = "Oral solution", "Formulation 2" = "IV solution")</code> .
events	List of EventSelection objects or raw lists.
observer_sets	List of ObserverSetSelection objects or raw lists.
observed_data_names	Character vector of observed-data names.
solver	A SolverSettings object or raw list. Defaults to PK-Sim defaults.
output_schema	An OutputSchema object or raw list. Defaults to an empty schema.
output_selections	Character vector of output quantity paths.
output_mappings	List of OutputMapping objects or raw lists.
parameters	List of LocalizedParameter objects (created with <code>create_parameter(path = ..., ...)</code>) or raw parameter lists.
advanced_parameters	List of AdvancedParameter objects or raw lists (population simulations only).
description	Character. Free-text description of the simulation.
allow_aging	Logical. Whether the simulation allows aging.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam") |>
  add_simulation(
    name = "Sim 1",
    individual = "Korean (Yu 2004 study)",
    compounds = list(list(
      name = "Rifampicin",
```

```

    protocol = "Yu 2004 - Rifampicin - 600 mg MD OD 10 days",
    formulation = "Oral solution",
    processes = c("Hepatic"),
    alternatives = c(solubility = "FaSSIF")
  ))
)

## End(Not run)

```

AdvancedParameter

AdvancedParameter class for Population advanced parameters

Description

An R6 class that represents an advanced parameter in a population.

Active bindings

data The raw data of the parameter (read-only)
name The name of the parameter
seed The seed used for parameter generation
distribution_type The distribution type of the parameter. Writable: one of "Normal", "LogNormal", "Uniform", "Discrete", "Unknown", or NULL to clear.
parameters The parameters of the distribution

Methods

Public methods:

- [AdvancedParameter\\$new\(\)](#)
- [AdvancedParameter\\$print\(\)](#)
- [AdvancedParameter\\$clone\(\)](#)

AdvancedParameter\$new(): Create a new AdvancedParameter object

Usage:

AdvancedParameter\$new(data)

Arguments:

data Raw parameter data from a population

Returns: A new AdvancedParameter object

AdvancedParameter\$print(): Print a summary of the advanced parameter

Usage:

AdvancedParameter\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the AdvancedParameter object for method chaining

AdvancedParameter\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

age	<i>Age value object</i>
-----	-------------------------

Description

Build an age value object for `create_individual()`'s age argument (or the writable Individual\$age / OriginData\$age field). Fills the OriginData Age characteristic (list(Value =, Unit =)).

Usage

```
age(value, unit = "year(s)")
```

Arguments

value	Numeric scalar. Age value.
unit	Character. Unit for age, validated against dimension "Age in years". Defaults to "year(s)".

Value

An age_spec object to pass to `create_individual()`.

See Also

`create_individual()`

Other value-object helpers: `error()`, `fraction_unbound()`, `gestational_age()`, `height()`, `intestinal_permeability()`, `lipophilicity()`, `permeability()`, `solubility()`, `time()`, `values()`, `weight()`

Examples

```
age(30)
create_individual(name = "Subject", age = age(30))
```

as_tibbles

*Convert a snapshot collection to a tibble or list of tibbles***Description**

Unified bridge from a Snapshot to the tidyverse for any building-block kind plus observed data. Replaces the nine per-kind `get_*_dfs()` functions as the canonical entry point; those remain available as thin wrappers. Omit `kind` to convert every kind of a snapshot in a single call.

Usage

```
as_tibbles(snapshot, kind = NULL)
```

Arguments

snapshot	A Snapshot object.
kind	Character vector naming the collection(s) to convert, or NULL (the default) to convert every kind. Each element must be one of "compounds", "individuals", "formulations", "populations", "events", "expression_profiles", "protocols", "observer_sets", "observed_data". A length-1 kind returns that kind's native shape directly (a tibble or a named list of tibbles); a length-2-or-more kind returns a named list keyed by the requested kinds, in request order; NULL returns a named list of all nine kinds in the order above.

Value

When `kind` names a single collection, a tibble or a named list of tibbles for that collection (see below). When `kind` names several collections, or is NULL (all nine kinds), a named list keyed by the requested kinds, each entry the native shape of that kind:

- "compounds": a list with `properties` and `processes` tibbles. `properties` carries one row per (compound, physicochemical property) pair (plus folded process rows for backwards compatibility); `processes` is the long-form, one row per (compound, process, parameter) triple.
- "protocols", "observed_data": a single tibble.
- "observer_sets": a list with `observer_sets` (one row per `ObserverSet`) and `observers` (one row per `Observer`, joinable to its parent via `observer_set_id` / `observer_set_name`).
- "individuals": a list with `individuals`, `individuals_parameters`, `individuals_expressions`.
- "formulations": a list with `formulations`, `formulations_parameters`.
- "populations": a list with `populations`, `populations_parameters`.
- "events": a list with `events`, `events_parameters`.
- "expression_profiles": a list with `expression_profiles`, `expression_profiles_parameters`.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam")

# List of tibbles (compounds returns properties + processes)
compounds <- as_tibbles(snapshot, "compounds")
compounds$properties
compounds$processes

# List of tibbles
individuals <- as_tibbles(snapshot, "individuals")
individuals$individuals_parameters

# Omit `kind` to convert every kind at once, keyed by kind name
all_kinds <- as_tibbles(snapshot)
names(all_kinds)

## End(Not run)
```

CalculationMethods	<i>CalculationMethods class for OSP snapshots</i>
--------------------	---

Description

An R6 class that wraps the set of calculation methods PK-Sim uses to derive compound quantities (e.g. partition coefficient method, cellular permeability method, body surface area method). In an OSP snapshot, the calculation methods are serialised as an array of method name strings. They appear in two places: directly on a [Compound](#) and inside an [OriginData](#) of an [Individual](#).

Active bindings

`names` Character vector of method names.

`length` Number of methods currently held (read-only).

Methods

Public methods:

- [CalculationMethods\\$new\(\)](#)
- [CalculationMethods\\$print\(\)](#)
- [CalculationMethods\\$add\(\)](#)
- [CalculationMethods\\$remove\(\)](#)
- [CalculationMethods\\$to_list\(\)](#)
- [CalculationMethods\\$clone\(\)](#)

`CalculationMethods$new()`: Create a new CalculationMethods object

Usage:

CalculationMethods\$new(names = character())

Arguments:

names Character vector of method names. May be NULL or an empty vector, both of which result in an empty set.

Returns: A new CalculationMethods object

CalculationMethods\$print(): Print a summary of the calculation methods

Usage:

CalculationMethods\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the CalculationMethods object

CalculationMethods\$add(): Add a method name. Duplicates are kept to mirror the snapshot's raw representation.

Usage:

CalculationMethods\$add(name)

Arguments:

name Character. Name of the calculation method to add.

Returns: Invisibly returns the CalculationMethods object

CalculationMethods\$remove(): Remove a method name. If name is not present, the set is unchanged. Note the asymmetry with \$add(): \$add() keeps duplicates to mirror the snapshot's raw representation, whereas \$remove() deletes every occurrence of name in a single call.

Usage:

CalculationMethods\$remove(name)

Arguments:

name Character. Name of the calculation method to remove.

Returns: Invisibly returns the CalculationMethods object

CalculationMethods\$to_list(): Convert back to the raw snapshot representation: a list of method-name strings, or NULL when empty. A list is used (rather than a character vector) so that single-method sets round-trip as JSON arrays.

Usage:

CalculationMethods\$to_list()

Returns: A list of method-name strings, or NULL when empty.

CalculationMethods\$clone(): The objects of this class are cloneable with this method.

Usage:

CalculationMethods\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Compound

*Compound class for OSP snapshot compounds***Description**

An R6 class that represents a compound in an OSP snapshot. This class provides methods to access different properties of a compound and display a summary of its information.

Compound processes are exposed via `$processes`, a flat named list of [Process](#) objects. The per-category tibble accessors (`$protein_binding_partners`, `$metabolizing_enzymes`, `$hepatic_clearance`, `$transporter_proteins`, `$renal_clearance`, `$biliary_clearance`, `$inhibition`, `$induction`) are `lifecycle::deprecate_soft()`-warned in favour of `$processes` and the long-form processes tibble returned by `get_compounds_dfs()`.

Active bindings

`data` The raw data of the compound (read-only). Refreshed from the embedded [CalculationMethods](#) and the cached [Process](#) objects so that mutations flow back to the export payload.

`name` The name of the compound. Writable: must be a non-empty scalar string.

`is_small_molecule` Whether the compound is a small molecule. Writable: a single logical value, or NULL to leave it unset (PK-Sim then defaults to TRUE).

`plasma_protein_binding_partner` The plasma protein binding partner of the compound. Writable: one of "Unknown", "Albumin", "Glycoprotein", or NULL to clear.

`molecular_weight` The molecular weight of the compound

`molecular_weight_unit` The unit of the molecular weight

`lipophilicity` The lipophilicity data of the compound. Writable: assign a [lipophilicity\(\)](#) object, a list of such objects to define several named alternatives (mark exactly one with `default = TRUE` to designate the group default; when none is marked, the first element is the default), a raw alternative list to set the array verbatim (the escape hatch for species-specific alternatives), or NULL to clear the property. A bare numeric scalar is rejected; use [lipophilicity\(\)](#) instead.

`fraction_unbound` The fraction unbound data of the compound. Writable: assign a [fraction_unbound\(\)](#) object, a list of such objects to define several named alternatives (mark exactly one with `default = TRUE` to designate the group default; when none is marked, the first element is the default), a raw alternative list to set the array verbatim, or NULL to clear the property. A bare numeric scalar is rejected; use [fraction_unbound\(\)](#) instead.

`solubility` The solubility data of the compound. Writable: assign a [solubility\(\)](#) object to express reference pH, gain per charge, or a pH/value table, a list of such objects to define several named alternatives (mark exactly one with `default = TRUE` to designate the group default; when none is marked, the first element is the default), a raw alternative list to set the array verbatim, or NULL to clear the property. A bare numeric scalar is rejected; use [solubility\(\)](#) instead.

`intestinal_permeability` The intestinal permeability data of the compound. Writable: assign an [intestinal_permeability\(\)](#) object, a list of such objects to define several named alternatives (mark exactly one with `default = TRUE` to designate the group default; when none is

marked, the first element is the default), a raw alternative list to set the array verbatim, or NULL to clear the property. A bare numeric scalar is rejected; use [intestinal_permeability\(\)](#) instead.

permeability The permeability data of the compound. Writable: assign a [permeability\(\)](#) object, a list of such objects to define several named alternatives (mark exactly one with `default = TRUE` to designate the group default; when none is marked, the first element is the default), a raw alternative list to set the array verbatim, or NULL to clear the property. A bare numeric scalar is rejected; use [permeability\(\)](#) instead.

pka_types The pKa types of the compound. Writable: assign a list of `list(type =, value =)` entries (the `create_compound()` pKa shape, converted to `list(Type =, Pka =)`), a raw `PkaType[]` list (entries carrying `Type/Pka`, set verbatim), or NULL / `list()` to clear the pKa types.

processes A flat named list of [Process](#) objects, one per entry in the compound's Processes array. Duplicate names are disambiguated with a numeric suffix (`_1`, `_2`, ...). The list is built once at construction so that state changes made on a [Process](#) persist across accesses. Writable: assign a list of [Process](#) objects and/or raw process lists to rebuild the processes (duplicate names are disambiguated as at construction), or NULL / `list()` to clear them.

calculation_methods A [CalculationMethods](#) object holding the compound's calculation methods.

parameters The additional parameters of the compound (excluding molecular weight)

protein_binding_partners Deprecated. Filter [get_compounds_dfs\(\)](#)\$processes on category == "protein_binding_partners", or iterate `self$processes` and check `process$category`.

metabolizing_enzymes Deprecated. See `$protein_binding_partners`.

hepatic_clearance Deprecated. See `$protein_binding_partners`.

transporter_proteins Deprecated. See `$protein_binding_partners`.

renal_clearance Deprecated. See `$protein_binding_partners`.

biliary_clearance Deprecated. See `$protein_binding_partners`.

inhibition Deprecated. See `$protein_binding_partners`.

induction Deprecated. See `$protein_binding_partners`.

Methods

Public methods:

- [Compound\\$new\(\)](#)
- [Compound\\$print\(\)](#)
- [Compound\\$to_df\(\)](#)
- [Compound\\$clone\(\)](#)

Compound\$new(): Create a new Compound object

Usage:

`Compound$new(data)`

Arguments:

data Raw compound data from a snapshot

Returns: A new Compound object

`Compound$print()`: Print a summary of the compound including its properties and parameters.

Usage:

`Compound$print(...)`

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the Compound object for method chaining

`Compound$to_df()`: Convert this compound's physicochemical properties and process parameters to a single long-form tibble (legacy shape).

Used by `get_compounds_dfs()` to assemble the compound-wide properties tibble. The process-derived rows produced here are `lifecycle::deprecate_soft()`-warned at the `get_compounds_dfs()` entry point; prefer the long-form processes tibble returned alongside.

Usage:

`Compound$to_df()`

Returns: A tibble with columns compound, category, type, parameter, value, unit, data_source, source.

`Compound$clone()`: The objects of this class are cloneable with this method.

Usage:

`Compound$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

CompoundProcessSelection

CompoundProcessSelection class for compound process selections

Description

An R6 class representing one entry in a `CompoundProperties`'s Processes array. Selects a compound process by some combination of name, molecule, metabolite, compound, and systemic process type; the resolver in PK-Sim picks the right placeholder when only a subset of fields is supplied.

Active bindings

`data` The raw data of the selection (read-only).

`name` The process name. Writable: a non-empty scalar string when supplied, or NULL to clear.

`molecule_name` The molecule involved in the process. Writable: a non-empty scalar string when supplied, or NULL to clear.

`metabolite_name` The metabolite produced by the process. Writable: a non-empty scalar string when supplied, or NULL to clear.

compound_name The other compound involved in the process. Writable: a non-empty scalar string when supplied, or NULL to clear.

systemic_process_type The systemic process type (e.g. "Hepatic", "Renal", "Biliary"). Writable: a non-empty scalar string when supplied, or NULL to clear. The closed set is not confirmable from the snapshot schema (only examples are given), so a required string is the strongest check available.

Methods

Public methods:

- [CompoundProcessSelection\\$new\(\)](#)
- [CompoundProcessSelection\\$clone\(\)](#)

CompoundProcessSelection\$new(): Create a new CompoundProcessSelection object.

Usage:

CompoundProcessSelection\$new(data)

Arguments:

data Raw CompoundProcessSelection data from a snapshot.

Returns: A new CompoundProcessSelection object.

CompoundProcessSelection\$clone(): The objects of this class are cloneable with this method.

Usage:

CompoundProcessSelection\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

convert_ospsuite_time_to_duration

Convert time value and unit to lubridate duration

Description

Convert a time value and ospsuite unit to a lubridate duration object. This function handles unit conversion and special cases like kiloseconds.

Usage

```
convert_ospsuite_time_to_duration(value, unit)
```

Arguments

value	The time value
unit	The ospsuite time unit

Value

A lubridate duration object

convert_ospsuite_time_unit_to_lubridate
Convert ospsuite time units to lubridate-compatible units

Description

Convert time units from ospsuite format to lubridate-compatible format. This function handles the mapping of units like "day(s)" to "days", etc.

Usage

```
convert_ospsuite_time_unit_to_lubridate(unit)
```

Arguments

unit The ospsuite time unit to convert

Value

A lubridate-compatible time unit

create_compound *Create a new compound*

Description

Create a [Compound](#) building block from named arguments. This is a thin factory around `Compound$new()` that builds the raw list shape for you, including the physicochemical properties, pKa, and processes.

Each of lipophilicity, fraction unbound, solubility (including reference pH, gain per charge, and table solubility), intestinal permeability, permeability, and pKa must be set through its dedicated argument (or by mutating the matching writable [Compound](#) field). The parameters argument is for additional/loose compound parameters only; it cannot set these physicochemical properties, because each lives in its own top-level alternative array or typed list rather than in Parameters.

Usage

```
create_compound(  
  name,  
  description = NULL,  
  is_small_molecule = NULL,  
  plasma_protein_binding_partner = NULL,  
  molecular_weight = NULL,  
  molecular_weight_unit = "g/mol",  
  calculation_methods = NULL,  
  parameters = NULL,
```

```

    lipophilicity = NULL,
    fraction_unbound = NULL,
    solubility = NULL,
    intestinal_permeability = NULL,
    permeability = NULL,
    pKa = NULL,
    processes = NULL
)

```

Arguments

name	Character. Name of the compound (required).
description	Character. Free-text description of the compound.
is_small_molecule	Logical. Whether the compound is a small molecule. Defaults to TRUE in PK-Sim when omitted.
plasma_protein_binding_partner	Character. Name of the plasma protein binding partner: one of "Unknown", "Albumin", "Glycoprotein".
molecular_weight	Numeric. Molecular weight value.
molecular_weight_unit	Character. Unit for molecular weight. Defaults to "g/mol".
calculation_methods	Character vector. Calculation method names that PK-Sim uses to derive compound quantities.
parameters	List of Parameter objects (created with create_parameter()) or raw parameter lists to attach as additional compound parameters. This does not set physico-chemical properties; use the dedicated arguments below for those.
lipophilicity	A lipophilicity() object, a list of such objects to define several named alternatives (mark exactly one with default = TRUE to designate the group default; when none is marked, the first element is the default), or NULL. When supplied, one Lipophilicity alternative is created per element.
fraction_unbound	A fraction_unbound() object, a list of such objects to define several named alternatives (mark exactly one with default = TRUE to designate the group default; when none is marked, the first element is the default), or NULL. When supplied, one FractionUnbound alternative is created per element.
solubility	A solubility() object, a list of such objects to define several named alternatives (mark exactly one with default = TRUE to designate the group default; when none is marked, the first element is the default), or NULL. Each object expresses either the scalar form (value at a reference pH, with optional gain per charge) or the table form (a pH/value table); a list may mix both forms. When supplied, one Solubility alternative is created per element. See solubility() for the scalar vs table forms and the mutual-exclusivity rule.

intestinal_permeability	An intestinal_permeability() object, a list of such objects to define several named alternatives (mark exactly one with default = TRUE to designate the group default; when none is marked, the first element is the default), or NULL. When supplied, one IntestinalPermeability alternative is created per element.
permeability	A permeability() object, a list of such objects to define several named alternatives (mark exactly one with default = TRUE to designate the group default; when none is marked, the first element is the default), or NULL. When supplied, one Permeability alternative is created per element.
pKa	List of typed pKa entries, each a list with a type (one of "Acid", "Base", "Neutral") and a numeric value, for example <code>list(list(type = "Base", value = 10.02))</code> . Order is preserved. Both NULL (the default) and an empty <code>list()</code> leave the created compound with no pKa types set. (The clearing semantics, where an empty <code>list()</code> removes existing values, apply to the writable <code>Compound\$pka_types</code> field, not to construction.)
processes	List of Process objects (created with create_process()) or raw process lists to attach to the compound.

Value

A [Compound](#) object.

See Also

[lipophilicity\(\)](#), [fraction_unbound\(\)](#), [solubility\(\)](#), [intestinal_permeability\(\)](#), [permeability\(\)](#) for the physicochemical property helpers.

Examples

```
# Create a minimal compound
compound <- create_compound(name = "Drug X")

# Create a small molecule with molecular weight and binding partner
compound <- create_compound(
  name = "Drug X",
  is_small_molecule = TRUE,
  molecular_weight = 250.3,
  plasma_protein_binding_partner = "Albumin"
)

# Set the single-parameter physicochemical properties
compound <- create_compound(
  name = "Drug X",
  lipophilicity = lipophilicity(2.5),
  fraction_unbound = fraction_unbound(0.1),
  intestinal_permeability = intestinal_permeability(1.14e-05),
  permeability = permeability(0.0069)
)
```

```
# Solubility with reference pH and gain per charge
compound <- create_compound(
  name = "Drug X",
  solubility = solubility(9999, reference_pH = 7, gain_per_charge = 1000)
)

# Table-based solubility (first column pH, second column value)
compound <- create_compound(
  name = "Drug X",
  solubility = solubility(
    table = data.frame(
      pH = c(3, 6, 6.8),
      value = c(5000, 3000, 90)
    )
  )
)

# Several named solubility alternatives (the first is the default)
compound <- create_compound(
  name = "Drug X",
  solubility = list(
    solubility(9999, name = "Aqueous"),
    solubility(200, name = "FaSSIF")
  )
)

# Several named solubility alternatives, explicitly marking the default
compound <- create_compound(
  name = "Drug X",
  solubility = list(
    solubility(9999, name = "Aqueous"),
    solubility(200, name = "FaSSIF", default = TRUE)
  )
)

# Multiple pKa entries
compound <- create_compound(
  name = "Drug X",
  pKa = list(
    list(type = "Base", value = 10.02),
    list(type = "Acid", value = 1.7)
  )
)

# Attach a process
compound <- create_compound(
  name = "Drug X",
  processes = list(
    create_process(
      internal_name = "GlomerularFiltration",
      data_source = "Publication X"
    )
  )
)
```

```
)  
  
# Additional/loose parameters (not physicochemical properties)  
compound <- create_compound(  
  name = "Drug X",  
  parameters = list(  
    create_parameter(name = "Cl_spec", value = 5, unit = "ml/min/kg")  
  )  
)
```

create_compound_process_selection

Create a compound process selection

Description

Build a [CompoundProcessSelection](#) entry, used inside a [CompoundProperties](#) to record which compound processes are selected for the simulation.

All arguments are optional and serialized only when supplied. PK-Sim resolves the right process placeholder from whichever combination of fields is set.

