propertyestimator Documentation

propertyestimator

Aug 23, 2019

USER GUIDE

The property estimator is a distributed framework from the Open Forcefield Consortium for storing, manipulating, and computing measured physical properties from simulation data.

Warning: This framework is still in **pre-alpha** and under heavy development. Although all steps have been taken to ensure correctness of the code and the results it produces, the authors accept no liability for any incorrectness any bugs or unintended behaviour may cause.

CHAPTER

ONE

INDEX

User Guide

1.1 Installing the Property Estimator

The Property Estimator is currently installable both from source and through `conda`. Whichever route is chosen, it is recommended to install the estimator within a conda environment, and allow the conda package manager to install the required dependencies.

More information about conda and instructions to perform a lightweight miniconda installation can be found here. It will be assumed that these have been followed and conda is available on your machine.

1.1.1 Installation from Conda

To install the `propertyestimator` from the `omnia` channel, simply run:

```
conda install -c omnia/label/rc propertyestimator
```

1.1.2 Installation from Source

To install Property Estimator from source, clone the repository from github:

```
git clone https://github.com/openforcefield/propertyestimator.git
cd propertyestimator
```

Create a custom conda environment which contains the required dependencies and activate it:

```
conda env create --name propertyestimator --file devtools/conda-envs/test_env.yaml
conda activate propertyestimator
```

The final step is to install the estimator itself:

```
python setup.py develop
```

And that's it!

1.2 Getting Started

The propertyestimator currently exists as two key components:

- a client object which the user can use to request the estimation of data sets of physical properties.
- a server object which accepts requests from a client and performs the estimations.

Warning: These instructions are still a work in progress, and may not run as expected.

1.2.1 Creating an Estimator Server

The PropertyEstimatorServer class creates objects that handle property estimation of all of the properties in a dataset given a set.

Create the file run_server.py. Tell server to log to file in case of failure:

```
setup_timestamp_logging()
```

Create directory structure to store intermediary results:

```
# Set the name of the directory in which all temporary files
# will be generated.
working_directory = 'working_directory'

# Remove any existing data.
if path.isdir(working_directory):
    shutil.rmtree(working_directory)
```

Set up a calculation backend. Different backends will take different optional arguments, but here is an example that will launch a backend with a single worker process:

```
# Create a calculation backend to perform workflow
# calculations on.
calculation_backend = DaskLocalCluster(1)
```

Set up storage the storage backend which will cache any generated simulation data:

```
# Create a backend to handle storing and retrieving
# cached simulation data.
storage_backend = LocalFileStorage()
```

Start the server running:

```
# Create a server instance.
property_server = server.PropertyEstimatorServer(calculation_backend,
                                                 storage_backend,
                                                 working_directory=working_directory)

# Tell the server to start listening for incoming
# estimation requests.
property_server.start_listening_loop()
```

To start the server, call the following command from the command line:

```
python run_server.py
```

The server will wait for requests until killed.

1.2.2 Submitting Estimation Requests

Create the file `run_client.py` Load in the data set of properties to estimate, and the force field parameters to use in the calculations:

```
# Load in the data set of interest.
data_set = ThermoMLDataSet.from_file(get_data_filename('properties/single_density.xml
↪'))

# Load in the force field to use.
force_field = smirnoff.ForceField('smirnoff99Frosst-1.1.0.offxml')
```

Create the client object and use it to send the estimation request to the server:

```
# Create the client object.
property_estimator = client.PropertyEstimatorClient()
# Submit the request to a running server.
result = property_estimator.request_estimate(data_set, force_field)
```

Query the result until all of the properties have either been estimated or have errored:

```
# Wait for the results synchronously.
results = request.results(True)
logging.info('The server has returned a response: {}'.format(result))
```

Save the results to a file:

```
with open('results.json', 'w') as file:

    json_results = json.dump(results, file, sort_keys=True, indent=2,
                             separators=(',', ': '), cls=TypedJSONEncoder)
```

1.3 Physical Property Measurements

Warning: This text is now out of date, but will be updated in future to reflect the latest version of the framework.

Physical property measurements are measured properties of a substance that provide some information about the physical parameters that define the interactions within the substance.

A physical property is defined by a combination of:

- A `Mixture` specifying the substance that the measurement was performed on
- A `ThermodynamicState` specifying the thermodynamic conditions under which the measurement was performed
- A `PhysicalProperty` is the physical property that was measured

- A MeasurementMethod specifying the kind of measurement that was performed

An example of each:

- Mixture: a 0.8 mole fraction mixture of ethanol and water
- ThermodynamicState: 298 kelvin, 1 atmosphere
- PhysicalProperty: mass density
- MeasurementMethod: vibrating tube method

1.3.1 Physical substances

We generally use the concept of a liquid or gas Mixture, which is a subclass of Substance.

A simple liquid has only one component:

```
liquid = Mixture()
liquid.add_component('water')
```

A binary mixture has two components:

```
binary_mixture = Mixture()
binary_mixture.add_component('water', mole_fraction=0.2)
binary_mixture.add_component('methanol') # assumed to be rest of mixture if no mole_
↪fraction specified
```

A ternary mixture has three components:

```
ternary_mixture = Mixture()
ternary_mixture.add_component('ethanol', mole_fraction=0.2)
ternary_mixture.add_component('methanol', mole_fraction=0.2)
ternary_mixture.add_component('water')
```

The infinite dilution of one solute within a solvent or mixture is also specified as a Mixture, where the solute has zero mole fraction:

```
infinite_dilution = Mixture()
infinite_dilution.add_component('phenol', impurity=True) # infinite dilution; one␣
↪copy only of the impurity
infinite_dilution.add_component('water')
```

You can iterate over the components in a mixture:

```
for component in mixture.components:
    print (component.iupac_name, component.mole_fraction)
```

retrieve a component by name:

```
component = mixture.components['ethanol']
```

or get the number of components in a mixture:

```
ncomponents = mixture.ncomponents
```

or check if a component is an impurity:

```
if component.impurity == True:
    ...
```

1.3.2 Thermodynamic states

A `ThermodynamicState` specifies a combination of thermodynamic parameters (e.g. temperature, pressure) at which a measurement is performed.

```
from simtk import unit
thermodynamic_state = ThermodynamicState(pressure=500*unit.kilopascals,␣
↪temperature=298.15*unit.kelvin)
```

We use the `simtk.unit` unit system from OpenMM for units (though we may later migrate to pint for portability).

1.3.3 Physical property measurements

A `MeasuredPhysicalProperty` is a combination of `Substance`, `ThermodynamicState`, and a unit-bearing measured property `value` and `uncertainty`:

```
# Define mixture
mixture = Mixture()
mixture.addComponent('water', mole_fraction=0.2)
mixture.addComponent('methanol')

# Define thermodynamic state
thermodynamic_state = ThermodynamicState(pressure=500*unit.kilopascals,␣
↪temperature=298.15*unit.kelvin)

# Define measurement
measurement = ExcessMolarEnthalpy(substance, thermodynamic_state, value=83.
↪3863244*unit.kilojoules_per_mole,
                                  uncertainty=0.1220794866*unit.kilojoules_per_mole)
```

The various properties are all subclasses of `MeasuredPhysicalProperty` and generally follow the `<ePropName/>` ThermoML tag names.

Some examples of `MeasuredPhysicalProperty`:

- `MassDensity` - mass density
- `ExcessMolarEnthalpy` - excess partial apparent molar enthalpy
- `HeatCapacity` - molar heat capacity at constant pressure

A roadmap of physical properties to be implemented) is available.

Please raise an issue if your physical property of interest is not listed!

Each `MeasuredPhysicalProperty` has several properties:

- `.substance` - the `Mixture` for which the measurement was made
- `.thermodynamic_state` - the `ThermodynamicState` at which the measurement was made
- `.measurement_method` - the `MeasurementMethod` used to measure the physical property
- `.value` - the unit-bearing measurement value
- `.uncertainty` - the standard uncertainty of the measurement

- `.reference` - the literature reference (if available) for the measurement
- `.DOI` - the literature reference DOI (if available) for the measurement

The value, uncertainty, reference, and DOI do not necessarily need to be defined for a dataset in order for property calculations to be performed.

1.4 Physical Property Data Sets

Warning: This text is now out of date, but will be updated in future to reflect the latest version of the framework.

A `PhysicalPropertyDataset` is a collection of `MeasuredPhysicalProperty` objects that are related in some way.

```
dataset = PhysicalPropertyDataset([measurement1, measurement2])
```

The dataset is iterable:

```
dataset = PhysicalPropertyDataset([measurement1, measurement2])

for measurement in dataset:
    print measurement.value
```

and has accessors to retrieve DOIs and references associated with measurements in the dataset:

```
# Print the DOIs associated with this dataset
print(dataset.DOIs)

# Print the references associated with this dataset
print(dataset.references)
```

For convenience, you can retrieve the dataset as a pandas DataFrame:

```
dataset.to_pandas()
```

1.4.1 ThermoML datasets

A `ThermoMLDataset` object represents a physical property dataset stored in the IUPAC-standard ThermoML) for specifying thermodynamic properties in XML format. `ThermoMLDataset` is a subclass of `PhysicalPropertyDataset`, and provides the same API interface (in addition to some ThermoML-specfic methods).

Direct access to the NIST ThermoML Archive is supported for obtaining physical property measurements in this format directly from the NIST TRC repository.

For example, to retrieve the ThermoML dataset that accompanies this paper, we can simply use the DOI `10.1016/j.jct.2005.03.012` as a key for creating a `PhysicalPropertyDataset` subclassed object from the ThermoML Archive:

```
dataset = ThermoMLDataset(doi='10.1016/j.jct.2005.03.012')
```

You can also specify multiple ThermoML Archive keys to create a dataset from multiple ThermoML files:

```
thermoml_keys = ['10.1021/acs.jced.5b00365', '10.1021/acs.jced.5b00474']
dataset = ThermoMLDataset(doi=thermoml_keys)
```

It is also possible to specify ThermoML datasets housed at other locations, such as

```
dataset = ThermoMLDataset(url='http://openforcefieldgroup.org/thermoml-datasets')
```

or

```
dataset = ThermoMLDataset(url='file:///Users/choderaj/thermoml')
```

or

```
dataset = ThermoMLDataset(doi=['10.1021/acs.jced.5b00365', '10.1021/acs.jced.5b00474
↪'],
                          url='http://openforcefieldgroup.org/thermoml-datasets')
```

or from ThermoML and a different URL:

```
dataset = ThermoMLDataset(doi=thermoml_keys)
dataset.retrieve(doi=local_keys, url='http://openforcefieldgroup.org/thermoml-datasets
↪')
```

You can see which DOIs contribute to the current `ThermoMLDataset` with the convenience functions:

```
print(dataset.DOIs)
```

NIST has compiled a JSON frame of corrections to uncertainties.

These can be used to update or correct data uncertainties and discard outliers using `applyNISTUncertainties()`:

```
# Modify uncertainties according to NIST evaluation
dataset.apply_nist_uncertainties(nist_uncertainties, adjust_uncertainties=True,␣
↪discard_outliers=True)
```

Todo:

- We should merge any other useful parts parts of the ThermoPyL API in here.

1.4.2 Other datasets

In future, we will add interfaces to other online datasets, such as

- BindingDB for retrieving host-guest binding affinity datasets.

Developer Documentation

- *API*
- *Release History*
- *Release Process*

1.5 API

A set of API documents for this projects classes and modules.

1.5.1 Client Side API

PropertyEstimatorClient	The PropertyEstimatorClient is the main object that users of the property estimator will interface with.
PropertyEstimatorOptions	Represents the options options that can be passed to the property estimation server backend.
PropertyEstimatorSubmission	Represents a set of properties to be estimated by the server backend, the parameters which will be used to estimate them, and options about how the properties will be estimated.
PropertyEstimatorResult	Represents the results of attempting to estimate a set of physical properties using the property estimator server backend.
ConnectionOptions	The set of options to use when connecting to a *PropertyEstimatorServer*

PropertyEstimatorClient

class propertyestimator.client.**PropertyEstimatorClient**(*connection_options=<propertyestimator.client.Conn object>*)

The PropertyEstimatorClient is the main object that users of the property estimator will interface with. It is responsible for requesting that a PropertyEstimatorServer estimates a set of physical properties, as well as querying for when those properties have been estimated.

The PropertyEstimatorClient supports two main workflows: one where a PropertyEstimatorServer lives on a remote supercomputing cluster where all of the expensive calculations will be run, and one where the users local machine acts as both the server and the client, and all calculations will be performed locally.

Warning: While the API of this class in now close to being final, the internals and implementation are still heavily under development and is subject to rapid changes.

Examples

Setting up the client instance:

```
>>> from propertyestimator.client import PropertyEstimatorClient
>>> property_estimator = PropertyEstimatorClient()
```

If the PropertyEstimatorServer is not running on the local machine, you will need to specify its address and the port that it is listening on:

```
>>> from propertyestimator.client import ConnectionOptions
>>>
>>> connection_options = ConnectionOptions(server_address='server_address',
>>>                                                          server_port=8000)
>>> property_estimator = PropertyEstimatorClient(connection_options)
```

To asynchronously submit a request to the running server using the default estimator options:

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>> # Filter the dataset to only include densities measured between 130-260 K
>>> from propertyestimator.properties import Density
>>>
>>> data_set.filter_by_property_types(Density)
>>> data_set.filter_by_temperature(min_temperature=130*unit.kelvin, max_
↪temperature=260*unit.kelvin)
>>>
>>> # Load initial parameters
>>> from openforcefield.typing.engines.smirnoff import ForceField
>>> parameters = ForceField('smirnoff99Frosst.offxml')
>>>
>>> request = property_estimator.request_estimate(data_set, parameters)
```

The status of the request can be asynchronously queried by calling

```
>>> results = request.results()
```

or the main thread can be blocked until the results are available by calling

```
>>> results = request.results(synchronous=True)
```

How the property set will be estimated can easily be controlled by passing a PropertyEstimatorOptions object to the estimate commands.

The calculations layers which will be used to estimate the properties can be controlled for example like so:

```
>>> from propertyestimator.layers import ReweightingLayer, SimulationLayer
>>>
>>> options = PropertyEstimatorOptions(allowed_calculation_layers =␣
↪[ReweightingLayer,
>>>                                                                  ␣
↪SimulationLayer])
>>>
>>> request = property_estimator.request_estimate(data_set, parameters, options)
```

Options for how properties should be estimated can be set on a per property, and per layer basis. For example, the relative uncertainty that properties should estimated to within by the SimulationLayer can be set as:

```
>>> from propertyestimator.workflow import WorkflowOptions
>>>
>>> workflow_options = WorkflowOptions(WorkflowOptions.ConvergenceMode.
↪RelativeUncertainty,
>>>                                    relative_uncertainty_fraction=0.1)
>>> options.workflow_options = {
>>>     'Density': {'SimulationLayer': workflow_options},
>>>     'Dielectric': {'SimulationLayer': workflow_options}
>>> }
```

Or alternatively, as absolute uncertainty tolerance can be set as:

```
>>> density_options = WorkflowOptions(WorkflowOptions.ConvergenceMode.
↪AbsoluteUncertainty,
>>>                                   absolute_uncertainty=0.0002 * unit.gram /␣
↪unit.milliliter)
```

(continues on next page)

(continued from previous page)

```
>>> dielectric_options = WorkflowOptions(WorkflowOptions.ConvergenceMode.
↪AbsoluteUncertainty,
>>>                                       absolute_uncertainty=0.02 * unit.
↪dimensionless)
>>>
>>> options.workflow_options = {
>>>     'Density': {'SimulationLayer': density_options},
>>>     'Dielectric': {'SimulationLayer': dielectric_options}
>>> }
```

The gradients of the observables of interest with respect to a number of chosen parameters can be requested by passing a *parameter_gradient_keys* parameter. In the below example, gradients will be calculated with respect to both the bond length parameter for the [#6:1]-[#8:2] chemical environment, and the bond angle parameter for the [*:1]-[#8:2]-[*:3] chemical environment:

