
OPSPiggybacker Documentation

Release 0.1

David W.H. Swenson

May 31, 2018

Contents

1	Installation	3
2	Overview	5
3	What OPSPiggybacker does	7
4	Partial input trajectories	9
5	Full input trajectories	11
6	API Reference	13

OPSPiggybacker is a tool to assist importing other simulation records into OpenPathSampling for analysis. Because of the extensive metadata that OPS tracks for each move, this isn't completely trivial. The idea of OPSPiggybacker is to request a reasonable set of inputs that a user of another path sampling simulation tool can give, and then create from that a file that can use most of OPS's standard analysis tools.

The current version, 0.1, only aims to cover one-way shooting moves in a single ensemble, as with TPS simulations. See the roadmap for future plans.

CHAPTER 1

Installation

TODO

CHAPTER 2

Overview

Since this combines OPS and some other sampler, you'll need a little familiarity with both. From the OPS side, you'll need to set up an OPS `TransitionNetwork` object that represents the simulation you'll be reading in. (For now, this only supports `TPSNetwork` (and possibly `FixedLengthTPSNetwork`, although I haven't tested that.) This includes creating collective variables and volumes as always done in OPS.

You'll also need to prepare your previous simulation in an input format that can be read by `OPSPiggybacker`. Currently, we only support TPS simulations with one-way shooting. Even within that, there are several options.

- create a one-way shooting summary file
- use the API with one-way (partial) trajectory input
- use the API with full trajectory input

In general, the summary file is probably the easiest approach for users to implement. The two approaches that directly use the API are

What OPSPiggybacker does

The overall approach is very similar to the setup of an OPS simulation. You use the standard OPS volume, collective variable, network, and ensemble objects. The only things that change are the move scheme/path movers, and the simulation object.

Instead of an OPS move scheme or OPS path movers, you build what we call a mover stub. Currently, the *ShootingStub* is the only mover stub supported. The *ShootingStub* is analogous to an *OneWayShootingMover* in *OpenPathSampling*. It is initialized with an ensemble, a selector (only *UniformSelector* is currently supported) and optionally an engine.

Once you've defined the mover stub, you create a pseudo-simulator. Currently, the only supported pseudo-simulator is the *ShootingPseudoSimulator*. The pseudo-simulator plays the same role as a *PathSampling* object in *OpenPathSampling*. It is initialized with a storage, a set of *initial_conditions* (in the form of an *openpathsampling.SampleSet*), a network, and a mover stub called *mover*, which takes the place of the move scheme used in OPS.

Once this is done, you simply use the *run* method of the *ShootingPseudoSimulator* to generate your file. However, whereas the *run* method of the *PathSampling* object in OPS takes an integer with a number of steps, in OPSPiggybacker, you must provide the output of your previous simulation to the *run* method of the pseudo-simulator. The following subsection will describe the moves.

CHAPTER 4

Partial input trajectories

Full input trajectories

One of the input options for shooting moves is to use full input trajectories (`pre_joined=True`). In this case, the input trajectory must be an OPS format trajectory for the full trial trajectory. In addition, *the frames which are shared with other trajectories must be identical in memory frames*. Since this is quite hard to do, it is usually easier to use the `pre_joined=False` version with partial input trajectories.

However, with full input trajectories, you don't need to specify whether a given trial was forward or backward: the OPSPiggybacker can figure that out for you.

This is in the format of a list of 4-tuples (`replica, trial_trajectory, shooting_point_index, accepted`), where each 4-tuple represents a trial move. In detail, the elements of the tuple are:

- `replica`: the replica ID. Currently always the same (usually 0).
- `trial_trajectory`: the generated trial trajectory, as an `openpathsampling.engine.Trajectory` object. Note that there are two tricky things here. First, this must be the *entire* trial trajectory (not just the part generated during one-way shooting). Second, frames which are shared between two trajectories must actually be the same object in memory. This means that you have to rebuild the shooting process for your trajectories. (TODO: find ways to make this part easier on people)
- `shooting_point_index`: the frame number of the shooting point from the *previous* trajectory (counting from 0).
- `accepted`: boolean as to whether this trial move was accepted.

That's it! If you can make that tuple for each of your moves, you can import those moves into OPS for analysis.

6.1 One-Way Shooting Converters

`class ops_piggybacker.TPSConverterOptions`

Parameters

- **trim** (*bool*) – whether to trim the file trajectories to minimum acceptable length (default True)
- **retrim_shooting** (*bool*) – whether the shooting point index given is based on an untrimmed trajectory, and therefore needs to be shifted (default False).
- **auto_reverse** (*bool*) – whether to reverse backward trajectories (if the file version is forward, instead of backward, default False)
- **includes_shooting_point** (*bool*) – whether the one-way trial trajectory includes the shooting point, and therefore must have it trimmed off (default True)
- **full_trajectory** (*bool*) – whether the input trajectories are the full trajectories, instead of the partial one-way trajectories (default False). Note that if you use `full_trajectory=True`, you should also use `auto_reverse=False`.

```
__add__
    x.__add__(y) <==> x+y

__contains__
    x.__contains__(y) <==> y in x

__delattr__
    x.__delattr__('name') <==> del x.name

__eq__
    x.__eq__(y) <==> x==y

__format__ ()
    default object formatter
```