Usage

```
create_compound_process_selection(  
  name = NULL,  
  molecule_name = NULL,  
  metabolite_name = NULL,  
  compound_name = NULL,  
  systemic_process_type = NULL  
)
```

Arguments

name	Character. Process name.
molecule_name	Character. Molecule involved in the process.
metabolite_name	Character. Metabolite produced by the process.
compound_name	Character. Other compound involved in the process.
systemic_process_type	Character. Systemic process type (e.g. "Hepatic", "Renal", "Biliary").

Value

A [CompoundProcessSelection](#) object.

Examples

```
create_compound_process_selection(  
    name = "P-gp-Collett 2004",  
    molecule_name = "P-gp"  
)  
  
create_compound_process_selection(systemic_process_type = "Hepatic")
```

create_descriptor_condition

Create a container criterion for an observer

Description

Build a single DescriptorCondition entry for an [Observer](#)'s ContainerCriteria. A container criterion matches the containers an observer applies to by a Tag and an optional match Type. This is a small builder that returns the raw named list; assemble several of them into a list for [create_observer\(\)](#) or observer\$container_criteria.

Usage

```
create_descriptor_condition(tag, type = NULL)
```

Arguments

tag	Character. The container tag to match (required).
type	Character. The match type. This is an open string, not a closed enum: PK-Sim writes values such as "InContainer", "NotInContainer", "InParent", or "MatchTag", and any of these (or another string PK-Sim emits) is preserved verbatim. Omitted when NULL.

Value

A named list with Tag and, when supplied, Type.

Examples

```
create_descriptor_condition("Brain", "InContainer")  
  
# The match type is optional  
create_descriptor_condition("Brain")
```

create_event	<i>Create a new event</i>
--------------	---------------------------

Description

Create an [Event](#) building block from named arguments. This is a thin factory around `Event$new()` that builds the raw list shape for you.

An [Event](#) is a discrete non-administration perturbation that fires at a specific simulation time (for example a meal, gallbladder emptying, or organ removal). PK-Sim creates events by cloning a named template and overriding parameters, so a template name is required.

Usage

```
create_event(name, template, parameters = NULL)
```

Arguments

name	Character. Name of the event (required). This package requires a name because events are added, referenced, and removed by name, so an unnamed event cannot be used. This is intentionally stricter than the PK-Sim snapshot specification, which marks the event name as optional.
template	Character. Event template name passed to the PK-Sim factory (required). Determines which event type is created.
parameters	List of Parameter objects (created with create_parameter()) or raw parameter lists overriding values on the cloned template.

Value

An [Event](#) object.

Examples

```
# Create a minimal event
event <- create_event(
  name = "Breakfast",
  template = "Meal: Standard (Human)"
)

# Create an event with parameter overrides
event <- create_event(
  name = "Breakfast",
  template = "Meal: Standard (Human)",
  parameters = list(
    create_parameter(name = "Meal energy content", value = 500, unit = "kcal"),
    create_parameter(name = "Meal volume", value = 0.3, unit = "l")
  )
)
```

`create_event_selection`*Create an event selection for a simulation*

Description

Build an [EventSelection](#) entry that references an [Event](#) building block by name and supplies a numeric start time (in hours) for the event inside the simulation.

Usage

```
create_event_selection(name, start_time)
```

Arguments

<code>name</code>	Character. Name of the event building block (required).
<code>start_time</code>	Numeric. Start time, in hours, when the event fires in the simulation (required).

Value

An [EventSelection](#) object.

Examples

```
create_event_selection(name = "GB emptying", start_time = 12)
```

`create_expression_profile`*Create a new expression profile*

Description

Create an [ExpressionProfile](#) building block from named arguments. This is a thin factory around `ExpressionProfile$new()` that builds the raw list shape for you.

An [ExpressionProfile](#)'s identity is the composite `Molecule|Species|Category`, so all three of molecule, species, and category are required.

Usage

```
create_expression_profile(
  molecule,
  species,
  category,
  type,
  localization = NULL,
  transport_type = NULL,
  ontogeny = NULL,
  parameters = NULL,
  expression = NULL,
  disease = NULL,
  description = NULL
)
```

Arguments

molecule	Character. Name of the molecule (for example "CYP3A4"). Required.
species	Character. Species (for example "Human"). Required.
category	Character. Category disambiguator (for example "Healthy" or "Variability"). Required.
type	Character. Molecule type, typically one of "Enzyme", "Transporter", or "OtherProtein". Required.
localization	Character. Expression localization for proteins.
transport_type	Character. Transporter type (for transporter profiles), for example "Efflux" or "Influx".
ontogeny	Character or list. Ontogeny name, or a raw ontogeny list. If a string is supplied it is wrapped as <code>list(Name = ontogeny)</code> .
parameters	List of Parameter objects (created with <code>create_parameter()</code>) or raw parameter lists describing relative expression and other per-organ values.
expression	Per-organ relative expression, written to the snapshot's Expression array. Supply a data frame (or tibble), one row per organ/container, with a required name column (organ/container name, for example "Liver") and the optional columns value (relative expression), compartment (for example "Intracellular"), and transport_direction (for transporter profiles). Missing (NA) or absent optional cells emit no key, so a row with only name and value produces a container with just those two fields. Row order is preserved and duplicate names are kept. Alternatively supply a raw list of container lists (each a named list with any of Name, Value, CompartmentName, TransportDirection) to pass through verbatim. NULL or an empty input writes nothing.
disease	Disease state, written to the snapshot's Disease object. Supply a named list with name (required, the DiseaseState name) and an optional parameters list of Parameter objects or raw parameter lists. NULL writes nothing.
description	Character. Free-text description of the profile.

Value

An [ExpressionProfile](#) object.

Examples

```
# Create a minimal enzyme expression profile
profile <- create_expression_profile(
  molecule = "CYP3A4",
  species = "Human",
  category = "Healthy",
  type = "Enzyme"
)

# Create a transporter expression profile with ontogeny
profile <- create_expression_profile(
  molecule = "P-gp",
  species = "Human",
  category = "Healthy",
  type = "Transporter",
  transport_type = "Efflux",
  ontogeny = "P-gp"
)

# Create an enzyme profile with per-organ relative expression
profile <- create_expression_profile(
  molecule = "CYP3A4",
  species = "Human",
  category = "Healthy",
  type = "Enzyme",
  expression = data.frame(
    name = c("Liver", "Kidney"),
    value = c(1, 0.5)
  )
)
```

create_formula_reference

Create a formula reference for an observer

Description

Build a single FormulaUsablePath entry for the References of an observer's ExplicitFormula. A formula reference binds an alias used in the formula expression to a model path. This is a small builder that returns the raw named list; assemble several of them into a list for [create_observer\(\)](#) or observer\$formula_references.

Usage

```
create_formula_reference(alias, path, dimension = NULL)
```

Arguments

alias	Character. The alias used in the formula expression (required).
path	Character. The model path the alias resolves to (required).
dimension	Character. The dimension of the referenced quantity. Omitted when NULL.

Value

A named list with Alias, Path, and, when supplied, Dimension.

Examples

```
create_formula_reference(
  "Conc_Br",
  "Organism|Brain|Plasma|Drug|Concentration"
)

# With an explicit dimension
create_formula_reference(
  "Conc_Br",
  "Organism|Brain|Plasma|Drug|Concentration",
  "Concentration (molar)"
)
```

create_formulation	<i>Create a new formulation</i>
--------------------	---------------------------------

Description

Create a new formulation with the specified properties. All arguments are optional except for name and type.

Usage

```
create_formulation(name, type, parameters = NULL)
```

Arguments

name	Character. Name of the formulation.
type	Character. Type of formulation. A curated human-readable alias or a known raw Formulation_* key resolves to that key: "Dissolved" - Immediate release solution "Weibull" - Weibull tablet formulation "Lint80" - Lint80 tablet formulation "Particle" - Particle formulation "Table" - Custom release profile table "Zero Order" - Zero-order release formulation

- "First Order" - First-order release formulation

Any other non-empty string is accepted verbatim and written straight to `FormulationType`, so a new or otherwise unknown PK-Sim template type can be authored (mirroring how `create_event()` treats `template`). The curated parameter vocabulary below only exists for the known types.

- parameters List. Accepts either of two mutually exclusive forms (they are never mixed in one call):
- Curated form: a named list drawn from the per-type alias vocabulary of a known type (documented in the sections below), with scalar/vector values. Defaults, informational messages, alias-to-Name synthesis, and the built-in Table shape apply. Invalid aliases error.
 - Raw form: a list of `Parameter` objects (built with `create_parameter()`) and/or raw `list(Name=, Value=, ...)` dicts. Each entry is written to `data$Parameters` with its `Name`, `Value`, `Unit`, `ValueOrigin`, and `TableFormula` preserved verbatim, for any type. This is the form that unlocks arbitrary real parameter names (including on `Dissolved`), per-parameter `ValueOrigin`, and a custom `TableFormula` on any type.

Only the raw form is valid for an unknown type (there is no alias vocabulary to interpret a curated-looking list against).

Value

A Formulation object

Weibull formulation parameters

- `dissolution_time` - Time to achieve 50% dissolution (numeric, default: 240)
- `dissolution_time_unit` - Unit for dissolution time (character, default: "min")
- `lag_time` - Lag time before dissolution starts (numeric, default: 0)
- `lag_time_unit` - Unit for lag time (character, default: "min")
- `dissolution_shape` - Dissolution shape parameter (numeric, default: 0.92)
- `suspension` - Whether to use as suspension (logical, default: TRUE)

Lint80 formulation parameters

- `dissolution_time` - Time to achieve 80% dissolution (numeric, default: 240)
- `dissolution_time_unit` - Unit for dissolution time (character, default: "min")
- `lag_time` - Lag time before dissolution starts (numeric, default: 0)
- `lag_time_unit` - Unit for lag time (character, default: "min")
- `suspension` - Whether to use as suspension (logical, default: TRUE)

Particle formulation parameters

- thickness - Thickness of unstirred water layer (numeric, default: 30)
- thickness_unit - Unit for thickness (character, default: "\u00b5m")
- distribution_type - Type of distribution, "mono" or "poly" (character, default: "mono")
- radius - Particle radius (mean or geometric mean) (numeric, default: 10)
- radius_unit - Unit for radius (character, default: "\u00b5m")

Parameters for polydisperse distribution (when distribution_type = "poly"):

- particle_size_distribution - "normal" or "lognormal" (character, default: "normal")
- radius_sd - Radius standard deviation, for normal distribution (numeric, default: 3)
- radius_sd_unit - Unit for radius SD (character, default: "\u00b5m")
- radius_cv - Coefficient of variation, for lognormal distribution (numeric, default: 1.5)
- radius_min - Minimum particle radius (numeric, default: 1)
- radius_min_unit - Unit for minimum radius (character, default: "\u00b5m")
- radius_max - Maximum particle radius (numeric, default: 19)
- radius_max_unit - Unit for maximum radius (character, default: "\u00b5m")
- n_bins - Number of bins (integer, default: 3)

Table formulation parameters

- tableX - Vector of time points for release profile in hours (numeric vector, required)
- tableY - Vector of fraction of dose at each time point (numeric vector, required)
- suspension - Whether to use as suspension (logical, default: TRUE)

ZeroOrder formulation parameters

- end_time - Time of administration end (numeric, default: 60)
- end_time_unit - Unit for end time (character, default: "min")

FirstOrder formulation parameters

- thalf - Half-life of the drug release (numeric, default: 0.01)
- thalf_unit - Unit for half-life (character, default: "min")

Examples

```
# Create a dissolved formulation (simplest type)
dissolved <- create_formulation(name = "Oral Solution", type = "Dissolved")

# Create a Weibull tablet formulation
tablet <- create_formulation(
  name = "Standard Tablet",
  type = "Weibull",
  parameters = list(
    dissolution_time = 60,
```

```

    dissolution_time_unit = "min",
    lag_time = 10,
    lag_time_unit = "min",
    dissolution_shape = 0.92,
    suspension = TRUE
  )
)

# Create a particle formulation with monodisperse distribution
particle <- create_formulation(
  name = "Suspension",
  type = "Particle",
  parameters = list(
    thickness = 25,
    thickness_unit = "µm",
    radius = 5,
    radius_unit = "µm"
  )
)

# Create a table formulation with custom release profile
custom <- create_formulation(
  name = "Custom Release",
  type = "Table",
  parameters = list(
    tableX = c(0, 0.5, 1, 2, 4, 8),
    tableY = c(0, 0.1, 0.3, 0.6, 0.9, 1.0),
    suspension = TRUE
  )
)

# Raw form: set an arbitrary parameter (with a ValueOrigin) on Dissolved
raw_dissolved <- create_formulation(
  name = "Suspension",
  type = "Dissolved",
  parameters = list(
    create_parameter(
      name = "Use as suspension",
      value = 1,
      source = "Lit",
      description = "Reference XYZ"
    )
  )
)

# Raw form on an unknown type carrying a custom TableFormula
custom_type <- create_formulation(
  name = "Novel",
  type = "Formulation_BrandNew",
  parameters = list(
    create_parameter(
      name = "Fraction (dose)",
      value = 0,

```

```
        table_points = list(list(x = 0, y = 0), list(x = 60, y = 1)),
        x_name = "Time",
        y_name = "Fraction",
        x_unit = "min",
        x_dimension = "Time",
        y_dimension = "Fraction"
    )
)
```

create_formulation_selection

Create a formulation selection for a simulation compound

Description

Build a [FormulationSelection](#) entry, used inside a [ProtocolSelection](#) to bind a formulation building block to one of the protocol's application slots.

Usage

```
create_formulation_selection(name, key)
```

Arguments

name	Character. Name of the formulation building block (required).
key	Character. Formulation key identifying which application in the protocol uses this formulation (required).

Value

A [FormulationSelection](#) object.

Examples

```
create_formulation_selection(name = "Oral solution", key = "Formulation")
```

create_individual	Create a new individual
-------------------	-------------------------

Description

Create a new individual with the specified properties. All arguments are optional and will use default values if not provided.

Usage

```
create_individual(
  name = "New Individual",
  species = NULL,
  population = NULL,
  gender = NULL,
  age = NULL,
  weight = NULL,
  height = NULL,
  gestational_age = NULL,
  calculation_methods = NULL,
  disease_state = NULL,
  disease_state_parameters = NULL,
  seed = NULL,
  expression_profiles = NULL,
  description = NULL,
  parameters = NULL
)
```

Arguments

name	Character. Name of the individual
species	Character. Species of the individual (must be valid ospsuite Species)
population	Character. Population of the individual (must be valid ospsuite HumanPopulation)
gender	Character. Gender of the individual (must be valid ospsuite Gender)
age	An age() object, or NULL.
weight	A weight() object, or NULL.
height	A height() object, or NULL.
gestational_age	A gestational_age() object, or NULL (for infant/preterm individuals).
calculation_methods	Character vector. Calculation methods used for the individual
disease_state	Character. Disease-state name of the individual (optional). Together with disease_state_parameters it feeds the modern Disease object (<code>{ Name, Parameters }</code>) emitted under <code>OriginData\$Disease</code> , matching create_expression_profile() .

disease_state_parameters	List. Parameters for the disease state (optional), keyed on Name in the emitted Disease object.
seed	Integer. Simulation seed (optional)
expression_profiles	Character vector of expression-profile composite names (Molecule Species Category) to attach to the individual. Default NULL (no profiles).
description	Character. Free-text description of the individual. Default NULL (no description).
parameters	List of localized parameters (each created with <code>create_parameter(path = ...)</code>), or the equivalent raw list form the loader consumes, applied as the individual's Parameters overrides. Each entry must be path-bearing. Default NULL (no parameter overrides).

Value

An Individual object

See Also

[age\(\)](#), [weight\(\)](#), [height\(\)](#), [gestational_age\(\)](#) for the demographic value-object helpers.

Examples

```
# Create a minimal individual
individual <- create_individual(name = "Test Individual")

# Create a complete individual
individual <- create_individual(
  name = "John Doe",
  species = "Human",
  population = "European_ICRP_2002",
  gender = "MALE",
  age = age(30),
  weight = weight(70),
  height = height(175)
)

# Create an individual with calculation methods
individual <- create_individual(
  name = "Test Individual",
  calculation_methods = c("Method 1", "Method 2", "Method 3")
)

# Create an individual with disease state (emitted as the modern
# `Disease` object: `OriginData$Disease = { Name, Parameters }`)
individual <- create_individual(
  name = "Patient",
  disease_state = "CKD",
  disease_state_parameters = list(
    list(Name = "eGFR", Value = 45.0, Unit = "ml/min/1.73m²")
  )
)
```

```

)

# Create an individual referencing expression profiles with a localized
# parameter and a description
individual <- create_individual(
  name = "Subject 1",
  expression_profiles = c("CYP3A4|Human|Healthy", "P-gp|Human|Healthy"),
  description = "Reference healthy adult",
  parameters = list(
    create_parameter(
      path = "Organism|Liver|EHC continuous fraction",
      value = 1
    )
  )
)

```

create_molecule_list *Create a molecule list for an observer*

Description

Build a MoleculeList for an [Observer](#), selecting which molecules the observer applies to. This is a small builder that returns the raw named list for [create_observer\(\)](#) or `observer$molecule_list`.

Usage

```
create_molecule_list(for_all = NULL, include = NULL, exclude = NULL)
```

Arguments

for_all	Logical (length 1). When TRUE, the observer applies to all molecules. Omitted when NULL.
include	Character vector of molecule names to include. Always serialises as a JSON array, even for a single name. Omitted when NULL.
exclude	Character vector of molecule names to exclude. Always serialises as a JSON array, even for a single name. Omitted when NULL.

Value

A named list with ForAll, MoleculeNamesToInclude, and MoleculeNamesToExclude for the arguments supplied (an empty named list when all are NULL).

Examples

```

create_molecule_list(for_all = FALSE, include = "Drug")

# Multiple names and an exclusion
create_molecule_list(include = c("Drug", "Metabolite"), exclude = "Water")

```

create_observed_data *Create a new observed data set*

Description

Create an `ospsuite::DataSet` from named arguments. This is a thin factory around `loadDataSetFromSnapshot()` that builds the raw snapshot observed-data shape for you and converts it into a `DataSet`.

An observed data entry is the time + value (+ optional error) series attached to a snapshot. It is referenced by name from a `Snapshot`'s simulations.

Usage

```
create_observed_data(  
  name,  
  time,  
  values,  
  error = NULL,  
  molecular_weight = NULL,  
  lloq = NULL,  
  metadata = NULL  
)
```

Arguments

name	Character. Name of the observed data set (required).
time	A <code>time()</code> object giving the time grid x-values and their unit (required).
values	A <code>values()</code> object giving the measurement y-values, their dimension, and an optional unit (required), same length as time. The dimension is supplied to <code>values()</code> and is required there.
error	A <code>error()</code> object giving the optional error y-values, an optional unit (defaulting to the values unit), and the auxiliary type. Same length as the values series. NULL for no error series.
molecular_weight	Numeric. Molecular weight to attach to the data set, in g/mol.
lloq	Numeric. Lower limit of quantification.
metadata	Named list. Extended properties (key/value pairs) stored alongside the data set.

Value

An `ospsuite::DataSet` object.

See Also

`time()`, `values()`, `error()` for the series value-object helpers.

Examples

```
# Create a minimal observed data set
obs <- create_observed_data(
  name = "Subject 001",
  time = time(c(0, 1, 2, 4, 8)),
  values = values(c(0, 12, 18, 11, 5), dimension = "Concentration (mass)")
)

# Create observed data with units and error
obs <- create_observed_data(
  name = "Subject 001",
  time = time(c(0, 1, 2, 4, 8), unit = "h"),
  values = values(
    c(0, 12, 18, 11, 5),
    unit = "mg/l",
    dimension = "Concentration (mass)"
  ),
  error = error(c(0, 1.2, 1.5, 1.1, 0.6), type = "ArithmeticStdDev")
)
```

create_observer

Create a new observer

Description

Create a single [Observer](#) from named arguments. This is a thin factory around `Observer$new()` that builds the raw list shape for you and composes with [create_observer_set\(\)](#) via `create_observer_set(observers = list(...))`.

An observer is a simulation-time formula that computes a derived quantity from the underlying model (for example an amount observer or a container observer). It is not itself a building block; it lives inside an [ObserverSet](#).

Usage

```
create_observer(
  name,
  type,
  dimension = NULL,
  formula = NULL,
  formula_references = NULL,
  container_criteria = NULL,
  molecule_list = NULL
)
```

Arguments

name Character. Name of the observer (required).

type	Character. The observer type, one of "Amount" (an AmountObserverBuilder) or "Container" (a ContainerObserverBuilder). Any other value aborts (required).
dimension	Character. The dimension of the observed quantity. Omitted when NULL.
formula	The observer's formula. Either a bare expression string (which becomes the inner Formula\$Formula), or a full ExplicitFormula raw list used as the whole Formula. Omitted when NULL.
formula_references	List of formula references, each a create_formula_reference() output or an equivalent raw {Alias, Path, Dimension} list. Written to the formula's References; supplying these without formula yields a Formula carrying only References. When formula is a full ExplicitFormula list that already carries its own References, this argument overrides them. Omitted when NULL.
container_criteria	List of container criteria, each a create_descriptor_condition() output or an equivalent raw {Tag, Type} list. Written to ContainerCriteria. Omitted when NULL.
molecule_list	A create_molecule_list() output or an equivalent raw list. Written to MoleculeList. Omitted when NULL.

Value

An [Observer](#) object.