```
>>> from propertyestimator.properties import ParameterGradientKey
>>>
>>> parameter_gradient_keys = [
>>>     ParameterGradientKey('Bonds', '[#6:1]-[#8:2]', 'length')
>>>     ParameterGradientKey('Angles', '[*:1]-[#8:2]-[*:3]', 'angle')
>>> ]
>>>
>>> request = property_estimator.request_estimate(data_set, parameters, options,␣
↪parameter_gradient_keys)
>>>
```

__init__(*connection_options=<propertyestimator.client.ConnectionOptions object>*)
Constructs a new PropertyEstimatorClient object.

Parameters **connection_options** (ConnectionOptions) – The options used when connecting to the calculation server.

Methods

__init__([connection_options])	Constructs a new PropertyEstimatorClient object.
request_estimate(property_set, force_field)	Requests that a PropertyEstimatorServer attempt to estimate the provided property set using the supplied force field and estimator options.

Attributes

server_address
server_port

class Request(*request_id*, *connection_options*, *client=None*)
An object representation of a estimation request which has been sent to a *PropertyEstimatorServer* instance. This object can be used to query and retrieve the results of the request, or be stored to retrieve the request at some point in the future.

property id
The id of the submitted request.
Type str

property server_address

The address of the server that the request was sent to.

Type str

property server_port

The port that the server is listening on.

json()

Returns a JSON representation of the *Request* object.

Returns The JSON representation of the *Request* object.

Return type str

classmethod from_json(*json_string*)

Creates a new *Request* object from a JSON representation.

Parameters **json_string** (`str`) – The JSON representation of the *Request* object.

Returns The created *Request* object.

Return type str

results(*synchronous=False*, *polling_interval=5*)

Retrieve the results of an estimate request.

Parameters

- **synchronous** (`bool`) – If true, this method will block the main thread until the server either returns a result or an error.
- **polling_interval** (`int`) – If running synchronously, this is the time interval (seconds) between checking if the calculation has finished.

Returns

Returns either the results of the requested estimate, or any exceptions which were raised.

If the method is run synchronously then this method will block the main thread until all of the requested properties have been estimated, or an exception is returned.

Return type *PropertyEstimatorResult* or PropertyEstimatorException

request_estimate(*property_set*, *force_field*, *options=None*, *parameter_gradient_keys=None*)

Requests that a PropertyEstimatorServer attempt to estimate the provided property set using the supplied force field and estimator options.

Parameters

- **property_set** (`PhysicalPropertyDataSet`) – The set of properties to attempt to estimate.
- **force_field** (`ForceField`) – The OpenFF force field to use for the calculations.
- **options** (`PropertyEstimatorOptions, optional`) – A set of estimator options. If None, default options will be used.
- **parameter_gradient_keys** (`list of ParameterGradientKey, optional`) – A list of references to all of the parameters which all observables should be differentiated with respect to.

Returns An object which will provide access the the results of the request.

Return type *PropertyEstimatorClient.Request*

PropertyEstimatorOptions

class `propertyestimator.client.`**`PropertyEstimatorOptions`**(*allowed_calculation_layers=None*, *allow_protocol_merging=True*)

Represents the options options that can be passed to the property estimation server backend.

> **Warning:**
>
> - The *gradient_properties* property is not implemented yet, and is meant only as a placeholder for future api development.
> - This class is still heavily under development and is subject to rapid changes.

`allowed_calculation_layers`

A list of allowed calculation layers. The order of the layers in the list is the order that the calculator will attempt to execute the layers in.

Type list of str or list of class

`workflow_schemas`

A dictionary of the WorkflowSchema which will be used to calculate any properties. The dictionary key represents the type of property the schema will calculate. The dictionary will be automatically populated with defaults if no entries are added.

Type dict of str and dict of str and WorkflowSchema

`workflow_options`

The set of options which will be used when setting up the default estimation workflows, where the string key here is the property for which the options apply. As an example, the target (relative or absolute) uncertainty of each property may be set using these options.

If None, a set of defaults will be applied when the properties are sent to a server for estimation. The current set of defaults will ensure that properties are estimated with an uncertainty which is less than or equal to the experimental uncertainty of a property.

Type dict of str and dict of str and WorkflowOptions, optional

`allow_protocol_merging`

If true, allows individual identical steps in a property estimation workflow to be merged.

Type bool, default = True

`__init__`(*allowed_calculation_layers=None*, *allow_protocol_merging=True*)

Constructs a new PropertyEstimatorOptions object.

Parameters

- **`allowed_calculation_layers`** (*`list of str or list of class`*) – A list of allowed calculation layers. The order of the layers in the list is the order that the calculator will attempt to execute the layers in.

  If None, all registered calculation layers are set as allowed.

- **`allow_protocol_merging`** (*`bool, default = True`*) – If true, allows individual identical steps in a property estimation workflow to be merged.

Methods

`__init__`([allowed_calculation_layers, …])	Constructs a new PropertyEstimatorOptions object.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (`str or bytes`) – The typed json string.
- **encoding** (`str`) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

PropertyEstimatorSubmission

class propertyestimator.client.**PropertyEstimatorSubmission**(*properties=None*, *force_field=None*, *options=None*, *parameter_gradient_keys=None*)
Represents a set of properties to be estimated by the server backend, the parameters which will be used to estimate them, and options about how the properties will be estimated.

Warning: This class is still heavily under development and is subject to rapid changes.

properties
The list of physical properties to estimate.

Type list of PhysicalProperty

options
The options which control how the *properties* are estimated.

Type *PropertyEstimatorOptions*

force_field
The force field parameters used during the calculations.

Type openforcefield.typing.engines.smirnoff.ForceField

__init__(*properties=None*, *force_field=None*, *options=None*, *parameter_gradient_keys=None*)
Constructs a new PropertyEstimatorSubmission object.

Parameters

- **properties** (`list of PhysicalProperty`) – The list of physical properties to estimate.

- **options** (PropertyEstimatorOptions) – The options which control how the *properties* are estimated.
- **force_field** (*openforcefield.typing.engines.smirnoff.ForceField*) – The force field parameters used during the calculations.
- **parameter_gradient_keys** (*list of ParameterGradientKey*) – A list of references to all of the parameters which all observables should be differentiated with respect to.

Methods

`__init__`([properties, force_field, options, . . .])	Constructs a new PropertyEstimatorSubmission object.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

PropertyEstimatorResult

class propertyestimator.client.**PropertyEstimatorResult**(*result_id=''*)
Represents the results of attempting to estimate a set of physical properties using the property estimator server backend.

Warning: This class is still heavily under development and is subject to rapid changes.

id
The unique id assigned to this result set by the server.

Type str

queued_properties
A dictionary of the properties which have yet to be estimated by the server.

Type dict of str and PhysicalProperty

estimated_properties
A dictionary of the properties which were successfully estimated, where the dictionary key is the unique id of the property being estimated.

Type dict of str and PhysicalProperty

unsuccessful_properties
A dictionary of the properties which could not be estimated by the server.

Type dict of str and PhysicalProperty

exceptions
A list of the exceptions that were raised when unsuccessfully carrying out this estimation request.

Type list of PropertyEstimatorException

__init__(*result_id=''*)
Constructs a new PropertyEstimatorResult object.

Parameters **result_id** (*str*) – The unique id assigned to this result set by the server.

Methods

__init__([result_id])	Constructs a new PropertyEstimatorResult object.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

ConnectionOptions

class propertyestimator.client.**ConnectionOptions**(*server_address='localhost'*, *server_port=8000*)
The set of options to use when connecting to a *PropertyEstimatorServer*

server_address
The address of the server to connect to.

Type str

server_port
The port number that the server is listening on.

Type int

> **Warning:** This class is still heavily under development and is subject to rapid changes.

`__init__`(*server_address='localhost'*, *server_port=8000*)

Constructs a new ConnectionOptions object.

Parameters

- **`server_address`** (*`str`*) – The address of the server to connect to.
- **`server_port`** (*`int`*) – The port number that the server is listening on.

Methods

`__init__`([server_address, server_port])	Constructs a new ConnectionOptions object.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

Attributes

`server_address`
`server_port`

`json`()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **`string_contents`** (*`str or bytes`*) – The typed json string.
- **`encoding`** (*`str`*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

1.5.2 Server Side API

`PropertyEstimatorServer`	The object responsible for coordinating all properties estimations to to be ran using the property estimator, in addition to deciding at which fidelity a property will be calculated.

PropertyEstimatorServer

class propertyestimator.server.**PropertyEstimatorServer**(*calculation_backend*, *storage_backend*, *port=8000*, *working_directory='working-data'*)

The object responsible for coordinating all properties estimations to to be ran using the property estimator, in addition to deciding at which fidelity a property will be calculated.

It acts as a server, which receives submitted jobs from clients launched via the property estimator.

> **Warning:** This class is still heavily under development and is subject to rapid changes.

Notes

Methods to handle the TCP messages are based on the StackOverflow response from A. Jesse Jiryu Davis: https://stackoverflow.com/a/40257248

Examples

Setting up a general server instance using a dask LocalCluster backend:

```
>>> # Create the backend which will be responsible for distributing the␣
↪calculations
>>> from propertyestimator.backends import DaskLocalCluster, ComputeResources
>>> calculation_backend = DaskLocalCluster(1)
>>>
>>> # Calculate the backend which will be responsible for storing and retrieving
>>> # the data from previous calculations
>>> from propertyestimator.storage import LocalFileStorage
>>> storage_backend = LocalFileStorage()
>>>
>>> # Create the server to which all estimation requests will be submitted
>>> from propertyestimator.server import PropertyEstimatorServer
>>> property_server = PropertyEstimatorServer(calculation_backend, storage_
↪backend)
>>>
>>> # Instruct the server to listen for incoming requests
>>> property_server.start_listening_loop()
```

__init__(*calculation_backend*, *storage_backend*, *port=8000*, *working_directory='working-data'*)
Constructs a new PropertyEstimatorServer object.

Parameters

- **calculation_backend** (`PropertyEstimatorBackend`) – The backend to use for executing calculations.
- **storage_backend** (`PropertyEstimatorStorage`) – The backend to use for storing information from any calculations.
- **port** (*int*) – The port on which to listen for incoming client requests.
- **working_directory** (*str*) – The local directory in which to store all local, temporary calculation data.

Methods

`__init__`(calculation_backend, storage_backend)	Constructs a new PropertyEstimatorServer object.
`add_socket`(socket)	Singular version of *add_sockets*.
`add_sockets`(sockets)	Makes this server start accepting connections on the given sockets.
`bind`(port[, address, family, backlog, . . .])	Binds this server to the given port on the given address.
`handle_stream`(stream, address)	A routine to handle incoming requests from a property estimator TCP client.
`listen`(port[, address])	Starts accepting connections on the given port.
`start`([num_processes])	Starts this server in the *.IOLoop*.
`start_listening_loop`()	Starts the main (blocking) server IOLoop which will run until the user kills the process.
`stop`()	Stops the property calculation server and it's provided backend.

class ServerEstimationRequest(*estimation_id=''*, *queued_properties=None*, *options=None*, *force_field_id=None*, *parameter_gradient_keys=None*)

Represents a request for the server to estimate a set of properties. Such requests are expected to only estimate properties for a single system (e.g. fixed components in a fixed ratio)

json()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (`str or bytes`) – The typed json string.
- **encoding** (`str`) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

async handle_stream(*stream*, *address*)

A routine to handle incoming requests from a property estimator TCP client.

Notes

This method is based on the StackOverflow response from A. Jesse Jiryu Davis: https://stackoverflow.com/a/40257248

Parameters

- **stream** (`IOStream`) – An IO stream used to pass messages between the server and client.
- **address** (`str`) – The address from which the request came.

start_listening_loop()

Starts the main (blocking) server IOLoop which will run until the user kills the process.

stop()

Stops the property calculation server and it's provided backend.

add_socket (*socket*)
Singular version of *add_sockets*. Takes a single socket object.

add_sockets (*sockets*)
Makes this server start accepting connections on the given sockets.

The `sockets` parameter is a list of socket objects such as those returned by *~tornado.netutil.bind_sockets*. *add_sockets* is typically used in combination with that method and *tornado.process.fork_processes* to provide greater control over the initialization of a multi-process server.

bind (*port*, *address=None*, *family=<AddressFamily.AF_UNSPEC: 0>*, *backlog=128*, *reuse_port=False*)
Binds this server to the given port on the given address.

To start the server, call *start*. If you want to run this server in a single process, you can call *listen* as a shortcut to the sequence of *bind* and *start* calls.

Address may be either an IP address or hostname. If it's a hostname, the server will listen on all IP addresses associated with the name. Address may be an empty string or None to listen on all available interfaces. Family may be set to either *socket.AF_INET* or *socket.AF_INET6* to restrict to IPv4 or IPv6 addresses, otherwise both will be used if available.

The `backlog` argument has the same meaning as for *socket.listen <socket.socket.listen>*. The `reuse_port` argument has the same meaning as for *.bind_sockets*.

This method may be called multiple times prior to *start* to listen on multiple ports or interfaces.

Changed in version 4.4: Added the `reuse_port` argument.

listen (*port*, *address=''*)
Starts accepting connections on the given port.

This method may be called more than once to listen on multiple ports. *listen* takes effect immediately; it is not necessary to call *TCPServer.start* afterwards. It is, however, necessary to start the *.IOLoop*.

start (*num_processes=1*)
Starts this server in the *.IOLoop*.

By default, we run the server in this process and do not fork any additional child process.

If num_processes is `None` or <= 0, we detect the number of cores available on this machine and fork that number of child processes. If num_processes is given and > 1, we fork that specific number of subprocesses.

Since we use processes and not threads, there is no shared memory between any server code.

Note that multiple processes are not compatible with the autoreload module (or the `autoreload=True` option to *tornado.web.Application* which defaults to True when `debug=True`). When using multiple processes, no IOLoops can be created or referenced until after the call to `TCPServer.start(n)`.

1.5.3 Physical Property API

`PhysicalProperty`	Represents the value of any physical property and it's uncertainty.
`PropertyPhase`	An enum describing the phase a property was collected in.
`Source`	Container class for information about how a property was measured / calculated.

Continued on next page

Table 11 – continued from previous page

MeasurementSource	Contains any metadata about how a physical property was measured by experiment.
CalculationSource	Contains any metadata about how a physical property was calculated.

PhysicalProperty

class propertyestimator.properties.**PhysicalProperty**(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

Represents the value of any physical property and it's uncertainty.

It additionally stores the thermodynamic state at which the property was collected, the phase it was collected in, information about the composition of the observed system, and metadata about how the property was collected.

__init__(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

Constructs a new PhysicalProperty object.

Parameters

- **thermodynamic_state** (ThermodynamicState) – The thermodynamic state that the property was measured in.
- **phase** (PropertyPhase) – The phase that the property was measured in.
- **substance** (Substance) – The composition of the substance that was measured.
- **value** (*unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*unit.Quantity*) – The uncertainty in the measured value.
- **source** (Source) – The source of this property.

Methods

__init__([thermodynamic_state, phase, . . .])	Constructs a new PhysicalProperty object.
get_default_workflow_schema(calculation_layer)	Returns the default workflow schema to use for a specific calculation layer.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
set_value(value, uncertainty)	Set the value and uncertainty of this property.

Attributes

metadata	Additional metadata associated with this property, such as file paths to coordinate files or . . .
pressure	The pressure at which the property was collected.
temperature	The temperature at which the property was collected.

property temperature

The temperature at which the property was collected.

Type propertyestimator.unit.Quantity or None

property pressure

The pressure at which the property was collected.

Type propertyestimator.unit.Quantity or None

property metadata

Additional metadata associated with this property, such as file paths to coordinate files or . . .

All property metadata will be made accessible to property estimation workflows.

Type dict of str and Any

set_value(*value*, *uncertainty*)

Set the value and uncertainty of this property.

Parameters

- **value** (*propertyestimator.unit.Quantity*) – The value of the property.
- **uncertainty** (*propertyestimator.unit.Quantity*) – The uncertainty in the properties value.

static get_default_workflow_schema(*calculation_layer*, *options=None*)

Returns the default workflow schema to use for a specific calculation layer.

Parameters

- **calculation_layer** (*str*) – The calculation layer which will attempt to execute the workflow defined by this schema.
- **options** (WorkflowOptions) – The options to use when setting up the default workflows.

Returns The default workflow schema.

Return type *WorkflowSchema*

json()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

PropertyPhase

class propertyestimator.properties.**PropertyPhase**

An enum describing the phase a property was collected in.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Attributes

Gas
Liquid
Solid
Undefined

Source

class propertyestimator.properties.**Source**
Container class for information about how a property was measured / calculated.

Todo: Swap this out with a more general provenance class.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

MeasurementSource

class propertyestimator.properties.**MeasurementSource**(*doi=''*, *reference=''*)
Contains any metadata about how a physical property was measured by experiment.

This class contains either the DOI and/or the reference, but must contain at least one as the observable must have a source, even if it was measured in lab.

doi
The DOI for the source, preferred way to identify for source

Type str or None, default None

reference
The long form description of the source if no DOI is available, or more information is needed or wanted.

Type str

__init__(*doi=''*, *reference=''*)
Constructs a new MeasurementSource object.

Parameters

- **doi** (*str or None, default None*) – The DOI for the source, preferred way to identify for source
- **reference** (*str*) – The long form description of the source if no DOI is available, or more information is needed or wanted.

Methods

__init__([doi, reference])	Constructs a new MeasurementSource object.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

CalculationSource

class propertyestimator.properties.**CalculationSource**(*fidelity=None*, *provenance=None*)
Contains any metadata about how a physical property was calculated.

This includes at which fidelity the property was calculated at (e.g Direct simulation, reweighting, . . .) in addition to the parameters which were used as part of the calculations.

fidelity
The fidelity at which the property was calculated

Type str

provenance
A dictionary containing information about how the property was calculated.

Type dict of str and Any

__init__(*fidelity=None*, *provenance=None*)
Constructs a new CalculationSource object.

Parameters

- **fidelity** (`str`) – The fidelity at which the property was calculated
- **provenance** (`dict of str and Any`) – A dictionary containing information about how the property was calculated.

Methods

`__init__`([fidelity, provenance])	Constructs a new CalculationSource object.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (`str or bytes`) – The typed json string.
- **encoding** (`str`) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

Built-in Properties

`Density`	A class representation of a density property
`DielectricConstant`	A class representation of a dielectric property
`EnthalpyOfMixing`	A class representation of an enthalpy of mixing property
`EnthalpyOfVaporization`	A class representation of an enthalpy of vaporization property
`HostGuestBindingAffinity`	A class representation of a host-guest binding affinity property

Density

class propertyestimator.properties.**Density**(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

A class representation of a density property

__init__(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

Constructs a new PhysicalProperty object.

Parameters

- **thermodynamic_state** (ThermodynamicState) – The thermodynamic state that the property was measured in.
- **phase** (PropertyPhase) – The phase that the property was measured in.
- **substance** (Substance) – The composition of the substance that was measured.
- **value** (*unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*unit.Quantity*) – The uncertainty in the measured value.
- **source** (Source) – The source of this property.

Methods

__init__([thermodynamic_state, phase, . . .])	Constructs a new PhysicalProperty object.
get_default_reweighting_workflow_schema(options)	Returns the default workflow to use when estimating this property by reweighting existing data.
get_default_simulation_workflow_schema([options])	Returns the default workflow to use when estimating this property from direct simulations.
get_default_workflow_schema(calculation_layer)	Returns the default workflow schema to use for a specific calculation layer.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
set_value(value, uncertainty)	Set the value and uncertainty of this property.

Attributes

metadata	Additional metadata associated with this property, such as file paths to coordinate files or . . .
multi_component_property	
pressure	The pressure at which the property was collected.
required_data_class	
temperature	The temperature at which the property was collected.

static get_default_workflow_schema(*calculation_layer*, *options=None*)

Returns the default workflow schema to use for a specific calculation layer.

Parameters

- **calculation_layer** (*str*) – The calculation layer which will attempt to execute the workflow defined by this schema.

- **options** (`WorkflowOptions`) – The options to use when setting up the default workflows.

Returns The default workflow schema.

Return type *WorkflowSchema*

static get_default_simulation_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property from direct simulations.

Parameters options (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

static get_default_reweighting_workflow_schema(*options*)
Returns the default workflow to use when estimating this property by reweighting existing data.

Parameters options (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

property metadata
Additional metadata associated with this property, such as file paths to coordinate files or ...

All property metadata will be made accessible to property estimation workflows.

Type dict of str and Any

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*`str or bytes`*) – The typed json string.
- **encoding** (*`str`*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

property pressure
The pressure at which the property was collected.

Type propertyestimator.unit.Quantity or None

set_value(*value*, *uncertainty*)
Set the value and uncertainty of this property.

Parameters

- **value** (*`propertyestimator.unit.Quantity`*) – The value of the property.
- **uncertainty** (*`propertyestimator.unit.Quantity`*) – The uncertainty in the properties value.

property temperature

The temperature at which the property was collected.

Type propertyestimator.unit.Quantity or None

DielectricConstant

class propertyestimator.properties.**DielectricConstant**(*thermodynamic_state=None, phase=, substance=None, value=None, uncertainty=None, gradients=None, source=None*)

A class representation of a dielectric property

__init__(*thermodynamic_state=None, phase=, substance=None, value=None, uncertainty=None, gradients=None, source=None*)

Constructs a new PhysicalProperty object.

Parameters

- **thermodynamic_state** (ThermodynamicState) – The thermodynamic state that the property was measured in.
- **phase** (PropertyPhase) – The phase that the property was measured in.
- **substance** (Substance) – The composition of the substance that was measured.
- **value** (*unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*unit.Quantity*) – The uncertainty in the measured value.
- **source** (Source) – The source of this property.

Methods

__init__([thermodynamic_state, phase, …])	Constructs a new PhysicalProperty object.
get_default_reweighting_workflow_schema([options])	Returns the default workflow to use when estimating this property by reweighting existing data.
get_default_simulation_workflow_schema([options])	Returns the default workflow to use when estimating this property from direct simulations.
get_default_workflow_schema(calculation_layer)	Returns the default workflow schema to use for a specific calculation layer.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
set_value(value, uncertainty)	Set the value and uncertainty of this property.

Attributes

metadata	Additional metadata associated with this property, such as file paths to coordinate files or …
multi_component_property	
pressure	The pressure at which the property was collected.
required_data_class	

Continued on next page

Table 22 – continued from previous page

temperature	The temperature at which the property was collected.

static get_default_workflow_schema(*calculation_layer*, *options=None*)
Returns the default workflow schema to use for a specific calculation layer.

Parameters

- **calculation_layer** (*str*) – The calculation layer which will attempt to execute the workflow defined by this schema.
- **options** (WorkflowOptions) – The options to use when setting up the default workflows.

Returns The default workflow schema.

Return type *WorkflowSchema*

static get_default_simulation_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property from direct simulations.

Parameters **options** (WorkflowOptions) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

static get_default_reweighting_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property by reweighting existing data.

Parameters **options** (WorkflowOptions) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

property metadata
Additional metadata associated with this property, such as file paths to coordinate files or . . .

All property metadata will be made accessible to property estimation workflows.

Type dict of str and Any

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

property pressure
The pressure at which the property was collected.

Type propertyestimator.unit.Quantity or None

set_value(*value*, *uncertainty*)

Set the value and uncertainty of this property.

Parameters

- **value** (*propertyestimator.unit.Quantity*) – The value of the property.
- **uncertainty** (*propertyestimator.unit.Quantity*) – The uncertainty in the properties value.

property temperature

The temperature at which the property was collected.

Type propertyestimator.unit.Quantity or None

EnthalpyOfMixing

class propertyestimator.properties.**EnthalpyOfMixing**(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

A class representation of an enthalpy of mixing property

__init__(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

Constructs a new PhysicalProperty object.

Parameters

- **thermodynamic_state** (ThermodynamicState) – The thermodynamic state that the property was measured in.
- **phase** (PropertyPhase) – The phase that the property was measured in.
- **substance** (Substance) – The composition of the substance that was measured.
- **value** (*unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*unit.Quantity*) – The uncertainty in the measured value.
- **source** (Source) – The source of this property.

Methods

__init__([thermodynamic_state, phase, …])	Constructs a new PhysicalProperty object.
get_default_reweighting_workflow_schema([…])	Returns the default workflow to use when estimating this property by reweighting existing data.
get_default_simulation_workflow_schema([options])	Returns the default workflow to use when estimating this property from direct simulations.
get_default_workflow_schema(calculation_layer)	Returns the default workflow schema to use for a specific calculation layer.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
set_value(value, uncertainty)	Set the value and uncertainty of this property.

Attributes

`metadata`	Additional metadata associated with this property, such as file paths to coordinate files or . . .
`multi_component_property`	
`pressure`	The pressure at which the property was collected.
`required_data_class`	
`temperature`	The temperature at which the property was collected.

EnthalpyWorkflow
alias of `EnthalpySchema`

static get_default_workflow_schema(*calculation_layer*, *options=None*)
Returns the default workflow schema to use for a specific calculation layer.

Parameters

- **calculation_layer** (*str*) – The calculation layer which will attempt to execute the workflow defined by this schema.
- **options** (`WorkflowOptions`) – The options to use when setting up the default workflows.

Returns The default workflow schema.

Return type *WorkflowSchema*

static get_default_simulation_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property from direct simulations.

Parameters **options** (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

static get_default_reweighting_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property by reweighting existing data.

Parameters **options** (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

property metadata
Additional metadata associated with this property, such as file paths to coordinate files or . . .

All property metadata will be made accessible to property estimation workflows.

Type dict of str and Any

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **`string_contents`** (*`str or bytes`*) – The typed json string.
- **`encoding`** (*`str`*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

`property pressure`
The pressure at which the property was collected.

Type propertyestimator.unit.Quantity or None

`set_value`(*value*, *uncertainty*)
Set the value and uncertainty of this property.

Parameters

- **`value`** (*`propertyestimator.unit.Quantity`*) – The value of the property.
- **`uncertainty`** (*`propertyestimator.unit.Quantity`*) – The uncertainty in the properties value.

`property temperature`
The temperature at which the property was collected.

Type propertyestimator.unit.Quantity or None

EnthalpyOfVaporization

`class` `propertyestimator.properties.`**`EnthalpyOfVaporization`**(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

A class representation of an enthalpy of vaporization property

`__init__`(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)
Constructs a new PhysicalProperty object.

Parameters

- **`thermodynamic_state`** (`ThermodynamicState`) – The thermodynamic state that the property was measured in.
- **`phase`** (`PropertyPhase`) – The phase that the property was measured in.
- **`substance`** (`Substance`) – The composition of the substance that was measured.
- **`value`** (*`unit.Quantity`*) – The value of the measured physical property.
- **`uncertainty`** (*`unit.Quantity`*) – The uncertainty in the measured value.
- **`source`** (`Source`) – The source of this property.

Methods

`__init__`([thermodynamic_state, phase, . . .])	Constructs a new PhysicalProperty object.
`get_default_reweighting_workflow_schema`(options)	Returns the default workflow to use when estimating this property by reweighting existing data.
`get_default_simulation_workflow_schema`([options])	Returns the default workflow to use when estimating this property from direct simulations.
`get_default_workflow_schema`(calculation_layer)	Returns the default workflow schema to use for a specific calculation layer.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
`set_value`(value, uncertainty)	Set the value and uncertainty of this property.

Attributes

`metadata`	Additional metadata associated with this property, such as file paths to coordinate files or . . .
`multi_component_property`	Returns whether this property is dependant on properties of the full mixed substance, or whether it is also dependant on the properties of the individual components also.
`pressure`	The pressure at which the property was collected.
`required_data_class`	
`temperature`	The temperature at which the property was collected.

property multi_component_property
Returns whether this property is dependant on properties of the full mixed substance, or whether it is also dependant on the properties of the individual components also.

static get_default_workflow_schema(*calculation_layer*, *options=None*)
Returns the default workflow schema to use for a specific calculation layer.

Parameters

- **calculation_layer** (`str`) – The calculation layer which will attempt to execute the workflow defined by this schema.
- **options** (`WorkflowOptions`) – The options to use when setting up the default workflows.

Returns The default workflow schema.

Return type *WorkflowSchema*

static get_default_simulation_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property from direct simulations.

Parameters **options** (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

static get_default_reweighting_workflow_schema(*options*)
Returns the default workflow to use when estimating this property by reweighting existing data.

Parameters **options** (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

json()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

property metadata

Additional metadata associated with this property, such as file paths to coordinate files or . . .

All property metadata will be made accessible to property estimation workflows.

Type dict of str and Any

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

property pressure

The pressure at which the property was collected.

Type propertyestimator.unit.Quantity or None

set_value(*value*, *uncertainty*)

Set the value and uncertainty of this property.

Parameters

- **value** (*propertyestimator.unit.Quantity*) – The value of the property.
- **uncertainty** (*propertyestimator.unit.Quantity*) – The uncertainty in the properties value.

property temperature

The temperature at which the property was collected.

Type propertyestimator.unit.Quantity or None

HostGuestBindingAffinity

class propertyestimator.properties.**HostGuestBindingAffinity**(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

A class representation of a host-guest binding affinity property

__init__(*thermodynamic_state=None*, *phase=*, *substance=None*, *value=None*, *uncertainty=None*, *gradients=None*, *source=None*)

Constructs a new PhysicalProperty object.

Parameters

- **thermodynamic_state** (`ThermodynamicState`) – The thermodynamic state that the property was measured in.
- **phase** (`PropertyPhase`) – The phase that the property was measured in.
- **substance** (`Substance`) – The composition of the substance that was measured.
- **value** (*unit.Quantity*) – The value of the measured physical property.
- **uncertainty** (*unit.Quantity*) – The uncertainty in the measured value.
- **source** (`Source`) – The source of this property.

Methods

`__init__`([thermodynamic_state, phase, . . .])	Constructs a new PhysicalProperty object.
`get_default_simulation_workflow_schema`([options])	Returns the default workflow to use when estimating this property from direct simulations.
`get_default_workflow_schema`(calculation_layer)	Returns the default workflow schema to use for a specific calculation layer.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
`set_value`(value, uncertainty)	Set the value and uncertainty of this property.

Attributes

`metadata`	Additional metadata associated with this property, such as file paths to coordinate files or . . .
`multi_component_property`	Returns whether this property is dependant on properties of the full mixed substance, or whether it is also dependant on the properties of the individual components also.
`pressure`	The pressure at which the property was collected.
`temperature`	The temperature at which the property was collected.

property multi_component_property
Returns whether this property is dependant on properties of the full mixed substance, or whether it is also dependant on the properties of the individual components also.

static get_default_workflow_schema (*calculation_layer*, *options=None*)
Returns the default workflow schema to use for a specific calculation layer.

Parameters

- **calculation_layer** (*str*) – The calculation layer which will attempt to execute the workflow defined by this schema.
- **options** (`WorkflowOptions`) – The options to use when setting up the default workflows.

Returns The default workflow schema.

Return type *WorkflowSchema*

static get_default_simulation_workflow_schema(*options=None*)
Returns the default workflow to use when estimating this property from direct simulations.

Parameters **options** (`WorkflowOptions`) – The default options to use when setting up the estimation workflow.

Returns The schema to follow when estimating this property.

Return type *WorkflowSchema*

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

property metadata
Additional metadata associated with this property, such as file paths to coordinate files or . . .

All property metadata will be made accessible to property estimation workflows.

Type dict of str and Any

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*`str or bytes`*) – The typed json string.
- **encoding** (*`str`*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

property pressure
The pressure at which the property was collected.

Type propertyestimator.unit.Quantity or None

set_value(*value*, *uncertainty*)
Set the value and uncertainty of this property.

Parameters

- **value** (*`propertyestimator.unit.Quantity`*) – The value of the property.
- **uncertainty** (*`propertyestimator.unit.Quantity`*) – The uncertainty in the properties value.

property temperature
The temperature at which the property was collected.

Type propertyestimator.unit.Quantity or None

Substance Definition

`Substance`	Defines the components, their amounts, and their roles in a system.

Substance

class propertyestimator.substances.**Substance**

Defines the components, their amounts, and their roles in a system.

Examples

A neat liquid containing only a single component:

```
>>> liquid = Substance()
>>> liquid.add_component(Substance.Component(smiles='O'), Substance.
↪MoleFraction(1.0))
```

A binary mixture containing two components, where the mole fractions are explicitly stated:

```
>>> binary_mixture = Substance()
>>> binary_mixture.add_component(Substance.Component(smiles='O'), Substance.
↪MoleFraction(0.2))
>>> binary_mixture.add_component(Substance.Component(smiles='CO'), Substance.
↪MoleFraction(0.8))
```

The infinite dilution of one molecule within a bulk solvent or mixture may also be specified by defining the exact number of copies of that molecule, rather than a mole fraction:

```
>>> benzene = Substance.Component(smiles='C1=CC=CC=C1', role=Substance.
↪ComponentRole.Solute)
>>> water = Substance.Component(smiles='O', role=Substance.ComponentRole.Solvent)
>>>
>>> infinite_dilution = Substance()
>>> infinite_dilution.add_component(component=benzene, amount=Substance.
↪ExactAmount(1)) # Infinite dilution.
>>> infinite_dilution.add_component(component=water, amount=Substance.
↪MoleFraction(1.0))
```

In this example we explicitly flag benzene as being the solute and the water component the solvent. This enables workflow's to easily identify key molecules of interest, such as the molecule which should be 'grown' into solution during solvation free energy calculations.

__init__()

Constructs a new Substance object.

Methods

__init__()	Constructs a new Substance object.
add_component(component, amount)	Add a component to the Substance.
calculate_aqueous_ionic_mole_fraction(...)	Determines what mole fraction of ions is needed to yield
get_amount(component)	Returns the amount of the component in this substance.
get_molecules_per_component(maximum_molecules)	Returns the number of molecules for each component in this substance, given a maximum total number of molecules.
json()	Creates a JSON representation of this class.

Continued on next page

Table 30 – continued from previous page

parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

Attributes

components	A list of all of the components in this substance.
identifier	A unique str representation of this substance, which encodes all components and their amounts in the substance.
number_of_components	The number of different components in this substance.

class ComponentRole

An enum which describes the role of a component in the system, such as whether the component is a solvent, a solute, a receptor etc.

These roles are mainly only used by specific protocols to identify the correct species in a system, such as when doing docking or performing solvation free energy calculations.

class Component(*smiles=None*, *label=None*, *role=None*)

Defines a single component in a system, as well as properties such as it's relative proportion in the system.

property identifier

A unique identifier for this component, which is either a smiles descriptor or the supplied label.

Type str

property label

A string label which describes this compound, for example, CB8.

Type str

property smiles

The smiles pattern which describes this component, which may be None for complex (e.g protein) molecules.

Type str

property role

The role of this component in the system, such as a ligand or a receptor.

Type *ComponentRole*

json()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

class Amount(*value=None*)

An abstract representation of the amount of a given component in a substance.

property value
The value of this amount.

property identifier
A string identifier for this amount.

abstract to_number_of_molecules(*total_substance_molecules*, *tolerance=None*)
Converts this amount to an exact number of molecules

Parameters

- **total_substance_molecules** (*int*) – The total number of molecules in the whole substance. This amount will contribute to a portion of this total number.
- **tolerance** (*float*) – The tolerance with which this amount should be in. As an example, when converting a mole fraction into a number of molecules, the total number of molecules may not be sufficently large enough to reproduce this amount.

Returns The number of molecules which this amount represents, given the *total_substance_molecules*.

Return type int

class MoleFraction(*value=1.0*)
Represents the amount of a component in a substance as a mole fraction.

property value
The value of this amount.

Type float

property identifier
A string identifier for this amount.

to_number_of_molecules(*total_substance_molecules*, *tolerance=None*)
Converts this amount to an exact number of molecules

Parameters

- **total_substance_molecules** (*int*) – The total number of molecules in the whole substance. This amount will contribute to a portion of this total number.
- **tolerance** (*float*) – The tolerance with which this amount should be in. As an example, when converting a mole fraction into a number of molecules, the total number of molecules may not be sufficently large enough to reproduce this amount.

Returns The number of molecules which this amount represents, given the *total_substance_molecules*.

Return type int

class ExactAmount(*value=1*)
Represents the amount of a component in a substance as an exact number of molecules.

The expectation is that this amount should be used for components which are infinitely dilute (such as ligands in binding calculations), and hence do not contribute to the total mole fraction of a substance

property value
The value of this amount.

Type int

property identifier
A string identifier for this amount.

to_number_of_molecules(*total_substance_molecules*, *tolerance=None*)
Converts this amount to an exact number of molecules

Parameters

- **total_substance_molecules** (*int*) – The total number of molecules in the whole substance. This amount will contribute to a portion of this total number.

- **tolerance** (*float*) – The tolerance with which this amount should be in. As an example, when converting a mole fraction into a number of molecules, the total number of molecules may not be sufficently large enough to reproduce this amount.

Returns The number of molecules which this amount represents, given the *total_substance_molecules*.

Return type int

property identifier
A unique str representation of this substance, which encodes all components and their amounts in the substance.

Type str

property components
A list of all of the components in this substance.

Type list of Substance.Component

property number_of_components
The number of different components in this substance.

Type int

add_component (*component*, *amount*)
Add a component to the Substance. If the component is already present in the substance, then the mole fraction will be added to the current mole fraction of that component.

Parameters

- **component** (`Substance.Component`) – The component to add to the system.
- **amount** (`Substance.Amount`) – The amount of this component in the substance.

get_amount (*component*)
Returns the amount of the component in this substance.

Parameters **component** (*str or* `Substance.Component`) – The component (or it's identifier) to retrieve the mole fraction of.

Returns The amount of the component in this substance.

Return type *Substance.Amount*

get_molecules_per_component (*maximum_molecules*)
Returns the number of molecules for each component in this substance, given a maximum total number of molecules.

Parameters **maximum_molecules** (*int*) – The maximum number of molecules.

Returns A dictionary of molecule counts per component, where each key is a component identifier.

Return type dict of str and int

static calculate_aqueous_ionic_mole_fraction (*ionic_strength*)

Determines what mole fraction of ions is needed to yield an aqueous system of a given ionic strength.

Parameters **ionic_strength** (*unit.Quantity*) – The ionic string in units of molar.

Returns The mole fraction of ions.

Return type float

json()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

State Definition

ThermodynamicState	Data specifying a physical thermodynamic state obeying Boltzmann statistics.

ThermodynamicState

class propertyestimator.thermodynamics.**ThermodynamicState**(*temperature=None*, *pressure=None*)

Data specifying a physical thermodynamic state obeying Boltzmann statistics.

temperature

The external temperature

Type propertyestimator.unit.Quantity with units compatible with kelvin

pressure

The external pressure

Type propertyestimator.unit.Quantity with units compatible with atmospheres

Examples

Specify an NPT state at 298 K and 1 atm pressure.

```
>>> state = ThermodynamicState(temperature=298.0*unit.kelvin, pressure=1.0*unit.
↪atmospheres)
```

Note that the pressure is only relevant for periodic systems.

__init__(*temperature=None*, *pressure=None*)

Constructs a new ThermodynamicState object.

Parameters

- **temperature** (*propertyestimator.unit.Quantity with units compatible with kelvin*) – The external temperature
- **pressure** (*propertyestimator.unit.Quantity with units compatible with atmospheres*) – The external pressure

Methods

`__init__`([temperature, pressure])	Constructs a new ThermodynamicState object.
`json`()	Creates a JSON representation of this class.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

Attributes

`beta`	Returns one divided by the temperature multiplied by the molar gas constant
`inverse_beta`	Returns the temperature multiplied by the molar gas constant

`property inverse_beta`
Returns the temperature multiplied by the molar gas constant

`property beta`
Returns one divided by the temperature multiplied by the molar gas constant

`json`()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

`classmethod parse_json`(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **`string_contents`** (`str or bytes`) – The typed json string.
- **`encoding`** (`str`) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

Metadata

`PropertyPhase`	An enum describing the phase a property was collected in.
`Source`	Container class for information about how a property was measured / calculated.
`MeasurementSource`	Contains any metadata about how a physical property was measured by experiment.
`CalculationSource`	Contains any metadata about how a physical property was calculated.
`ParameterGradientKey`	
`ParameterGradient`	

ParameterGradientKey

class propertyestimator.properties.**ParameterGradientKey**(*tag=None*, *smirks=None*, *attribute=None*)

__init__(*tag=None*, *smirks=None*, *attribute=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([tag, smirks, attribute])	Initialize self.

Attributes

attribute
smirks
tag

ParameterGradient

class propertyestimator.properties.**ParameterGradient**(*key=None*, *value=None*)

__init__(*key=None*, *value=None*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([key, value])	Initialize self.

Attributes

key
value

1.5.4 Data Set API

PhysicalPropertyDataSet	An object for storing and curating data sets of both physical property measurements and estimated.

PhysicalPropertyDataSet

class propertyestimator.datasets.**PhysicalPropertyDataSet**
An object for storing and curating data sets of both physical property measurements and estimated. This class defines a number of convenience functions for filtering out unwanted properties, and for generating general statistics (such as the number of properties per substance) about the set.

__init__()

Constructs a new PhysicalPropertyDataSet object.

Methods

`__init__`()	Constructs a new PhysicalPropertyDataSet object.
`filter_by_components`(number_of_components)	Filter the data set based on a minimum and maximum temperature.
`filter_by_elements`(*allowed_elements)	Filters out those properties which were estimated for
`filter_by_function`(filter_function)	Filter the data set using a given filter function.
`filter_by_phases`(phases)	Filter the data set based on the phase of the property (e.g liquid).
`filter_by_pressure`(min_pressure, max_pressure)	Filter the data set based on a minimum and maximum pressure.
`filter_by_property_types`(*property_type)	Filter the data set based on the type of property (e.g Density).
`filter_by_smiles`(*allowed_smiles)	Filters out those properties which were estimated for
`filter_by_temperature`(min_temperature, ...)	Filter the data set based on a minimum and maximum temperature.
`json`()	Creates a JSON representation of this class.
`merge`(data_set)	Merge another data set into the current one.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

Attributes

`number_of_properties`	The number of properties in the data set.
`properties`	A list of all of the properties within this set, partitioned by substance identifier.
`sources`	The list of sources from which the properties were gathered

`property properties`

A list of all of the properties within this set, partitioned by substance identifier.

TODO: Add a link to Substance.identifier when have access to sphinx docs. TODO: Investigate why PhysicalProperty is not cross-linking.

See also:

`Substance.identifier`

Type dict of str and list of PhysicalProperty

`property sources`

The list of sources from which the properties were gathered

Type list of Source

`property number_of_properties`

The number of properties in the data set.

Type int

merge (*data_set*)
Merge another data set into the current one.

Parameters **data_set** (PhysicalPropertyDataSet) – The secondary data set to merge into this one.

filter_by_function (*filter_function*)
Filter the data set using a given filter function.

Parameters **filter_function** (*lambda*) – The filter function.

filter_by_property_types (**property_type*)
Filter the data set based on the type of property (e.g Density).

Parameters **property_type** (*PropertyType or str*) – The type of property which should be retained.

Examples

Filter the dataset to only contain densities and static dielectric constants

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> # Filter the dataset to only include densities and dielectric constants.
>>> from propertyestimator.properties import Density, DielectricConstant
>>> data_set.filter_by_property_types(Density, DielectricConstant)
```

or

```
>>> data_set.filter_by_property_types('Density', 'DielectricConstant')
```

filter_by_phases (*phases*)
Filter the data set based on the phase of the property (e.g liquid).