__ge__
`x.__ge__(y) <==> x>=y`

__getattr__
`x.__getattr__('name') <==> x.name`

__getitem__
`x.__getitem__(y) <==> x[y]`

__getslice__
`x.__getslice__(i, j) <==> x[i - j]`
 Use of negative indices is not supported.

__gt__
`x.__gt__(y) <==> x>y`

__hash__

__iter__

__le__
`x.__le__(y) <==> x<=y`

__len__

__lt__
`x.__lt__(y) <==> x<y`

__mul__
`x.__mul__(n) <==> x*n`

__ne__
`x.__ne__(y) <==> x!=y`

__reduce__ ()
 helper for pickle

__reduce_ex__ ()
 helper for pickle

__rmul__
`x.__rmul__(n) <==> n*x`

__setattr__
`x.__setattr__('name', value) <==> x.name = value`

__sizeof__ () → int
 size of object in memory, in bytes

__str__

auto_reverse
 Alias for field number 2

count (value) → integer – return number of occurrences of value

full_trajectory
 Alias for field number 4

includes_shooting_point
 Alias for field number 3

index (value[, start[, stop]]) → integer – return first index of value.
 Raises ValueError if the value is not present.

retrim_shooting

Alias for field number 1

trim

Alias for field number 0

class ops_piggybacker.OneWayTPSConverter(*storage, initial_file, mover, network, options=None, options_rejected=None*)

Bases: ops_piggybacker.simulation_stubs.ShootingPseudoSimulator

Single-ensemble network shooting pseudo-simulator from external trajectories.

This object handles a wide variety of external simulators. The idea is that the user must create a “simulation summary” file, which contains the information we need to perform the pseudo-simulation, where the trajectories are loaded via mdtraj.

__delattr__

x.__delattr__('name') <==> del x.name

__format__()

default object formatter

__getattr__

x.__getattr__('name') <==> x.name

__reduce__()

helper for pickle

__reduce_ex__()

helper for pickle

__repr__

__setattr__

x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int

size of object in memory, in bytes

__str__

classmethod args()

Return a list of args of the `__init__` function of a class

Returns the list of argument names. No information about defaults is included.

Return type list of str

classmethod base()

Return the most parent class actually derived from StorableObject

Important to determine which store should be used for storage

Returns the base class

Return type `type`

base_cls

Return the base class

Returns the base class

Return type `type`

See also:

`base()`

base_cls_name

Return the name of the base class

Returns the string representation of the base class

Return type `str`

cls

Return the class name as a string

Returns the class name

Return type `str`

static count_weaks ()

Return number of objects subclassed from StorableObject still in memory

This includes objects not yet recycled by the garbage collector.

Returns **dict of str** – the dictionary which assigns the base class name of each references objects the integer number of objects still present

Return type `int`

default_name

Return the default name.

Usually derived from the objects class

Returns the default name

Return type `str`

classmethod descendants ()

Return a list of all subclassed objects

Returns list of subclasses of a storable object

Return type list of type

fix_name ()

Set the objects name to be immutable.

Usually called after load and save to fix the stored state.

classmethod from_dict (dct)

Reconstruct an object from a dictionary representaiton

Parameters **dct** (*dict*) – the dictionary containing a state representaion of the class.

Returns the reconstructed storable object

Return type `openpathsampling.netcdfplus.StorableObject`

idx (store)

Return the index which is used for the object in the given store.

Once you store a storable object in a store it gets assigned a unique number that can be used to retrieve the object back from the store. This function will ask the given store if the object is stored if so what the used index is.

Parameters **store** (`openpathsampling.netcdfplus.ObjectStore`) – the store in which to ask for the index

Returns the integer index for the object of it exists or *None* else

Return type `int` or *None*

is_named

True if this object has a custom name.

This distinguishes default algorithmic names from assigned names.

name

Return the current name of the object.

If no name has been set a default generated name is returned.

Returns the name of the object

Return type `str`

named (*name*)

Name an unnamed object.

This only renames the object if it does not yet have a name. It can be used to chain the naming onto the object creation. It should also be used when naming things algorithmically: directly setting the `.name` attribute could override a user-defined name.

Parameters **name** (*str*) – the name to be used for the object. Can only be set once

Examples

```
>>> import openpathsampling as p
>>> full = p.FullVolume().named('myFullVolume')
```

static objects ()

Returns a dictionary of all storable objects

Returns **dict of str** – a dictionary of all subclassed objects from `StorableObject`. The name points to the class

Return type `type`

parse_summary_line (*line*)

Parse a line from the summary file.

To control the parsing, set the `OneWayTPSConverter.options` (see `TPSConverterOptions`).

Parameters **line** (*str*) – the input line

Returns

- **replica** (*int*) – always zero for now
- **trial_trajectory** (*openpathsampling.Trajectory*) – one-way trial segments
- **shooting_point_index** (*int*) – index of the shooting point based on the previous trajectory (None if no previous trajectory)
- **accepted** (*bool*) – whether the trial was accepted
- **direction** (*1 or -1*) – positive if forward shooting, negative if backward

run (*summary_file_name, n_trajs_per_block=None*)

Parameters **step_info_list** (*list of tuple*) – (replica, trial_trajectory, shooting_point_index, accepted) or (replica, one_way_trial_segment, shooting_point_index, accepted, direction)

save_initial_step ()

Save the initial state as an `MCStep` to the storage

static set_observer (*active*)

(De-)Activate observing creation of storable objects

This can be used to track which storable objects are still alive and hence look for memory leaks and inspect caching. Use `openpathsampling.netcdfplus.base.StorableObject.count_weaks()` to get the current summary of created objects

Parameters **active** (*bool*) – if *True* then observing is enabled. *False* disables observing. Per default observing is disabled.