Examples

```
# Minimal amount observer
create_observer(name = "drug_amount", type = "Amount")

# Container observer with all sub-properties
obs <- create_observer(
  name = "brain_plasma_conc",
  type = "Container",
  dimension = "Concentration (molar)",
  formula = "Conc_Br",
  formula_references = list(
    create_formula_reference(
      "Conc_Br",
      "Organism|Brain|Plasma|Drug|Concentration"
    )
  ),
  container_criteria = list(
    create_descriptor_condition("Brain", "InContainer")
  ),
  molecule_list = create_molecule_list(for_all = FALSE, include = "Drug")
)

# Compose into an observer set
create_observer_set(name = "BrainPlasmaConcentration", observers = list(obs))
```

create_observer_set	Create a new observer set
---------------------	---------------------------

Description

Create an [ObserverSet](#) building block from named arguments. This is a thin factory around `ObserverSet$new()` that builds the raw list shape for you.

An [ObserverSet](#) bundles a collection of [Observer](#) objects (each one a simulation-time formula that computes a derived quantity from the underlying model) so a simulation can reference the whole bundle by name.

Usage

```
create_observer_set(name, observers = NULL)
```

Arguments

name	Character. Name of the observer set (required).
observers	List of Observer objects or raw observer lists to include in the set. Entries that inherit from Observer contribute their underlying raw data; other entries are assumed to already be raw observer lists.

Value

An [ObserverSet](#) object.

Examples

```
# Create an empty observer set
empty_set <- create_observer_set(name = "BrainPlasmaConcentration")

# Create an observer set from raw observer lists
raw_set <- create_observer_set(
  name = "BrainPlasmaConcentration",
  observers = list(
    list(
      Name = "brain_plasma_conc",
      Type = "Container",
      Dimension = "Concentration (molar)",
      Formula = list(Formula = "Conc_Br")
    )
  )
)

# Create an observer set from Observer R6 objects
observer <- Observer$new(list(
  Name = "brain_plasma_conc",
  Type = "Container",
```

```
    Dimension = "Concentration (molar)",
    Formula = list(Formula = "Conc_Br")
  ))
observer_set <- create_observer_set(
  name = "BrainPlasmaConcentration",
  observers = list(observer)
)
```

`create_observer_set_selection`*Create an observer-set selection for a simulation*

Description

Build an [ObserverSetSelection](#) entry that references an [ObserverSet](#) building block by name inside a [Simulation](#).

Usage

```
create_observer_set_selection(name)
```

Arguments

name	Character. Name of the observer-set building block (required).
------	--

Value

An [ObserverSetSelection](#) object.

Examples

```
create_observer_set_selection(name = "BrainPlasmaConcentration")
```

`create_output_interval`*Create an output interval for a simulation output schema*

Description

Build an [OutputInterval](#) entry, used inside an [OutputSchema](#) to record one time window of output reporting (start time, end time, resolution). All three time parameters are required; PK-Sim's default units ("h" for times and "pts/h" for resolution) are used.

Usage

```
create_output_interval(name = NULL, start_time, end_time, resolution)
```

Arguments

name	Character. Optional interval name.
start_time	Numeric. Interval start time, in hours (required).
end_time	Numeric. Interval end time, in hours (required).
resolution	Numeric. Reporting resolution, in points per hour (required).

Value

An [OutputInterval](#) object.

Examples

```
create_output_interval(start_time = 0, end_time = 24, resolution = 20)
```

```
create_output_mapping
```

Create an output-to-observed-data mapping for a simulation

Description

Build an [OutputMapping](#) entry, used inside a [Simulation](#) to link a simulation output quantity to an observed-data set for downstream comparison and parameter identification.

All arguments are optional and serialized only when supplied.

Usage

```
create_output_mapping(
  path = NULL,
  observed_data = NULL,
  scaling = NULL,
  weight = NULL,
  weights = NULL
)
```

Arguments

path	Character. Simulation output quantity path.
observed_data	Character. Observed-data repository name.
scaling	Character. Scaling used for the mapping: one of "Linear", "Log".
weight	Numeric. Single weight applied to all points.
weights	Numeric vector. Per-point weights.

Value

An [OutputMapping](#) object.

Examples

```
create_output_mapping(  
  path = "Organism|PeripheralVenousBlood|Drug|Plasma",  
  observed_data = "Study A",  
  scaling = "Log",  
  weight = 1  
)
```

create_output_schema	Create an output schema for a simulation
----------------------	--

Description

Build an [OutputSchema](#) from a list of [OutputInterval](#) objects.

Usage

```
create_output_schema(intervals = list())
```

Arguments

intervals	List of OutputInterval objects (created with create_output_interval()) or raw lists. Defaults to an empty list.
-----------	--

Value

An [OutputSchema](#) object.

Examples

```
create_output_schema(  
  intervals = list(  
    create_output_interval(start_time = 0, end_time = 2, resolution = 20),  
    create_output_interval(start_time = 2, end_time = 24, resolution = 20)  
  )  
)
```

create_parameter	Create a new parameter
------------------	------------------------

Description

Create a new parameter with the specified properties. All arguments except name and value are optional.

Returns a [LocalizedParameter](#) when a non-NULL path is supplied (i.e. the parameter is identified by its position within a target’s parameter tree); otherwise returns a plain [Parameter](#).

Usage

```
create_parameter(  
    name,  
    value,  
    path = NULL,  
    unit = NULL,  
    source = NULL,  
    description = NULL,  
    source_id = NULL,  
    source_method = NULL,  
    table_formula = NULL,  
    table_points = NULL,  
    x_name = NULL,  
    y_name = NULL,  
    x_unit = NULL,  
    x_dimension = NULL,  
    y_dimension = NULL  
)
```

Arguments

name	Character. Name of the parameter.
value	Numeric. Value of the parameter.
path	Character. Full container path of the parameter within its parameter tree. Supply this to obtain a LocalizedParameter (used in Individual, ExpressionProfile, and Simulation parameter sections).
unit	Character. Unit of the parameter (optional).
source	Character. Source of the value (optional).
description	Character. Description of the value origin (optional).
source_id	Integer. ID of the source (optional).
source_method	Character. Value-origin determination method (optional), mapped to ValueOrigin.Method in the emitted snapshot. One of the ValueOriginDeterminationMethodId values: "Undefined", "Unknown", "Assumption", "ManualFit", "ParameterIdentification", "InVitro", "InVivo", or "Other".

table_formula	List. Table formula data for table parameters (optional).
table_points	List. Points for table parameters, a list of x,y pairs (optional).
x_name	Character. Name of X axis for table parameters (optional).
y_name	Character. Name of Y axis for table parameters (optional).
x_unit	Character. Unit for X axis for table parameters (optional).
x_dimension	Character. Dimension for X axis for table parameters (optional).
y_dimension	Character. Dimension for Y axis for table parameters (optional).

Value

A [Parameter](#) object, or a [LocalizedParameter](#) when path is supplied.

Data shape

Plain Parameter objects carry the identifier in data\$Name, matching the JSON shape used by Parameter slots in Compound, Formulation, Protocol, and Event (per snapshot-spec.md). LocalizedParameter objects carry the identifier in data\$Path. The factory writes whichever field matches the returned class so the raw data shape reflects the kind of parameter unambiguously.

Examples

```
# Create a basic parameter
param <- create_parameter(
  name = "Organism|Liver|Volume",
  value = 1.5
)

# Create a parameter with unit
param <- create_parameter(
  name = "Organism|Liver|Volume",
  value = 1.5,
  unit = "L"
)

# Create a parameter with value origin
param <- create_parameter(
  name = "Organism|Liver|Volume",
  value = 1.5,
  unit = "L",
  source = "Publication",
  description = "Reference XYZ",
  source_method = "ParameterIdentification"
)

# Create a localized parameter (path-bearing)
localized <- create_parameter(
  path = "Organism|Liver|Volume",
  value = 1.5,
  unit = "L"
)
```

```
# Create a table parameter
param <- create_parameter(
  name = "Fraction (dose)",
  value = 0.0,
  table_points = list(
    list(x = 0.0, y = 0.0),
    list(x = 1.0, y = 0.5),
    list(x = 2.0, y = 0.8),
    list(x = 4.0, y = 1.0)
  ),
  x_name = "Time",
  y_name = "Fraction (dose)",
  x_unit = "h",
  x_dimension = "Time",
  y_dimension = "Dimensionless"
)
```

create_population	<i>Create a new population</i>
-------------------	--------------------------------

Description

Create a [Population](#) building block from named arguments. This is a thin factory around `Population$new()` that builds the raw Settings shape for you.

A [Population](#) is a recipe for generating subjects: it owns the settings PK-Sim uses to sample a cohort at simulation time. Subjects themselves are not stored in the snapshot.

Usage

```
create_population(
  name,
  number_of_individuals,
  proportion_of_females = 50,
  individual_name = NULL,
  species = NULL,
  source_population = NULL,
  age_range = NULL,
  weight_range = NULL,
  height_range = NULL,
  bmi_range = NULL,
  seed = NULL,
  advanced_parameters = NULL,
  description = NULL,
  gestational_age_range = NULL,
  disease_state_parameters = NULL,
  individual = NULL
)
```

Arguments

name	Character. Name of the population (required).
number_of_individuals	Integer. Number of subjects to generate (required, a whole number between 2 and 10000, matching PK-Sim's population-creation bounds).
proportion_of_females	Numeric. Percentage of females in the population, between 0 and 100. Defaults to 50.
individual_name	Character. Name of the base Individual building block the population samples from.
species	Character. Species used for the base individual when individual_name is not provided.
source_population	Character. Population name used for the base individual (for example "European_ICRP_2002").
age_range	A Range object describing the age bounds (see range()).
weight_range	A Range object describing the weight bounds.
height_range	A Range object describing the height bounds.
bmi_range	A Range object describing the BMI bounds.
seed	Integer. Random seed for population generation.
advanced_parameters	List of AdvancedParameter objects or raw advanced-parameter lists that override default variability distributions.
description	Character. Free-text description of the population.
gestational_age_range	A Range object describing the gestational age range used for population generation (see range()). This is the population-settings range, distinct from the base individual's own scalar gestational age.
disease_state_parameters	A named list mapping a disease-state parameter name to a Range object, for example <code>list(eGFR = range(60, 120, "ml/min/1.73m²"))</code> . Each entry becomes one range in the population settings. This population-level name-to-Range map is distinct from the base individual's own <code>disease_state_parameters</code> , which carries scalar parameter values.
individual	An Individual object (typically from create_individual()) used as the base individual the population samples from. When supplied, it fully configures the base individual (gender, origin data, calculation methods, seed, description, parameters) and cannot be combined with individual_name, species, or source_population.

Value

A [Population](#) object.

Examples

```
# Create a minimal population
pop <- create_population(
  name = "Adults",
  number_of_individuals = 100
)

# Create a population with demographic ranges
pop <- create_population(
  name = "Healthy Adults",
  number_of_individuals = 50,
  proportion_of_females = 50,
  species = "Human",
  source_population = "European_ICRP_2002",
  age_range = range(20, 60, "year(s)",
  weight_range = range(50, 90, "kg")
)

# Compose a base individual and set a disease-state parameter
pop <- create_population(
  name = "Renal Impairment",
  number_of_individuals = 25,
  individual = create_individual(
    name = "Base",
    species = "Human",
    population = "European_ICRP_2002",
    gender = "MALE"
  ),
  gestational_age_range = range(37, 42, "week(s)",
  disease_state_parameters = list(
    eGFR = range(60, 120, "ml/min/1.73m²")
  ),
  description = "Adults with reduced renal function"
)
```

create_process

Create a new compound process

Description

Create a [Process](#) (a PK-Sim CompoundProcess) from named arguments. This is a thin factory around `Process$new()` that builds the raw list shape for you.

For richer process structures (such as parameters with full value origins), build the raw list directly or load a snapshot and copy the relevant entry.

Usage

```
create_process(
  internal_name,
```

```
    data_source,  
    molecule = NULL,  
    metabolite = NULL,  
    species = NULL,  
    parameters = NULL  
  )
```

Arguments

internal_name	Character. The PK-Sim InternalName for the process template (e.g. "SpecificBinding", "MetabolizationSpecific_MM", "GlomerularFiltration"). Required.
data_source	Character. The DataSource string identifying the process. Required.
molecule	Character. Optional molecule name for partial processes (e.g. the enzyme catalysing the reaction).
metabolite	Character. Optional metabolite name for enzymatic processes.
species	Character. Optional species name for species-dependent processes.
parameters	List of Parameter objects (created with create_parameter()) or raw parameter lists to attach to the process.

Value

A [Process](#) object.

Examples

```
# Minimal protein-binding process  
p <- create_process(  
  internal_name = "SpecificBinding",  
  data_source = "Publication X",  
  molecule = "Albumin"  
)  
  
# Metabolizing enzyme with a parameter  
p <- create_process(  
  internal_name = "MetabolizationSpecific_MM",  
  data_source = "Optimized",  
  molecule = "CYP3A4",  
  metabolite = "Hydroxy-Drug",  
  parameters = list(  
    create_parameter(name = "Km", value = 1.2, unit = "μmol/l")  
  )  
)
```

create_protocol	Create a new protocol
-----------------	-----------------------

Description

Create a [Protocol](#) building block from named arguments. This is a thin factory around `Protocol$new()` that builds the raw list shape for you.

By default this creates a Simple Protocol with a single application type and dosing interval. To create an Advanced Protocol pass a list of schemas via `schemas` instead of `application_type` and `dosing_interval`.

Usage

```
create_protocol(
  name,
  application_type = NULL,
  dosing_interval = NULL,
  target_organ = NULL,
  target_compartment = NULL,
  parameters = NULL,
  schemas = NULL,
  time_unit = NULL,
  dose = NULL,
  dose_unit = "mg",
  start_time = NULL,
  start_time_unit = "h",
  end_time = NULL
)
```

Arguments

<code>name</code>	Character. Name of the protocol (required).
<code>application_type</code>	Character. Application type for a Simple Protocol. Optional; when supplied it must be one of the canonical PK-Sim application types: "Oral", "Intravenous", "IntravenousBolus", or "UserDefined". Mutually exclusive with <code>schemas</code> .
<code>dosing_interval</code>	Character. Dosing interval identifier for a Simple Protocol. Optional; when supplied it must be one of the fixed PK-Sim <code>DosingIntervalId</code> values: "Single", "DI_6_6_6", "DI_6_6_12", "DI_8_8_8", "DI_12_12", or "DI_24".
<code>target_organ</code>	Character. Target organ for the dose. Only valid when <code>application_type</code> is "UserDefined".
<code>target_compartment</code>	Character. Target compartment for the dose. Only valid when <code>application_type</code> is "UserDefined".

parameters	List of Parameter objects (created with create_parameter()) or raw parameter lists. This is the free-form escape hatch for any Simple-Protocol parameter that has no dedicated argument. Dose, start time, and end time have the dedicated dose, start_time, and end_time arguments; supplying one of those settings both here and via its dedicated argument is an error.
schemas	List of schemas for an Advanced Protocol. Entries may be Schema objects (created with create_schema()) or raw schema lists with Name, Parameters, and SchemaItems. If provided, the protocol is created as an Advanced Protocol.
time_unit	Character. Display time unit for the protocol, validated against dimension "Time"; valid units are those in <code>ospsuite::ospUnits\$Time</code> .
dose	Numeric scalar. Optional dose for a Simple Protocol. When supplied it is emitted as a single <code>InputDose</code> parameter carrying dose and dose_unit. Mutually exclusive with schemas, and with an <code>InputDose</code> entry in parameters.
dose_unit	Character. Display unit for dose, default "mg". Validated as a dose-family unit: the unit must belong to one of the mass, amount, dose per body weight, or dose per body surface area dimensions, and the unit alone selects the family (the emitted parameter is always a single <code>InputDose</code>). There is no single <code>ospsuite::ospUnits\$Dose</code> member, so the accepted units are the union of <code>ospsuite::ospUnits\$Mass</code> , <code>ospsuite::ospUnits\$Amount</code> , <code>ospsuite::ospUnits[["Dose per body weight"]]</code> , and <code>ospsuite::ospUnits[["Dose per body surface area"]]</code> . Only consulted when dose is supplied.
start_time	Numeric scalar. Optional start time for a Simple Protocol. When supplied it is emitted as a <code>Start time</code> parameter. Mutually exclusive with schemas, and with a <code>Start time</code> entry in parameters.
start_time_unit	Character. Display unit for start_time, default "h", validated against dimension "Time"; valid units are those in <code>ospsuite::ospUnits\$Time</code> . Only consulted when start_time is supplied.
end_time	Numeric scalar. Optional end time for a Simple Protocol. When supplied it is emitted as an <code>End time</code> parameter. Its display unit is taken from time_unit, falling back to "h" when time_unit is unset; there is no separate end-time unit argument. Mutually exclusive with schemas, and with an <code>End time</code> entry in parameters.

Value

A [Protocol](#) object.

Examples

```
# Create a simple oral protocol with one dose via promoted arguments
protocol <- create_protocol(
  name = "Single dose 10mg",
  application_type = "Oral",
  dosing_interval = "Single",
  dose = 10,
  dose_unit = "mg",
  start_time = 0
```

```

)

# The same protocol via the free-form `parameters` escape hatch
protocol <- create_protocol(
  name = "Single dose 10mg",
  application_type = "Oral",
  dosing_interval = "Single",
  parameters = list(
    create_parameter(name = "Start time", value = 0, unit = "h"),
    create_parameter(name = "InputDose", value = 10, unit = "mg")
  )
)

# Create a twice-daily intravenous protocol
protocol <- create_protocol(
  name = "BID IV",
  application_type = "IntravenousBolus",
  dosing_interval = "DI_12_12",
  parameters = list(
    create_parameter(name = "InputDose", value = 5, unit = "mg")
  )
)

# Create an Advanced Protocol from Schema objects
protocol <- create_protocol(
  name = "Advanced",
  schemas = list(
    create_schema(
      name = "Schema 1",
      parameters = list(
        create_parameter(name = "NumberOfRepetitions", value = 1)
      ),
      items = list(
        create_schema_item(
          name = "Item 1",
          application_type = "Oral",
          parameters = list(
            create_parameter(name = "InputDose", value = 5, unit = "mg")
          )
        )
      )
    )
  ),
  time_unit = "h"
)

```

create_protocol_selection

Create a protocol selection for a simulation compound

Description

Build a [ProtocolSelection](#) entry, used inside a [CompoundProperties](#) to assign a [Protocol](#) building block to the compound and (optionally) map each application slot to a [Formulation](#) via [FormulationSelection](#).

Usage

```
create_protocol_selection(name, formulations = NULL)
```

Arguments

name	Character. Name of the protocol building block (required).
formulations	List of FormulationSelection objects (created with create_formulation_selection()) or raw lists.

Value

A [ProtocolSelection](#) object.

Examples

```
create_protocol_selection(  
  name = "Yu 2004 - Rifampicin - 600 mg MD OD 10 days",  
  formulations = list(  
    create_formulation_selection(name = "Oral solution", key = "Formulation")  
  )  
)
```

create_schema

Create a new schema

Description

Create a [Schema](#) for an Advanced [Protocol](#) from named arguments. This is a thin factory around `Schema$new()` that builds the raw list shape for you.

A schema is a repeatable block inside an Advanced [Protocol](#): it has a name, schema-level parameters (typically `NumberOfRepetitions`, `TimeBetweenRepetitions`, and `Start time`), and an ordered list of [SchemaItem](#) applications. The three repetition parameters are available as the plain, unit-aware arguments `number_of_repetitions`, `time_between_repetitions`, and `start_time`; anything else is supplied through the parameters escape hatch.

Usage

```

create_schema(
  name,
  parameters = NULL,
  items = NULL,
  number_of_repetitions = NULL,
  time_between_repetitions = NULL,
  time_between_repetitions_unit = "h",
  start_time = NULL,
  start_time_unit = "h"
)

```

Arguments

name	Character. Name of the schema (required).
parameters	List of Parameter objects (created with create_parameter()) or raw parameter lists. This is the escape hatch for any schema-level parameter not promoted to a plain argument. Each promoted argument is mutually exclusive with a matching entry here (an entry named "NumberOfRepetitions", "TimeBetweenRepetitions", or "Start time"): supply a repetition parameter either as the plain argument or as a parameters entry, not both.
items	List of SchemaItem objects (created with create_schema_item()) or raw schema item lists. These define the applications inside the schema.
number_of_repetitions	Numeric. Optional count of schema repetitions (NumberOfRepetitions). A single finite whole number (dimensionless, no unit). Mutually exclusive with a "NumberOfRepetitions" entry in parameters.
time_between_repetitions	Numeric. Optional time between repetitions (TimeBetweenRepetitions). A single finite numeric. Mutually exclusive with a "TimeBetweenRepetitions" entry in parameters.
time_between_repetitions_unit	Character. Display unit for time_between_repetitions, default "h", validated against the "Time" dimension.
start_time	Numeric. Optional schema start time (Start time). A single finite numeric. Mutually exclusive with a "Start time" entry in parameters.
start_time_unit	Character. Display unit for start_time, default "h", validated against the "Time" dimension.

Value

A [Schema](#) object.