Parameters **phases** (PropertyPhase) – The phase of property which should be retained.

Examples

Filter the dataset to only include liquid properties.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> from propertyestimator.properties import PropertyPhase
>>> data_set.filter_by_temperature(PropertyPhase.Liquid)
```

filter_by_temperature (*min_temperature*, *max_temperature*)
Filter the data set based on a minimum and maximum temperature.

Parameters

- **min_temperature** (*unit.Quantity*) – The minimum temperature.
- **max_temperature** (*unit.Quantity*) – The maximum temperature.

Examples

Filter the dataset to only include properties measured between 130-260 K.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> from propertyestimator import unit
>>> data_set.filter_by_temperature(min_temperature=130*unit.kelvin, max_
↪temperature=260*unit.kelvin)
```

`filter_by_pressure`(*min_pressure*, *max_pressure*)
Filter the data set based on a minimum and maximum pressure.

Parameters

- **`min_pressure`** (*`unit.Quantity`*) – The minimum pressure.
- **`max_pressure`** (*`unit.Quantity`*) – The maximum pressure.

Examples

Filter the dataset to only include properties measured between 70-150 kPa.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> from propertyestimator import unit
>>> data_set.filter_by_temperature(min_pressure=70*unit.kilopascal, max_
↪temperature=150*unit.kilopascal)
```

`filter_by_components`(*number_of_components*)
Filter the data set based on a minimum and maximum temperature.

Parameters **`number_of_components`** (*`int`*) – The allowed number of components in the mixture.

Examples

Filter the dataset to only include pure substance properties.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> data_set.filter_by_components(number_of_components=1)
```

`filter_by_elements`(**allowed_elements*)

Filters out those properties which were estimated for compounds which contain elements outside of those defined in *allowed_elements*.

Parameters **`allowed_elements`** (*`str`*) – The symbols (e.g. C, H, Cl) of the elements to retain.

filter_by_smiles (**allowed_smiles*)

Filters out those properties which were estimated for compounds which do not appear in the allowed *smiles* list.

Parameters **allowed_smiles** (*str*) – The smiles identifiers of the compounds to keep after filtering.

json ()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json (*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

NIST ThermoML Archive

ThermoMLDataSet	A dataset of physical property measurements created from a ThermoML dataset.
register_thermoml_property	A decorator which registers information on how to parse a given ThermoML property

ThermoMLDataSet

class propertyestimator.datasets.**ThermoMLDataSet**

A dataset of physical property measurements created from a ThermoML dataset.

Examples

For example, we can use the DOI *10.1016/j.jct.2005.03.012* as a key for retrieving the dataset from the ThermoML Archive:

```
>>> dataset = ThermoMLDataSet.from_doi('10.1016/j.jct.2005.03.012')
```

You can also specify multiple ThermoML Archive keys to create a dataset from multiple ThermoML files:

```
>>> thermoml_keys = ['10.1021/acs.jced.5b00365', '10.1021/acs.jced.5b00474']
>>> dataset = ThermoMLDataSet.from_doi(*thermoml_keys)
```

__init__ ()

Constructs a new ThermoMLDataSet object.

Methods

`__init__`()	Constructs a new ThermoMLDataSet object.
`filter_by_components`(number_of_components)	Filter the data set based on a minimum and maximum temperature.
`filter_by_elements`(*allowed_elements)	Filters out those properties which were estimated for
`filter_by_function`(filter_function)	Filter the data set using a given filter function.
`filter_by_phases`(phases)	Filter the data set based on the phase of the property (e.g liquid).
`filter_by_pressure`(min_pressure, max_pressure)	Filter the data set based on a minimum and maximum pressure.
`filter_by_property_types`(*property_type)	Filter the data set based on the type of property (e.g Density).
`filter_by_smiles`(*allowed_smiles)	Filters out those properties which were estimated for
`filter_by_temperature`(min_temperature, . . .)	Filter the data set based on a minimum and maximum temperature.
`from_doi`(*doi_list)	Load a ThermoML data set from a list of DOIs
`from_file`(*file_list)	Load a ThermoML data set from a list of files
`from_url`(*url_list)	Load a ThermoML data set from a list of URLs
`from_xml`(xml, source)	Load a ThermoML data set from an xml object.
`json`()	Creates a JSON representation of this class.
`merge`(data_set)	Merge another data set into the current one.
`parse_json`(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

Attributes

`number_of_properties`	The number of properties in the data set.
`properties`	A list of all of the properties within this set, partitioned by substance identifier.
`sources`	The list of sources from which the properties were gathered

classmethod from_doi(**doi_list*)
Load a ThermoML data set from a list of DOIs

Parameters **doi_list** (`str`) – The list of DOIs to pull data from

Returns The loaded data set.

Return type *ThermoMLDataSet*

classmethod from_url(**url_list*)
Load a ThermoML data set from a list of URLs

Parameters **url_list** (`str`) – The list of URLs to pull data from

Returns The loaded data set.

Return type *ThermoMLDataSet*

classmethod from_file(**file_list*)
Load a ThermoML data set from a list of files

Parameters **file_list** (`str`) – The list of files to pull data from

Returns The loaded data set.

Return type *ThermoMLDataSet*

filter_by_components(*number_of_components*)
Filter the data set based on a minimum and maximum temperature.

Parameters **number_of_components** (*int*) – The allowed number of components in the mixture.

Examples

Filter the dataset to only include pure substance properties.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> data_set.filter_by_components(number_of_components=1)
```

filter_by_elements(**allowed_elements*)

Filters out those properties which were estimated for compounds which contain elements outside of those defined in *allowed_elements*.

Parameters **allowed_elements** (*str*) – The symbols (e.g. C, H, Cl) of the elements to retain.

filter_by_function(*filter_function*)
Filter the data set using a given filter function.

Parameters **filter_function** (*lambda*) – The filter function.

filter_by_phases(*phases*)
Filter the data set based on the phase of the property (e.g liquid).

Parameters **phases** (*PropertyPhase*) – The phase of property which should be retained.

Examples

Filter the dataset to only include liquid properties.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> from propertyestimator.properties import PropertyPhase
>>> data_set.filter_by_temperature(PropertyPhase.Liquid)
```

filter_by_pressure(*min_pressure*, *max_pressure*)
Filter the data set based on a minimum and maximum pressure.

Parameters

- **min_pressure** (*unit.Quantity*) – The minimum pressure.
- **max_pressure** (*unit.Quantity*) – The maximum pressure.

Examples

Filter the dataset to only include properties measured between 70-150 kPa.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> from propertyestimator import unit
>>> data_set.filter_by_temperature(min_pressure=70*unit.kilopascal, max_
↪temperature=150*unit.kilopascal)
```

filter_by_property_types (**property_type*)

Filter the data set based on the type of property (e.g Density).

Parameters **property_type** (*PropertyType or str*) – The type of property which should be retained.

Examples

Filter the dataset to only contain densities and static dielectric constants

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> # Filter the dataset to only include densities and dielectric constants.
>>> from propertyestimator.properties import Density, DielectricConstant
>>> data_set.filter_by_property_types(Density, DielectricConstant)
```

or

```
>>> data_set.filter_by_property_types('Density', 'DielectricConstant')
```

filter_by_smiles (**allowed_smiles*)

Filters out those properties which were estimated for compounds which do not appear in the allowed *smiles* list.

Parameters **allowed_smiles** (*str*) – The smiles identifiers of the compounds to keep after filtering.

filter_by_temperature (*min_temperature, max_temperature*)

Filter the data set based on a minimum and maximum temperature.

Parameters

- **min_temperature** (*unit.Quantity*) – The minimum temperature.
- **max_temperature** (*unit.Quantity*) – The maximum temperature.

Examples

Filter the dataset to only include properties measured between 130-260 K.

```
>>> # Load in the data set of properties which will be used for comparisons
>>> from propertyestimator.datasets import ThermoMLDataSet
>>> data_set = ThermoMLDataSet.from_doi('10.1016/j.jct.2016.10.001')
>>>
>>> from propertyestimator import unit
>>> data_set.filter_by_temperature(min_temperature=130*unit.kelvin, max_
↪temperature=260*unit.kelvin)
```

classmethod from_xml(*xml*, *source*)

Load a ThermoML data set from an xml object.

Parameters

- **xml** (*str*) – The xml string to parse.
- **source** (Source) – The source of the xml object.

Returns The loaded ThermoML data set.

Return type *ThermoMLDataSet*

json()

Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

merge(*data_set*)

Merge another data set into the current one.

Parameters **data_set** (PhysicalPropertyDataSet) – The secondary data set to merge into this one.

property number_of_properties

The number of properties in the data set.

Type int

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

property properties

A list of all of the properties within this set, partitioned by substance identifier.

TODO: Add a link to Substance.identifier when have access to sphinx docs. TODO: Investigate why PhysicalProperty is not cross-linking.

See also:

`Substance.identifier`

Type dict of str and list of PhysicalProperty

property sources

The list of sources from which the properties were gathered

Type list of Source

propertyestimator.datasets.register_thermoml_property

`propertyestimator.datasets.`**`register_thermoml_property`**(*thermoml_string*, *supported_phases*)

A decorator which registers information on how to parse a given ThermoML property

For now this only takes input of a thermoML string, but in future will give greater control over exactly how ThermoML XML gets parsed to an actual property.

Parameters

- **`thermoml_string`** (*`str`*) – The ThermoML string identifier (ePropName) for this property.
- **`supported_phases`** (*`PropertyPhase:`*) – An enum which encodes all of the phases for which this property supports being estimated in.

1.5.5 Calculation Layers API

`PropertyCalculationLayer`	An abstract representation of a calculation layer whose goal is to estimate a set of physical properties using a single approach, such as a layer which employs direct simulations to estimate properties, or one which reweights cached simulation data to the same end.
`register_calculation_layer`	A decorator which registers a class as being a calculation layer which may be used in property calculations.

PropertyCalculationLayer

class `propertyestimator.layers.`**`PropertyCalculationLayer`**

An abstract representation of a calculation layer whose goal is to estimate a set of physical properties using a single approach, such as a layer which employs direct simulations to estimate properties, or one which reweights cached simulation data to the same end.

Notes

Calculation layers must inherit from this class, and must override the *schedule_calculation* method.

See also:

`TODO` Link to a general page outlining what calculation layers are and how they are used.

`__init__`()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
schedule_calculation(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.

static schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *data_model*, *callback*, *synchronous=False*)

Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (PropertyEstimatorBackend) – The backend to the submit the calculations to.
- **storage_backend** (PropertyEstimatorStorage) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The local directory in which to store all local, temporary calculation data from this layer.
- **data_model** (PropertyEstimatorServer.ServerEstimationRequest) – The data model encoding the proposed calculation.
- **callback** (*function*) – The function to call when the backend returns the results (or an error).
- **synchronous** (*bool*) – If true, this function will block until the calculation has completed. This is mainly intended for debugging purposes.

propertyestimator.layers.register_calculation_layer

propertyestimator.layers.**register_calculation_layer**()

A decorator which registers a class as being a calculation layer which may be used in property calculations.

See also:

TODO() add documentation for plugin support

Built-in Calculation Layers

ReweightingLayer	A calculation layer which aims to calculate physical properties by reweighting the results of previous calculations.
SimulationLayer	A calculation layer which aims to calculate physical properties directly from molecular simulation.

ReweightingLayer

class propertyestimator.layers.**ReweightingLayer**

A calculation layer which aims to calculate physical properties by reweighting the results of previous calculations.

Warning: This class is still heavily under development and is subject to rapid changes.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
schedule_calculation(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.

static schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *data_model*, *callback*, *synchronous=False*)
Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (PropertyEstimatorBackend) – The backend to the submit the calculations to.
- **storage_backend** (PropertyEstimatorStorage) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The local directory in which to store all local, temporary calculation data from this layer.
- **data_model** (PropertyEstimatorServer.ServerEstimationRequest) – The data model encoding the proposed calculation.
- **callback** (*function*) – The function to call when the backend returns the results (or an error).
- **synchronous** (*bool*) – If true, this function will block until the calculation has completed. This is mainly intended for debugging purposes.

SimulationLayer

class propertyestimator.layers.**SimulationLayer**
A calculation layer which aims to calculate physical properties directly from molecular simulation.

Warning: This class is experimental and should not be used in a production environment.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
schedule_calculation(calculation_backend, ...)	Submit the proposed calculation to the backend of choice.

static schedule_calculation(*calculation_backend*, *storage_backend*, *layer_directory*, *data_model*, *callback*, *synchronous=False*)
Submit the proposed calculation to the backend of choice.

Parameters

- **calculation_backend** (PropertyEstimatorBackend) – The backend to the submit the calculations to.
- **storage_backend** (PropertyEstimatorStorage) – The backend used to store / retrieve data from previous calculations.
- **layer_directory** (*str*) – The local directory in which to store all local, temporary calculation data from this layer.
- **data_model** (PropertyEstimatorServer.ServerEstimationRequest) – The data model encoding the proposed calculation.
- **callback** (*function*) – The function to call when the backend returns the results (or an error).
- **synchronous** (*bool*) – If true, this function will block until the calculation has completed. This is mainly intended for debugging purposes.

1.5.6 Calculation Backends API

PropertyEstimatorBackend	An abstract base representation of a property estimator backend.
ComputeResources	An object which stores how many of each type of computational resource (threads or gpu's) is available to a calculation worker.
QueueWorkerResources	An extended resource object with properties specific to calculations which will run on queue based resources, such as LSF, PBS or SLURM.

PropertyEstimatorBackend

class propertyestimator.backends.**PropertyEstimatorBackend**(*number_of_workers=1, resources_per_worker=<propertyestimator.backer object>*)

An abstract base representation of a property estimator backend. A backend is responsible for coordinating, distributing and running calculations on the available hardware. This may range from a single machine to a multinode cluster, but *not* accross multiple cluster or physical locations.

Notes

All estimator backend classes must inherit from this class, and must implement the *start*, *stop*, and *submit_task* method.

__init__(*number_of_workers=1, resources_per_worker=<propertyestimator.backends.backends.ComputeResources object>*)

Constructs a new PropertyEstimatorBackend object.

Parameters

- **number_of_workers** (*int*) – The number of works to run the calculations on. One worker can perform a single task (e.g run a simulation) at once.
- **resources_per_worker** (ComputeResources) – The number of resources to request per worker.

Methods

`__init__`([number_of_workers, . . .])	Constructs a new PropertyEstimatorBackend object.
`start`()	Start the calculation backend.
`stop`()	Stop the calculation backend.
`submit_task`(function, *args, **kwargs)	Submit a task to the compute resources managed by this backend.

start()
Start the calculation backend.

stop()
Stop the calculation backend.

submit_task(*function*, **args*, ***kwargs*)
Submit a task to the compute resources managed by this backend.

Parameters **function** (*function*) – The function to run.

Returns Returns a future object which will eventually point to the results of the submitted task.

Return type Future

ComputeResources

class propertyestimator.backends.**ComputeResources**(*number_of_threads=1*, *number_of_gpus=0*, *preferred_gpu_toolkit=None*)
An object which stores how many of each type of computational resource (threads or gpu's) is available to a calculation worker.

TODO: The use of the terminology here is questionable, and is used interchangable with process which may lead to some confusion.

__init__(*number_of_threads=1*, *number_of_gpus=0*, *preferred_gpu_toolkit=None*)
Constructs a new ComputeResources object.

Parameters

- **number_of_threads** (*int*) – The number of threads available to a calculation worker.
- **number_of_gpus** (*int*) – The number of GPUs available to a calculation worker.
- **preferred_gpu_toolkit** (`ComputeResources.GPUToolkit`, *optional*) – The preferred toolkit to use when running on GPUs.

Methods

`__init__`([number_of_threads, . . .])	Constructs a new ComputeResources object.

Attributes

`gpu_device_indices`	The indices of the GPUs to run on.

Continued on next page

Table 54 – continued from previous page

number_of_gpus	The number of GPUs available to a calculation worker.
number_of_threads	The number of threads available to a calculation worker.
preferred_gpu_toolkit	The preferred toolkit to use when running on GPUs.

class GPUToolkit
An enumeration of the different GPU toolkits to make available to different calculations.

property number_of_threads
The number of threads available to a calculation worker.

Type int

property number_of_gpus
The number of GPUs available to a calculation worker.

Type int

property preferred_gpu_toolkit
The preferred toolkit to use when running on GPUs.

Type *ComputeResources.GPUToolkit*

property gpu_device_indices
The indices of the GPUs to run on. This is purely an internal implementation detail and should not be relied upon externally.

Type str

QueueWorkerResources

class propertyestimator.backends.**QueueWorkerResources**(*number_of_threads=1*, *number_of_gpus=0*, *preferred_gpu_toolkit=None*, *per_thread_memory_limit=<Quantity(1, 'gigabyte')>*, *wallclock_time_limit='01:00'*)
An extended resource object with properties specific to calculations which will run on queue based resources, such as LSF, PBS or SLURM.

__init__(*number_of_threads=1*, *number_of_gpus=0*, *preferred_gpu_toolkit=None*, *per_thread_memory_limit=<Quantity(1, 'gigabyte')>*, *wallclock_time_limit='01:00'*)
Constructs a new ComputeResources object.

Notes

Both the requested *number_of_threads* and the *number_of_gpus* must be less than or equal to the number of threads (/cpus/cores) and GPUs available to each compute node in the cluster respectively, such that a single worker is able to be accommodated by a single compute node.

Parameters

- **per_thread_memory_limit** (*simtk.Quantity*) – The maximum amount of memory available to each thread.