See also:

`openpathsampling.netcdfplus.base.StorableObject.count_weaks()`

sync_storage ()

Will sync all collective variables and the storage to disk

to_dict ()

Convert object into a dictionary representation

Used to convert the dictionary into JSON string for serialization

Returns the dictionary representing the (immutable) state of the object

Return type `dict`

class `ops_piggybacker.GromacsOneWayTPSConverter` (*storage*, *network*, *initial_file*,
topology_file, *options=None*, *options_rejected=None*)

Bases: `ops_piggybacker.one_way_tps_converters.OneWayTPSConverter`

__delattr__

`x.__delattr__('name') <==> del x.name`

__format__ ()

default object formatter

__getattr__

`x.__getattr__('name') <==> x.name`

__reduce__ ()

helper for pickle

__reduce_ex__ ()

helper for pickle

__repr__

__setattr__

`x.__setattr__('name', value) <==> x.name = value`

__sizeof__ () → int

size of object in memory, in bytes

__str__

classmethod args ()

Return a list of args of the `__init__` function of a class

Returns the list of argument names. No information about defaults is included.

Return type list of str

classmethod base ()

Return the most parent class actually derived from `StorableObject`

Important to determine which store should be used for storage

Returns the base class

Return type `type`

base_cls

Return the base class

Returns the base class

Return type `type`

See also:

`base()`

base_cls_name

Return the name of the base class

Returns the string representation of the base class

Return type `str`

cls

Return the class name as a string

Returns the class name

Return type `str`

static count_weaks()

Return number of objects subclassed from StorableObject still in memory

This includes objects not yet recycled by the garbage collector.

Returns **dict of str** – the dictionary which assigns the base class name of each references objects the integer number of objects still present

Return type `int`

default_name

Return the default name.

Usually derived from the objects class

Returns the default name

Return type `str`

classmethod descendants()

Return a list of all subclassed objects

Returns list of subclasses of a storable object

Return type list of type

fix_name()

Set the objects name to be immutable.

Usually called after load and save to fix the stored state.

classmethod from_dict(dct)

Reconstruct an object from a dictionary representaiton

Parameters **dct** (*dict*) – the dictionary containing a state representaion of the class.

Returns the reconstructed storable object

Return type `openpathsampling.netcdfplus.StorableObject`

idx (*store*)

Return the index which is used for the object in the given store.

Once you store a storable object in a store it gets assigned a unique number that can be used to retrieve the object back from the store. This function will ask the given store if the object is stored if so what the used index is.

Parameters **store** (`openpathsampling.netcdfplus.ObjectStore`) – the store in which to ask for the index

Returns the integer index for the object if it exists or *None* else

Return type `int` or *None*

is_named

True if this object has a custom name.

This distinguishes default algorithmic names from assigned names.

load_trajectory (*file_name*)

Creates an OPS trajectory from the given file

name

Return the current name of the object.

If no name has been set a default generated name is returned.

Returns the name of the object

Return type `str`

named (*name*)

Name an unnamed object.

This only renames the object if it does not yet have a name. It can be used to chain the naming onto the object creation. It should also be used when naming things algorithmically: directly setting the `.name` attribute could override a user-defined name.

Parameters **name** (*str*) – the name to be used for the object. Can only be set once

Examples

```
>>> import openpathsampling as p
>>> full = p.FullVolume().named('myFullVolume')
```

static objects ()

Returns a dictionary of all storable objects

Returns **dict of str** – a dictionary of all subclassed objects from `StorableObject`. The name points to the class

Return type `type`

parse_summary_line (*line*)

Parse a line from the summary file.

To control the parsing, set the `OneWayTPSConverter.options` (see *TPSConverterOptions*).

Parameters **line** (*str*) – the input line

Returns

- **replica** (*0*) – always zero for now
- **trial_trajectory** (*openpathsampling.Trajectory*) – one-way trial segments
- **shooting_point_index** (*int*) – index of the shooting point based on the previous trajectory (None if no previous trajectory)
- **accepted** (*bool*) – whether the trial was accepted
- **direction** (*1 or -1*) – positive if forward shooting, negative if backward

run (*summary_file_name, n_trajs_per_block=None*)

Parameters **step_info_list** (*list of tuple*) – (replica, trial_trajectory, shooting_point_index, accepted) or (replica, one_way_trial_segment, shooting_point_index, accepted, direction)

save_initial_step ()

Save the initial state as an MCStep to the storage

static set_observer (*active*)

(De-)Activate observing creation of storable objects

This can be used to track which storable objects are still alive and hence look for memory leaks and inspect caching. Use `openpathsampling.netcdfplus.base.StorableObject.count_weaks()` to get the current summary of created objects

Parameters **active** (*bool*) – if *True* then observing is enabled. *False* disables observing. Per default observing is disabled.

See also:

`openpathsampling.netcdfplus.base.StorableObject.count_weaks()`

sync_storage ()

Will sync all collective variables and the storage to disk

to_dict ()

Convert object into a dictionary representation

Used to convert the dictionary into JSON string for serialization

Returns the dictionary representing the (immutable) state of the object

Return type `dict`

6.2 Mover Stubs

```
class ops_piggybacker.ShootingStub (ensemble, selector=None, engine=None,  
                                     pre_joined=True)
```

Bases: `openpathsampling.pathmover.PathMover`

Stub to mimic a shooting move.

Parameters

- **ensemble** (*paths.Ensemble*) – the ensemble for the shooting mover
- **selector** (*paths.ShootingPointSelector or None*) – the selector for the shooting point. Default *None* creates a *UniformSelector*. Currently, only *UniformSelector* is supported.