Examples

```
# A schema using the promoted repetition arguments
```

```

schema <- create_schema(
  name = "Schema 1",
  number_of_repetitions = 3,
  time_between_repetitions = 24,
  start_time = 0
)

# A once-daily schema with a single oral application, using the
# `parameters` escape hatch for the schema-level parameters
schema <- create_schema(
  name = "Schema 1",
  parameters = list(
    create_parameter(name = "NumberOfRepetitions", value = 1),
    create_parameter(name = "TimeBetweenRepetitions", value = 0, unit = "h")
  ),
  items = list(
    create_schema_item(
      name = "Item 1",
      application_type = "Oral",
      parameters = list(
        create_parameter(name = "InputDose", value = 5, unit = "mg")
      )
    )
  )
)

```

create_schema_item	Create a new schema item
--------------------	--------------------------

Description

Create a [SchemaItem](#) for an Advanced [Protocol](#) from named arguments. This is a thin factory around `SchemaItem$new()` that builds the raw list shape for you.

A schema item is one application inside a [Schema](#) of an Advanced Protocol. It carries an application type, an optional formulation key, target organ and compartment, and the application-level parameters (dose, start time, ...).

Usage

```

create_schema_item(
  name,
  application_type,
  formulation_key = NULL,
  target_organ = NULL,
  target_compartment = NULL,
  parameters = NULL,
  dose = NULL,
  dose_unit = "mg",
  start_time = NULL,

```

```

    start_time_unit = "h"
  )

```

Arguments

name	Character. Name of the schema item (required).
application_type	Character. Application type for the schema item (required). Must be one of the canonical PK-Sim application types: "Oral", "Intravenous", "IntravenousBolus", or "UserDefined".
formulation_key	Character. Formulation key linking the schema item to a formulation selection in the owning simulation.
target_organ	Character. Target organ for the application. Only valid when application_type is "UserDefined".
target_compartment	Character. Target compartment for the application. Only valid when application_type is "UserDefined".
parameters	List of Parameter objects (created with create_parameter()) or raw parameter lists. These become the application-level parameters (dose, start time, ...). The promoted dose and start_time arguments flow into the same "InputDose" and "Start time" parameters, so supplying a setting both as a promoted argument and as the matching parameters entry is an error.
dose	Numeric scalar dose for the application, written as a single InputDose parameter. NULL (default) emits no dose parameter. The dose family (plain dose, per body weight, or per body surface area) is selected by dose_unit.
dose_unit	Character. Unit for dose, default "mg". Must be a dose-family unit (a Mass, Amount, dose-per-body-weight, or dose-per-body-surface-area unit, for example "mg", "mg/kg", or "mg/m ² "). Consulted only when dose is supplied.
start_time	Numeric scalar application start time, written as a "Start time" parameter. NULL (default) emits no start-time parameter. Zero and negative values are allowed.
start_time_unit	Character. Unit for start_time, default "h", validated against the "Time" dimension. Consulted only when start_time is supplied.

Value

A [SchemaItem](#) object.

Examples

```

# An oral schema item using the promoted dose and start-time arguments
item <- create_schema_item(
  name = "Item 1",
  application_type = "Oral",
  dose = 10,
  dose_unit = "mg",

```

```

    start_time = 0
  )

  # The same, authored through the free-form `parameters` escape hatch
  item <- create_schema_item(
    name = "Item 1",
    application_type = "Oral",
    parameters = list(
      create_parameter(name = "Start time", value = 0, unit = "h"),
      create_parameter(name = "InputDose", value = 10, unit = "mg")
    )
  )

  # A user-defined schema item targeting the venous blood plasma
  item <- create_schema_item(
    name = "User defined",
    application_type = "UserDefined",
    target_organ = "VenousBlood",
    target_compartment = "Plasma",
    dose = 5
  )

```

create_snapshot

Create an empty snapshot

Description

Create an in-memory [Snapshot](#) from scratch, carrying the current supported PK-Sim version and no building blocks. This is the snapshot-level constructor that pairs with [load_snapshot\(\)](#) and [export_snapshot\(\)](#): rather than loading an existing project, it starts an empty one you can then populate.

The result touches no files and has no path. Mutate it with the `add_*`() verbs (for example [add_compound\(\)](#)) and serialize it with [export_snapshot\(\)](#).

Usage

```
create_snapshot(name = NULL, description = NULL)
```

Arguments

name	Character. Optional snapshot name. When supplied, it is written to the snapshot's Name; when NULL, no Name is set. An empty string "" is written verbatim rather than treated as absent.
description	Character. Optional snapshot description. Follows the same NULL-versus-supplied and empty-string semantics as name.

Value

A [Snapshot](#) object.

See Also

[load_snapshot\(\)](#), [export_snapshot\(\)](#)

Examples

```
# An empty snapshot
snapshot <- create_snapshot()

# A named and described snapshot
snapshot <- create_snapshot(name = "My Project", description = "Notes")

# Populate it with a building block
snapshot <- add_compound(create_snapshot(), create_compound(name = "Drug X"))
```

create_solver_settings

Create a solver settings object for a simulation

Description

Build a [SolverSettings](#) object, used inside a [Simulation](#). All arguments are optional; absent fields fall back to PK-Sim solver defaults at load time.

Usage

```
create_solver_settings(
  abs_tol = NULL,
  rel_tol = NULL,
  use_jacobian = NULL,
  h0 = NULL,
  h_min = NULL,
  h_max = NULL,
  mx_step = NULL
)
```

Arguments

abs_tol	Numeric. Absolute solver tolerance.
rel_tol	Numeric. Relative solver tolerance.
use_jacobian	Logical. Whether to use the Jacobian during integration.
h0	Numeric. Initial step size.
h_min	Numeric. Minimum step size.
h_max	Numeric. Maximum step size.
mx_step	Integer. Maximum number of internal solver steps.

Value

A [SolverSettings](#) object.

Examples

```
create_solver_settings()
```

```
create_solver_settings(abs_tol = 1e-9, rel_tol = 1e-9, mx_step = 100000)
```

error	<i>Error series value object</i>
-------	----------------------------------

Description

Build an error-series value object for [create_observed_data\(\)](#)'s error argument. Carries the error y-values, an optional unit, and an auxiliary type.

The unit is not validated here: the error dimension is the values-series dimension, which this helper does not see. [create_observed_data\(\)](#) validates the error unit against the values dimension when a unit is present, and defaults the error unit to the values unit when unit is NULL.

Usage

```
error(value, unit = NULL, type = "ArithmeticStdDev")
```

Arguments

value	Numeric vector. Error y-values, same length as the values series.
unit	Character. Optional unit for the error. Defaults (in the factory) to the values-series unit when NULL.
type	Character. Auxiliary type for the error, one of the schema AuxiliaryType values: "ArithmeticStdDev" (the default), "GeometricStdDev", "ArithmeticMeanPop", or "GeometricMeanPop".

Value

An error_spec object to pass to [create_observed_data\(\)](#).

See Also

[create_observed_data\(\)](#)

Other value-object helpers: [age\(\)](#), [fraction_unbound\(\)](#), [gestational_age\(\)](#), [height\(\)](#), [intestinal_permeability\(\)](#), [lipophilicity\(\)](#), [permeability\(\)](#), [solubility\(\)](#), [time\(\)](#), [values\(\)](#), [weight\(\)](#)

Examples

```
error(c(0, 1.2, 1.5, 1.1, 0.6))
```

Event

Event class for OSP snapshot events

Description

An R6 class that represents an event in an OSP snapshot. This class provides methods to access different properties of an event and display a summary of its information.

Active bindings

`data` The raw data of the event (read-only)

`name` The name of the event. Writable: must be a non-empty scalar string.

`template` The template of the event. Writable: must be a non-empty scalar string.

`parameters` The list of parameter objects. Writable: must be a list, or NULL to clear.

Methods

Public methods:

- [Event\\$new\(\)](#)
- [Event\\$print\(\)](#)
- [Event\\$to_dataframe\(\)](#)
- [Event\\$clone\(\)](#)

`Event$new()`: Create a new Event object

Usage:

```
Event$new(data)
```

Arguments:

`data` Raw event data from a snapshot

Returns: A new Event object

`Event$print()`: Print a summary of the event

Usage:

```
Event$print(...)
```

Arguments:

`...` Additional arguments passed to print methods

Returns: Invisibly returns the object

`Event$to_dataframe()`: Convert the event to a data frame for easier manipulation

Usage:

```
Event$to_dataframe()
```

Returns: A tibble with event information

Event\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

EventSelection

EventSelection class for Simulation event references

Description

An R6 class representing one entry in a [Simulation](#)'s Events array. Resolves to an [Event](#) building block by name and carries the simulation-local StartTime [Parameter](#) used by the event.

Active bindings

data The raw data of the selection (read-only). Rebuilt from the cached [Parameter](#) so that mutations on \$start_time flow back to the export payload.

name The name of the event building block. Writable: must be a non-empty scalar string.

start_time The [Parameter](#) used as start time for the event.

Methods

Public methods:

- [EventSelection\\$new\(\)](#)
- [EventSelection\\$clone\(\)](#)

EventSelection\$new(): Create a new EventSelection object.

Usage:

```
EventSelection$new(data)
```

Arguments:

data Raw EventSelection data from a snapshot.

Returns: A new EventSelection object.

EventSelection\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

export_snapshot	<i>Export a snapshot to a JSON file</i>
-----------------	---

Description

Export a Snapshot object to a JSON file. This is a convenient wrapper around the `$export()` method of the Snapshot class.

Usage

```
export_snapshot(snapshot, path)
```

Arguments

snapshot	A Snapshot object
path	Character string. Path where to save the JSON file

Value

Invisibly returns the snapshot object

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("path/to/snapshot.json")

# Export to a new file
export_snapshot(snapshot, "path/to/exported_snapshot.json")

## End(Not run)
```

ExpressionProfile	<i>ExpressionProfile class for OSP snapshot expression profiles</i>
-------------------	---

Description

An R6 class that represents an expression profile in an OSP snapshot. This class provides methods to access different properties of an expression profile and display a summary of its information.

Active bindings

data The raw data of the expression profile (read-only).
type The type of the expression profile
species The species of the expression profile
molecule The molecule name of the expression profile
category The category of the expression profile
localization The localization of the expression profile
transportType The transport type of the expression profile
ontogeny The ontogeny information of the expression profile
parameters The parameters of the expression profile
expression The per-organ relative expression of the profile. Reading returns the raw ExpressionContainer[] list (or NULL when unset). Assigning accepts the same shapes as the expression argument of [create_expression_profile\(\)](#) (a data frame of container rows, a raw list, or NULL/empty to clear it).
disease The disease state of the profile. Reading returns the raw DiseaseState list (or NULL when unset). Assigning accepts the same shape as the disease argument of [create_expression_profile\(\)](#) (a named list with name and optional parameters, or NULL to clear it).
id The unique identifier of the expression profile

Methods**Public methods:**

- [ExpressionProfile\\$new\(\)](#)
- [ExpressionProfile\\$print\(\)](#)
- [ExpressionProfile\\$to_df\(\)](#)
- [ExpressionProfile\\$clone\(\)](#)

ExpressionProfile\$new(): Create a new ExpressionProfile object

Usage:

ExpressionProfile\$new(data)

Arguments:

data Raw expression profile data from a snapshot

Returns: A new ExpressionProfile object

ExpressionProfile\$print(): Print a summary of the expression profile

Usage:

ExpressionProfile\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the object

`ExpressionProfile$to_df()`: Convert the expression profile to data frames for easier manipulation

Usage:

`ExpressionProfile$to_df()`

Returns: A list with basic information and parameters as tibbles

`ExpressionProfile$clone()`: The objects of this class are cloneable with this method.

Usage:

`ExpressionProfile$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Formulation

Formulation class for OSP snapshot formulations

Description

An R6 class that represents a formulation in an OSP snapshot. This class provides methods to access different properties of a formulation and display a summary of its information.

Active bindings

`data` The raw data of the formulation (read-only)

`name` The name of the formulation

`formulation_type` The formulation type identifier

`parameters` The list of parameter objects. Writable: must be a list, or NULL to clear.

Methods

Public methods:

- `Formulation$new()`
- `Formulation$print()`
- `Formulation$to_df()`
- `Formulation$get_human_formulation_type()`
- `Formulation$clone()`

`Formulation$new()`: Create a new Formulation object

Usage:

`Formulation$new(data)`

Arguments:

`data` Raw formulation data from a snapshot

Returns: A new Formulation object

`Formulation$print()`: Print a summary of the formulation including its properties and parameters.

Usage:

```
Formulation$print(...)
```

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the Formulation object for method chaining

`Formulation$to_df()`: Convert formulation data to tibbles

Usage:

```
Formulation$to_df(type = "all")
```

Arguments:

type Character. Type of data to convert: "all" (default) or "parameters"

Returns: A list of tibbles containing the requested data:

- `formulations`: Basic formulation information (ID, name, type)
- `formulations_parameters`: All parameters including table parameter points

The `formulations_parameters` tibble includes the following columns:

- `formulation_id`: ID of the formulation
- `name`: Name of the parameter
- `value`: Value of the parameter (NA for table points)
- `unit`: Unit of the parameter (NA for table points)
- `is_table_point`: TRUE for table parameter points, FALSE for regular parameters
- `x_value`: X-axis value for table points (NA for regular parameters)
- `y_value`: Y-axis value for table points (NA for regular parameters)
- `table_name`: Name of the table (usually "Time" for release profiles)
- `source`: Source of the parameter (NA if not available)
- `description`: Description of the parameter (NA if not available)
- `source_id`: ID of the source (NA if not available)

`Formulation$get_human_formulation_type()`: Get human-readable formulation type

Usage:

```
Formulation$get_human_formulation_type()
```

Returns: Character string with human-readable formulation type

`Formulation$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

FormulationSelection	<i>FormulationSelection class for Simulation compound formulations</i>
----------------------	--

Description

An R6 class representing one entry in a [ProtocolSelection](#)'s Formulations array. Each entry maps a formulation building-block name to a formulation key used by an application within the protocol.

Active bindings

data The raw data of the selection (read-only).

name The name of the formulation building block. Writable: must be a non-empty scalar string.

key The formulation key (application slot) inside the protocol. Writable: must be a non-empty scalar string.

Methods

Public methods:

- [FormulationSelection\\$new\(\)](#)
- [FormulationSelection\\$clone\(\)](#)

`FormulationSelection$new()`: Create a new `FormulationSelection` object.

Usage:

`FormulationSelection$new(data)`

Arguments:

data Raw `FormulationSelection` data from a snapshot.

Returns: A new `FormulationSelection` object.

`FormulationSelection$clone()`: The objects of this class are cloneable with this method.

Usage:

`FormulationSelection$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

fraction_unbound	<i>Fraction unbound value object</i>
------------------	--------------------------------------

Description

Build a fraction-unbound value object for `create_compound()`'s `fraction_unbound` argument (or the writable `Compound$fraction_unbound` field). Produces one default `FractionUnbound` alternative with parameter "Fraction unbound (plasma, reference value)". Fraction unbound stores a bare value with no unit, so this helper takes no unit argument.

Fraction unbound is the one species-dependent single-parameter group, so the emitted alternative carries a `Species` field (see `species`), which PK-Sim requires to load the snapshot.

Usage

```
fraction_unbound(
  value,
  name = "User defined",
  default = FALSE,
  species = "Human"
)
```

Arguments

value	Numeric scalar. Fraction-unbound value.
name	Character. Name of the created alternative. Defaults to "User defined".
default	Logical. Whether this alternative is the group's default when it appears in a list of alternatives passed to the matching <code>create_compound()</code> argument or <code>Compound</code> field. Defaults to <code>FALSE</code> . Ignored for a single (non-list) value object, which is always the default. When a list has no element marked <code>default = TRUE</code> , the first element is the default (unchanged behaviour); marking two or more elements <code>default = TRUE</code> in the same list is an error.
species	Character. PK-Sim species emitted as the alternative's <code>Species</code> field. Defaults to "Human" and is validated against <code>ospsuite::Species</code> . Fraction unbound is species-dependent, and PK-Sim rejects a snapshot whose fraction-unbound alternative has no resolvable species, so this is emitted on every fraction-unbound alternative. A user hand-building a raw alternative list (the <code>Compound\$fraction_unbound</code> escape hatch) is responsible for its own <code>Species</code> .

Value

A `fraction_unbound_spec` object to pass to `create_compound()`.

See Also

`create_compound()`

Other value-object helpers: `age()`, `error()`, `gestational_age()`, `height()`, `intestinal_permeability()`, `lipophilicity()`, `permeability()`, `solubility()`, `time()`, `values()`, `weight()`

Examples

```

fraction_unbound(0.1)
create_compound(name = "Drug X", fraction_unbound = fraction_unbound(0.1))

# Several alternatives, the second marked as the default
create_compound(
  name = "Drug X",
  fraction_unbound = list(
    fraction_unbound(0.1, name = "Plasma"),
    fraction_unbound(0.2, name = "Microsomal", default = TRUE)
  )
)

```

gestational_age	<i>Gestational age value object</i>
-----------------	-------------------------------------

Description

Build a gestational-age value object for `create_individual()`'s `gestational_age` argument (or the writable `Individual$gestational_age` / `OriginData$gestational_age` field). Fills the `OriginData` `GestationalAge` characteristic (`list(Value =, Unit =)`).

Usage

```
gestational_age(value, unit = "week(s)")
```

Arguments

value	Numeric scalar. Gestational-age value.
unit	Character. Unit for gestational age, validated against dimension "Time". Defaults to "week(s)".

Value

A `gestational_age_spec` object to pass to `create_individual()`.

See Also

`create_individual()`

Other value-object helpers: `age()`, `error()`, `fraction_unbound()`, `height()`, `intestinal_permeability()`, `lipophilicity()`, `permeability()`, `solubility()`, `time()`, `values()`, `weight()`

Examples

```

gestational_age(38)
create_individual(name = "Preterm", gestational_age = gestational_age(30))

```

get_compounds_dfs	<i>Get all compounds in a snapshot as data frames</i>
-------------------	---

Description

Thin wrapper around [as_tibbles\(\)](#) with kind = "compounds". Prefer [as_tibbles\(\)](#) in new code.

Usage

```
get_compounds_dfs(snapshot)
```

Arguments

snapshot	A Snapshot object.
----------	--------------------

Value

A list with properties and processes tibbles; see [as_tibbles\(\)](#) for the column contract.

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
dfs <- get_compounds_dfs(snapshot)
dfs$properties
dfs$processes

## End(Not run)
```

get_default_alternative	<i>Get the default alternative of a compound property</i>
-------------------------	---

Description

Returns the name of the default alternative in a compound physicochemical-property group, for example the Solubility, Lipophilicity, or FractionUnbound field of a [Compound](#). The default is the alternative whose IsDefault flag is TRUE (an absent flag is treated as TRUE, per the snapshot schema); when no alternative sets it, the "User defined" alternative is used, and a group with no resolvable default returns NULL.

Usage

```
get_default_alternative(alternatives)
```

Arguments

alternatives A compound physicochemical-property group, such as `compound$solubility`, `compound$lipophilicity`, or any list of named alternatives each carrying an optional `IsDefault` flag.

Value

The default alternative's name as a character string, or `NULL` when the group has no resolvable default.

Examples

```
compound <- create_compound(
  name = "Drug X",
  solubility = list(
    solubility(9999, name = "Aqueous"),
    solubility(200, name = "FaSSIF", default = TRUE)
  )
)
get_default_alternative(compound$solubility)
```

get_events_dfs

Get all events in a snapshot as data frames

Description

Thin wrapper around `as_tibbles()` with `kind = "events"`. Prefer `as_tibbles()` in new code.

Usage

```
get_events_dfs(snapshot)
```

Arguments

snapshot A Snapshot object.

Value

A list with events and `events_parameters` tibbles.

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
dfs <- get_events_dfs(snapshot)

## End(Not run)
```

`get_expression_profiles_dfs`*Get all expression profiles in a snapshot as data frames*

Description

Thin wrapper around `as_tibbles()` with `kind = "expression_profiles"`. Prefer `as_tibbles()` in new code.

Usage

```
get_expression_profiles_dfs(snapshot)
```

Arguments

`snapshot` A Snapshot object.

Value

A list with `expression_profiles` and `expression_profiles_parameters` tibbles.

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
dfs <- get_expression_profiles_dfs(snapshot)

## End(Not run)
```

`get_formulations_dfs` *Get all formulations in a snapshot as data frames*

Description

Thin wrapper around `as_tibbles()` with `kind = "formulations"`. Prefer `as_tibbles()` in new code.

Usage

```
get_formulations_dfs(snapshot)
```

Arguments

`snapshot` A Snapshot object.

Value

A list with formulations and formulations_parameters tibbles. Table parameter points have `is_table_point = TRUE` and carry `x_value`, `y_value`, and `table_name`.

Examples

```
## Not run:  
snapshot <- load_snapshot("path/to/snapshot.json")  
dfs <- get_formulations_dfs(snapshot)  
  
## End(Not run)
```

get_individuals_dfs	<i>Get all individuals in a snapshot as data frames</i>
---------------------	---

Description

Thin wrapper around `as_tibbles()` with `kind = "individuals"`. Prefer `as_tibbles()` in new code.

Usage

```
get_individuals_dfs(snapshot)
```

Arguments

`snapshot` A Snapshot object.

Value

A list with `individuals`, `individuals_parameters`, and `individuals_expressions` tibbles.

Examples

```
## Not run:  
snapshot <- load_snapshot("path/to/snapshot.json")  
dfs <- get_individuals_dfs(snapshot)  
  
## End(Not run)
```

get_observed_data_dfs *Get all observed data in a snapshot as a tibble*

Description

Thin wrapper around `as_tibbles()` with `kind = "observed_data"`. Prefer `as_tibbles()` in new code.

Usage

```
get_observed_data_dfs(snapshot)
```

Arguments

snapshot A Snapshot object.