- **wallclock_time_limit** (*str*) – The maximum amount of wall clock time that a worker can run for. This should be a string of the form *HH:MM* where HH is the number of hours and MM the number of minutes

Methods

__init__([number_of_threads, …])	Constructs a new ComputeResources object.

Attributes

gpu_device_indices	The indices of the GPUs to run on.
number_of_gpus	The number of GPUs available to a calculation worker.
number_of_threads	The number of threads available to a calculation worker.
per_thread_memory_limit	The maximum amount of memory available to each thread, such that the total memory limit will be *per_cpu_memory_limit * number_of_threads*.
preferred_gpu_toolkit	The preferred toolkit to use when running on GPUs.
wallclock_time_limit	The maximum amount of wall clock time that a worker can run for.

property per_thread_memory_limit
The maximum amount of memory available to each thread, such that the total memory limit will be *per_cpu_memory_limit * number_of_threads*.

Type simtk.Quantity

property wallclock_time_limit
The maximum amount of wall clock time that a worker can run for. This should be a string of the form *HH:MM* where HH is the number of hours and MM the number of minutes

Type str

class GPUToolkit
An enumeration of the different GPU toolkits to make available to different calculations.

property gpu_device_indices
The indices of the GPUs to run on. This is purely an internal implementation detail and should not be relied upon externally.

Type str

property number_of_gpus
The number of GPUs available to a calculation worker.

Type int

property number_of_threads
The number of threads available to a calculation worker.

Type int

property preferred_gpu_toolkit
The preferred toolkit to use when running on GPUs.

Type *ComputeResources.GPUToolkit*

Dask Backends

`BaseDaskBackend`	A base *dask* backend class, which implements functionality which is common to all other *dask* based backends.
`DaskLocalCluster`	A property estimator backend which uses a *dask LocalCluster* object to run calculations on a single machine.
`DaskLSFBackend`	A property estimator backend which uses a *dask_jobqueue LSFCluster* object to run calculations within an existing LSF queue.

BaseDaskBackend

class propertyestimator.backends.**BaseDaskBackend**(*number_of_workers=1, resources_per_worker=<propertyestimator.backends.backends.*
object>)

A base *dask* backend class, which implements functionality which is common to all other *dask* based backends.

__init__(*number_of_workers=1, resources_per_worker=<propertyestimator.backends.backends.ComputeResources object>*)

Constructs a new BaseDaskBackend object.

Methods

`__init__`([number_of_workers, ...])	Constructs a new BaseDaskBackend object.
`start`()	Start the calculation backend.
`stop`()	Stop the calculation backend.
`submit_task`(function, *args, **kwargs)	Submit a task to the compute resources managed by this backend.

start()

Start the calculation backend.

stop()

Stop the calculation backend.

submit_task(*function, *args, **kwargs*)

Submit a task to the compute resources managed by this backend.

Parameters **function** (`function`) – The function to run.

Returns Returns a future object which will eventually point to the results of the submitted task.

Return type Future

DaskLocalCluster

class propertyestimator.backends.**DaskLocalCluster**(*number_of_workers=1, resources_per_worker=<propertyestimator.backends.backend*
object>)

A property estimator backend which uses a *dask LocalCluster* object to run calculations on a single machine.

See also:

`dask.LocalCluster`

__init__(*number_of_workers=1*, *resources_per_worker=<propertyestimator.backends.backends.ComputeResources object>*)

Constructs a new DaskLocalCluster

Methods

__init__([number_of_workers, . . .])	Constructs a new DaskLocalCluster
start()	Start the calculation backend.
stop()	Stop the calculation backend.
submit_task(function, *args, **kwargs)	Submit a task to the compute resources managed by this backend.

start()

Start the calculation backend.

submit_task(*function*, **args*, ***kwargs*)

Submit a task to the compute resources managed by this backend.

Parameters **function** (*function*) – The function to run.

Returns Returns a future object which will eventually point to the results of the submitted task.

Return type Future

stop()

Stop the calculation backend.

DaskLSFBackend

class propertyestimator.backends.**DaskLSFBackend**(*minimum_number_of_workers=1*, *maximum_number_of_workers=1*, *resources_per_worker=<propertyestimator.backends.backends.Q object>*, *queue_name='default'*, *setup_script_commands=None*, *extra_script_options=None*, *adaptive_interval='10000ms'*, *disable_nanny_process=False*)

A property estimator backend which uses a *dask_jobqueue LSFCluster* object to run calculations within an existing LSF queue.

See also:

`dask_jobqueue.LSFCluster`

__init__(*minimum_number_of_workers=1*, *maximum_number_of_workers=1*, *resources_per_worker=<propertyestimator.backends.backends.QueueWorkerResources object>*, *queue_name='default'*, *setup_script_commands=None*, *extra_script_options=None*, *adaptive_interval='10000ms'*, *disable_nanny_process=False*)

Constructs a new DaskLSFBackend object

Parameters

- **minimum_number_of_workers** (*int*) – The minimum number of workers to request from the queue system.
- **maximum_number_of_workers** (*int*) – The maximum number of workers to request from the queue system.

- **resources_per_worker** (QueueWorkerResources) – The resources to request per worker.
- **queue_name** (*str*) – The name of the queue which the workers will be requested from.
- **setup_script_commands** (*list of str*) – A list of bash script commands to call within the queue submission script before the call to launch the dask worker.

  This may include activating a python environment, or loading an environment module
- **extra_script_options** (*list of str*) – A list of extra job specific options to include in the queue submission script. These will get added to the script header in the form

  #BSUB <extra_script_options[x]>
- **adaptive_interval** (*str*) – The interval between attempting to either scale up or down the cluster, of of the from 'XXXms'.
- **disable_nanny_process** (*bool*) – If true, dask workers will be started in *–no-nanny* mode. This is required if using multiprocessing code within submitted tasks.

  This has not been fully tested yet and my lead to stability issues with the workers.

Examples

To create an LSF queueing compute backend which will attempt to spin up workers which have access to a single GPU.

```
>>> # Create a resource object which will request a worker with
>>> # one gpu which will stay alive for five hours.
>>> from propertyestimator.backends import QueueWorkerResources
>>>
>>> resources = QueueWorkerResources(number_of_threads=1,
>>>                                  number_of_gpus=1,
>>>                                  preferred_gpu_
↪toolkit=QueueWorkerResources.GPUToolkit.CUDA,
>>>                                  wallclock_time_limit='05:00')
>>>
>>> # Define the set of commands which will set up the correct environment
>>> # for each of the workers.
>>> setup_script_commands = [
>>>     'module load cuda/9.2',
>>> ]
>>>
>>> # Define extra options to only run on certain node groups
>>> extra_script_options = [
>>>     '-m "ls-gpu lt-gpu"'
>>> ]
>>>
>>>
>>> # Create the backend which will adaptively try to spin up between one and
>>> # ten workers with the requested resources depending on the calculation␣
↪load.
>>> from propertyestimator.backends import DaskLSFBackend
>>>
>>> lsf_backend = DaskLSFBackend(minimum_number_of_workers=1,
>>>                              maximum_number_of_workers=10,
>>>                              resources_per_worker=resources,
```

(continues on next page)

(continued from previous page)

```
>>>                                queue_name='gpuqueue',
>>>                                setup_script_commands=setup_script_commands,
>>>                                extra_script_options=extra_script_options)
```

Methods

`__init__`([minimum_number_of_workers, . . .])	Constructs a new DaskLSFBackend object
`start`()	Start the calculation backend.
`stop`()	Stop the calculation backend.
`submit_task`(function, *args, **kwargs)	Submit a task to the compute resources managed by this backend.

start()
Start the calculation backend.

submit_task(*function*, **args*, ***kwargs*)
Submit a task to the compute resources managed by this backend.

Parameters **function** (`function`) – The function to run.

Returns Returns a future object which will eventually point to the results of the submitted task.

Return type Future

stop()
Stop the calculation backend.

1.5.7 Storage Backends API

`PropertyEstimatorStorage`	An abstract base representation of how the property estimator will interact with and store simulation data.

PropertyEstimatorStorage

class propertyestimator.storage.**PropertyEstimatorStorage**
An abstract base representation of how the property estimator will interact with and store simulation data.

Notes

Any inheriting class must provide an implementation for the *store_object*, *retrieve_object* and *has_object* methods

__init__()
Constructs a new PropertyEstimatorStorage object.

Methods

`__init__`()	Constructs a new PropertyEstimatorStorage object.

Continued on next page

Table 62 – continued from previous page

has_force_field(force_field)	Checks whether the force field has been previously stored in the force field directory.
retrieve_force_field(unique_id)	Retrieves a force field from storage, if it exists.
retrieve_simulation_data(substance[, . . .])	Retrieves any data that has been stored for a given substance.
retrieve_simulation_data_by_id(unique_id)	Attempts to retrieve a storage piece of simulation data from it's unique id.
store_force_field(force_field)	Store the force field in the cached force field directory.
store_simulation_data(data_object, . . .)	Store the simulation data.

has_force_field(*force_field*)
Checks whether the force field has been previously stored in the force field directory.

Parameters **force_field** (*ForceField*) – The force field to check for.

Returns None if the force field has not been cached, otherwise the unique id of the cached force field.

Return type str, optional

retrieve_force_field(*unique_id*)
Retrieves a force field from storage, if it exists.

Parameters **unique_id** (*str*) – The unique id of the force field to retrieve

Returns The force field if present in the storage system with the given key, otherwise None.

Return type openforcefield.typing.engines.smirnoff.ForceField, optional

store_force_field(*force_field*)
Store the force field in the cached force field directory.

Parameters **force_field** (*openforcefield.typing.engines.smirnoff.ForceField*) – The force field to store.

Returns The unique id of the stored force field.

Return type str

retrieve_simulation_data_by_id(*unique_id*)
Attempts to retrieve a storage piece of simulation data from it's unique id.

Parameters **unique_id** (*str*) – The unique id assigned to the data.

Returns

- *BaseStoredData* – The stored data object.
- *str* – The path to the data's corresponding directory.

retrieve_simulation_data(*substance*, *include_component_data=True*, *data_class=<class 'propertyestimator.storage.dataclasses.StoredSimulationData'>*)
Retrieves any data that has been stored for a given substance.

Parameters

- **substance** (*Substance*) – The substance to check for.
- **include_component_data** (*bool*) – If the substance if a mixture where has multiple components and *include_component_data* is True, data will be returned for both the mixed system, and for the individual components, otherwise only data for the mixed system will be returned.

- **data_class** (*subclass of BaseStoredData*) – The type of data to retrieve.

Returns A dictionary of the stored data objects and their corresponding directory paths partitioned by substance id.

Return type dict of str and tuple of BaseStoredData and str

store_simulation_data(*data_object*, *data_directory*)
Store the simulation data.

Notes

If the storage system already contains equivalent information (i.e data stored for the same substance, thermodynamic state and parameter set) then the data will be merged according to the data objects *merge* method.

Parameters

- **data_object** (BaseStoredData) – The data object being stored.
- **data_directory** (*str*) – The directory which stores files associated with the data object such as trajectory files.

Returns The unique id of the stored data.

Return type str

Built-in Storage Backends

LocalFileStorage	A storage backend which stores files in directories on the local disk.

LocalFileStorage

class propertyestimator.storage.**LocalFileStorage**(*root_directory='stored_data'*)
A storage backend which stores files in directories on the local disk.

__init__(*root_directory='stored_data'*)
Constructs a new PropertyEstimatorStorage object.

Methods

__init__([root_directory])	Constructs a new PropertyEstimatorStorage object.
has_force_field(force_field)	Checks whether the force field has been previously stored in the force field directory.
retrieve_force_field(unique_id)	Retrieves a force field from storage, if it exists.
retrieve_simulation_data(substance[, . . .])	Retrieves any data that has been stored for a given substance.
retrieve_simulation_data_by_id(unique_id)	Attempts to retrieve a storage piece of simulation data from it's unique id.
store_force_field(force_field)	Store the force field in the cached force field directory.
store_simulation_data(data_object, . . .)	Store the simulation data.

Attributes

root_directory	Returns the directory in which all stored objects are located.

property root_directory
Returns the directory in which all stored objects are located.

Type str

store_simulation_data(*data_object*, *data_directory*)
Store the simulation data.

Notes

If the storage system already contains equivalent information (i.e data stored for the same substance, thermodynamic state and parameter set) then the data will be merged according to the data objects *merge* method.

Parameters

- **data_object** (BaseStoredData) – The data object being stored.
- **data_directory** (*str*) – The directory which stores files associated with the data object such as trajectory files.

Returns The unique id of the stored data.

Return type str

retrieve_simulation_data_by_id(*unique_id*)
Attempts to retrieve a storage piece of simulation data from it's unique id.

Parameters **unique_id** (*str*) – The unique id assigned to the data.

Returns

- *BaseStoredData* – The stored data object.
- *str* – The path to the data's corresponding directory.

retrieve_simulation_data(*substance*, *include_component_data=True*, *data_class=<class 'propertyestimator.storage.dataclasses.StoredSimulationData'>*)
Retrieves any data that has been stored for a given substance.

Parameters

- **substance** (Substance) – The substance to check for.
- **include_component_data** (*bool*) – If the substance if a mixture where has multiple components and *include_component_data* is True, data will be returned for both the mixed system, and for the individual components, otherwise only data for the mixed system will be returned.
- **data_class** (*subclass of BaseStoredData*) – The type of data to retrieve.

Returns A dictionary of the stored data objects and their corresponding directory paths partitioned by substance id.

Return type dict of str and tuple of BaseStoredData and str

has_force_field(*force_field*)

Checks whether the force field has been previously stored in the force field directory.

Parameters **force_field** (*ForceField*) – The force field to check for.

Returns None if the force field has not been cached, otherwise the unique id of the cached force field.

Return type str, optional

retrieve_force_field(*unique_id*)

Retrieves a force field from storage, if it exists.

Parameters **unique_id** (*str*) – The unique id of the force field to retrieve

Returns The force field if present in the storage system with the given key, otherwise None.

Return type openforcefield.typing.engines.smirnoff.ForceField, optional

store_force_field(*force_field*)

Store the force field in the cached force field directory.

Parameters **force_field** (*openforcefield.typing.engines.smirnoff.ForceField*) – The force field to store.

Returns The unique id of the stored force field.

Return type str

Data Classes

BaseStoredData	A base representation of cached data to be stored by a storage backend.
StoredSimulationData	A representation of data which has been cached from a single previous simulation.
StoredDataCollection	A collection of stored *StoredSimulationData* objects, all generated at the same state and using the same force field parameters.

BaseStoredData

class propertyestimator.storage.dataclasses.**BaseStoredData**

A base representation of cached data to be stored by a storage backend.

The expectation is that stored data will exist in storage as two parts:

1) A JSON serialized representation of this class (or a subclass), which contains lightweight information such as the state and composition of the system. Any larger pieces of data, such as coordinates or trajectories, should be referenced by this class as a filename.

2) A directory like structure (either directly a directory, or some NetCDF like compressed archive) of ancillary files which do not easily lend themselves to be serialized within a JSON object, whose files are referenced by name by the data object.

substance

A description of the composition of the stored system.

Type *Substance*

thermodynamic_state

The state at which the data was collected.

Type *ThermodynamicState*

source_calculation_id
The server id of the calculation which yielded this data.

Type str

provenance
A dictionary containing the provenance information about how this data was generated.

Type dict of str and Any

force_field_id
The server assigned unique id of the force field parameters used to generate the data.

Type str

__init__()
Constructs a new BaseStoredData object

Methods

__init__()	Constructs a new BaseStoredData object
can_merge(other_data)	Checks whether this piece of data stores the same amount of compatible information (or more) than another piece of stored data, and hence whether the two can be merged together.
merge(stored_data_1, stored_data_2)	Collapse two pieces of compatible stored data into one.

can_merge(*other_data*)
Checks whether this piece of data stores the same amount of compatible information (or more) than another piece of stored data, and hence whether the two can be merged together.

Parameters **other_data** (`BaseStoredData`) – The other stored data to compare against.

Returns Returns *True* if this piece of data stores the same amount of information or more than another piece of data, or false if it contains less or incompatible data.

Return type bool

classmethod merge(*stored_data_1*, *stored_data_2*)
Collapse two pieces of compatible stored data into one.

Parameters

- **stored_data_1** (`BaseStoredData`) – The first piece of stored data.
- **stored_data_2** (`BaseStoredData`) – The second piece of stored data.

Returns The merged stored data.

Return type *BaseStoredData*

StoredSimulationData

class propertyestimator.storage.dataclasses.**StoredSimulationData**
A representation of data which has been cached from a single previous simulation.

Notes

The ancillary directory which stores larger information such as trajectories should be of the form:

```
|--- data_object.json
|--- data_directory
     |--- coordinate_file_name.pdb
     |--- trajectory_file_name.dcd
     |--- statistics_file_name.csv
```

coordinate_file_name
The name of a coordinate file which encodes the toplogy information of the system.

Type str

trajectory_file_name
The name of a .dcd trajectory file containing configurations generated by the simulation.

Type str

statistics_file_name
The name of a *StatisticsArray* csv file, containing statistics generated by the simulation.

Type str

statistical_inefficiency
The statistical inefficiency of the collected data.

Type float

total_number_of_molecules
The total number of molecules in the system.

Type int

__init__()
Constructs a new StoredSimulationData object

Methods

__init__()	Constructs a new StoredSimulationData object
can_merge(other_data)	Checks whether this piece of data stores the same amount of compatible information (or more) than another piece of stored data, and hence whether the two can be merged together.
merge(stored_data_1, stored_data_2)	Collapse two pieces of compatible stored data into one, by only retaining the data with the longest autocorrelation time.

classmethod merge(*stored_data_1*, *stored_data_2*)
Collapse two pieces of compatible stored data into one, by only retaining the data with the longest autocorrelation time.

Parameters

- **stored_data_1** (StoredSimulationData) – The first piece of stored data.
- **stored_data_2** (StoredSimulationData) – The second piece of stored data.

Returns The merged stored data.

Return type *StoredSimulationData*

can_merge(*other_data*)

Checks whether this piece of data stores the same amount of compatible information (or more) than another piece of stored data, and hence whether the two can be merged together.

Parameters **other_data** (BaseStoredData) – The other stored data to compare against.

Returns Returns *True* if this piece of data stores the same amount of information or more than another piece of data, or false if it contains less or incompatible data.

Return type bool

StoredDataCollection

class propertyestimator.storage.dataclasses.**StoredDataCollection**

A collection of stored *StoredSimulationData* objects, all generated at the same state and using the same force field parameters.

The ancillary directory which stores larger information such as trajectories should be of the form:

```
|--- data_object.json
|--- data_directory
    |--- data_key_1
         |--- coordinate_file_name.pdb
         |--- trajectory_file_name.dcd
         |--- statistics_file_name.csv
    |--- data_key_2
         |--- coordinate_file_name.pdb
         |--- trajectory_file_name.dcd
         |--- statistics_file_name.csv
    |--- data_key_3
         |--- coordinate_file_name.pdb
         |--- trajectory_file_name.dcd
         |--- statistics_file_name.csv
```

data

A dictionary of stored simulation data objects which have been given a unique key.

Type dict of str and StoredSimulationData

__init__()

Constructs a new StoredDataCollection object

Methods

__init__()	Constructs a new StoredDataCollection object
can_merge(other_data_collection)	**param other_data_collection** The other stored data to compare against.
merge(stored_data_1, stored_data_2)	Collapse two pieces of compatible stored data into one, by only retaining the data with the longest auto-correlation time.

can_merge(*other_data_collection*)

Parameters **other_data_collection** (`StoredDataCollection`) – The other stored data to compare against.

classmethod merge(*stored_data_1*, *stored_data_2*)

Collapse two pieces of compatible stored data into one, by only retaining the data with the longest autocorrelation time.