- **engine** (*paths.engines.DynamicsEngine*) – the engine to report as the source of the dynamics
- **pre_joined** (*bool*) – whether the input trial trajectories are pre-joined into complete trajectories, or take partial one-way segments which should be dynamically joined. Currently defaults to `pre_joined=True` (likely to change soon, though).

mimic

paths.OneWayShootingMover – the mover that this stub mimics

__delattr__

`x.__delattr__('name') <==> del x.name`

__format__ ()

default object formatter

__getattr__

`x.__getattr__('name') <==> x.name`

__reduce__ ()

helper for pickle

__reduce_ex__ ()

helper for pickle

__repr__

__setattr__

`x.__setattr__('name', value) <==> x.name = value`

__sizeof__ () → int

size of object in memory, in bytes

classmethod args ()

Return a list of args of the `__init__` function of a class

Returns the list of argument names. No information about defaults is included.

Return type list of str

classmethod base ()

Return the most parent class actually derived from `StorableObject`

Important to determine which store should be used for storage

Returns the base class

Return type `type`

base_cls

Return the base class

Returns the base class

Return type `type`

See also:

`base()`

base_cls_name

Return the name of the base class

Returns the string representation of the base class

Return type `str`

cls

Return the class name as a string

Returns the class name

Return type `str`

static count_weak ()

Return number of objects subclassed from StorableObject still in memory

This includes objects not yet recycled by the garbage collector.

Returns **dict of str** – the dictionary which assigns the base class name of each references objects the integer number of objects still present

Return type `int`

depth_post_order (*fnc*, *level=0*, ***kwargs*)

Traverse the tree in post-order applying a function with depth

This traverses the underlying tree and applies the given function at each node returning a list of the results. Post-order means that subnodes are called BEFORE the node itself is evaluated.

Parameters

- **fnc** (*function* (*node*, ***kwargs*)) – the function run at each node. It is given the node and the optional (fixed) parameters
- **level** (*int*) – the initial level
- **kwargs** (*named arguments*) – optional arguments added to the function

Returns flattened list of tuples of results of the map. First part of the tuple is the level, second part is the function result.

Return type list of tuple(level, func(node, **kwargs))

See also:

`map_pre_order()`, `map_post_order()`, `level_pre_order()`, `level_post_order()`

depth_pre_order (*fnc*, *level=0*, *only_canonical=False*, ***kwargs*)

Traverse the tree of node in pre-order applying a function

This traverses the underlying tree applies the given function at each node returning a list of the results. Pre-order means that subnodes are called AFTER the node itself is evaluated.

Parameters

- **fnc** (*function* (*node*, ***kwargs*)) – the function run at each node. It is given the node and the optional parameters
- **level** (*int*) – the initial level
- **only_canonical** (*bool*, *default: False*) – if *True* the recursion stops at canonical movers and will hence be more compact
- **kwargs** (*named arguments*) – optional arguments added to the function

Returns flattened list of tuples of results of the map. First part of the tuple is the level, second part is the function result.

Return type list of tuple(level, fnc(node, **kwargs))

See also:

`map_pre_order()`, `map_post_order()`, `level_pre_order()`, `level_post_order()`

classmethod descendants()

Return a list of all subclassed objects

Returns list of subclasses of a storable object

Return type list of type

fix_name()

Set the objects name to be immutable.

Usually called after load and save to fix the stored state.

classmethod from_dict(dct)

Reconstruct an object from a dictionary representaiton

Parameters *dct* (*dict*) – the dictionary containing a state representaion of the class.

Returns the reconstructed storable object

Return type `openpathsampling.netcdfplus.StorableObject`

idx(store)

Return the index which is used for the object in the given store.

Once you store a storable object in a store it gets assigned a unique number that can be used to retrieve the object back from the store. This function will ask the given store if the object is stored if so what the used index is.

Parameters *store* (`openpathsampling.netcdfplus.ObjectStore`) – the store in which to ask for the index

Returns the integer index for the object if it exists or *None* else

Return type `int` or *None*

in_out

List the input -> output relation for ensembles

A mover will pick one or more replicas from specific ensembles. Alter them (or not) and place these (or additional ones) in specific ensembles. This relation can be visualized as a mapping of input to output ensembles. Like

ReplicaExchange ens1 -> ens2 ens2 -> ens1

EnsembleHop (A sample in ens1 will disappear and appear in ens2) ens1 -> ens2

DuplicateMover (create a copy with a new replica number) Not used yet! ens1 -> ens1 None -> ens1

Returns

- **list of list of tuple** (`((openpathsampling.Ensemble,)`
- `openpathsampling.Ensemble)` – a list of possible lists of tuples of ensembles.

Notes

The default implementation will (1) in case of a single input and output connect the two, (2) return nothing if there are no out_ensembles and (3) for more then two require implementation

input_ensembles

Return a list of possible used ensembles for this mover

This list contains all Ensembles from which this mover might pick samples. This is very useful to determine on which ensembles a mover acts for analysis and sanity checking.

Returns the list of input ensembles

Return type list of `openpathsampling.Ensemble`

is_named

True if this object has a custom name.

This distinguishes default algorithmic names from assigned names.

static join_one_way (*input_trajectory, partial_trial, shooting_point, direction*)

Create a one-way trial trajectory

Parameters

- **input_trajectory** (*paths.Trajectory*) – the previous complete trajectory
- **partial_trial** (*paths.Trajectory*) – The partial (one-way) trial trajectory. Must *not* include the shooting point.
- **shooting_point** (*paths.Snapshot*) – the snapshot for the shooting point – must be a member of the input trajectory
- **direction** (*+1 or -1*) – if positive, treat as forward shooting; if negative, treat as backward shooting

Returns the complete trial trajectory

Return type `paths.Trajectory`

keylist ()

Return a list of key : subtree tuples

Returns A list of all subtrees with their respective keys

Return type list of tuple(key, subtree)

static legal_sample_set (*sample_set, ensembles=None, replicas='all'*)

This returns all the samples from *sample_set* which are in both *self.replicas* and the parameter ensembles. If *ensembles* is *None*, we use *self.ensembles*. If you want all ensembles allowed, pass *ensembles='all'*.