Value

A tibble in long format with columns `name`, `xValues`, `yValues`, `yErrorValues`, `xDimension`, `xUnit`, `yDimension`, `yUnit`, `yErrorType`, `yErrorUnit`, `molWeight`, `lloq`.

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
observed_data_df <- get_observed_data_dfs(snapshot)

## End(Not run)
```

get_observer_sets_dfs *Get all observer sets in a snapshot as data frames*

Description

Thin wrapper around `as_tibbles()` with `kind = "observer_sets"`. Prefer `as_tibbles()` in new code.

Usage

```
get_observer_sets_dfs(snapshot)
```

Arguments

snapshot A Snapshot object.

Value

A list with two tibbles. `observer_sets` has one row per `ObserverSet` with columns `observer_set_id`, `name`, `n_observers`. `observers` has one row per `Observer` with columns `observer_set_id`, `observer_set_name`, `name`, `type`, `dimension`, `formula_expression`, `formula_dimension`, `formula_references`, `container_tags`; rows join back to their parent `ObserverSet` by `observer_set_id` or `observer_set_name`. `formula_references` flattens the underlying `ExplicitFormula` references to "alias=path" pairs joined with |.

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
observer_sets_df <- get_observer_sets_dfs(snapshot)

## End(Not run)
```

get_populations_dfs	<i>Get all populations in a snapshot as data frames</i>
---------------------	---

Description

Thin wrapper around `as_tibbles()` with `kind = "populations"`. Prefer `as_tibbles()` in new code.

Usage

```
get_populations_dfs(snapshot)
```

Arguments

`snapshot` A Snapshot object.

Value

A list with `populations` and `populations_parameters` tibbles.

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
dfs <- get_populations_dfs(snapshot)

## End(Not run)
```

get_protocols_dfs	<i>Get all protocols in a snapshot as a single consolidated data frame</i>
-------------------	--

Description

Thin wrapper around `as_tibbles()` with `kind = "protocols"`. Prefer `as_tibbles()` in new code.

Usage

```
get_protocols_dfs(snapshot)
```

Arguments

snapshot A Snapshot object.

Value

A tibble with one row per protocol parameter (or per schema item parameter for advanced protocols).

Examples

```
## Not run:
snapshot <- load_snapshot("path/to/snapshot.json")
protocols_df <- get_protocols_dfs(snapshot)

## End(Not run)
```

height	<i>Height value object</i>
--------	----------------------------

Description

Build a height value object for `create_individual()`'s height argument (or the writable `Individual$height / OriginData$height` field). Fills the `OriginData Height` characteristic (`list(Value =, Unit =)`).

Usage

```
height(value, unit = "cm")
```

Arguments

value Numeric scalar. Height value.
unit Character. Unit for height, validated against dimension "Length". Defaults to "cm".

Value

A height_spec object to pass to `create_individual()`.

See Also

`create_individual()`

Other value-object helpers: `age()`, `error()`, `fraction_unbound()`, `gestational_age()`, `intestinal_permeability()`, `lipophilicity()`, `permeability()`, `solubility()`, `time()`, `values()`, `weight()`

Examples

```
height(175)
create_individual(name = "Subject", height = height(175))
```

Individual	<i>Individual class for OSP snapshot individuals</i>
------------	--

Description

An R6 class that represents an individual in an OSP snapshot. This class provides methods to access different properties of an individual and display a summary of its information.

Active bindings

`data` The raw data of the individual (read-only)

`name` The name of the individual. Writable: must be a non-empty scalar string.

`seed` The simulation seed for the individual

`origin_data` The [OriginData](#) object holding species, population, gender, physiological parameters, and calculation methods.

`species` The species of the individual

`population` The population of the individual

`gender` The gender of the individual

`age` The age value of the individual. Writable: assign an [age\(\)](#) object; see [OriginData](#)'s age field for the full contract (a bare numeric scalar is rejected).

`age_unit` The age unit of the individual

`weight` The weight value of the individual. Writable: assign a [weight\(\)](#) object; see [OriginData](#)'s weight field for the full contract (a bare numeric scalar is rejected).

`weight_unit` The weight unit of the individual

`height` The height value of the individual. Writable: assign a [height\(\)](#) object; see [OriginData](#)'s height field for the full contract (a bare numeric scalar is rejected).

`height_unit` The height unit of the individual

`gestational_age` The gestational age value of the individual. Writable: assign a `gestational_age()` object; see `OriginData`'s `gestational_age` field for the full contract (a bare numeric scalar is rejected).

`gestational_age_unit` The gestational age unit of the individual

`disease_state` The disease state of the individual

`disease_state_parameters` The disease state parameters of the individual

`parameters` The list of parameter objects with a custom print method. Writable: must be a list, or NULL to clear.

`calculation_methods` The calculation methods of the individual, returned as a character vector for backwards compatibility. Use `$origin_data$calculation_methods` to access the `CalculationMethods` object directly.

`expression_profiles` Read/write. The expression profiles attached to the individual, as a character vector of composite names (Molecule|Species|Category) or NULL when none are attached. Assigning a character vector replaces the attached profiles; assigning NULL or `character(0)` removes them (no empty array is serialized).

`description` Read/write. Free-text description of the individual, or NULL when none is set. Assigning a length-1 character sets it; assigning NULL removes it.

Methods

Public methods:

- `Individual$new()`
- `Individual$print()`
- `Individual$to_df()`
- `Individual$clone()`

`Individual$new()`: Create a new `Individual` object

Usage:

`Individual$new(data)`

Arguments:

`data` Raw individual data from a snapshot

Returns: A new `Individual` object

`Individual$print()`: Print a summary of the individual including its properties and parameters.

Usage:

`Individual$print(...)`

Arguments:

`...` Additional arguments passed to print methods

Returns: Invisibly returns the `Individual` object for method chaining

`Individual$to_df()`: Convert individual data to tibbles

Usage:

`Individual$to_df(type = "all")`

Arguments:

type Character. Type of data to convert: "all" (default), "individuals", "individuals_parameters", or "individuals_expressions"

Returns: A list of tibbles containing the requested data

Individual\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

intestinal_permeability

Intestinal permeability value object

Description

Build an intestinal-permeability value object for `create_compound()`'s `intestinal_permeability` argument (or the writable `Compound$intestinal_permeability` field). Produces one default `IntestinalPermeability` alternative with parameter "Specific intestinal permeability (transcellular)".

Usage

```
intestinal_permeability(
  value,
  unit = "cm/min",
  name = "User defined",
  default = FALSE
)
```

Arguments

value	Numeric scalar. Intestinal-permeability value.
unit	Character. Unit for the parameter, validated against dimension "Velocity". Defaults to "cm/min".
name	Character. Name of the created alternative. Defaults to "User defined".
default	Logical. Whether this alternative is the group's default when it appears in a list of alternatives passed to the matching <code>create_compound()</code> argument or <code>Compound</code> field. Defaults to FALSE. Ignored for a single (non-list) value object, which is always the default. When a list has no element marked <code>default = TRUE</code> , the first element is the default (unchanged behaviour); marking two or more elements <code>default = TRUE</code> in the same list is an error.

Value

An `intestinal_permeability_spec` object to pass to `create_compound()`.

See Also[create_compound\(\)](#)

Other value-object helpers: [age\(\)](#), [error\(\)](#), [fraction_unbound\(\)](#), [gestational_age\(\)](#), [height\(\)](#), [lipophilicity\(\)](#), [permeability\(\)](#), [solubility\(\)](#), [time\(\)](#), [values\(\)](#), [weight\(\)](#)

Examples

```
intestinal_permeability(1.14e-05)
create_compound(
  name = "Drug X",
  intestinal_permeability = intestinal_permeability(1.14e-05)
)

# Several alternatives, the second marked as the default
create_compound(
  name = "Drug X",
  intestinal_permeability = list(
    intestinal_permeability(1.14e-05, name = "Caco-2"),
    intestinal_permeability(2e-05, name = "PAMPA", default = TRUE)
  )
)
```

lipophilicity	<i>Lipophilicity value object</i>
---------------	-----------------------------------

Description

Build a lipophilicity value object for [create_compound\(\)](#)'s lipophilicity argument (or the writable Compound\$lipophilicity field). Produces one default Lipophilicity alternative with parameter "Lipophilicity".

Usage

```
lipophilicity(
  value,
  unit = "Log Units",
  name = "User defined",
  default = FALSE
)
```

Arguments

value	Numeric scalar. Lipophilicity value.
unit	Character. Unit for the lipophilicity parameter, validated against dimension "Log Units". Defaults to "Log Units".
name	Character. Name of the created alternative. Defaults to "User defined".

default Logical. Whether this alternative is the group's default when it appears in a list of alternatives passed to the matching `create_compound()` argument or Compound field. Defaults to FALSE. Ignored for a single (non-list) value object, which is always the default. When a list has no element marked default = TRUE, the first element is the default (unchanged behaviour); marking two or more elements default = TRUE in the same list is an error.

Value

A lipophilicity_spec object to pass to `create_compound()`.

See Also

`create_compound()`

Other value-object helpers: `age()`, `error()`, `fraction_unbound()`, `gestational_age()`, `height()`, `intestinal_permeability()`, `permeability()`, `solubility()`, `time()`, `values()`, `weight()`

Examples

```
lipophilicity(2.5)
create_compound(name = "Drug X", lipophilicity = lipophilicity(2.5))

# Several alternatives, the second marked as the default
create_compound(
  name = "Drug X",
  lipophilicity = list(
    lipophilicity(2.5, name = "Measured"),
    lipophilicity(3.1, name = "Predicted", default = TRUE)
  )
)
```

load_snapshot

Load a snapshot from various sources

Description

Conveniently load an OSP snapshot from a local file, URL, or predefined template name.

Usage

```
load_snapshot(source, upgrade = FALSE)
```

Arguments

source Character string. Can be:

- Path to a local file (.json)
- URL to a remote snapshot file
- Name of a template from the OSPSuite.BuildingBlockTemplates repository

upgrade	Logical, default FALSE. When TRUE, a below-floor snapshot (Version 74-78) is migrated up to the version the installed PK-Sim core emits via a round trip through ospsuite (slow, several minutes, and requires a compatible installed core). When FALSE, such a snapshot reports how to migrate and does not load. In-band snapshots (Version 79-81) are never migrated regardless of this argument.
---------	--

Details

Available templates can be listed with `osp_models()`.

Value

A Snapshot object

Examples

```
## Not run:
# Load from local file
snapshot <- load_snapshot("path/to/local/snapshot.json")

# Load from URL
snapshot <- load_snapshot("https://example.com/snapshot.json")

# Load a predefined template by name
snapshot <- load_snapshot("Midazolam")

## End(Not run)
```

loadDataSetFromSnapshot

Load DataSet from OSP snapshot observed data

Description

Creates a DataSet object (from ospsuite package) from observed data in an OSP snapshot. This function converts snapshot observed data format to the standardized DataSet format used throughout the OSP ecosystem.

Usage

```
loadDataSetFromSnapshot(observedDataStructure)
```

Arguments

observedDataStructure	Raw observed data structure from a snapshot JSON
-----------------------	--

Details

This function follows the same pattern as other DataSet loading functions in `ospsuite`:

- `ospsuite::loadDataSetFromPKML()`
- `ospsuite::loadDataSetFromExcel()`
- `osp.snapshots::loadDataSetFromSnapshot()` (this function)

`ObservedData` is a backwards-compatible alias for `loadDataSetFromSnapshot()` and may be retired in a future release if this functionality migrates to the `ospsuite` package. Prefer the `loadDataSetFromSnapshot()` name in new code.

Value

A DataSet object from the `ospsuite` package

Migration Plan

This function is designed for eventual migration to the `ospsuite` package. When migrated, the usage pattern will be:

```
# Future usage (when moved to ospsuite):
dataset <- ospsuite::loadDataSetFromSnapshot(observedDataStructure)

# Current usage:
dataset <- osp.snapshots::loadDataSetFromSnapshot(observedDataStructure)

# Migration steps:
# 1. Copy this function to ospsuite package
# 2. Update osp.snapshots to import from ospsuite
# 3. Eventually remove from osp.snapshots
```

LocalizedParameter	<i>LocalizedParameter class for path-bearing OSP snapshot parameters</i>
--------------------	--

Description

An R6 class that represents a Localized parameter in an OSP snapshot. A Localized parameter is a [Parameter](#) identified by its full path within a target's parameter tree (typically a Simulation or Individual). The path locates where the override applies.

`LocalizedParameter` inherits from [Parameter](#); everything that works on a `Parameter` also works on a `LocalizedParameter`. The only addition is a required `path` field, populated from the `Path` element of the raw data.

Snapshot version assumption

osp.snapshots only supports v11+ snapshots (numeric Version ≥ 79); the check is enforced by Snapshot\$new() at load time. In v11+ snapshots, path segments named Applications are migrated to Events. The migration runs in initialize(); for v11+ data it is a no-op because PK-Sim already uses Events, and the constructor itself does not re-check the version.

Legacy Name fallback

A v11+ snapshot's localized parameter always carries Path. As a convenience for hand-rolled data, initialize() falls back to data\$Name when data\$Path is missing and emits a deprecation warning so the substitution is not silent; the resulting object stores the value in data\$Path and drops data\$Name. Real snapshots never trigger this branch.

Super class

Parameter -> LocalizedParameter

Methods

Public methods:

- LocalizedParameter\$new()
- LocalizedParameter\$clone()

LocalizedParameter\$new(): Create a new LocalizedParameter object.

Requires a non-empty path supplied through data\$Path. If data\$Path is missing but data\$Name is present, the name is used as the path and a deprecation warning is emitted; data\$Name is then cleared so the resulting raw shape is unambiguous. Path segments named Applications are rewritten to Events (legacy migration kept for robustness; no-op on v11+ data).

Usage:

LocalizedParameter\$new(data)

Arguments:

data Raw parameter data from a snapshot. Must contain a Path.

Returns: A new LocalizedParameter object.

LocalizedParameter\$clone(): The objects of this class are cloneable with this method.

Usage:

LocalizedParameter\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

Observer

Observer class for OSP snapshot observer sets

Description

An R6 class that represents a single observer inside an [ObserverSet](#). An observer is a simulation-time formula that computes a derived quantity from the underlying model (for example an amount observer or a container observer), used to expose values that are not natural model outputs.

In an OSP snapshot, each observer is one entry of the `Observers` array nested inside an `ObserverSets` building block. An observer is not itself a building block; it lives inside an `ObserverSet`.

Active bindings

`data` The raw `Observer` list as it appears in the snapshot JSON (read-only).

`name` The name of the observer. Writable: must be a non-empty scalar string.

`type` The observer type. PK-Sim recognises "Amount" (an `AmountObserverBuilder`) and "Container" (a `ContainerObserverBuilder`). On assignment, a non-NULL value is validated against these two; assigning NULL removes the type.

`dimension` The dimension of the observed quantity, resolved in PK-Sim via `IDimensionRepository.DimensionByName()`. Writable: must be a non-empty scalar string, or NULL to clear. There is no closed set of dimension names to validate against, so a required string is the strongest check available.

`formula` The full `ExplicitFormula` object backing the observer, as a list with `Name`, `Formula` (the expression string), `Dimension`, and `References` (a list of `FormulaUsablePath` entries, each with `Alias`, `Path`, and `Dimension`). NULL when the observer carries no formula. Setting this field replaces the whole structure; pass NULL to drop it. For the inner expression string or dimension only, use `formula_expression` or `formula_dimension`.

`formula_expression` The expression string of the underlying `ExplicitFormula` (`Formula$Formula` in the snapshot JSON). Setting writes back to the inner `Formula$Formula` field and preserves the sibling `Name`, `Dimension`, and `References` entries. To build a complete `ExplicitFormula` (including `Name` and `References`), assign to `formula` instead.

`formula_dimension` The output dimension of the underlying `ExplicitFormula` (`Formula$Dimension` in the snapshot JSON), resolved in PK-Sim via `IDimensionRepository.DimensionByName()`. Setting writes back to the inner `Formula$Dimension` field.

`formula_references` The `References` list of the underlying `ExplicitFormula`, where each entry is a named list with `Alias`, `Path`, and `Dimension`. Assign a list of `create_formula_reference()` outputs (or raw `{Alias, Path, Dimension}` lists) to replace it, preserving the sibling `Formula$Name`, `Formula$Formula`, and `Formula$Dimension` (the `Formula` is created if absent). Assign NULL to remove the references; when no `Formula` exists this is a no-op that does not create an empty `Formula`.

`container_tags` The `Tag` values from the observer's `ContainerCriteria`, joined with `|`. There is no `ContainerPath` field in the snapshot JSON; this binding is synthesized from the tags found in `ContainerCriteria`. NULL when the observer carries no container criteria.

- container_criteria** The full `ContainerCriteria` list, each entry a named list with `Tag` and (optionally) `Type`. Unlike `container_tags`, this preserves each condition's `Type` verbatim (including non-enum values such as `"MatchTag"`). Assign a list of `create_descriptor_condition()` outputs (or raw `{Tag, Type}` lists) to replace it; assign `NULL` to remove it.
- molecule_list** The full `MoleculeList` object (`ForAll`, `MoleculeNamesToInclude`, `MoleculeNamesToExclude`) or `NULL`. Assign a `create_molecule_list()` output (or an equivalent raw list) to replace it; assign `NULL` to remove it.

Methods

Public methods:

- `Observer$new()`
- `Observer$print()`
- `Observer$to_df()`
- `Observer$clone()`

Observer\$new(): Create a new `Observer` object.

Usage:

```
Observer$new(data = list())
```

Arguments:

data Raw `Observer` list from a snapshot. May be `NULL` or an empty list, both of which create an empty observer.

Returns: A new `Observer` object

Observer\$print(): Print a short summary of the observer.

Usage:

```
Observer$print(...)
```

Arguments:

... Additional arguments (unused).

Returns: Invisibly returns the `Observer` object.

Observer\$to_df(): Convert the observer to a single-row tibble suitable for the tibble-layer exporter. Columns are `name`, `type`, `dimension`, `formula_expression`, `formula_dimension`, `formula_references`, and `container_tags`. `formula_references` collapses each `FormulaUsablePath` entry (`Alias`, `Path`, `Dimension`) to `"alias=path"` and joins entries with `|`; `NA` when the observer carries no references.

Usage:

```
Observer$to_df()
```

Returns: A tibble with one row.

Observer\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
observer <- Observer$new(list(
  Name = "brain_plasma_conc",
  Type = "Container",
  Dimension = "Concentration (molar)",
  Formula = list(
    Name = "brain_plasma_conc_formula",
    Formula = "Conc_Br",
    Dimension = "Concentration (molar)",
    References = list(list(
      Alias = "Conc_Br",
      Path = "Organism|Brain|Plasma|Drug|Concentration",
      Dimension = "Concentration (molar)"
    ))
  )
))
observer$name
observer$type
observer$formula_expression
observer$formula_dimension
observer$formula_references
```

ObserverSet

ObserverSet class for OSP snapshot observer sets

Description

An R6 class that represents an ObserverSet building block in an OSP snapshot. An ObserverSet is a named bundle of [Observer](#) objects that simulations can reference by name.

Active bindings

data The raw data of the observer set, refreshed from the wrapped [Observer](#) objects so mutations through the R6 surface flow back into the snapshot payload (read-only).

name The name of the observer set. Writable: must be a non-empty scalar string.

observers A named list of [Observer](#) objects keyed by each observer's \$name. Duplicate names are disambiguated with `_{n}` suffixes. Assigning accepts either a list of [Observer](#) objects or a list of raw observer dicts; both are normalised to [Observer](#) objects and the underlying raw data is kept in sync.

Methods

Public methods:

- [ObserverSet\\$new\(\)](#)
- [ObserverSet\\$print\(\)](#)
- [ObserverSet\\$clone\(\)](#)

ObserverSet\$new(): Create a new ObserverSet object

Usage:

ObserverSet\$new(data)

Arguments:

data Raw observer set data from a snapshot

Returns: A new ObserverSet object

ObserverSet\$print(): Print a summary of the observer set

Usage:

ObserverSet\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the object

ObserverSet\$clone(): The objects of this class are cloneable with this method.

Usage:

ObserverSet\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

ObserverSetSelection *ObserverSetSelection class for Simulation observer-set references*

Description

An R6 class representing one entry in a [Simulation](#)'s ObserverSets array. Resolves to an [ObserverSet](#) building block by name.

Active bindings

data The raw data of the selection (read-only).

name The name of the observer-set building block. Writable: must be a non-empty scalar string.

Methods

Public methods:

- [ObserverSetSelection\\$new\(\)](#)
- [ObserverSetSelection\\$clone\(\)](#)

ObserverSetSelection\$new(): Create a new ObserverSetSelection object.

Usage:

ObserverSetSelection\$new(data)

Arguments:

data Raw ObserverSetSelection data from a snapshot.

Returns: A new ObserverSetSelection object.

ObserverSetSelection\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

OriginData

OriginData class for OSP snapshot individuals

Description

An R6 class that represents the demographic starting point of an [Individual](#): species, population, gender, and the physiological parameters (age, gestational age, weight, height) that PK-Sim uses when creating the subject. The set of calculation methods used while deriving the individual is exposed as a [CalculationMethods](#) object. Optional disease state metadata is preserved as-is for round-trip fidelity.

In an OSP snapshot, the JSON object is named OriginData and lives under each entry of the Individuals array.