Parameters

- **stored_data_1** (`StoredDataCollection`) – The first piece of stored data.
- **stored_data_2** (`StoredDataCollection`) – The second piece of stored data.

Returns The merged stored data.

Return type *StoredDataCollection*

1.5.8 Workflow API

`Workflow`	Encapsulates and prepares a workflow which is able to estimate a physical property.
`WorkflowGraph`	A hierarchical structure for storing and submitting the workflows which will estimate a set of physical properties..
`WorkflowOptions`	A set of convenience options used when creating estimation workflows.
`IWorkflowProperty`	Defines the interface a property must implement to be estimable by a workflow.

Workflow

class propertyestimator.workflow.**Workflow**(*physical_property*, *global_metadata*, *workflow_uuid=None*)

Encapsulates and prepares a workflow which is able to estimate a physical property.

__init__(*physical_property*, *global_metadata*, *workflow_uuid=None*)

Constructs a new Workflow object.

Parameters

- **physical_property** (`PhysicalProperty`) – The property which this workflow aims to calculate.
- **global_metadata** (*`dict of str and Any`*) – A dictionary of the global metadata available to each of the workflow properties.
- **workflow_uuid** (*`str, optional`*) – An optional uuid to assign to this workflow. If none is provided, one will be chosen at random.

Methods

`__init__`(physical_property, global_metadata)	Constructs a new Workflow object.
`generate_default_metadata`(physical_property, …)	Generates a default global metadata dictionary.

Continued on next page

Table 71 – continued from previous page

replace_protocol(old_protocol, new_protocol)	Replaces an existing protocol with a new one, while updating all input and local references to point to the new protocol.

Attributes

schema

replace_protocol (*old_protocol*, *new_protocol*)

Replaces an existing protocol with a new one, while updating all input and local references to point to the new protocol.

The main use of this method is when merging multiple protocols into one.

Parameters

- **old_protocol** (protocols.BaseProtocol *or* *str*) – The protocol (or its id) to replace.
- **new_protocol** (protocols.BaseProtocol *or* *str*) – The new protocol (or its id) to use.

static generate_default_metadata (*physical_property*, *force_field_path*, *parameter_gradient_keys=None*, *workflow_options=None*)

Generates a default global metadata dictionary.

Parameters

- **physical_property** (PhysicalProperty) – The physical property whose arguments are available in the global scope.
- **force_field_path** (*str*) – The path to the force field parameters to use in the workflow.
- **parameter_gradient_keys** (*list of ParameterGradientKey*) – A list of references to all of the parameters which all observables should be differentiated with respect to.
- **workflow_options** (WorkflowOptions, *optional*) – The options provided when an estimate request was submitted.

Returns

The metadata dictionary, with the following keys / types:

- **thermodynamic_state: *ThermodynamicState* - The state (T,p) at which the** property is being computed
- substance: *Substance* - The composition of the system of interest.
- **components: list of *Substance* - The components present in the system for** which the property is being estimated.
- **target_uncertainty: propertyestimator.unit.Quantity - The target uncertainty with which** properties should be estimated.
- **per_component_uncertainty: propertyestimator.unit.Quantity - The target uncertainty divided** by the sqrt of the number of components in the system + 1
- **force_field_path: str - A path to the force field parameters with which the** property should be evaluated with.

- **parameter_gradient_keys: list of ParameterGradientKey - A list of references to all of the** parameters which all observables should be differentiated with respect to.

Return type dict of str, Any

WorkflowGraph

class propertyestimator.workflow.**WorkflowGraph**(*root_directory=''*)

A hierarchical structure for storing and submitting the workflows which will estimate a set of physical properties..

__init__(*root_directory=''*)

Constructs a new WorkflowGraph

Parameters **root_directory** (*str*) – The root directory in which to store all outputs from this graph.

Methods

__init__([root_directory])	Constructs a new WorkflowGraph
add_workflow(workflow)	Insert a workflow into the workflow graph.
submit(backend[, include_uncertainty_check])	Submits the protocol graph to the backend of choice.

add_workflow(*workflow*)

Insert a workflow into the workflow graph.

Parameters **workflow** (Workflow) – The workflow to insert.

submit(*backend*, *include_uncertainty_check=True*)

Submits the protocol graph to the backend of choice.

Parameters

- **backend** (PropertyEstimatorBackend) – The backend to execute the graph on.
- **include_uncertainty_check** (*bool*) – If true, the uncertainty of each estimated property will be checked to ensure it is below the target threshold set in the workflow metadata. If an uncertainty is not included in the workflow metadata, then this parameter will be ignored.

Returns The futures of the submitted protocols.

Return type list of Future

WorkflowOptions

class propertyestimator.workflow.**WorkflowOptions**(*convergence_mode=<ConvergenceMode.RelativeUncertainty 'RelativeUncertainty'>*, *relative_uncertainty_fraction=1.0*, *absolute_uncertainty=None*)

A set of convenience options used when creating estimation workflows.

__init__(*convergence_mode=<ConvergenceMode.RelativeUncertainty: 'RelativeUncertainty'>*, *relative_uncertainty_fraction=1.0*, *absolute_uncertainty=None*)

Constructs a new WorkflowOptions object.

Parameters

- **convergence_mode** (`WorkflowOptions.ConvergenceMode`) – The mode which governs how workflows should decide when they have reached convergence.
- **relative_uncertainty_fraction** (*float, optional*) – If the convergence mode is set to *RelativeUncertainty*, then workflows will by default run simulations until the estimated uncertainty is less than

  relative_uncertainty_fraction * property_to_estimate.uncertainty
- **absolute_uncertainty** (*propertyestimator.unit.Quantity, optional*) – If the convergence mode is set to *AbsoluteUncertainty*, then workflows will by default run simulations until the estimated uncertainty is less than the *absolute_uncertainty*

Methods

__init__([convergence_mode, . . .])	Constructs a new WorkflowOptions object.

class ConvergenceMode

The available options for deciding when a workflow has converged. For now, these options include running until the computed uncertainty of a property is within a relative fraction of the measured uncertainty (*ConvergenceMode.RelativeUncertainty*) or is less than some absolute value (*ConvergenceMode.AbsoluteUncertainty*).

IWorkflowProperty

class propertyestimator.workflow.**IWorkflowProperty**

Defines the interface a property must implement to be estimable by a workflow.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
get_default_workflow_schema(...)	

Schema

WorkflowSchema	Outlines the workflow which should be followed when calculating a certain property.
ProtocolSchema	A json serializable representation of a workflow protocol.
ProtocolGroupSchema	A json serializable representation of a workflow protocol group.
ProtocolReplicator	A protocol replicator contains the information necessary to replicate parts of a property estimation workflow.
WorkflowOutputToStore	An object which describes which data should be cached after a workflow has finished executing, and from which completed protocols should the data be collected from.

Continued on next page

Table 76 – continued from previous page

WorkflowSimulationDataToStore	An object which describes which data should be cached after a workflow has finished executing, and from which completed protocols should the data be collected from.
WorkflowDataCollectionToStore	An object which describes which data should be cached after a workflow has finished executing, and from which completed protocols should the data be collected from.

WorkflowSchema

class propertyestimator.workflow.schemas.**WorkflowSchema**(*property_type=None*)
Outlines the workflow which should be followed when calculating a certain property.

__init__(*property_type=None*)
Constructs a new WorkflowSchema object.

Parameters **property_type** (*str*) – The type of property which this workflow aims to estimate.

Methods

__init__([property_type])	Constructs a new WorkflowSchema object.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
validate_interfaces()	Validates the flow of the data between protocols, ensuring that inputs and outputs correctly match up.

validate_interfaces()
Validates the flow of the data between protocols, ensuring that inputs and outputs correctly match up.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

ProtocolSchema

class propertyestimator.workflow.schemas.**ProtocolSchema**
A json serializable representation of a workflow protocol.

__init__()
Constructs a new ProtocolSchema object.

Methods

__init__()	Constructs a new ProtocolSchema object.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

ProtocolGroupSchema

class propertyestimator.workflow.schemas.**ProtocolGroupSchema**
A json serializable representation of a workflow protocol group.

__init__()
Constructs a new ProtocolGroupSchema object.

Methods

__init__()	Constructs a new ProtocolGroupSchema object.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)
Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

ProtocolReplicator

class propertyestimator.workflow.schemas.**ProtocolReplicator**(*replicator_id=”*)

A protocol replicator contains the information necessary to replicate parts of a property estimation workflow.

Any protocol whose id includes *$(replicator.id)* (where *replicator.id* is the id of a replicator) will be cloned for each value present in *template_values*. Protocols that are being replicated will also have any ReplicatorValue inputs replaced with the actual value taken from *template_values*.

When the protocol is replicated, the *$(replicator.id)* placeholder in the protocol id will be replaced an integer which corresponds to the index of a value in the *template_values* array.

Any protocols which take input from a replicated protocol will be updated to instead take a list of value, populated by the outputs of the replicated protocols.

Notes

- The *template_values* property must be a list of either constant values, or *ProtocolPath* objects which take their value from the *global* scope.
- If children of replicated protocols are also flagged as to be replicated, they will only have their ids changed to match the index of the parent protocol, as opposed to being fully replicated.

__init__(*replicator_id=”*)

Constructs a new ProtocolReplicator object.

Parameters **replicator_id** (*str*) – The id of this replicator.

Methods

__init__([replicator_id])	Constructs a new ProtocolReplicator object.
apply(protocols[, template_values, …])	Applies this replicator to the provided set of protocols and any of their children.
json()	Creates a JSON representation of this class.
parse_json(string_contents[, encoding])	Parses a typed json string into the corresponding class structure.
update_references(protocols, …)	Redirects the input references of protocols to the replicated versions.

Attributes

placeholder_id	The string which protocols to be replicated should include in their ids.

property placeholder_id
The string which protocols to be replicated should include in their ids.

apply(*protocols*, *template_values=None*, *template_index=-1*, *template_value=None*)
Applies this replicator to the provided set of protocols and any of their children.

This protocol should be followed by a call to *update_references* to ensure that all protocols which take their input from a replicated protocol get correctly updated.

Parameters

- **protocols** (*dict of str and BaseProtocol*) – The protocols to apply the replicator to.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by this replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by this replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

Returns

- *dict of str and BaseProtocol* – The replicated protocols.
- *dict of ProtocolPath and list of tuple of ProtocolPath and int* – A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

update_references(*protocols*, *replication_map*, *template_values*)
Redirects the input references of protocols to the replicated versions.

Parameters

- **protocols** (*dict of str and BaseProtocol*) – The protocols which have had this replicator applied to them.
- **replication_map** (*dict of ProtocolPath and list of tuple of ProtocolPath and int*) – A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.
- **template_values** (*List of Any*) – A list of the values which will be inserted into the newly replicated protocols.

json()
Creates a JSON representation of this class.

Returns The JSON representation of this class.

Return type str

classmethod parse_json(*string_contents*, *encoding='utf8'*)

Parses a typed json string into the corresponding class structure.

Parameters

- **string_contents** (*str or bytes*) – The typed json string.
- **encoding** (*str*) – The encoding of the *string_contents*.

Returns The parsed class.

Return type Any

WorkflowOutputToStore

class propertyestimator.workflow.schemas.**WorkflowOutputToStore**

An object which describes which data should be cached after a workflow has finished executing, and from which completed protocols should the data be collected from.

A *WorkflowOutputToStore* maps to the *BaseStoredData* stored data class.

substance

A reference to the composition of the collected data.

Type *ProtocolPath*

__init__()

Constructs a new WorkflowOutputToStore object.

Methods

__init__()	Constructs a new WorkflowOutputToStore object.

WorkflowSimulationDataToStore

class propertyestimator.workflow.schemas.**WorkflowSimulationDataToStore**

An object which describes which data should be cached after a workflow has finished executing, and from which completed protocols should the data be collected from.

A *WorkflowSimulationDataToStore* maps to the creation of a *StoredSimulationData* stored data class.

coordinate_file_path

A reference to the file path of a coordinate file which encodes the topology of the system.

Type *ProtocolPath*

trajectory_file_path

A reference to the file path of a .dcd trajectory file containing configurations generated by the simulation.

Type *ProtocolPath*

statistics_file_path

A reference to the file path of of a *StatisticsArray* csv file, containing statistics generated by the simulation.

Type *ProtocolPath*

statistical_inefficiency

A reference to the statistical inefficiency of the collected data.

Type *ProtocolPath*

total_number_of_molecules
A reference to the total number of molecules in the system.

Type *ProtocolPath*

__init__()
Constructs a new WorkflowSimulationDataToStore object.

Methods

__init__()	Constructs a new WorkflowSimulationDataToStore object.

WorkflowDataCollectionToStore

class propertyestimator.workflow.schemas.**WorkflowDataCollectionToStore**
An object which describes which data should be cached after a workflow has finished executing, and from which completed protocols should the data be collected from.

A *WorkflowDataCollectionToStore* maps to the creation of a *StoredDataCollection* stored data class.

data
A dictionary of stored simulation data objects which have been given a unique key.

Type dict of str and WorkflowSimulationDataToStore

__init__()
Constructs a new WorkflowDataCollectionToStore object.

Methods

__init__()	Constructs a new WorkflowDataCollectionToStore object.

Base Protocol API

BaseProtocol	The base class for a protocol which would form one step of a larger property calculation workflow.

BaseProtocol

class propertyestimator.workflow.protocols.**BaseProtocol**(*protocol_id*)
The base class for a protocol which would form one step of a larger property calculation workflow.

A protocol may for example:

- create the coordinates of a mixed simulation box
- set up a bound ligand-protein system
- build the simulation topology
- perform an energy minimisation

An individual protocol may require a set of inputs, which may either be set as constants

```
>>> from propertyestimator.protocols.simulation import RunOpenMMSimulation
>>>
>>> npt_equilibration = RunOpenMMSimulation('npt_equilibration')
>>> npt_equilibration.ensemble = RunOpenMMSimulation.Ensemble.NPT
```

or from the output of another protocol, pointed to by a ProtocolPath

```
>>> npt_production = RunOpenMMSimulation('npt_production')
>>> # Use the coordinate file output by the npt_equilibration protocol
>>> # as the input to the npt_production protocol
>>> npt_production.input_coordinate_file = ProtocolPath('output_coordinate_file',
>>>                                                     npt_equilibration.id)
```

In this way protocols may be chained together, thus defining a larger property calculation workflow from simple, reusable building blocks.

Warning: This class is still heavily under development and is subject to rapid changes.

`__init__`(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(protocol_id)	Initialize self.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.

Continued on next page

Table 87 – continued from previous page

schema	A serializable schema for this object.

property id
The unique id of this protocol.

Type str

property schema
A serializable schema for this object.

Type *ProtocolSchema*

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (`str`) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters value (`str`) – The uuid of the parent calculation.

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (`str`) – The id of the old input protocol.
- **new_id** (`str`) – The id of the new input protocol.

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters other (*BaseProtocol*) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

set_value(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (`ProtocolPath`) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

Input / Output Utilities

`PlaceholderInput`	A class to act as a place holder for a protocols input value, for when the value of an input is not known a priori, and does not come from another protocol.
`ReplicatorValue`	A placeholder value which will be set by a protocol replicator with the specified id.
`ProtocolPath`	Represents a pointer to the output of another protocol.

PlaceholderInput

class propertyestimator.workflow.utils.**PlaceholderInput**

A class to act as a place holder for a protocols input value, for when the value of an input is not known a priori, and does not come from another protocol.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`	Initialize self.

ReplicatorValue

class propertyestimator.workflow.utils.**ReplicatorValue**(*replicator_id=”*)

A placeholder value which will be set by a protocol replicator with the specified id.

__init__(*replicator_id=”*)

Constructs a new ReplicatorValue object

Parameters **replicator_id** (*str*) – The id of the replicator which will set this value.

Methods

__init__([replicator_id])	Constructs a new ReplicatorValue object

ProtocolPath

class propertyestimator.workflow.utils.**ProtocolPath**(*property_name=”*, **protocol_ids*)

Represents a pointer to the output of another protocol.

__init__(*property_name=”*, **protocol_ids*)

Constructs a new ProtocolPath object.

Parameters

- **property_name** (*str*) – The property name referenced by the path.
- **protocol_ids** (*str*) – An args list of protocol ids in the order in which they will appear in the path.

Methods

__init__([property_name])	Constructs a new ProtocolPath object.
append_uuid(uuid)	Appends a uuid to each of the protocol id’s in the path
from_string(existing_path_string)	
pop_next_in_path()	Pops and then returns the leading protocol id from the path.
prepend_protocol_id(id_to_prepend)	Prepend a new protocol id onto the front of the path.
replace_protocol(old_id, new_id)	Redirect the input to point at a new protocol.
to_components(path_string)	Splits a protocol path string into the property name, and the individual protocol ids.
validate(v)	

Attributes

full_path	The full path referenced by this object.
is_global	
last_protocol	The leading protocol id of the path.
path_separator	
property_name	The property name pointed to by the path.

Continued on next page

Table 92 – continued from previous page

`property_separator`	
`protocol_path`	The full path referenced by this object excluding the property name.
`start_protocol`	The leading protocol id of the path.

property property_name
The property name pointed to by the path.

Type str

property start_protocol
The leading protocol id of the path.

Type str

property last_protocol
The leading protocol id of the path.

Type str

property protocol_path
The full path referenced by this object excluding the property name.

Type str

property full_path
The full path referenced by this object.

Type str

static to_components(*path_string*)
Splits a protocol path string into the property name, and the individual protocol ids.

Parameters **path_string** (*str*) – The protocol path to split.

Returns A tuple of the property name, and a list of the protocol ids in the path.

Return type str, list of str

prepend_protocol_id(*id_to_prepend*)
Prepend a new protocol id onto the front of the path.

Parameters **id_to_prepend** (*str*) – The protocol id to prepend to the path

pop_next_in_path()
Pops and then returns the leading protocol id from the path.

Returns The previously leading protocol id.

Return type str

append_uuid(*uuid*)
Appends a uuid to each of the protocol id's in the path

Parameters **uuid** (*str*) – The uuid to append.

replace_protocol(*old_id*, *new_id*)
Redirect the input to point at a new protocol.

The main use of this method is when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the protocol to replace.

- **new_id** (*str*) – The id of the new protocol to use.

Decorators

protocol_input	A custom decorator used to mark a protocol attribute as a possible input.
protocol_output	A custom decorator used to mark a protocol attribute as an output of the protocol.
BaseProtocolInputObject	A custom decorator used to mark class attributes as either a required input, or output, of a protocol.
MergeBehaviour	A enum which describes how attributes should be handled when attempting to merge similar protocols.

propertyestimator.workflow.decorators.protocol_input

propertyestimator.workflow.decorators.**protocol_input**(*value_type*, *merge_behavior=<MergeBehaviour.ExactlyEqual: (0,)>*)

A custom decorator used to mark a protocol attribute as a possible input.

Examples

To mark an attribute as an input:

```
>>> from propertyestimator.substances import Substance
>>>
>>> @protocol_input(value_type=Substance)
>>> def substance(self, value):
>>>     pass
```

To control how this input should behave when protocols are being / considered being merged, use the merge_behavior attribute:

```
>>> @protocol_input(value_type=int, merge_behavior=MergeBehaviour.GreatestValue)
>>> def simulation_steps(self, value):
>>>     pass
```

propertyestimator.workflow.decorators.protocol_output

propertyestimator.workflow.decorators.**protocol_output**(*value_type*)

A custom decorator used to mark a protocol attribute as an output of the protocol.