Parameters

- **sample_set** (*openpathsampling.SampleSet*) – the sampleset from which to pick specific samples matching certain criteria
- **ensembles** (list of *openpathsampling.Ensembles*) – the ensembles to pick from
- **replicas** (list of int or *all*) – the replicas to pick or *'all'* for all

map_post_order (*fnc, **kwargs*)

Traverse the tree in post-order applying a function

This traverses the underlying tree and applies the given function at each node returning a list of the results. Post-order means that subnodes are called BEFORE the node itself is evaluated.

Parameters

- **fnc** (*function(node, kwargs)*) – the function run at each node. It is given the node and the optional (fixed) parameters
- **kwargs** (*named arguments*) – optional arguments added to the function

Returns flattened list of the results of the map

Return type list (*fnc(node, **kwargs)*)

Notes

This uses the same order as *reversed()*

See also:

map_pre_order(), *map_post_order()*, *level_pre_order()*, *level_post_order()*

map_pre_order (*fnc*, ***kwargs*)

Traverse the tree in pre-order applying a function

This traverses the underlying tree applies the given function at each node returning a list of the results. Pre-order means that subnodes are called AFTER the node itself is evaluated.

Parameters

- **fnc** (*function* (*node*, ***kwargs*)) – the function run at each node. It is given the node and the optional parameters
- **kwargs** (*named arguments*) – optional arguments added to the function

Returns flattened list of the results of the map

Return type *list* (*fnc*(*node*, ***kwargs*))

Notes

This uses the same order as *iter()*

See also:

map_pre_order(), *map_post_order()*, *level_pre_order()*, *level_post_order()*

map_tree (*fnc*)

Apply a function to each node and return a nested tree of results

Parameters

- **fnc** (*function* (*node*, *args*, *kwargs*)) – the function run at each node node. It is given the node and the optional (fixed) parameters
- **kwargs** (*named arguments*) – optional arguments added to the function

Returns nested list of the results of the map

Return type *tree* (*fnc*(*node*, ***kwargs*))

move (*input_sample*, *trial_trajectory*, *shooting_point*, *accepted*, *direction=None*)

Fake a move.

Parameters

- **input_sample** (*paths.Sample*) – the input sample for this shooting move
- **trial_trajectory** (*paths.Trajectory*) – the trial trajectory generated by this move
- **shooting_point** (*paths.Snapshot*) – the shooting point snapshot for this trial
- **accepted** (*bool*) – whether the trial was accepted
- **direction** (*+1*, *-1*, or *None*) – direction of the shooting move (positive is forward, negative is backward). If *self.pre_joined* is *True*, the trial trajectory is reconstructed from the parts. To use the exact input trial trajectory with *self.pre_joined == True*, set *direction=None*

name

Return the current name of the object.

If no name has been set a default generated name is returned.

Returns the name of the object

Return type `str`

named (*name*)

Name an unnamed object.

This only renames the object if it does not yet have a name. It can be used to chain the naming onto the object creation. It should also be used when naming things algorithmically: directly setting the `.name` attribute could override a user-defined name.

Parameters **name** (*str*) – the name to be used for the object. Can only be set once

Examples

```
>>> import openpathsampling as p
>>> full = p.FullVolume().named('myFullVolume')
```

static objects ()

Returns a dictionary of all storable objects

Returns **dict of str** – a dictionary of all subclassed objects from StorableObject. The name points to the class

Return type `type`

output_ensembles

Return a list of possible returned ensembles for this mover

This list contains all Ensembles for which this mover might return samples. This is very useful to determine on which ensembles a mover affects in later steps for analysis and sanity checking.

Returns the list of output ensembles

Return type list of Ensemble

static select_sample (*sample_set*, *ensembles=None*, *replicas=None*)

Returns one of the legal samples given self.replica and the ensemble set in ensembles.

Parameters

- **sample_set** (*openpathsampling.SampleSet*) – the sampleset from which to pick specific samples matching certain criteria
- **ensembles** (list of *openpathsampling.Ensembles* or *None*) – the ensembles to pick from or *None* for all
- **replicas** (list of *int* or *None*) – the replicas to pick or *None* for all

static set_observer (*active*)

(De-)Activate observing creation of storable objects

This can be used to track which storable objects are still alive and hence look for memory leaks and inspect caching. Use `openpathsampling.netcdfplus.base.StorableObject.count_weaks()` to get the current summary of created objects

Parameters **active** (*bool*) – if *True* then observing is enabled. *False* disables observing. Per default observing is disabled.

See also:

`openpathsampling.netcdfplus.base.StorableObject.count_weeks()`

sub_replica_state (*replica_states*)

Return set of replica states that a submover might be called with

Parameters **replica_states** (set of *openpathsampling.pathmover_inout.ReplicaState*) –

Returns

Return type list of set of *ReplicaState*

submovers

Returns a list of submovers

Returns the list of sub-movers

Return type list of *openpathsampling.PathMover*

to_dict ()

Convert object into a dictionary representation

Used to convert the dictionary into JSON string for serialization

Returns the dictionary representing the (immutable) state of the object

Return type `dict`

tree ()

Return the object as a tree structure of nested lists of nodes

Returns the tree in nested list format

Return type nested list of nodes

6.3 Simulation Stubs

class `ops_piggybacker.ShootingPseudoSimulator` (*storage, initial_conditions, mover, network*)

Bases: *openpathsampling.pathsimulator.PathSimulator*

Pseudo-simulator for shooting-only mimics.