Active bindings

data The raw OriginData list as it appears in the snapshot JSON, refreshed from the embedded [CalculationMethods](#) object (read-only).

species The species of the individual.

population The population of the individual.

gender The gender of the individual.

age Numeric age value of the individual (in age_unit). Writable: assign an [age\(\)](#) object to set the value and unit together. A bare numeric scalar is rejected; use [age\(\)](#) instead.

age_unit Unit string for age.

gestational_age Numeric gestational age value (in gestational_age_unit), used for preterm individuals. Writable: assign a [gestational_age\(\)](#) object to set the value and unit together. A bare numeric scalar is rejected; use [gestational_age\(\)](#) instead.

gestational_age_unit Unit string for gestational_age.

weight Numeric weight value of the individual (in weight_unit). Writable: assign a [weight\(\)](#) object to set the value and unit together. A bare numeric scalar is rejected; use [weight\(\)](#) instead.

weight_unit Unit string for weight.

- height** Numeric height value of the individual (in **height_unit**). Writable: assign a [height\(\)](#) object to set the value and unit together. A bare numeric scalar is rejected; use [height\(\)](#) instead.
- height_unit** Unit string for height.
- calculation_methods** A [CalculationMethods](#) object holding the calculation methods PK-Sim applies while creating the individual.
- disease_state** Optional disease-state name. Read from the modern Disease object (`Disease$Name`) or, for a loaded legacy snapshot, the legacy `DiseaseState` key. Writing sets the modern `Disease$Name`.
- disease_state_parameters** Optional list of disease-state parameters. Read from the modern Disease object (`Disease$Parameters`) or, for a loaded legacy snapshot, the legacy `DiseaseStateParameters` key. Writing sets the modern `Disease$Parameters`.

Methods

Public methods:

- [OriginData\\$new\(\)](#)
- [OriginData\\$print\(\)](#)
- [OriginData\\$clone\(\)](#)

OriginData\$new(): Create a new OriginData object

Usage:

```
OriginData$new(data = list())
```

Arguments:

data Raw OriginData list from a snapshot. May be NULL or an empty list, both of which create an empty OriginData.

Returns: A new OriginData object

OriginData\$print(): Print a summary of the origin data

Usage:

```
OriginData$print(...)
```

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the OriginData object

OriginData\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

osp_models

Browse available OSPSuite building block templates

Description

Browse and display all available building block templates from the OSPSuite.BuildingBlockTemplates repository. This function helps users discover what templates are available for loading with `load_snapshot()`.

Usage

```
osp_models(pattern = NULL)
```

Arguments

pattern Character string. Optional pattern to filter template names (case insensitive). Uses regular expression matching.

Value

A data.frame with template information (invisibly), while also printing a formatted list to the console.

Examples

```
## Not run:
# Browse all available templates
osp_models()

# Filter templates containing "midazolam"
osp_models(pattern = "midazolam")

# Get the data frame of templates for programmatic use
templates_df <- osp_models()

## End(Not run)
```

OutputInterval

OutputInterval class for Simulation output schemas

Description

An R6 class representing one entry in an [OutputSchema](#)'s Intervals array. Each interval carries optional name and the three [Parameters](#) Start time, End time, and Resolution.

Active bindings

data The raw data of the interval (read-only). Rebuilt from the cached [Parameter](#) list so that mutations flow back to the export payload.

name The interval name.

parameters Named list of [Parameter](#) objects, keyed by name.

Methods**Public methods:**

- [OutputInterval\\$new\(\)](#)
- [OutputInterval\\$clone\(\)](#)

OutputInterval\$new(): Create a new OutputInterval object.

Usage:

`OutputInterval$new(data)`

Arguments:

data Raw OutputInterval data from a snapshot.

Returns: A new OutputInterval object.

OutputInterval\$clone(): The objects of this class are cloneable with this method.

Usage:

`OutputInterval$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

OutputMapping

OutputMapping class for simulation output-to-observed-data mappings

Description

An R6 class representing one entry in a [Simulation](#)'s OutputMappings array. Each entry maps a simulation output quantity path to an observed-data repository name and records the scaling and weight(s) used when comparing the two.

Active bindings

data The raw data of the mapping (read-only).

path The simulation output quantity path. Writable: must be a non-empty scalar string, or NULL to clear.

observed_data The observed-data repository name. Writable: must be a non-empty scalar string, or NULL to clear.

scaling The scaling used for the mapping. Writable: one of "Linear", "Log", or NULL to clear.

weight A single weight applied to all points of the mapping. Writable: a single numeric value, or NULL to clear.

weights A per-point weight vector. Writable: a numeric vector, or NULL to clear.

Methods**Public methods:**

- [OutputMapping\\$new\(\)](#)
- [OutputMapping\\$clone\(\)](#)

[OutputMapping\\$new\(\)](#): Create a new [OutputMapping](#) object.

Usage:

`OutputMapping$new(data)`

Arguments:

`data` Raw [OutputMapping](#) data from a snapshot.

Returns: A new [OutputMapping](#) object.

[OutputMapping\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

`OutputMapping$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

OutputSchema

OutputSchema class for Simulation output schedules

Description

An R6 class representing the [OutputSchema](#) field of a [Simulation](#), a collection of [OutputIntervals](#). The raw JSON shape is a bare array of intervals (no enclosing object); the `intervals` active binding exposes the wrapped list directly.

Active bindings

`data` The raw data of the schema (read-only). Rebuilt from the cached [OutputInterval](#) list so that mutations flow back to the export payload. Returns a bare list of interval named lists.

`intervals` List of [OutputInterval](#) objects.

Methods**Public methods:**

- [OutputSchema\\$new\(\)](#)
- [OutputSchema\\$clone\(\)](#)

[OutputSchema\\$new\(\)](#): Create a new [OutputSchema](#) object.

Usage:

`OutputSchema$new(data)`

Arguments:

`data` Raw OutputSchema data from a snapshot. May be a bare list of interval named lists or NULL for an empty schema.

Returns: A new OutputSchema object.

`OutputSchema$clone()`: The objects of this class are cloneable with this method.

Usage:

`OutputSchema$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Parameter

Parameter class for OSP snapshot parameters

Description

An R6 class that represents a parameter in an OSP snapshot. This class provides methods to access different properties of a parameter and display a summary of its information.

Active bindings

`data` The raw data of the parameter (read-only).

`path` The path of the parameter

`name` The name of the parameter (same as path)

`value` The value of the parameter

`unit` The unit of the parameter (if any). Writable: a non-NULL value must be a single non-empty character string, or NULL to clear. A bare `Parameter` carries no dimension, so a dimension-aware `validate_unit()` check is not universally applicable here; only the shape is enforced.

`value_origin` The origin information for the parameter value

`table_formula` The table formula data for table parameters

Methods**Public methods:**

- `Parameter$new()`
- `Parameter$print()`
- `Parameter$to_df()`
- `Parameter$clone()`

`Parameter$new()`: Create a new Parameter object

Usage:

`Parameter$new(data)`

Arguments:

data Raw parameter data from a snapshot

Returns: A new Parameter object

Parameter\$print(): Print a summary of the parameter

Usage:

Parameter\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the object

Parameter\$to_df(): Convert parameter data to a tibble row

Usage:

Parameter\$to_df()

Returns: A tibble with one row containing the parameter data

Parameter\$clone(): The objects of this class are cloneable with this method.

Usage:

Parameter\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

permeability

Permeability value object

Description

Build a permeability value object for `create_compound()`'s permeability argument (or the writable Compound\$permeability field). Produces one default Permeability alternative with parameter "Permeability".

Usage

```
permeability(value, unit = "cm/min", name = "User defined", default = FALSE)
```

Arguments

value	Numeric scalar. Permeability value.
unit	Character. Unit for the parameter, validated against dimension "Velocity". Defaults to "cm/min".
name	Character. Name of the created alternative. Defaults to "User defined".
default	Logical. Whether this alternative is the group's default when it appears in a list of alternatives passed to the matching <code>create_compound()</code> argument or Compound field. Defaults to FALSE. Ignored for a single (non-list) value object, which is always the default. When a list has no element marked default = TRUE, the first element is the default (unchanged behaviour); marking two or more elements default = TRUE in the same list is an error.

Value

A permeability_spec object to pass to `create_compound()`.

See Also

`create_compound()`

Other value-object helpers: `age()`, `error()`, `fraction_unbound()`, `gestational_age()`, `height()`, `intestinal_permeability()`, `lipophilicity()`, `solubility()`, `time()`, `values()`, `weight()`

Examples

```
permeability(0.0069)
create_compound(name = "Drug X", permeability = permeability(0.0069))

# Several alternatives, the second marked as the default
create_compound(
  name = "Drug X",
  permeability = list(
    permeability(0.0069, name = "Measured"),
    permeability(0.008, name = "Predicted", default = TRUE)
  )
)
```

Population

Population class for OSP snapshot populations

Description

An R6 class that represents a population in an OSP snapshot. This class provides methods to access different properties of a population and display a summary of its information.

Active bindings

`data` The raw data of the population (read-only)

`name` The name of the population. Writable: must be a non-empty scalar string.

`description` A free-text description of the population

`source_population` The source population name (read-only)

`individual_name` The individual name (read-only)

`individual` The base individual the population samples from, returned as an [Individual](#) object built from the base individual in the population settings, or NULL when none is set (read-only). Author it at construction time via the `individual` argument of `create_population()`.

`seed` The seed used for population generation

`number_of_individuals` The number of individuals in the population. Writable: must be a single positive whole number.

`proportion_of_females` The percentage of females in the population

age_range The age range for the population. Writable: the Range's unit must be a valid unit for dimension "Age in years".

weight_range The weight range for the population. Writable: the Range's unit must be a valid unit for dimension "Mass".

height_range The height range for the population. Writable: the Range's unit must be a valid unit for dimension "Length".

bmi_range The BMI range for the population. Writable: when the "BMI" dimension resolves in the installed opsuite, the Range's unit must be a valid unit for it; otherwise the unit is left unvalidated (a documented, opportunistic limitation).

gestational_age_range The gestational age range used for population generation. Writable: the Range's unit must be a valid unit for dimension "Time".

egfr_range The eGFR range for the population (if available). Convenience over the case-insensitive "egfr" entry of `disease_state_parameters`; writes persist into the population settings so they survive export.

disease_state_parameters The population disease-state parameters as a named list mapping each parameter name to a [Range](#) object. Assign a named list of Range objects to replace them, or NULL to clear them. This population-level name-to-Range map is distinct from the base individual's own `disease_state_parameters`, which carries scalar parameter values.

advanced_parameters Advanced parameters for the population

Methods

Public methods:

- [Population\\$new\(\)](#)
- [Population\\$print\(\)](#)
- [Population\\$to_df\(\)](#)
- [Population\\$clone\(\)](#)

Population\$new(): Create a new Population object

Usage:

```
Population$new(data)
```

Arguments:

data Raw population data from a snapshot

Returns: A new Population object

Population\$print(): Print a summary of the population including its properties and settings.

Usage:

```
Population$print(...)
```

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the Population object for method chaining

Population\$to_df(): Convert the population to a list of data frames for easier analysis

Usage:

Population\$to_df()

Returns: A list containing data frames with population information:

- characteristics: Population characteristics including basic information and physiological parameters
- parameters: Advanced parameters information

Population\$clone(): The objects of this class are cloneable with this method.

Usage:

Population\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

```
print.biliary_clearance
```

Print method for biliary clearance processes

Description

Print method for biliary clearance processes

Usage

```
## S3 method for class 'biliary_clearance'
print(x, ...)
```

Arguments

x	A biliary_clearance object
...	Additional arguments (unused)

```
print.compound_additional_parameters
```

Print method for compound additional parameters

Description

Print method for compound additional parameters

Usage

```
## S3 method for class 'compound_additional_parameters'
print(x, ...)
```

Arguments

- x A compound_additional_parameters object
- ... Additional arguments (unused)

print.compound_processes
<i>S3 print method for compound processes</i>

Description

S3 print method for compound processes

Usage

```
## S3 method for class 'compound_processes'  
print(x, ...)
```

Arguments

- x A compound_processes object
- ... Additional arguments passed to print methods

Value

Invisibly returns the compound processes

print.hepatic_clearance
<i>Print method for hepatic clearance processes</i>

Description

Print method for hepatic clearance processes

Usage

```
## S3 method for class 'hepatic_clearance'  
print(x, ...)
```

Arguments

- x A hepatic_clearance object
- ... Additional arguments (unused)

print.induction	<i>Print method for induction processes</i>
-----------------	---

Description

Print method for induction processes

Usage

```
## S3 method for class 'induction'  
print(x, ...)
```

Arguments

x	An induction object
...	Additional arguments (unused)

print.inhibition	<i>Print method for inhibition processes</i>
------------------	--

Description

Print method for inhibition processes

Usage

```
## S3 method for class 'inhibition'  
print(x, ...)
```

Arguments

x	An inhibition object
...	Additional arguments (unused)

```
print.metabolizing_enzymes
```

Print method for metabolizing enzymes

Description

Print method for metabolizing enzymes

Usage

```
## S3 method for class 'metabolizing_enzymes'  
print(x, ...)
```

Arguments

x	A metabolizing_enzymes object
...	Additional arguments (unused)

```
print.parameter_collection
```

Print method for parameter collections

Description

Print method for parameter collections

Usage

```
## S3 method for class 'parameter_collection'  
print(x, ...)
```

Arguments

x	A parameter_collection object
...	Additional arguments passed to print methods

Value

Invisibly returns the object

```
print.physicochemical_property
```

S3 print method for physicochemical properties

Description

S3 print method for physicochemical properties

Usage

```
## S3 method for class 'physicochemical_property'
print(x, ...)
```

Arguments

x	A physicochemical_property object
...	Additional arguments passed to print methods

Value

Invisibly returns the physicochemical property

```
print.protein_binding_partners
```

Print method for protein binding partners

Description

Print method for protein binding partners

Usage

```
## S3 method for class 'protein_binding_partners'
print(x, ...)
```

Arguments

x	A protein_binding_partners object
...	Additional arguments (unused)

```
print.renal_clearance
```

Print method for renal clearance processes

Description

Print method for renal clearance processes

Usage

```
## S3 method for class 'renal_clearance'
print(x, ...)
```

Arguments

x	A renal_clearance object
...	Additional arguments (unused)

```
print.snapshot_collection
```

Generic print method for snapshot collections

Description

Shared S3 print method that renders any building-block collection: a header with the kind and item count, one bullet per item with a per-kind summary, and a friendly message when the collection is empty. Per-item summary lines and per-kind labels are dispatched to internal generics `collection_kind_info()` and `collection_item_label()`, each with one method per collection class.

Usage

```
## S3 method for class 'snapshot_collection'
print(x, n = 5, ...)
```

Arguments

x	A snapshot collection (a snapshot_collection named list).
n	Maximum number of items to display before truncating with "... and X more". Honoured only when the collection's <code>collection_kind_info()</code> method returns <code>truncate = TRUE</code> ; the default is to show every item. Currently only the <code>observed_data_collection</code> opts in.
...	Additional arguments passed to print methods.

Value

Invisibly returns the collection.

```
print.transporter_proteins
```

Print method for transporter proteins

Description

Print method for transporter proteins

Usage

```
## S3 method for class 'transporter_proteins'  
print(x, ...)
```

Arguments

x	A transporter_proteins object
...	Additional arguments (unused)

Process	<i>Process class for OSP snapshot compound processes</i>
---------	--

Description

An R6 class that represents a single compound process in an OSP snapshot.

A Process wraps one entry of a Compound's Processes array (PK-Sim CompoundProcess). The PK-Sim JSON does not bucket processes by subtype; subtypes are inferred from internal_name. The derived category field surfaces that bucketing for R consumers: "protein_binding_partners", "metabolizing_enzymes", "hepatic_clearance", "transporter_proteins", "renal_clearance", "biliary_clearance", "inhibition", "induction", or NA when the internal_name does not match any known pattern.

Active bindings

data The raw underlying data list (read-only).

internal_name The PK-Sim InternalName (process template key). Writable: must be a non-empty scalar string.

data_source The DataSource string identifying the process. Writable: must be a non-empty scalar string.

molecule Optional Molecule field (for partial processes). Writable: a non-empty scalar string when supplied, or NULL to clear.

metabolite Optional Metabolite field (for enzymatic processes). Writable: a non-empty scalar string when supplied, or NULL to clear.

species Optional Species field (for species-dependent processes). Writable: a non-empty scalar string when supplied, or NULL to clear. This is a free molecule/species string (not the Individual/OriginData species enum), so `validate_species()` does not apply here.

parameters A named list of [Parameter](#) objects (one entry per Parameter in the snapshot JSON), keyed by parameter name. Assigning accepts either a list of [Parameter](#) objects or raw parameter dicts; the underlying raw data and the [Parameter](#) cache are kept in sync.

category Derived category string. One of "protein_binding_partners", "metabolizing_enzymes", "hepatic_clearance", "transporter_proteins", "renal_clearance", "biliary_clearance", "inhibition", "induction", or NA_character_.

Methods

Public methods:

- [Process\\$new\(\)](#)
- [Process\\$print\(\)](#)
- [Process\\$clone\(\)](#)

Process\$new(): Create a new Process object.

Usage:

```
Process$new(data)
```

Arguments:

data Raw process data (a CompoundProcess entry from a snapshot).

Returns: A new Process object.

Process\$print(): Print a short summary of the process.

Usage:

```
Process$print(...)
```

Arguments:

... Additional arguments (unused).

Returns: Invisibly returns the Process object.

Process\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

Protocol

*Protocol class for OSP snapshot protocols***Description**

An R6 class that represents a protocol in an OSP snapshot. This class provides methods to access different properties of a protocol and display a summary of its information. Protocols can be either simple (with dosing intervals) or advanced (with schemas and schema items).

An Advanced Protocol's schemas are exposed as a named list of [Schema](#) objects, each of which owns a list of [SchemaItem](#) objects. Simple Protocols expose their single application directly through the `application_type`, `dosing_interval`, and `parameters` fields.

Active bindings

`data` The raw data of the protocol, refreshed from the wrapped [Schema](#) objects so mutations through the R6 surface flow back into the snapshot payload (read-only).

`name` The name of the protocol. Writable: must be a non-empty scalar string.

`is_advanced` Whether the protocol is advanced (schema-based)

`application_type` The application type (for simple protocols). Writable: must be one of the canonical PK-Sim application types (see [create_protocol\(\)](#)'s `application_type` argument), or NULL to clear.

`dosing_interval` The dosing interval (for simple protocols). Writable: must be one of the fixed PK-Sim dosing intervals (see [create_protocol\(\)](#)'s `dosing_interval` argument), or NULL to clear.

`time_unit` The time unit for the protocol. Writable: must be a valid unit for dimension "Time", or NULL to clear.

`parameters` The parameters of the protocol (for simple protocols). Writable: must be a list, or NULL to clear.

`schemas` A named list of [Schema](#) objects for advanced protocols. Names are taken from each schema's `name` field (duplicates are disambiguated by `make.unique`). Simple protocols return an empty list.

Methods**Public methods:**

- [Protocol\\$new\(\)](#)
- [Protocol\\$print\(\)](#)
- [Protocol\\$to_df\(\)](#)
- [Protocol\\$get_human_application_type\(\)](#)
- [Protocol\\$get_human_dosing_interval\(\)](#)
- [Protocol\\$clone\(\)](#)

`Protocol$new()`: Create a new Protocol object

Usage:

```
Protocol$new(data)
```

Arguments:

data Raw protocol data from a snapshot

Returns: A new Protocol object

Protocol\$print(): Print a summary of the protocol including its properties and parameters.

Usage:

```
Protocol$print(...)
```

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the Protocol object for method chaining

Protocol\$to_df(): Convert protocol data to a single consolidated tibble. Advanced protocols emit one row per [SchemaItem](#) and join back to the protocol via protocol_name. Simple protocols emit a single row.

Usage:

```
Protocol$to_df()
```

Returns: A tibble containing all protocol data in a single data frame

Protocol\$get_human_application_type(): Get human-readable application type

Usage:

```
Protocol$get_human_application_type()
```

Returns: Character string with human-readable application type

Protocol\$get_human_dosing_interval(): Get human-readable dosing interval

Usage:

```
Protocol$get_human_dosing_interval()
```

Returns: Character string with human-readable dosing interval

Protocol\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

ProtocolSelection	<i>ProtocolSelection class for Simulation compound protocol assignments</i>
-------------------	---

Description

An R6 class representing the Protocol field of a [CompoundProperties](#). References a [Protocol](#) building block by name and lists per-application [FormulationSelections](#) mapping formulation keys to formulation building blocks.

Active bindings

data The raw data of the selection (read-only). Rebuilt from the cached [FormulationSelection](#) list so that mutations flow back to the export payload.

name The name of the protocol building block. Writable: must be a non-empty scalar string.

formulations List of [FormulationSelection](#) objects.

Methods

Public methods:

- [ProtocolSelection\\$new\(\)](#)
- [ProtocolSelection\\$clone\(\)](#)

ProtocolSelection\$new(): Create a new ProtocolSelection object.

Usage:

```
ProtocolSelection$new(data)
```

Arguments:

data Raw ProtocolSelection data from a snapshot.

Returns: A new ProtocolSelection object.

ProtocolSelection\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

range*Create a range object for physiological parameters*

Description

Create a new [Range](#) object with the specified min, max, and unit values. Either min or max can be NULL to represent ranges with only an upper or lower bound. min and max can also be equal to represent a single value.

Usage

```
range(min, max, unit)
```

Arguments

min	Numeric or NULL. Minimum value, can be NULL for ranges with only an upper bound.
max	Numeric or NULL. Maximum value, can be NULL for ranges with only a lower bound.
unit	Character. Unit for the range.

Value

A [Range](#) object.