Examples

To mark a property as an output:

```
>>> @protocol_output(value_type=str)
>>> def coordinate_file_path(self):
>>>     pass
```

BaseProtocolInputObject

class propertyestimator.workflow.decorators.**BaseProtocolInputObject**(*class_attribute*)

A custom decorator used to mark class attributes as either a required input, or output, of a protocol.

Notes

This decorator expects the protocol to have a matching private field in addition to the public attribute. For example if a protocol has an attribute *substance*, by default the protocol must also have a _*substance* field.

__init__(*class_attribute*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(class_attribute)	Initialize self.

MergeBehaviour

class propertyestimator.workflow.decorators.**MergeBehaviour**

A enum which describes how attributes should be handled when attempting to merge similar protocols.

Notes

Any attributes marked with a merge behavior of *ExactlyEqual* must be exactly for two protocols to merge.

__init__()
Initialize self. See help(type(self)) for accurate signature.

Attributes

`ExactlyEqual`
`GreatestValue`
`SmallestValue`

1.5.9 Built-in Workflow Protocols

Coordinate Generation

`BuildCoordinatesPackmol`	Creates a set of 3D coordinates with a specified composition.
`SolvateExistingStructure`	Creates a set of 3D coordinates with a specified composition.
`BuildDockedCoordinates`	Creates a set of coordinates for a ligand bound to some receptor.

BuildCoordinatesPackmol

class propertyestimator.protocols.coordinates.**BuildCoordinatesPackmol**(*protocol_id*)
Creates a set of 3D coordinates with a specified composition.

Notes

The coordinates are created using packmol.

__init__(*protocol_id*)
Constructs a new BuildCoordinatesPackmol object.

Methods

__init__(protocol_id)	Constructs a new BuildCoordinatesPackmol object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
box_aspect_ratio	The aspect ratio of the simulation box.
coordinate_file_path	The file path to the created PDB coordinate file.
dependencies	A list of pointers to the protocols which this protocol takes input from.
final_number_of_molecules	The file path to the created PDB coordinate file.
id	The unique id of this protocol.
mass_density	The target density of the created system.
max_molecules	The maximum number of molecules to be added to the system.
retain_packmol_files	If True, packmol will not delete all of the temporary files it creates while building the coordinates.
schema	A serializable schema for this object.
substance	The composition of the system to build.

Continued on next page

Table 98 – continued from previous page

verbose_packmol	If True, packmol will be allowed to log verbose information to the logger, and any working packmol files will be retained.

max_molecules
The maximum number of molecules to be added to the system.

mass_density
The target density of the created system.

box_aspect_ratio
The aspect ratio of the simulation box. The default is [1.0, 1.0, 1.0], i.e a cubic box.

substance
The composition of the system to build.

verbose_packmol
If True, packmol will be allowed to log verbose information to the logger, and any working packmol files will be retained.

retain_packmol_files
If True, packmol will not delete all of the temporary files it creates while building the coordinates.

final_number_of_molecules
The file path to the created PDB coordinate file.

coordinate_file_path
The file path to the created PDB coordinate file.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (`str`) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*

- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`
The unique id of this protocol.

Type str

`merge`(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`replace_protocol`(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

SolvateExistingStructure

class propertyestimator.protocols.coordinates.**SolvateExistingStructure**(*protocol_id*)

Creates a set of 3D coordinates with a specified composition.

Notes

The coordinates are created using packmol.

__init__(*protocol_id*)

Constructs a new BuildCoordinatesPackmol object.

Methods

`__init__`(protocol_id)	Constructs a new BuildCoordinatesPackmol object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`box_aspect_ratio`	The aspect ratio of the simulation box.
`coordinate_file_path`	The file path to the created PDB coordinate file.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`final_number_of_molecules`	The file path to the created PDB coordinate file.
`id`	The unique id of this protocol.
`mass_density`	The target density of the created system.
`max_molecules`	The maximum number of molecules to be added to the system.
`retain_packmol_files`	If True, packmol will not delete all of the temporary files it creates while building the coordinates.
`schema`	A serializable schema for this object.
`solute_coordinate_file`	A file path to the solute to solvate.

Continued on next page

Table 100 – continued from previous page

substance	The composition of the system to build.
verbose_packmol	If True, packmol will be allowed to log verbose information to the logger, and any working packmol files will be retained.

solute_coordinate_file
A file path to the solute to solvate.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`box_aspect_ratio`
The aspect ratio of the simulation box. The default is [1.0, 1.0, 1.0], i.e a cubic box.

`can_merge`(*other*)
Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`coordinate_file_path`
The file path to the created PDB coordinate file.

`property dependencies`
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`final_number_of_molecules`
The file path to the created PDB coordinate file.

`get_attribute_type`(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

`get_value_references`(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

mass_density
The target density of the created system.

max_molecules
The maximum number of molecules to be added to the system.

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

retain_packmol_files
If True, packmol will not delete all of the temporary files it creates while building the coordinates.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (`ProtocolPath`) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

substance
The composition of the system to build.

verbose_packmol
If True, packmol will be allowed to log verbose information to the logger, and any working packmol files will be retained.

BuildDockedCoordinates

class propertyestimator.protocols.coordinates.**BuildDockedCoordinates**(*protocol_id*)
Creates a set of coordinates for a ligand bound to some receptor.

Notes

This protocol currently only supports docking with the OpenEye OEDocking framework.

__init__(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

activate_site_location	Defines the method by which the activate site is identified.
allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
docked_complex_coordinate_path	The file path to the docked ligand-receptor complex.
docked_ligand_coordinate_path	The file path to the coordinates of the ligand in it's docked pose, aligned with the initial *receptor_coordinate_file*.

Continued on next page

Table 102 – continued from previous page

id	The unique id of this protocol.
ligand_residue_name	The residue name assigned to the docked ligand.
ligand_substance	A substance containing only the ligand to dock.
number_of_ligand_conformers	The number of conformers to try and dock into the receptor structure.
receptor_coordinate_file	The file path to the coordinates of the receptor molecule.
receptor_residue_name	The residue name assigned to the receptor.
schema	A serializable schema for this object.

class ActivateSiteLocation
An enum which describes the methods by which a receptors activate site(s) is located.

ligand_substance
A substance containing only the ligand to dock.

number_of_ligand_conformers
The number of conformers to try and dock into the receptor structure.

receptor_coordinate_file
The file path to the coordinates of the receptor molecule.

activate_site_location
Defines the method by which the activate site is identified. Currently the only available option is *ActivateSiteLocation.ReceptorCenterOfMass*

docked_ligand_coordinate_path
The file path to the coordinates of the ligand in it's docked pose, aligned with the initial *receptor_coordinate_file*.

docked_complex_coordinate_path
The file path to the docked ligand-receptor complex.

ligand_residue_name
The residue name assigned to the docked ligand.

receptor_residue_name
The residue name assigned to the receptor.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

Force Field Assignment

BuildSmirnoffSystem	Parametrise a set of molecules with a given smirnoff force field.

BuildSmirnoffSystem

class propertyestimator.protocols.forcefield.**BuildSmirnoffSystem**(*protocol_id*)
Parametrise a set of molecules with a given smirnoff force field.

__init__(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.

Continued on next page

Table 105 – continued from previous page

apply_known_charges	If true, formal the formal charges of ions, and the charges of the selected water model will be automatically applied to any matching molecules in the system.
charged_molecule_paths	File paths to mol2 files which contain the charges assigned to molecules in the system.
coordinate_file_path	The file path to the coordinate file which defines the system to which the force field parameters will be assigned.
dependencies	A list of pointers to the protocols which this protocol takes input from.
force_field_path	The file path to the force field parameters to assign to the system.
id	The unique id of this protocol.
schema	A serializable schema for this object.
substance	The composition of the system.
system_path	The assigned system.
water_model	The water model to apply, if any water molecules are present.

class WaterModel

An enum which describes which water model is being used, so that correct charges can be applied.

Warning: This is only a temporary addition until library charges are introduced into the openforcefield toolkit.

force_field_path

The file path to the force field parameters to assign to the system.

coordinate_file_path

The file path to the coordinate file which defines the system to which the force field parameters will be assigned.

charged_molecule_paths

File paths to mol2 files which contain the charges assigned to molecules in the system. This input is helpful when dealing with large molecules (such as hosts in host-guest binding calculations) whose charges may by needed in multiple places, and hence should only be calculated once.

substance

The composition of the system.

water_model

The water model to apply, if any water molecules are present.

Warning: This is only a temporary addition until library charges are introduced into the openforcefield toolkit.

apply_known_charges

If true, formal the formal charges of ions, and the charges of the selected water model will be automatically applied to any matching molecules in the system.

system_path

The assigned system.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters value (*str*) – The uuid of the parent calculation.

set_value(*reference_path, value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

Simulation

RunEnergyMinimisation	A protocol to minimise the potential energy of a system.
RunOpenMMSimulation	Performs a molecular dynamics simulation in a given ensemble using an OpenMM backend.
BaseYankProtocol	An abstract base class for protocols which will performs a set of alchemical free energy simulations using the YANK framework.
LigandReceptorYankProtocol	An abstract base class for protocols which will performs a set of alchemical free energy simulations using the YANK framework.

RunEnergyMinimisation

class propertyestimator.protocols.simulation.**RunEnergyMinimisation**(*protocol_id*)
A protocol to minimise the potential energy of a system.

__init__(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.

Continued on next page

Table 107 – continued from previous page

`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`enable_pbc`	If true, periodic boundary conditions will be enabled.
`id`	The unique id of this protocol.
`input_coordinate_file`	The coordinates to minimise.
`max_iterations`	The maximum number of iterations to perform.
`output_coordinate_file`	The file path to the minimised coordinates.
`schema`	A serializable schema for this object.
`system_path`	The path to the XML system object which defines the forces present in the system.
`tolerance`	The energy tolerance to which the system should be minimized.

input_coordinate_file
The coordinates to minimise.

tolerance
The energy tolerance to which the system should be minimized.

max_iterations
The maximum number of iterations to perform. If this is 0, minimization is continued until the results converge without regard to how many iterations it takes.

system_path
The path to the XML system object which defines the forces present in the system.

enable_pbc
If true, periodic boundary conditions will be enabled.

output_coordinate_file
The file path to the minimised coordinates.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging

If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path, value*)
Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

RunOpenMMSimulation

`class` `propertyestimator.protocols.simulation.`**`RunOpenMMSimulation`**(*protocol_id*)
Performs a molecular dynamics simulation in a given ensemble using an OpenMM backend.

`__init__`(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(protocol_id)	Initialize self.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to

Continued on next page

Table 109 – continued from previous page

set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_gpu_platforms	If true, OpenMM will be allowed to run using a GPU if available, otherwise it will be constrained to only using CPUs.
allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
enable_pbc	If true, periodic boundary conditions will be enabled.
ensemble	The thermodynamic ensemble to simulate in.
high_precision	If true, OpenMM will be run using a platform with high precision settings.
id	The unique id of this protocol.
input_coordinate_file	The file path to the starting coordinates.
output_coordinate_file	The file path to the coordinates of the final system configuration.
output_frequency	The frequency with which to write to the output statistics and trajectory files.
save_rolling_statistics	If True, the statisitics file will be written to every *output_frequency* number of steps, rather than just once at the end of the simulation.
schema	A serializable schema for this object.
statistics_file_path	The file path to the statistics sampled during the simulation.
steps	The number of timesteps to evolve the system by.
system_path	A path to the XML system object which defines the forces present in the system.
thermodynamic_state	The thermodynamic conditions to simulate under
thermostat_friction	The thermostat friction coefficient.
timestep	The timestep to evolve the system by at each step.
trajectory_file_path	The file path to the trajectory sampled during the simulation.

steps
The number of timesteps to evolve the system by.

thermostat_friction
The thermostat friction coefficient.

timestep
The timestep to evolve the system by at each step.

output_frequency
The frequency with which to write to the output statistics and trajectory files.

ensemble
The thermodynamic ensemble to simulate in.

thermodynamic_state
The thermodynamic conditions to simulate under

`input_coordinate_file`
The file path to the starting coordinates.

`system_path`
A path to the XML system object which defines the forces present in the system.

`enable_pbc`
If true, periodic boundary conditions will be enabled.

`save_rolling_statistics`
If True, the statisitics file will be written to every *output_frequency* number of steps, rather than just once at the end of the simulation.

Notes

In future when either saving the statistics to file has been optimised, or an option for the frequency to save to the file has been added, this option will be removed.

`allow_gpu_platforms`
If true, OpenMM will be allowed to run using a GPU if available, otherwise it will be constrained to only using CPUs.

`high_precision`
If true, OpenMM will be run using a platform with high precision settings. This will be the Reference platform when only a CPU is available, or double precision mode when a GPU is available.

`output_coordinate_file`
The file path to the coordinates of the final system configuration.

`trajectory_file_path`
The file path to the trajectory sampled during the simulation.

`statistics_file_path`
The file path to the statistics sampled during the simulation.

`execute`(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **`directory`** (`str`) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

`allow_merging`
If true, this protocol is allowed to merge with other identical protocols.

Type bool

`apply_replicator`(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

- **value** (*Any*) – The value to set.

BaseYankProtocol

class propertyestimator.protocols.simulation.**BaseYankProtocol**(*protocol_id*)

An abstract base class for protocols which will performs a set of alchemical free energy simulations using the YANK framework.

Protocols which inherit from this base must implement the abstract *_get_yank_options* methods.

__init__(*protocol_id*)

Constructs a new BaseYankProtocol object.

Methods

__init__(protocol_id)	Constructs a new BaseYankProtocol object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
checkpoint_interval	The number of iterations between saving YANK checkpoint files.
dependencies	A list of pointers to the protocols which this protocol takes input from.
estimated_free_energy	The estimated free energy value and its uncertainty returned by YANK.
force_field_path	The path to the force field to use for the calculations
id	The unique id of this protocol.
number_of_iterations	The number of YANK iterations to perform.
schema	A serializable schema for this object.
steps_per_iteration	The number of steps per YANK iteration to perform.
thermodynamic_state	The state at which to run the calculations.

Continued on next page

Table 112 – continued from previous page

timestep	The length of the timestep to take.
verbose	Controls whether or not to run YANK at high verbosity.

thermodynamic_state
The state at which to run the calculations.

number_of_iterations
The number of YANK iterations to perform.

steps_per_iteration
The number of steps per YANK iteration to perform.

checkpoint_interval
The number of iterations between saving YANK checkpoint files.

timestep
The length of the timestep to take.

force_field_path
The path to the force field to use for the calculations

verbose
Controls whether or not to run YANK at high verbosity.

estimated_free_energy
The estimated free energy value and its uncertainty returned by YANK.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (`str`) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (`list of Any`) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*

- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`
The unique id of this protocol.

Type str

`merge`(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`replace_protocol`(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

LigandReceptorYankProtocol

class propertyestimator.protocols.simulation.**LigandReceptorYankProtocol**(*protocol_id*)

An abstract base class for protocols which will performs a set of alchemical free energy simulations using the YANK framework.

Protocols which inherit from this base must implement the abstract *_get_*_dictionary* methods.

__init__(*protocol_id*)

Constructs a new LigandReceptorYankProtocol object.

Methods

`__init__`(protocol_id)	Constructs a new LigandReceptorYankProtocol object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`apply_restraints`	Determines whether the ligand should be explicitly restrained to the receptor in order to stop the ligand from temporarily unbinding.
`checkpoint_interval`	The number of iterations between saving YANK checkpoint files.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`estimated_free_energy`	The estimated free energy value and its uncertainty returned by YANK.
`force_field_path`	The path to the force field to use for the calculations
`id`	The unique id of this protocol.
`ligand_residue_name`	The residue name of the ligand.
`number_of_iterations`	The number of YANK iterations to perform.
`receptor_residue_name`	The residue name of the receptor.

Continued on next page

Table 114 – continued from previous page

restraint_type	The type of ligand restraint applied, provided that *apply_restraints* is *True*
schema	A serializable schema for this object.
solvated_complex_coordinates	The file path to the solvated complex coordinates.
solvated_complex_system	The file path to the solvated complex system object.
solvated_complex_trajectory_path	The file path to the generated ligand trajectory.
solvated_ligand_coordinates	The file path to the solvated ligand coordinates.
solvated_ligand_system	The file path to the solvated ligand system object.
solvated_ligand_trajectory_path	The file path to the generated ligand trajectory.
steps_per_iteration	The number of steps per YANK iteration to perform.
thermodynamic_state	The state at which to run the calculations.
timestep	The length of the timestep to take.
verbose	Controls whether or not to run YANK at high verbosity.

class RestraintType
The types of ligand restraints available within yank.

ligand_residue_name
The residue name of the ligand.

receptor_residue_name
The residue name of the receptor.

solvated_ligand_coordinates
The file path to the solvated ligand coordinates.

solvated_ligand_system
The file path to the solvated ligand system object.

solvated_complex_coordinates
The file path to the solvated complex coordinates.

solvated_complex_system
The file path to the solvated complex system object.

apply_restraints
Determines whether the ligand should be explicitly restrained to the receptor in order to stop the ligand from temporarily unbinding.

restraint_type
The type of ligand restraint applied, provided that *apply_restraints* is *True*

solvated_ligand_trajectory_path
The file path to the generated ligand trajectory.

solvated_complex_trajectory_path
The file path to the generated ligand trajectory.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging

If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

checkpoint_interval

The number of iterations between saving YANK checkpoint files.

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

estimated_free_energy
The estimated free energy value and its uncertainty returned by YANK.

force_field_path
The path to the force field to use for the calculations

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

number_of_iterations
The number of YANK iterations to perform.

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path, value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

steps_per_iteration

The number of steps per YANK iteration to perform.

thermodynamic_state

The state at which to run the calculations.

timestep

The length of the timestep to take.

verbose

Controls whether or not to run YANK at high verbosity.

Simulation Analysis

AveragePropertyProtocol	An abstract base class for protocols which will calculate the average of a property and its uncertainty via bootstrapping.
AverageTrajectoryProperty	An abstract base class for protocols which will calculate the average of a property from a simulation trajectory.
ExtractAverageStatistic	Extracts the average value from a statistics file which was generated during a simulation.
ExtractUncorrelatedData	An abstract base class for protocols which will subsample a data set, yielding only equilibrated, uncorrelated data.
ExtractUncorrelatedTrajectoryData	A protocol which will subsample frames from a trajectory, yielding only uncorrelated frames as determined from a provided statistical inefficiency and equilibration time.

Continued on next page

Table 115 – continued from previous page

ExtractUncorrelatedStatisticsData	A protocol which will subsample entries from a statistics array, yielding only uncorrelated entries as determined from a provided statistical inefficiency and equilibration time.

AveragePropertyProtocol

class propertyestimator.protocols.analysis.**AveragePropertyProtocol**(*protocol_id*)

An abstract base class for protocols which will calculate the average of a property and its uncertainty via bootstrapping.

__init__(*protocol_id*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
bootstrap_iterations	The number of bootstrap iterations to perform.
bootstrap_sample_size	The relative sample size to use for bootstrapping.
dependencies	A list of pointers to the protocols which this protocol takes input from.
equilibration_index	The index in the data set after which the data is stationary.
id	The unique id of this protocol.
schema	A serializable schema for this object.
statistical_inefficiency	The statistical inefficiency in the data set.

Continued on next page

Table 117 – continued from previous page

`uncorrelated_values`	The uncorrelated values which the average was calculated from.
`value`	The averaged value.