Parameters

- **storage** (*openpathsampling.netcdfplus.Storage*) – file to store OPS-ready analysis
- **initial_conditions** (*openpathsampling.SampleSet*) – sample set giving the OPS version of the initial conditions
- **mover** (*ShootingStub*) – stub to mimic the shooting mover
- **network** (*openpathsampling.TransitionNetwork*) – transition network with information about this system

__delattr__

`x.__delattr__('name') <==> del x.name`

__format__ ()

default object formatter

__getattr__

`x.__getattr__('name') <==> x.name`

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__

__setattr__
x.__setattr__('name', value) <==> x.name = value

__sizeof__() → int
size of object in memory, in bytes

__str__

classmethod args()
Return a list of args of the `__init__` function of a class

Returns the list of argument names. No information about defaults is included.

Return type list of str

classmethod base()
Return the most parent class actually derived from StorableObject

Important to determine which store should be used for storage

Returns the base class

Return type `type`

base_cls
Return the base class

Returns the base class

Return type `type`

See also:

[`base\(\)`](#)

base_cls_name
Return the name of the base class

Returns the string representation of the base class

Return type `str`

cls
Return the class name as a string

Returns the class name

Return type `str`

static count_weak()
Return number of objects subclassed from StorableObject still in memory

This includes objects not yet recycled by the garbage collector.

Returns **dict of str** – the dictionary which assigns the base class name of each references objects
the integer number of objects still present

Return type `int`

default_name

Return the default name.

Usually derived from the objects class

Returns the default name

Return type `str`

classmethod descendants()

Return a list of all subclassed objects

Returns list of subclasses of a storable object

Return type list of type

fix_name()

Set the objects name to be immutable.

Usually called after load and save to fix the stored state.

classmethod from_dict(dict)

Reconstruct an object from a dictionary representaiton

Parameters `dct` (`dict`) – the dictionary containing a state representaion of the class.

Returns the reconstructed storable object

Return type `openpathsampling.netcdfplus.StorableObject`

idx(store)

Return the index which is used for the object in the given store.

Once you store a storable object in a store it gets assigned a unique number that can be used to retrieve the object back from the store. This function will ask the given store if the object is stored if so what the used index is.

Parameters `store` (`openpathsampling.netcdfplus.ObjectStore`) – the store in which to ask for the index

Returns the integer index for the object if it exists or *None* else

Return type `int` or `None`

is_named

True if this object has a custom name.

This distinguishes default algorithmic names from assigned names.

name

Return the current name of the object.

If no name has been set a default generated name is returned.

Returns the name of the object

Return type `str`

named(name)

Name an unnamed object.

This only renames the object if it does not yet have a name. It can be used to chain the naming onto the object creation. It should also be used when naming things algorithmically: directly setting the `.name` attribute could override a user-defined name.

Parameters `name` (`str`) – the name to be used for the object. Can only be set once

Examples

```
>>> import openpathsampling as p
>>> full = p.FullVolume().named('myFullVolume')
```

static `objects()`

Returns a dictionary of all storable objects

Returns dict of str – a dictionary of all subclassed objects from StorableObject. The name points to the class

Return type `type`

run (*step_info_list*)

Parameters *step_info_list* (*list of tuple*) – (replica, trial_trajectory, shooting_point_index, accepted) or (replica, one_way_trial_segment, shooting_point_index, accepted, direction)

save_initial_step ()

Save the initial state as an MCStep to the storage

static `set_observer` (*active*)

(De-)Activate observing creation of storable objects

This can be used to track which storable objects are still alive and hence look for memory leaks and inspect caching. Use `openpathsampling.netcdfplus.base.StorableObject.count_weaks()` to get the current summary of created objects

Parameters *active* (*bool*) – if *True* then observing is enabled. *False* disables observing. Per default observing is disabled.

See also:

`openpathsampling.netcdfplus.base.StorableObject.count_weaks()`

sync_storage ()

Will sync all collective variables and the storage to disk

to_dict ()