Examples

```
## Not run:
# Create a weight range with both bounds
weight_range <- range(60, 90, "kg")

# Create a weight range with only a lower bound
weight_min_range <- range(60, NULL, "kg")

# Create a weight range with only an upper bound
weight_max_range <- range(NULL, 90, "kg")

# Create a weight range with equal min and max (representing a single value)
weight_fixed_range <- range(70, 70, "kg")

# Assign to a population
population$weight_range <- weight_range

## End(Not run)
```

Range-class

*Range class for physiological parameters***Description**

An R6 class that represents a range of values with a minimum, maximum and unit. Used for age, weight, height, and BMI ranges in populations. Either min or max can be NULL to represent ranges with only an upper or lower bound.

Use `range()` as the user-facing constructor; `Range$new()` is the equivalent direct call on the R6 generator.

Active bindings

`min` The minimum value of the range

`max` The maximum value of the range

`unit` The unit of the range. Range is dimension-agnostic (it holds no dimension of its own), so `unit` is not validated here; dimension-aware unit validation happens at the `Population` range setters that consume a `Range` (`age_range`, `weight_range`, `height_range`, `gestational_age_range`, `bmi_range`), where the field's dimension is known. Do not add dimension validation to this binding.

Methods**Public methods:**

- `Range$new()`
- `Range$print()`
- `Range$clone()`

`Range$new()`: Create a new Range object

Usage:

`Range$new(min, max, unit)`

Arguments:

`min` Numeric or NULL. Minimum value, can be NULL for ranges with only an upper bound

`max` Numeric or NULL. Maximum value, can be NULL for ranges with only a lower bound

`unit` Character. Unit for the range

Returns: A new Range object

`Range$print()`: Print a summary of the range

Usage:

`Range$print(...)`

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the Range object for method chaining

Range\$clone(): The objects of this class are cloneable with this method.

Usage:

Range\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

remove_compound	<i>Remove compounds from a snapshot</i>
-----------------	---

Description

Remove one or more compounds from a [Snapshot](#) by name.

Usage

```
remove_compound(snapshot, compound_name)
```

Arguments

snapshot A [Snapshot](#) object.

compound_name Character vector of compound names to remove.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam") |>
  remove_compound("Midazolam")

## End(Not run)
```

remove_event	<i>Remove events from a snapshot</i>
--------------	--------------------------------------

Description

Remove one or more events from a [Snapshot](#) by name.

Usage

```
remove_event(snapshot, event_name)
```

Arguments

snapshot	A Snapshot object.
event_name	Character vector of event names to remove.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:  
snapshot <- load_snapshot("Midazolam") |>  
  remove_event("Breakfast")  
  
## End(Not run)
```

remove_expression_profile	<i>Remove expression profiles from a snapshot</i>
---------------------------	---

Description

Remove one or more expression profiles from a Snapshot by ID (molecule|species|category).

Usage

```
remove_expression_profile(snapshot, profile_id)
```

Arguments

snapshot	A Snapshot object
profile_id	Character vector of expression profile IDs to remove

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Remove a single expression profile by ID
snapshot <- remove_expression_profile(snapshot, "CYP3A4|Human|Healthy")

# Remove multiple expression profiles
snapshot <- remove_expression_profile(
  snapshot,
  c("CYP3A4|Human|Healthy", "P-gp|Human|Healthy")
)

## End(Not run)
```

remove_formulation	<i>Remove formulations from a snapshot</i>
--------------------	--

Description

Remove one or more formulations from a Snapshot by name.

Usage

```
remove_formulation(snapshot, formulation_name)
```

Arguments

snapshot	A Snapshot object
formulation_name	Character vector of formulation names to remove

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Remove a single formulation
snapshot <- remove_formulation(snapshot, "Tablet")
```

```
# Remove multiple formulations
snapshot <- remove_formulation(
  snapshot,
  c("Tablet", "Oral solution")
)

## End(Not run)
```

remove_individual	<i>Remove individuals from a snapshot</i>
-------------------	---

Description

Remove one or more individuals from a Snapshot by name.

Usage

```
remove_individual(snapshot, individual_name)
```

Arguments

snapshot	A Snapshot object
individual_name	Character vector of individual names to remove

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Remove a single individual
snapshot <- remove_individual(snapshot, "Subject_001")

# Remove multiple individuals
snapshot <- remove_individual(
  snapshot,
  c("Subject_001", "Subject_002")
)

## End(Not run)
```

remove_observed_data	<i>Remove observed data from a snapshot</i>
----------------------	---

Description

Remove one or more observed-data entries from a [Snapshot](#) by name.

Usage

```
remove_observed_data(snapshot, observed_data_name)
```

Arguments

snapshot	A Snapshot object.
observed_data_name	Character vector of observed-data names to remove.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:  
snapshot <- load_snapshot("Midazolam") |>  
  remove_observed_data("Study A")  
  
## End(Not run)
```

remove_observer_set	<i>Remove observer sets from a snapshot</i>
---------------------	---

Description

Remove one or more ObserverSet building blocks from a Snapshot by name. Names not present in the snapshot trigger a warning rather than an error so callers can run idempotent cleanup pipelines.

Usage

```
remove_observer_set(snapshot, observer_set_name)
```

Arguments

snapshot	A Snapshot object.
observer_set_name	Character vector of observer set names to remove.

Value

The updated Snapshot object, invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam")

snapshot |> remove_observer_set("BrainPlasmaConcentration")

snapshot |>
  remove_observer_set(c("BrainPlasmaConcentration", "LiverObservers"))

## End(Not run)
```

remove_population	<i>Remove populations from a snapshot</i>
-------------------	---

Description

Remove one or more populations from a Snapshot by name.

Usage

```
remove_population(snapshot, population_name)
```

Arguments

snapshot	A Snapshot object
population_name	Character vector of population names to remove

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
# Load a snapshot
snapshot <- load_snapshot("Midazolam")

# Remove a single population
snapshot <- remove_population(snapshot, "pop_1")

# Remove multiple populations
snapshot <- remove_population(
  snapshot,
  c("pop_1", "pop_2")
)
```

```
)
## End(Not run)
```

remove_protocol	<i>Remove protocols from a snapshot</i>
-----------------	---

Description

Remove one or more protocols from a [Snapshot](#) by name.

Usage

```
remove_protocol(snapshot, protocol_name)
```

Arguments

snapshot	A Snapshot object.
protocol_name	Character vector of protocol names to remove.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam") |>
  remove_protocol("IV bolus 1mg")

## End(Not run)
```

remove_simulation	<i>Remove simulations from a snapshot</i>
-------------------	---

Description

Remove one or more simulations from a [Snapshot](#) by name.

Usage

```
remove_simulation(snapshot, simulation_name)
```

Arguments

snapshot	A Snapshot object.
simulation_name	Character vector of simulation names to remove.

Value

The updated [Snapshot](#) object, returned invisibly.

Examples

```
## Not run:
snapshot <- load_snapshot("Midazolam") |>
  remove_simulation("simulation1")

## End(Not run)
```

 Schema

Schema class for OSP advanced protocols

Description

An R6 class that represents a repeatable block inside an Advanced [Protocol](#). A schema has its own name, schema-level parameters (typically Start time, NumberOfRepetitions, TimeBetweenRepetitions), and an ordered list of [SchemaItem](#) applications.

In an OSP snapshot, each schema is one entry of the Schemas array inside a Protocols building block.

Active bindings

data The raw Schema list as it appears in the snapshot JSON, refreshed from the wrapped schema items and parameters (read-only).

name The name of the schema. Writable: must be a non-empty scalar string.

parameters The schema-level [Parameter](#) objects (number of repetitions, time between repetitions, ...). Writable: must be a list, or NULL to clear.

items List of [SchemaItem](#) objects in the schema, in declaration order.

Methods**Public methods:**

- [Schema\\$new\(\)](#)
- [Schema\\$print\(\)](#)
- [Schema\\$clone\(\)](#)

Schema\$new(): Create a new Schema object.

Usage:

```
Schema$new(data = list())
```

Arguments:

data Raw Schema list from a snapshot. May be NULL or an empty list, both of which create an empty schema.

Returns: A new Schema object

Schema\$print(): Print a summary of the schema.

Usage:

Schema\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the Schema object

Schema\$clone(): The objects of this class are cloneable with this method.

Usage:

Schema\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

SchemaItem

SchemaItem class for OSP advanced protocols

Description

An R6 class that represents one application inside a [Schema](#) of an Advanced [Protocol](#). A schema item carries an application type, an optional formulation key, target organ and compartment, and the application-level parameters (dose, start time, ...).

In an OSP snapshot, each schema item is one entry of the SchemaItems array nested inside a Schemas entry of a Protocols building block.

Active bindings

data The raw SchemaItem list as it appears in the snapshot JSON, refreshed from the wrapped parameters (read-only).

name The name of the schema item. Writable: must be a non-empty scalar string.

application_type The application type of the schema item (for example "Oral", "IntravenousBolus"). Writable: must be one of the canonical PK-Sim application types (see [create_schema_item\(\)](#)'s application_type argument), or NULL to clear.

formulation_key The formulation key, linking the schema item to a formulation selection in the owning simulation.

target_organ The target organ for the application.

target_compartment The target compartment for the application.

parameters The schema item's application-level [Parameter](#) objects (dose, start time, ...). Writable: must be a list, or NULL to clear.

Methods**Public methods:**

- [SchemaItem\\$new\(\)](#)
- [SchemaItem\\$print\(\)](#)
- [SchemaItem\\$clone\(\)](#)

SchemaItem\$new(): Create a new SchemaItem object.

Usage:

```
SchemaItem$new(data = list())
```

Arguments:

data Raw SchemaItem list from a snapshot. May be NULL or an empty list, both of which create an empty schema item.

Returns: A new SchemaItem object

SchemaItem\$print(): Print a summary of the schema item.

Usage:

```
SchemaItem$print(...)
```

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the SchemaItem object

SchemaItem\$clone(): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

Simulation

Simulation class for OSP snapshot simulations

Description

An R6 class that represents a simulation in an OSP snapshot. A simulation references building blocks (compounds, events, formulations, observer sets, observed data, protocols, plus exactly one of an individual or a population) and binds them together with a solver configuration, an output schema, and parameter overrides.

Subject type

Exactly one of Individual / Population must be non-empty. Setting or clearing one through the active bindings is the user's responsibility; the initialiser only checks the inbound raw data.

Passthrough fields

Four fields are preserved byte-equivalent from the input rather than parsed: Interactions, AlteredBuildingBlocks, IndividualAnalyses, and PopulationAnalyses. They survive round-trip without modelling.

Active bindings

data The raw data of the simulation (read-only). Rebuilt from the nested caches so that mutations on wrapped objects flow back to the export payload. Passthrough fields are preserved byte-equivalent.

name The name of the simulation.

description Free-text description of the simulation.

model The PK-Sim model used to build the simulation (e.g. "4Comp").

allow_aging Whether the simulation allows aging. Writable: a single logical value, or NULL to clear.

individual Name of the individual building block (or NULL for a population simulation).

population Name of the population building block (or NULL for an individual simulation).

solver The [SolverSettings](#) object.

output_schema The [OutputSchema](#) object.

compounds List of [CompoundProperties](#) objects.

events List of [EventSelection](#) objects.

observer_sets List of [ObserverSetSelection](#) objects.

observed_data_names Character vector of observed-data building-block names referenced by this simulation.

output_selections Character vector of output quantity paths selected for reporting.

output_mappings List of [OutputMapping](#) objects.

parameters Named list of [LocalizedParameter](#) objects (keyed by path) overriding parameters in the simulation's tree.

advanced_parameters List of [AdvancedParameter](#) objects (population simulations only).

Methods

Public methods:

- [Simulation\\$new\(\)](#)
- [Simulation\\$print\(\)](#)
- [Simulation\\$clone\(\)](#)

Simulation\$new(): Create a new Simulation object.

Usage:

`Simulation$new(data)`

Arguments:

data Raw simulation data from a snapshot.

Returns: A new Simulation object.

`Simulation$print()`: Print a summary of the simulation.

Usage:

`Simulation$print(...)`

Arguments:

... Additional arguments passed to print methods.

Returns: Invisibly returns the simulation object.

`Simulation$clone()`: The objects of this class are cloneable with this method.

Usage:

`Simulation$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

Snapshot

Snapshot class for OSP snapshots

Description

An R6 class that represents an OSP snapshot file. This class provides methods to access different components of the snapshot and visualize its structure.

Supported snapshot versions

Snapshot accepts PK-Sim snapshots whose integer Version is within the supported band 79 (v11.2) to 81 (v13), inclusive (see `osp.snapshots:::SUPPORTED_VERSION_MIN` and `osp.snapshots:::SUPPORTED_VERSION_MAX`). A snapshot above the ceiling aborts as not supported in this version. A below-floor snapshot in the 74–78 band can be migrated by passing `upgrade = TRUE`, which round-trips it through PK-Sim up to the version the installed `osp suite` core emits; migration requires that core to emit a supported version (79 to 81) and aborts before converting when it would emit something above the ceiling. Without `upgrade = TRUE`, a below-floor snapshot reports how to migrate and does not load. Snapshots below 74 are too old to migrate and abort. A snapshot newer than the installed `osp suite` core (but still in band) loads with a warning that it may not load or run there. Hand-rolled list input must supply Version.

Active bindings

`data` The aggregated data of the snapshot from all components

`pksim_version` The human-readable PKSIM version corresponding to the snapshot version

`path` The path to the snapshot file relative to the working directory

`compounds` List of Compound objects in the snapshot

`expression_profiles` List of ExpressionProfile objects in the snapshot

individuals List of Individual objects in the snapshot
formulations List of Formulation objects in the snapshot
populations List of Population objects in the snapshot
events List of Event objects in the snapshot
observer_sets List of ObserverSet objects in the snapshot
protocols List of Protocol objects in the snapshot
observed_data List of DataSet objects (observed data) in the snapshot
simulations List of Simulation objects in the snapshot

Methods

Public methods:

- `Snapshot$new()`
- `Snapshot$print()`
- `Snapshot$export()`
- `Snapshot$add_individual()`
- `Snapshot$remove_individual()`
- `Snapshot$add_formulation()`
- `Snapshot$remove_formulation()`
- `Snapshot$remove_population()`
- `Snapshot$add_expression_profile()`
- `Snapshot$remove_expression_profile()`
- `Snapshot$add_observer_set()`
- `Snapshot$remove_observer_set()`
- `Snapshot$add_observed_data()`
- `Snapshot$remove_observed_data()`
- `Snapshot$add_compound()`
- `Snapshot$remove_compound()`
- `Snapshot$add_population()`
- `Snapshot$add_protocol()`
- `Snapshot$remove_protocol()`
- `Snapshot$add_event()`
- `Snapshot$remove_event()`
- `Snapshot$add_simulation()`
- `Snapshot$remove_simulation()`
- `Snapshot$clone()`

`Snapshot$new()`: Create a new Snapshot object from a JSON file or a list

Usage:

`Snapshot$new(input, upgrade = FALSE)`

Arguments:

input Path to the snapshot JSON file, URL, template name, or a list containing snapshot data. The parsed data must contain an integer **Version** field. Versions 79 to 81 load normally; versions 74 to 78 load only with **upgrade = TRUE** (otherwise they report how to migrate and abort); all other or missing versions abort (see the "Supported snapshot versions" section).

upgrade Logical, default **FALSE**. When **TRUE** and the snapshot's **Version** is in the below-floor migration band (74–78), the snapshot is round-tripped through the installed PK-Sim core to upgrade it to a supported version before loading (slow, several minutes). When **FALSE**, a below-floor snapshot reports how to migrate and aborts. Ignored for in-band snapshots, which are never migrated.

Returns: A new Snapshot object

Snapshot\$print(): Print a summary of the snapshot

Usage:

Snapshot\$print(...)

Arguments:

... Additional arguments passed to print methods

Returns: Invisibly returns the snapshot object

Snapshot\$export(): Export the snapshot to a JSON file

Usage:

Snapshot\$export(path)

Arguments:

path Path to save the JSON file

Returns: Invisibly returns the object

Snapshot\$add_individual(): Add one or more Individual objects to the snapshot. References to expression profiles not yet in the snapshot trigger one informational warning per individual; the add proceeds either way.

Usage:

Snapshot\$add_individual(individual)

Arguments:

individual An Individual object created with **create_individual()**, or a list of such objects.

Returns: Invisibly returns the object

Examples:

```
# Create a new individual
ind <- create_individual(name = "New Patient", age = 35, weight = 70)

# Add the individual to a snapshot
snapshot$add_individual(ind)

# Add several at once
patients <- list(
  create_individual("Patient_A", age = 25),
  create_individual("Patient_B", age = 45)
)
snapshot$add_individual(patients)
```

`Snapshot$remove_individual()`: Remove an individual from the snapshot by name

Usage:

```
Snapshot$remove_individual(individual_name)
```

Arguments:

`individual_name` Character vector of individual name(s) to remove

Returns: Invisibly returns the object

Examples:

```
# Remove an individual from the snapshot
snapshot$remove_individual("Subject_001")
```

`Snapshot$add_formulation()`: Add one or more Formulation objects to the snapshot.

Usage:

```
Snapshot$add_formulation(formulation)
```

Arguments:

`formulation` A Formulation object created with `create_formulation()`, or a list of such objects.

Returns: Invisibly returns the object

Examples:

```
# Create a new formulation
form <- create_formulation(name = "Tablet", type = "Weibull")

# Add the formulation to a snapshot
snapshot$add_formulation(form)

# Add several at once
forms <- list(
  create_formulation("Tablet", type = "Weibull"),
  create_formulation("Oral solution", type = "First Order")
)
snapshot$add_formulation(forms)
```

`Snapshot$remove_formulation()`: Remove a formulation from the snapshot by name

Usage:

```
Snapshot$remove_formulation(formulation_name)
```

Arguments:

`formulation_name` Character vector of formulation name(s) to remove

Returns: Invisibly returns the object

Examples:

```
# Remove a formulation from the snapshot
snapshot$remove_formulation("Tablet")
```

`Snapshot$remove_population()`: Remove a population from the snapshot by name

Usage:

```
Snapshot$remove_population(population_name)
```

Arguments:

population_name Character vector of population name(s) to remove

Returns: Invisibly returns the object

Examples:

```
# Remove a population from the snapshot
snapshot$remove_population("pop_1")
```

Snapshot\$add_expression_profile(): Add one or more ExpressionProfile objects to the snapshot.

Usage:

```
Snapshot$add_expression_profile(expression_profile)
```

Arguments:

expression_profile An ExpressionProfile object, or a list of such objects.

Returns: Invisibly returns the object

Examples:

```
# Create a new expression profile
profile_data <- list(
  Type = "Enzyme",
  Species = "Human",
  Molecule = "CYP3A4",
  Category = "Healthy",
  Parameters = list()
)
profile <- ExpressionProfile$new(profile_data)

# Add the expression profile to a snapshot
snapshot$add_expression_profile(profile)

# Add several at once
snapshot$add_expression_profile(list(profile, profile))
```

Snapshot\$remove_expression_profile(): Remove expression profiles from the snapshot by ID

Usage:

```
Snapshot$remove_expression_profile(profile_id)
```

Arguments:

profile_id Character vector of expression profile IDs to remove

Returns: Invisibly returns the object

Examples:

```
# Remove an expression profile from the snapshot
snapshot$remove_expression_profile("CYP3A4_Human_Healthy")
```

Snapshot\$add_observer_set():

Usage:

Snapshot\$add_observer_set(observer_set)

Snapshot\$remove_observer_set():

Usage:

Snapshot\$remove_observer_set(observer_set_name)

Snapshot\$add_observed_data(): Add one or more DataSet objects (observed data) to the snapshot.

Usage:

Snapshot\$add_observed_data(observed_data)

Arguments:

observed_data A DataSet object created from snapshot observed data, or a list of such objects.

Returns: Invisibly returns the object

Snapshot\$remove_observed_data(): Remove observed data from the snapshot by name

Usage:

Snapshot\$remove_observed_data(observed_data_name)

Arguments:

observed_data_name Character vector of observed data names to remove

Returns: Invisibly returns the object

Snapshot\$add_compound():

Usage:

Snapshot\$add_compound(compound)

Snapshot\$remove_compound():

Usage:

Snapshot\$remove_compound(compound_name)

Snapshot\$add_population():

Usage:

Snapshot\$add_population(population)

Snapshot\$add_protocol():

Usage:

Snapshot\$add_protocol(protocol)

Snapshot\$remove_protocol():

Usage:

Snapshot\$remove_protocol(protocol_name)

Snapshot\$add_event():

Usage:

```
Snapshot$add_event(event)
```

```
Snapshot$remove_event():
```

Usage:

```
Snapshot$remove_event(event_name)
```

`Snapshot$add_simulation()`: Build a simulation from named arguments and attach it, or attach a pre-built simulation.

The build mode is the entry point: with the snapshot in hand it resolves compound references and derives defaults (calculation methods, formulation key, alternatives) before attaching. Supply name plus exactly one of individual / population to build; each inline compounds entry is a config list (`list(name =, protocol =, formulation =, processes =, calculation_methods =, alternatives =)`). Alternatively supply a pre-built [Simulation](#) (or a list of them) through `simulation`. References to building blocks not yet in the snapshot trigger one informational warning per simulation; the add proceeds either way.

Usage:

```
Snapshot$add_simulation(
  simulation = NULL,
  name = NULL,
  model = "4Comp",
  individual = NULL,
  population = NULL,
  compounds = NULL,
  events = NULL,
  observer_sets = NULL,
  observed_data_names = NULL,
  solver = NULL,
  output_schema = NULL,
  output_selections = NULL,
  output_mappings = NULL,
  parameters = NULL,
  advanced_parameters = NULL,
  description = NULL,
  allow_aging = NULL
)
```

Arguments:

`simulation` A pre-built [Simulation](#) object, or a list of such objects. Leave `NULL` to build a simulation from the named arguments instead.

`name` Character. Simulation name (required in build mode).

`model` Character. PK-Sim model name (defaults to "4Comp").

`individual` Character. Name of the individual building block. Mutually exclusive with `population`.

`population` Character. Name of the population building block. Mutually exclusive with `individual`.

`compounds` List of inline compound-config lists (`list(name =, protocol =, formulation =, processes =, calculation_methods =, alternatives =)`). `alternatives` is a named character vector (or named list of length-one strings) mapping a friendly property name

(lipophilicity, fraction_unbound, solubility, intestinal_permeability, permeability) to the alternative label to select on that compound, for example `alternatives = c(solubility = "FaSSIF")`; it overrides the derived default for the named groups only, every other group is still defaulted from the compound. `formulation` accepts a single string (bound to the protocol's inferred first slot key, unchanged) or a named character vector (or named list of length-one strings) mapping application-slot key to formulation name for a multi-slot protocol, for example `formulation = c(Formulation = "Oral solution", "Formulation 2" = "IV solution")`.

`events` List of [EventSelection](#) objects or raw lists.

`observer_sets` List of [ObserverSetSelection](#) objects or raw lists.

`observed_data_names` Character vector of observed-data names.

`solver` A [SolverSettings](#) object or raw list. Defaults to PK-Sim defaults.

`output_schema` An [OutputSchema](#) object or raw list. Defaults to an empty schema.

`output_selections` Character vector of output quantity paths.

`output_mappings` List of [OutputMapping](#) objects or raw lists.

`parameters` List of [LocalizedParameter](#) objects (created with `create_parameter(path = ... , ...)`) or raw parameter lists.

`advanced_parameters` List of [AdvancedParameter](#) objects or raw lists (population simulations only).

`description` Character. Free-text description of the simulation.

`allow_aging` Logical. Whether the simulation allows aging.

Returns: Invisibly returns the object

`Snapshot$remove_simulation()`: Remove a simulation from the snapshot by name

Usage:

```
Snapshot$remove_simulation(simulation_name)
```

Arguments:

`simulation_name` Character vector of simulation name(s) to remove

Returns: Invisibly returns the object

`Snapshot$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## -----
## Method `Snapshot$add_individual()`
## -----

## Not run:
# Create a new individual
ind <- create_individual(name = "New Patient", age = 35, weight = 70)
```

```

# Add the individual to a snapshot
snapshot$add_individual(ind)

# Add several at once
patients <- list(
  create_individual("Patient_A", age = 25),
  create_individual("Patient_B", age = 45)
)
snapshot$add_individual(patients)

## End(Not run)

## -----
## Method `Snapshot$remove_individual()`
## -----

## Not run:
# Remove an individual from the snapshot
snapshot$remove_individual("Subject_001")

## End(Not run)

## -----
## Method `Snapshot$add_formulation()`
## -----

## Not run:
# Create a new formulation
form <- create_formulation(name = "Tablet", type = "Weibull")

# Add the formulation to a snapshot
snapshot$add_formulation(form)

# Add several at once
forms <- list(
  create_formulation("Tablet", type = "Weibull"),
  create_formulation("Oral solution", type = "First Order")
)
snapshot$add_formulation(forms)

## End(Not run)

## -----
## Method `Snapshot$remove_formulation()`
## -----

## Not run:
# Remove a formulation from the snapshot
snapshot$remove_formulation("Tablet")

## End(Not run)

```

```
## -----
## Method `Snapshot$remove_population()`
## -----

## Not run:
# Remove a population from the snapshot
snapshot$remove_population("pop_1")

## End(Not run)

## -----
## Method `Snapshot$add_expression_profile()`
## -----

## Not run:
# Create a new expression profile
profile_data <- list(
  Type = "Enzyme",
  Species = "Human",
  Molecule = "CYP3A4",
  Category = "Healthy",
  Parameters = list()
)
profile <- ExpressionProfile$new(profile_data)

# Add the expression profile to a snapshot
snapshot$add_expression_profile(profile)

# Add several at once
snapshot$add_expression_profile(list(profile, profile))

## End(Not run)

## -----
## Method `Snapshot$remove_expression_profile()`
## -----

## Not run:
# Remove an expression profile from the snapshot
snapshot$remove_expression_profile("CYP3A4_Human_Healthy")

## End(Not run)
```

solubility

Solubility value object

Description

Build a solubility value object for `create_compound()`'s solubility argument (or the writable `Compound$solubility` field). Expresses either the scalar form (a single solubility value at a ref-

erence pH, optionally with a solubility gain per charge) or the mutually exclusive table form (a pH/value table).

The scalar-form arguments (value, reference_pH, gain_per_charge) and table are mutually exclusive: supplying table together with any of them is an error. Unlike the previous `create_compound()` behaviour, where reference_pH / gain_per_charge were silently ignored on the table path, this helper rejects the combination so the intent is unambiguous.

Usage

```
solubility(
  value = NULL,
  unit = "mg/l",
  reference_pH = NULL,
  gain_per_charge = NULL,
  table = NULL,
  name = "User defined",
  default = FALSE
)
```

Arguments

value	Numeric scalar. Solubility-at-reference-pH value for the scalar form. Leave NULL for the table form.
unit	Character. Unit for the solubility value (scalar form) and for the table Y values, validated against dimension "Concentration (mass)". Defaults to "mg/l".
reference_pH	Numeric scalar. Reference pH added to the scalar Solubility alternative. Scalar form only.
gain_per_charge	Numeric scalar. Optional Solubility gain per charge parameter added to the scalar Solubility alternative. Scalar form only.
table	Two-column data frame giving table-based solubility: the first column is pH and the second is the solubility value. When supplied, the table form is built and the scalar-form arguments must be left unset.
name	Character. Name of the created alternative (scalar or table). Defaults to "User defined".
default	Logical. Whether this alternative is the group's default when it appears in a list of alternatives passed to the matching <code>create_compound()</code> argument or Compound field. Defaults to FALSE. Ignored for a single (non-list) value object, which is always the default. When a list has no element marked default = TRUE, the first element is the default (unchanged behaviour); marking two or more elements default = TRUE in the same list is an error.

Value

A `solubility_spec` object to pass to `create_compound()`.

See Also

[create_compound\(\)](#)

Other value-object helpers: [age\(\)](#), [error\(\)](#), [fraction_unbound\(\)](#), [gestational_age\(\)](#), [height\(\)](#), [intestinal_permeability\(\)](#), [lipophilicity\(\)](#), [permeability\(\)](#), [time\(\)](#), [values\(\)](#), [weight\(\)](#)

Examples

```
# Scalar form with reference pH and gain per charge
solubility(9999, reference_pH = 7, gain_per_charge = 1000)

# Table form (first column pH, second column value)
solubility(table = data.frame(pH = c(3, 6, 6.8), value = c(5000, 3000, 90)))

# Several alternatives, the second marked as the default
create_compound(
  name = "Drug X",
  solubility = list(
    solubility(9999, name = "Aqueous"),
    solubility(200, name = "FaSSIF", default = TRUE)
  )
)
```

SolverSettings

SolverSettings class for simulation solver configuration

Description

An R6 class representing the Solver block of a [Simulation](#). Each field is optional and inherits the PK-Sim solver default when absent; the underlying raw data list stays sparse so missing fields remain absent on export.

Active bindings

data The raw data of the solver settings (read-only). An empty solver is always returned as a named-empty list so the JSON serialiser writes it as `{}` (an object), not `[]` (an array). PK-Sim's snapshot mapper requires the object shape.

abs_tol Absolute solver tolerance. Writable: a single numeric value, or NULL to clear.

rel_tol Relative solver tolerance. Writable: a single numeric value, or NULL to clear.

use_jacobian Whether to use the Jacobian during integration. Writable: a single logical value, or NULL to clear.

h0 Initial step size. Writable: a single numeric value, or NULL to clear.

h_min Minimum step size. Writable: a single numeric value, or NULL to clear.

h_max Maximum step size. Writable: a single numeric value, or NULL to clear.

mx_step Maximum number of internal solver steps. Writable: a single positive whole number (stored as an integer), or NULL to clear.

Methods**Public methods:**

- [SolverSettings\\$new\(\)](#)
- [SolverSettings\\$clone\(\)](#)

[SolverSettings\\$new\(\)](#): Create a new SolverSettings object.

Usage:

[SolverSettings\\$new](#)(data)

Arguments:

data Raw Solver data from a snapshot (may be an empty list).

Returns: A new SolverSettings object.

[SolverSettings\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

[SolverSettings\\$clone](#)(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

time

Time series value object

Description

Build a time-series value object for [create_observed_data\(\)](#)'s time argument. Carries the time grid x-values and their unit.

Usage

```
time(value, unit = "h")
```

Arguments

value Numeric vector. Time grid x-values.

unit Character. Unit for value, validated against dimension "Time". Defaults to "h".

Value

A time_spec object to pass to [create_observed_data\(\)](#).

See Also

[create_observed_data\(\)](#)

Other value-object helpers: [age\(\)](#), [error\(\)](#), [fraction_unbound\(\)](#), [gestational_age\(\)](#), [height\(\)](#), [intestinal_permeability\(\)](#), [lipophilicity\(\)](#), [permeability\(\)](#), [solubility\(\)](#), [values\(\)](#), [weight\(\)](#)

Examples

```
time(c(0, 1, 2, 4, 8))
```

validate_gender	<i>Validate that a gender is valid</i>
-----------------	--

Description

Check if a gender is valid using ospsuite Gender enum. Throws an error if the gender is invalid.

Usage

```
validate_gender(gender)
```

Arguments

gender Gender value to validate

Value

TRUE if valid, throws an error if invalid

validate_population	<i>Validate that a population is valid</i>
---------------------	--

Description

Check if a population is valid using ospsuite HumanPopulation enum. Throws an error if the population is invalid.

Usage

```
validate_population(population)
```

Arguments

population Population value to validate

Value

TRUE if valid, throws an error if invalid

validate_snapshot	<i>Validate that an object is a Snapshot</i>
-------------------	--

Description

Check if an object is a Snapshot instance. Throws an error if the object is not a Snapshot.

Usage

```
validate_snapshot(snapshot)
```

Arguments

snapshot Object to validate

Value

TRUE if valid, throws an error if invalid

validate_species	<i>Validate that a species is valid</i>
------------------	---

Description

Check if a species is valid using ospsuite Species enum. Throws an error if the species is invalid.

Usage

```
validate_species(species)
```

Arguments

species Species value to validate. Must be a single, non-NA value.

Value

TRUE if valid, throws an error if invalid

validate_unit	<i>Validate that a unit is valid for a given dimension</i>
---------------	--

Description

Check if a unit is valid for a given dimension using ospsuite units. Throws an error if the unit is invalid.

Usage

```
validate_unit(unit, dimension)
```

Arguments

unit	Unit to validate
dimension	Dimension the unit should belong to

Value

TRUE if valid, throws an error if invalid

values	<i>Measurement values series value object</i>
--------	---

Description

Build a measurement-values value object for [create_observed_data\(\)](#)'s values argument. Carries the measurement y-values, their dimension, and an optional unit.

The dimension is required and has no default; it gates unit validation and is written to the data-set column. When unit is supplied it is validated against dimension via [validate_unit\(\)](#).

Usage

```
values(value, unit = NULL, dimension)
```

Arguments

value	Numeric vector. Measurement y-values.
unit	Character. Optional unit for value. When supplied, validated against dimension.
dimension	Character. Dimension for value (for example "Concentration (mass)"). Required: pass one of the names of ospsuite::ospDimensions.

Value

A values_spec object to pass to [create_observed_data\(\)](#).

See Also

[create_observed_data\(\)](#)
Other value-object helpers: [age\(\)](#), [error\(\)](#), [fraction_unbound\(\)](#), [gestational_age\(\)](#), [height\(\)](#), [intestinal_permeability\(\)](#), [lipophilicity\(\)](#), [permeability\(\)](#), [solubility\(\)](#), [time\(\)](#), [weight\(\)](#)

Examples

```
values(c(0, 12, 18, 11, 5), unit = "mg/l", dimension = "Concentration (mass)")
```

weight	<i>Weight value object</i>
--------	----------------------------

Description

Build a weight value object for [create_individual\(\)](#)'s weight argument (or the writable Individual\$weight / OriginData\$weight field). Fills the OriginData Weight characteristic (list(Value =, Unit =)).

Usage

```
weight(value, unit = "kg")
```

Arguments

value	Numeric scalar. Weight value.
unit	Character. Unit for weight, validated against dimension "Mass". Defaults to "kg".

Value

A weight_spec object to pass to [create_individual\(\)](#).

See Also

[create_individual\(\)](#)
Other value-object helpers: [age\(\)](#), [error\(\)](#), [fraction_unbound\(\)](#), [gestational_age\(\)](#), [height\(\)](#), [intestinal_permeability\(\)](#), [lipophilicity\(\)](#), [permeability\(\)](#), [solubility\(\)](#), [time\(\)](#), [values\(\)](#)

Examples

```
weight(70)
create_individual(name = "Subject", weight = weight(70))
```

Index

* value-object helpers

- age, [15](#)
- error, [63](#)
- fraction_unbound, [71](#)
- gestational_age, [72](#)
- height, [79](#)
- intestinal_permeability, [82](#)
- lipophilicity, [83](#)
- permeability, [98](#)
- solubility, [134](#)
- time, [137](#)
- values, [140](#)
- weight, [141](#)

add_compound, [4](#)
add_compound(), [61](#)
add_event, [5](#)
add_expression_profile, [6](#)
add_formulation, [7](#)
add_individual, [8](#)
add_observed_data, [9](#)
add_observer_set, [9](#)
add_population, [10](#)
add_protocol, [11](#)
add_simulation, [12](#)
AdvancedParameter, [13](#), [14](#), [51](#), [124](#), [132](#)
age, [15](#)
age(), [38](#), [39](#), [63](#), [71](#), [72](#), [80](#), [83](#), [84](#), [92](#), [99](#),
[136](#), [137](#), [141](#)
as_tibbles, [16](#)
as_tibbles(), [73–79](#)

CalculationMethods, [17](#), [19](#), [20](#), [81](#), [92](#), [93](#)
Compound, [4](#), [17](#), [19](#), [23](#), [25](#), [73](#)
CompoundProcessSelection, [21](#), [27](#)
CompoundProperties, [21](#), [27](#), [57](#), [111](#), [124](#)
convert_ospsuite_time_to_duration, [22](#)
convert_ospsuite_time_unit_to_lubridate,
[23](#)
create_compound, [23](#)

create_compound(), [4](#), [71](#), [82–84](#), [98](#), [99](#),
[134–136](#)
create_compound_process_selection, [27](#)
create_descriptor_condition, [28](#)
create_descriptor_condition(), [43](#)
create_event, [29](#)
create_event(), [5](#), [34](#)
create_event_selection, [30](#)
create_expression_profile, [30](#)
create_expression_profile(), [38](#), [67](#)
create_formula_reference, [32](#)
create_formula_reference(), [43](#)
create_formulation, [33](#)
create_formulation(), [7](#)
create_formulation_selection, [37](#)
create_formulation_selection(), [57](#)
create_individual, [38](#)
create_individual(), [8](#), [15](#), [51](#), [72](#), [79](#), [80](#),
[141](#)
create_molecule_list, [40](#)
create_molecule_list(), [43](#)
create_observed_data, [41](#)
create_observed_data(), [9](#), [63](#), [137](#), [140](#),
[141](#)
create_observer, [42](#)
create_observer(), [28](#), [32](#), [40](#)
create_observer_set, [44](#)
create_observer_set(), [42](#)
create_observer_set_selection, [45](#)
create_output_interval, [45](#)
create_output_interval(), [47](#)
create_output_mapping, [46](#)
create_output_schema, [47](#)
create_parameter, [48](#)
create_parameter(), [24](#), [29](#), [31](#), [34](#), [53](#), [55](#),
[58](#), [60](#)
create_population, [50](#)
create_population(), [10](#), [99](#)
create_process, [52](#)

- create_process(), 25
- create_protocol, 54
- create_protocol(), 11, 109
- create_protocol_selection, 56
- create_schema, 57
- create_schema(), 55
- create_schema_item, 59
- create_schema_item(), 58, 122
- create_snapshot, 61
- create_solver_settings, 62

- error, 63
- error(), 15, 41, 71, 72, 80, 83, 84, 99, 136, 137, 141
- Event, 5, 29, 30, 64, 65
- EventSelection, 13, 30, 65, 124, 132
- export_snapshot, 66
- export_snapshot(), 61, 62
- ExpressionProfile, 6, 30, 32, 66

- Formulation, 7, 57, 68
- FormulationSelection, 37, 57, 70, 111
- fraction_unbound, 71
- fraction_unbound(), 15, 19, 24, 25, 63, 72, 80, 83, 84, 99, 136, 137, 141

- gestational_age, 72
- gestational_age(), 15, 38, 39, 63, 71, 80, 81, 83, 84, 92, 99, 136, 137, 141
- get_compounds_dfs, 73
- get_compounds_dfs(), 19–21
- get_default_alternative, 73
- get_events_dfs, 74
- get_expression_profiles_dfs, 75
- get_formulations_dfs, 75
- get_individuals_dfs, 76
- get_observed_data_dfs, 77
- get_observer_sets_dfs, 77
- get_populations_dfs, 78
- get_protocols_dfs, 79

- height, 79
- height(), 15, 38, 39, 63, 71, 72, 80, 83, 84, 93, 99, 136, 137, 141

- Individual, 8, 17, 51, 80, 92, 99
- intestinal_permeability, 82
- intestinal_permeability(), 15, 19, 20, 25, 63, 71, 72, 80, 84, 99, 136, 137, 141

- lifecycle::deprecate_soft(), 19, 21
- lipophilicity, 83
- lipophilicity(), 15, 19, 24, 25, 63, 71, 72, 80, 83, 99, 136, 137, 141
- load_snapshot, 84
- load_snapshot(), 61, 62
- loadDataSetFromSnapshot, 85
- loadDataSetFromSnapshot(), 9, 41, 86
- LocalizedParameter, 13, 48, 49, 86, 124, 132

- ObservedData (loadDataSetFromSnapshot), 85
- Observer, 28, 40, 42–44, 88, 90
- ObserverSet, 42, 44, 45, 88, 90, 91
- ObserverSetSelection, 13, 45, 91, 124, 132
- OriginData, 17, 80, 81, 92
- osp_models, 94
- OutputInterval, 45–47, 94, 96
- OutputMapping, 13, 46, 95, 124, 132
- OutputSchema, 13, 45, 47, 94, 96, 124, 132

- Parameter, 24, 29, 31, 34, 48, 49, 53, 55, 58, 60, 65, 86, 87, 94, 95, 97, 108, 121, 122
- permeability, 98
- permeability(), 15, 20, 25, 63, 71, 72, 80, 83, 84, 136, 137, 141
- Population, 10, 50, 51, 99
- print.biliary_clearance, 101
- print.compound_additional_parameters, 101
- print.compound_processes, 102
- print.hepatic_clearance, 102
- print.induction, 103
- print.inhibition, 103
- print.metabolizing_enzymes, 104
- print.parameter_collection, 104
- print.physicochemical_property, 105
- print.protein_binding_partners, 105
- print.renal_clearance, 106
- print.snapshot_collection, 106
- print.transporter_proteins, 107
- Process, 19, 20, 25, 52, 53, 107
- Protocol, 11, 54, 55, 57, 59, 109, 111, 121, 122
- ProtocolSelection, 37, 57, 70, 111

- Range, 51, 100, 112
- Range (Range-class), 113

range, [112](#)
range(), [51](#), [113](#)
Range-class, [113](#)
remove_compound, [114](#)
remove_event, [115](#)
remove_expression_profile, [115](#)
remove_formulation, [116](#)
remove_individual, [117](#)
remove_observed_data, [118](#)
remove_observer_set, [118](#)
remove_population, [119](#)
remove_protocol, [120](#)
remove_simulation, [120](#)

Schema, [55](#), [57–59](#), [109](#), [121](#), [122](#)
SchemaItem, [57–60](#), [109](#), [110](#), [121](#), [122](#)
Simulation, [12](#), [45](#), [46](#), [62](#), [65](#), [91](#), [95](#), [96](#),
[123](#), [131](#), [136](#)
Snapshot, [4–13](#), [41](#), [61](#), [114–121](#), [125](#)
solubility, [134](#)
solubility(), [15](#), [19](#), [24](#), [25](#), [63](#), [71](#), [72](#), [80](#),
[83](#), [84](#), [99](#), [137](#), [141](#)
SolverSettings, [13](#), [62](#), [63](#), [124](#), [132](#), [136](#)

time, [137](#)
time(), [15](#), [41](#), [63](#), [71](#), [72](#), [80](#), [83](#), [84](#), [99](#), [136](#),
[141](#)

validate_gender, [138](#)
validate_population, [138](#)
validate_snapshot, [139](#)
validate_species, [139](#)
validate_unit, [140](#)
validate_unit(), [97](#), [140](#)
values, [140](#)
values(), [15](#), [41](#), [63](#), [71](#), [72](#), [80](#), [83](#), [84](#), [99](#),
[136](#), [137](#), [141](#)

weight, [141](#)
weight(), [15](#), [38](#), [39](#), [63](#), [71](#), [72](#), [80](#), [83](#), [84](#),
[92](#), [99](#), [136](#), [137](#), [141](#)