Convert object into a dictionary representation

Used to convert the dictionary into JSON string for serialization

Returns the dictionary representing the (immutable) state of the object

Return type `dict`

Symbols

<code>__add__</code> (ops_piggybacker.TPSConverterOptions attribute), 13	<code>__getslice__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__contains__</code> (ops_piggybacker.TPSConverterOptions attribute), 13	<code>__gt__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__delattr__</code> (ops_piggybacker.GromacsOneWayTPSConverter attribute), 18	<code>__hash__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__delattr__</code> (ops_piggybacker.OneWayTPSConverter attribute), 15	<code>__iter__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__delattr__</code> (ops_piggybacker.ShootingPseudoSimulator attribute), 28	<code>__le__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__delattr__</code> (ops_piggybacker.ShootingStub attribute), 22	<code>__len__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__delattr__</code> (ops_piggybacker.TPSConverterOptions attribute), 13	<code>__lt__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__eq__</code> (ops_piggybacker.TPSConverterOptions attribute), 13	<code>__mul__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__format__</code> () (ops_piggybacker.GromacsOneWayTPSConverter method), 18	<code>__ne__</code> (ops_piggybacker.TPSConverterOptions attribute), 14
<code>__format__</code> () (ops_piggybacker.OneWayTPSConverter method), 15	<code>__reduce__</code> () (ops_piggybacker.GromacsOneWayTPSConverter method), 18
<code>__format__</code> () (ops_piggybacker.ShootingPseudoSimulator method), 28	<code>__reduce__</code> () (ops_piggybacker.OneWayTPSConverter method), 15
<code>__format__</code> () (ops_piggybacker.ShootingStub method), 22	<code>__reduce__</code> () (ops_piggybacker.ShootingPseudoSimulator method), 29
<code>__format__</code> () (ops_piggybacker.TPSConverterOptions method), 13	<code>__reduce__</code> () (ops_piggybacker.ShootingStub method), 22
<code>__ge__</code> (ops_piggybacker.TPSConverterOptions attribute), 13	<code>__reduce__</code> () (ops_piggybacker.TPSConverterOptions method), 14
<code>__getattr__</code> (ops_piggybacker.GromacsOneWayTPSConverter attribute), 18	<code>__reduce_ex__</code> () (ops_piggybacker.GromacsOneWayTPSConverter method), 18
<code>__getattr__</code> (ops_piggybacker.OneWayTPSConverter attribute), 15	<code>__reduce_ex__</code> () (ops_piggybacker.OneWayTPSConverter method), 15
<code>__getattr__</code> (ops_piggybacker.ShootingPseudoSimulator attribute), 28	<code>__reduce_ex__</code> () (ops_piggybacker.ShootingPseudoSimulator method), 29
<code>__getattr__</code> (ops_piggybacker.ShootingStub attribute), 22	<code>__reduce_ex__</code> () (ops_piggybacker.ShootingStub method), 22
<code>__getattr__</code> (ops_piggybacker.TPSConverterOptions attribute), 14	<code>__reduce_ex__</code> () (ops_piggybacker.TPSConverterOptions method), 14
<code>__getitem__</code> (ops_piggybacker.TPSConverterOptions attribute), 14	<code>__repr__</code> (ops_piggybacker.GromacsOneWayTPSConverter attribute), 14

attribute), 18
 __repr__ (ops_piggybacker.OneWayTPSConverter attribute), 15
 __repr__ (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 __repr__ (ops_piggybacker.ShootingStub attribute), 22
 __rmul__ (ops_piggybacker.TPSConverterOptions attribute), 14
 __setattr__ (ops_piggybacker.GromacsOneWayTPSConverter attribute), 18
 __setattr__ (ops_piggybacker.OneWayTPSConverter attribute), 15
 __setattr__ (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 __setattr__ (ops_piggybacker.ShootingStub attribute), 22
 __setattr__ (ops_piggybacker.TPSConverterOptions attribute), 14
 __sizeof__ () (ops_piggybacker.GromacsOneWayTPSConverter method), 18
 __sizeof__ () (ops_piggybacker.OneWayTPSConverter method), 15
 __sizeof__ () (ops_piggybacker.ShootingPseudoSimulator method), 29
 __sizeof__ () (ops_piggybacker.ShootingStub method), 22
 __sizeof__ () (ops_piggybacker.TPSConverterOptions method), 14
 __str__ (ops_piggybacker.GromacsOneWayTPSConverter attribute), 18
 __str__ (ops_piggybacker.OneWayTPSConverter attribute), 15
 __str__ (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 __str__ (ops_piggybacker.TPSConverterOptions attribute), 14
 base() (ops_piggybacker.ShootingStub class method), 22
 base_cls (ops_piggybacker.GromacsOneWayTPSConverter attribute), 19
 base_cls (ops_piggybacker.OneWayTPSConverter attribute), 15
 base_cls (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 base_cls (ops_piggybacker.ShootingStub attribute), 22
 base_cls_name (ops_piggybacker.GromacsOneWayTPSConverter attribute), 19
 base_cls_name (ops_piggybacker.OneWayTPSConverter attribute), 16
 base_cls_name (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 base_cls_name (ops_piggybacker.ShootingStub attribute), 22
 cls (ops_piggybacker.GromacsOneWayTPSConverter attribute), 19
 cls (ops_piggybacker.OneWayTPSConverter attribute), 16
 cls (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 cls (ops_piggybacker.ShootingStub attribute), 22
 count() (ops_piggybacker.TPSConverterOptions method), 14
 count_weak() (ops_piggybacker.GromacsOneWayTPSConverter static method), 19
 count_weak() (ops_piggybacker.OneWayTPSConverter static method), 16
 count_weak() (ops_piggybacker.ShootingPseudoSimulator static method), 29
 count_weak() (ops_piggybacker.ShootingStub static method), 23

A

args() (ops_piggybacker.GromacsOneWayTPSConverter class method), 18
 args() (ops_piggybacker.OneWayTPSConverter class method), 15
 args() (ops_piggybacker.ShootingPseudoSimulator class method), 29
 args() (ops_piggybacker.ShootingStub class method), 22
 auto_reverse (ops_piggybacker.TPSConverterOptions attribute), 14

B

base() (ops_piggybacker.GromacsOneWayTPSConverter class method), 18
 base() (ops_piggybacker.OneWayTPSConverter class method), 15
 base() (ops_piggybacker.ShootingPseudoSimulator class method), 29

D

default_name (ops_piggybacker.GromacsOneWayTPSConverter attribute), 19
 default_name (ops_piggybacker.OneWayTPSConverter attribute), 16
 default_name (ops_piggybacker.ShootingPseudoSimulator attribute), 29
 depth_post_order() (ops_piggybacker.ShootingStub method), 23
 depth_pre_order() (ops_piggybacker.ShootingStub method), 23
 descendants() (ops_piggybacker.GromacsOneWayTPSConverter class method), 19
 descendants() (ops_piggybacker.OneWayTPSConverter class method), 16
 descendants() (ops_piggybacker.ShootingPseudoSimulator class method), 30

descendants() (ops_piggybacker.ShootingStub class method), 23

F

fix_name() (ops_piggybacker.GromacsOneWayTPSConverter method), 19

fix_name() (ops_piggybacker.OneWayTPSConverter method), 16

fix_name() (ops_piggybacker.ShootingPseudoSimulator method), 30

fix_name() (ops_piggybacker.ShootingStub method), 24

from_dict() (ops_piggybacker.GromacsOneWayTPSConverter class method), 19

from_dict() (ops_piggybacker.OneWayTPSConverter class method), 16

from_dict() (ops_piggybacker.ShootingPseudoSimulator class method), 30

from_dict() (ops_piggybacker.ShootingStub class method), 24

full_trajectory (ops_piggybacker.TPSConverterOptions attribute), 14

G

GromacsOneWayTPSConverter (class in ops_piggybacker), 18

I

idx() (ops_piggybacker.GromacsOneWayTPSConverter method), 20

idx() (ops_piggybacker.OneWayTPSConverter method), 16

idx() (ops_piggybacker.ShootingPseudoSimulator method), 30

idx() (ops_piggybacker.ShootingStub method), 24

in_out (ops_piggybacker.ShootingStub attribute), 24

includes_shooting_point (ops_piggybacker.TPSConverterOptions attribute), 14

index() (ops_piggybacker.TPSConverterOptions method), 14

input_ensembles (ops_piggybacker.ShootingStub attribute), 24

is_named (ops_piggybacker.GromacsOneWayTPSConverter attribute), 20

is_named (ops_piggybacker.OneWayTPSConverter attribute), 16

is_named (ops_piggybacker.ShootingPseudoSimulator attribute), 30

is_named (ops_piggybacker.ShootingStub attribute), 25

J

join_one_way() (ops_piggybacker.ShootingStub static method), 25

K

keylist() (ops_piggybacker.ShootingStub method), 25

L

legal_sample_set() (ops_piggybacker.ShootingStub static method), 25

load_trajectory() (ops_piggybacker.GromacsOneWayTPSConverter method), 20

M

map_post_order() (ops_piggybacker.ShootingStub method), 25

map_pre_order() (ops_piggybacker.ShootingStub method), 26

map_tree() (ops_piggybacker.ShootingStub method), 26

mimic (ops_piggybacker.ShootingStub attribute), 22

move() (ops_piggybacker.ShootingStub method), 26

N

name (ops_piggybacker.GromacsOneWayTPSConverter attribute), 20

name (ops_piggybacker.OneWayTPSConverter attribute), 17

name (ops_piggybacker.ShootingPseudoSimulator attribute), 30

name (ops_piggybacker.ShootingStub attribute), 26

named() (ops_piggybacker.GromacsOneWayTPSConverter method), 20

named() (ops_piggybacker.OneWayTPSConverter method), 17

named() (ops_piggybacker.ShootingPseudoSimulator method), 30

named() (ops_piggybacker.ShootingStub method), 27

O

objects() (ops_piggybacker.GromacsOneWayTPSConverter static method), 20

objects() (ops_piggybacker.OneWayTPSConverter static method), 17

objects() (ops_piggybacker.ShootingPseudoSimulator static method), 31

objects() (ops_piggybacker.ShootingStub static method), 27

OneWayTPSConverter (class in ops_piggybacker), 15

output_ensembles (ops_piggybacker.ShootingStub attribute), 27

P

parse_summary_line() (ops_piggybacker.GromacsOneWayTPSConverter method), 20

parse_summary_line() (ops_piggybacker.OneWayTPSConverter method), 17

R

retrim_shooting (ops_piggybacker.TPSCConverterOptions attribute), 14

run() (ops_piggybacker.GromacsOneWayTPSCConverter method), 21

run() (ops_piggybacker.OneWayTPSCConverter method), 17

run() (ops_piggybacker.ShootingPseudoSimulator method), 31

S

save_initial_step() (ops_piggybacker.GromacsOneWayTPSCConverter method), 21

save_initial_step() (ops_piggybacker.OneWayTPSCConverter method), 17

save_initial_step() (ops_piggybacker.ShootingPseudoSimulator method), 31

select_sample() (ops_piggybacker.ShootingStub static method), 27

set_observer() (ops_piggybacker.GromacsOneWayTPSCConverter static method), 21

set_observer() (ops_piggybacker.OneWayTPSCConverter static method), 17

set_observer() (ops_piggybacker.ShootingPseudoSimulator static method), 31

set_observer() (ops_piggybacker.ShootingStub static method), 27

ShootingPseudoSimulator (class in ops_piggybacker), 28

ShootingStub (class in ops_piggybacker), 21

sub_replica_state() (ops_piggybacker.ShootingStub method), 28

submovers (ops_piggybacker.ShootingStub attribute), 28

sync_storage() (ops_piggybacker.GromacsOneWayTPSCConverter method), 21

sync_storage() (ops_piggybacker.OneWayTPSCConverter method), 18

sync_storage() (ops_piggybacker.ShootingPseudoSimulator method), 31

T

to_dict() (ops_piggybacker.GromacsOneWayTPSCConverter method), 21

to_dict() (ops_piggybacker.OneWayTPSCConverter method), 18

to_dict() (ops_piggybacker.ShootingPseudoSimulator method), 31

to_dict() (ops_piggybacker.ShootingStub method), 28

TPSCConverterOptions (class in ops_piggybacker), 13

tree() (ops_piggybacker.ShootingStub method), 28

trim (ops_piggybacker.TPSCConverterOptions attribute), 15