`bootstrap_iterations`
The number of bootstrap iterations to perform.

`bootstrap_sample_size`
The relative sample size to use for bootstrapping.

`value`
The averaged value.

`equilibration_index`
The index in the data set after which the data is stationary.

`statistical_inefficiency`
The statistical inefficiency in the data set.

`uncorrelated_values`
The uncorrelated values which the average was calculated from.

`execute`(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **`directory`** (`str`) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

`allow_merging`
If true, this protocol is allowed to merge with other identical protocols.

Type bool

`apply_replicator`(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (*`list of Any`*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

AverageTrajectoryProperty

class propertyestimator.protocols.analysis.**AverageTrajectoryProperty**(*protocol_id*)
An abstract base class for protocols which will calculate the average of a property from a simulation trajectory.

__init__(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(protocol_id)	Initialize self.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`bootstrap_iterations`	The number of bootstrap iterations to perform.
`bootstrap_sample_size`	The relative sample size to use for bootstrapping.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`equilibration_index`	The index in the data set after which the data is stationary.
`id`	The unique id of this protocol.
`input_coordinate_file`	The file path to the starting coordinates of a trajectory.
`schema`	A serializable schema for this object.
`statistical_inefficiency`	The statistical inefficiency in the data set.
`trajectory_path`	The file path to the trajectory to average over.
`uncorrelated_values`	The uncorrelated values which the average was calculated from.
`value`	The averaged value.

`input_coordinate_file`
The file path to the starting coordinates of a trajectory.

`trajectory_path`
The file path to the trajectory to average over.

`execute` (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

bootstrap_iterations
The number of bootstrap iterations to perform.

bootstrap_sample_size
The relative sample size to use for bootstrapping.

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

equilibration_index
The index in the data set after which the data is stationary.

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path, value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

statistical_inefficiency
The statistical inefficiency in the data set.

uncorrelated_values
The uncorrelated values which the average was calculated from.

value
The averaged value.

ExtractAverageStatistic

class propertyestimator.protocols.analysis.**ExtractAverageStatistic**(*protocol_id*)
Extracts the average value from a statistics file which was generated during a simulation.

__init__(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.

Continued on next page

Table 120 – continued from previous page

get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
bootstrap_iterations	The number of bootstrap iterations to perform.
bootstrap_sample_size	The relative sample size to use for bootstrapping.
dependencies	A list of pointers to the protocols which this protocol takes input from.
divisor	A divisor to divide the statistic by.
equilibration_index	The index in the data set after which the data is stationary.
id	The unique id of this protocol.
schema	A serializable schema for this object.
statistical_inefficiency	The statistical inefficiency in the data set.
statistics_path	The file path to the trajectory to average over.
statistics_type	The file path to the trajectory to average over.
uncorrelated_values	The uncorrelated values which the average was calculated from.
value	The averaged value.

statistics_path
The file path to the trajectory to average over.

statistics_type
The file path to the trajectory to average over.

divisor
A divisor to divide the statistic by. This is useful if a statistic (such as enthalpy) needs to be normalised by the number of molecules.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

bootstrap_iterations
The number of bootstrap iterations to perform.

bootstrap_sample_size
The relative sample size to use for bootstrapping.

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

equilibration_index
The index in the data set after which the data is stationary.

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters value (*str*) – The uuid of the parent calculation.

set_value(*reference_path, value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (*ProtocolPath*) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

statistical_inefficiency

The statistical inefficiency in the data set.

uncorrelated_values

The uncorrelated values which the average was calculated from.

value

The averaged value.

ExtractUncorrelatedData

class propertyestimator.protocols.analysis.**ExtractUncorrelatedData**(*protocol_id*)

An abstract base class for protocols which will subsample a data set, yielding only equilibrated, uncorrelated data.

__init__(*protocol_id*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.

Continued on next page

Table 122 – continued from previous page

get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
equilibration_index	The index in the data set after which the data is stationary.
id	The unique id of this protocol.
number_of_uncorrelated_samples	The number of uncorrelated samples.
schema	A serializable schema for this object.
statistical_inefficiency	The statistical inefficiency in the data set.

equilibration_index
The index in the data set after which the data is stationary.

statistical_inefficiency
The statistical inefficiency in the data set.

number_of_uncorrelated_samples
The number of uncorrelated samples.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains

the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (*`list of Any`*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **`template_value`** (*`Any, optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)
Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`property dependencies`
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`get_attribute_type`(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id

The unique id of this protocol.

Type str

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

ExtractUncorrelatedTrajectoryData

class propertyestimator.protocols.analysis.**ExtractUncorrelatedTrajectoryData**(*protocol_id*)

A protocol which will subsample frames from a trajectory, yielding only uncorrelated frames as determined from a provided statistical inefficiency and equilibration time.

__init__(*protocol_id*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(protocol_id)	Initialize self.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
equilibration_index	The index in the data set after which the data is stationary.
id	The unique id of this protocol.
input_coordinate_file	The file path to the starting coordinates of a trajectory.
input_trajectory_path	The file path to the trajectory to subsample.
number_of_uncorrelated_samples	The number of uncorrelated samples.
output_trajectory_path	The file path to the subsampled trajectory.
schema	A serializable schema for this object.

Continued on next page

Table 125 – continued from previous page

statistical_inefficiency	The statistical inefficiency in the data set.

input_coordinate_file
The file path to the starting coordinates of a trajectory.

input_trajectory_path
The file path to the trajectory to subsample.

output_trajectory_path
The file path to the subsampled trajectory.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the

actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)
Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`property dependencies`
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`equilibration_index`
The index in the data set after which the data is stationary.

`get_attribute_type`(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

`get_value_references`(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

number_of_uncorrelated_samples
The number of uncorrelated samples.

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

statistical_inefficiency
The statistical inefficiency in the data set.

ExtractUncorrelatedStatisticsData

class propertyestimator.protocols.analysis.**ExtractUncorrelatedStatisticsData**(*protocol_id*)
A protocol which will subsample entries from a statistics array, yielding only uncorrelated entries as determined from a provided statistical inefficiency and equilibration time.

__init__(*protocol_id*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(protocol_id)	Initialize self.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`equilibration_index`	The index in the data set after which the data is stationary.
`id`	The unique id of this protocol.
`input_statistics_path`	The file path to the statistics to subsample.
`number_of_uncorrelated_samples`	The number of uncorrelated samples.
`output_statistics_path`	The file path to the subsampled statistics.
`schema`	A serializable schema for this object.
`statistical_inefficiency`	The statistical inefficiency in the data set.

`input_statistics_path`
The file path to the statistics to subsample.

`output_statistics_path`
The file path to the subsampled statistics.

`execute` (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **`directory`** (`str`) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

equilibration_index
The index in the data set after which the data is stationary.

`get_attribute_type`(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

`get_value_references`(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`
The unique id of this protocol.

Type str

`merge`(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`number_of_uncorrelated_samples`
The number of uncorrelated samples.

`replace_protocol`(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

`statistical_inefficiency`
The statistical inefficiency in the data set.

Reweighting

`ConcatenateTrajectories`	A protocol which concatenates multiple trajectories into a single one.
`ConcatenateStatistics`	A protocol which concatenates multiple trajectories into a single one.
`CalculateReducedPotentialOpenMM`	Calculates the reduced potential for a given set of configurations.
`BaseMBARProtocol`	Reweights a set of observables using MBAR to calculate the average value of the observables at a different state than they were originally measured.
`ReweightStatistics`	Reweights a set of observables from a *StatisticsArray* using MBAR.

ConcatenateTrajectories

`class` `propertyestimator.protocols.reweighting.`**`ConcatenateTrajectories`**(*protocol_id*)
A protocol which concatenates multiple trajectories into a single one.

`__init__`(*protocol_id*)
Constructs a new ConcatenateTrajectories object.

Methods

`__init__`(protocol_id)	Constructs a new ConcatenateTrajectories object.

Continued on next page

Table 129 – continued from previous page

apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
input_coordinate_paths	A list of paths to the starting coordinates for each of the trajectories.
input_trajectory_paths	A list of paths to the trajectories to concatenate.
output_coordinate_path	The path the coordinate file which contains the topology of the concatenated trajectory.
output_trajectory_path	The path to the concatenated trajectory.
schema	A serializable schema for this object.

input_coordinate_paths
A list of paths to the starting coordinates for each of the trajectories.

input_trajectory_paths
A list of paths to the trajectories to concatenate.

output_coordinate_path
The path the coordinate file which contains the topology of the concatenated trajectory.

output_trajectory_path
The path to the concatenated trajectory.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.

- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.

- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters value (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

ConcatenateStatistics

class propertyestimator.protocols.reweighting.**ConcatenateStatistics**(*protocol_id*)
A protocol which concatenates multiple trajectories into a single one.

__init__(*protocol_id*)
Constructs a new ConcatenateStatistics object.

Methods

__init__(protocol_id)	Constructs a new ConcatenateStatistics object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
input_statistics_paths	A list of paths to the different statistics arrays.
output_statistics_path	The path the csv file which contains the concatenated statistics.
schema	A serializable schema for this object.

input_statistics_paths
A list of paths to the different statistics arrays.

output_statistics_path
The path the csv file which contains the concatenated statistics.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

CalculateReducedPotentialOpenMM

class propertyestimator.protocols.reweighting.**CalculateReducedPotentialOpenMM**(*protocol_id*)
Calculates the reduced potential for a given set of configurations.

__init__(*protocol_id*)
Constructs a new CalculateReducedPotentialOpenMM object.

Methods

__init__(protocol_id)	Constructs a new CalculateReducedPotentialOpenMM object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
coordinate_file_path	The path to the coordinate file which contains topology information about the system.
dependencies	A list of pointers to the protocols which this protocol takes input from.
enable_pbc	If true, periodic boundary conditions will be enabled.
high_precision	If true, OpenMM will be run in double precision mode.
id	The unique id of this protocol.
kinetic_energies_path	The file path to a statistics array which contain the kinetic energies of each frame in the trajectory.
schema	A serializable schema for this object.
statistics_file_path	A file path to the StatisticsArray file which contains the reduced potentials, and the potential, kinetic and total energies and enthalpies evaluated at the specified state and using the specified system object.
system_path	The path to the system object which describes the systems potential energy function.
thermodynamic_state	The state to calculate the reduced potential at.
trajectory_file_path	The path to the trajectory file which contains the configurations to calculate the energies of.
use_internal_energy	If true the internal energy, rather than the potential energy will be used when calculating the reduced potential.

thermodynamic_state
The state to calculate the reduced potential at.

system_path

The path to the system object which describes the systems potential energy function.

enable_pbc
If true, periodic boundary conditions will be enabled.

coordinate_file_path
The path to the coordinate file which contains topology information about the system.

trajectory_file_path
The path to the trajectory file which contains the configurations to calculate the energies of.

kinetic_energies_path
The file path to a statistics array which contain the kinetic energies of each frame in the trajectory.

high_precision
If true, OpenMM will be run in double precision mode.

use_internal_energy
If true the internal energy, rather than the potential energy will be used when calculating the reduced potential. This is required when reweighting properties which depend on the total energy, such as enthalpy.

statistics_file_path
A file path to the StatisticsArray file which contains the reduced potentials, and the potential, kinetic and total energies and enthalpies evaluated at the specified state and using the specified system object.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (`str`) – The directory to store output data in.
- **available_resources** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (`Any, optional`) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (`bool`) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (`propertyestimator.workflow.utils.ProtocolPath`) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id, new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path, value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (`ProtocolPath`) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

BaseMBARProtocol

class propertyestimator.protocols.reweighting.**BaseMBARProtocol**(*protocol_id*)
Reweights a set of observables using MBAR to calculate the average value of the observables at a different state than they were originally measured.

__init__(*protocol_id*)
Constructs a new BaseMBARProtocol object.

Methods

`__init__`(protocol_id)	Constructs a new BaseMBARProtocol object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`bootstrap_iterations`	The number of bootstrap iterations to perform if bootstraped uncertainties have been requested
`bootstrap_sample_size`	The relative bootstrap sample size to use if bootstraped uncertainties have been requested
`bootstrap_uncertainties`	If true, bootstrapping will be used to estimated the total uncertainty
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`effective_sample_indices`	The indices of those samples which have a non-zero weight.
`effective_samples`	The number of effective samples which were reweighted.
`id`	The unique id of this protocol.
`reference_reduced_potentials`	A list of paths to the reduced potentials of each reference state.
`required_effective_samples`	The minimum number of MBAR effective samples for the reweighted value to be trusted.
`schema`	A serializable schema for this object.
`target_reduced_potentials`	A list of paths to the reduced potentials of the target state.
`value`	The reweighted average value of the observable at the target state.

`reference_reduced_potentials`

A list of paths to the reduced potentials of each reference state.

target_reduced_potentials
A list of paths to the reduced potentials of the target state.

bootstrap_uncertainties
If true, bootstrapping will be used to estimated the total uncertainty

bootstrap_iterations
The number of bootstrap iterations to perform if bootstraped uncertainties have been requested

bootstrap_sample_size
The relative bootstrap sample size to use if bootstraped uncertainties have been requested

required_effective_samples
The minimum number of MBAR effective samples for the reweighted value to be trusted. If this minimum is not met then the uncertainty will be set to sys.float_info.max

value
The reweighted average value of the observable at the target state.

effective_samples
The number of effective samples which were reweighted.

effective_sample_indices
The indices of those samples which have a non-zero weight.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (`Any, optional`) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (`bool`) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (`ProtocolPath`) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

ReweightStatistics

class propertyestimator.protocols.reweighting.**ReweightStatistics**(*protocol_id*)
Reweights a set of observables from a *StatisticsArray* using MBAR.

__init__(*protocol_id*)
Constructs a new ReweightWithMBARProtocol object.

Methods

`__init__`(protocol_id)	Constructs a new ReweightWithMBARProtocol object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`bootstrap_iterations`	The number of bootstrap iterations to perform if bootstraped uncertainties have been requested
`bootstrap_sample_size`	The relative bootstrap sample size to use if bootstraped uncertainties have been requested
`bootstrap_uncertainties`	If true, bootstrapping will be used to estimated the total uncertainty
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`effective_sample_indices`	The indices of those samples which have a non-zero weight.
`effective_samples`	The number of effective samples which were reweighted.
`frame_counts`	An optional list which describes how many of the statistics in the array belong to each reference state.
`id`	The unique id of this protocol.
`reference_reduced_potentials`	A list of paths to the reduced potentials of each reference state.
`required_effective_samples`	The minimum number of MBAR effective samples for the reweighted value to be trusted.
`schema`	A serializable schema for this object.
`statistics_paths`	The file paths to the statistics array which contains the observables of interest from each state.
`statistics_type`	The type of observable to reweight.

Continued on next page

Table 138 – continued from previous page

`target_reduced_potentials`	A list of paths to the reduced potentials of the target state.
`value`	The reweighted average value of the observable at the target state.

`statistics_paths`
The file paths to the statistics array which contains the observables of interest from each state. If the observable of interest is dependant on the changing variable (e.g. the potential energy) then this must be a path to the observable re-evaluated at the new state.

`statistics_type`
The type of observable to reweight.

`frame_counts`
An optional list which describes how many of the statistics in the array belong to each reference state. If this input is used, only a single file path should be passed to the *statistics_paths* input.

`execute`(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **`directory`** (`str`) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

`allow_merging`
If true, this protocol is allowed to merge with other identical protocols.

Type bool

`apply_replicator`(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (`list of Any`) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (`int, optional`) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **`template_value`** (`Any, optional`) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

bootstrap_iterations
The number of bootstrap iterations to perform if bootstraped uncertainties have been requested

bootstrap_sample_size
The relative bootstrap sample size to use if bootstraped uncertainties have been requested

bootstrap_uncertainties
If true, bootstrapping will be used to estimated the total uncertainty

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

effective_sample_indices
The indices of those samples which have a non-zero weight.

effective_samples
The number of effective samples which were reweighted.

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id

The unique id of this protocol.

Type str

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

reference_reduced_potentials

A list of paths to the reduced potentials of each reference state.

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

required_effective_samples

The minimum number of MBAR effective samples for the reweighted value to be trusted. If this minimum is not met then the uncertainty will be set to sys.float_info.max

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

target_reduced_potentials
A list of paths to the reduced potentials of the target state.

value
The reweighted average value of the observable at the target state.

Gradients

GradientReducedPotentials	A protocol to estimates the gradient of an observable with respect to a number of specified force field parameters.
CentralDifferenceGradient	A protocol which employs the central diference method to estimate the gradient of an observable A, such that
DivideGradientByScalar	A protocol which divides a gradient by a specified scalar
MultiplyGradientByScalar	A protocol which multiplies a gradient by a specified scalar
AddGradients	A temporary protocol to add together multiple gradients.
SubtractGradients	A temporary protocol to add together two gradients.

GradientReducedPotentials

class propertyestimator.protocols.gradients.**GradientReducedPotentials**(*protocol_id*)
A protocol to estimates the gradient of an observable with respect to a number of specified force field parameters.

__init__(*protocol_id*)
Constructs a new EstimateParameterGradients object.

Methods

__init__(protocol_id)	Constructs a new EstimateParameterGradients object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.

Continued on next page

Table 140 – continued from previous page

`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`coordinate_file_path`	A path to the initial coordinates of the simulation trajectory which was used to estimate the observable of interest.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`effective_sample_indices`	NOTE - this is currently a placeholder input ONLY, and currently is not used for anything.
`enable_pbc`	If true, periodic boundary conditions will be enabled when re-evaluating the reduced potentials.
`force_field_path`	A path to the force field which contains the parameters to differentiate the observable with respect to.
`forward_parameter_value`	
`forward_potentials_path`	
`id`	The unique id of this protocol.
`parameter_key`	A list of the parameters to differentiate with respect to.
`perturbation_scale`	The amount to perturb the parameter by, such that p_new = p_old * (1 +/- perturbation_scale)
`reference_force_field_paths`	A list of path to the force field file which were originally used to estimate the observable of interest.
`reference_potential_paths`	
`reference_statistics_path`	An optional path to the statistics array which was generated alongside the observable of interest, which will be used to correct the potential energies at the reverse and forward states.
`reverse_parameter_value`	
`reverse_potentials_path`	
`schema`	A serializable schema for this object.
`substance`	The substance which describes the composition of the system.
`thermodynamic_state`	The thermodynamic state to estimate the gradients at.
`trajectory_file_path`	A path to the simulation trajectory which was used to estimate the observable of interest.
`use_subset_of_force_field`	If true, the reduced potential will be estimated using an OpenMM system which only contains the parameter of interest.

reference_force_field_paths
A list of path to the force field file which were originally used to estimate the observable of interest.

reference_statistics_path
An optional path to the statistics array which was generated alongside the observable of interest, which will be used to correct the potential energies at the reverse and forward states.

This is only really needed when the observable of interest is an energy.

force_field_path
A path to the force field which contains the parameters to differentiate the observable with respect to.

enable_pbc
If true, periodic boundary conditions will be enabled when re-evaluating the reduced potentials.

substance
The substance which describes the composition of the system.

thermodynamic_state
The thermodynamic state to estimate the gradients at.

coordinate_file_path
A path to the initial coordinates of the simulation trajectory which was used to estimate the observable of interest.

trajectory_file_path
A path to the simulation trajectory which was used to estimate the observable of interest.

parameter_key
A list of the parameters to differentiate with respect to.

perturbation_scale
The amount to perturb the parameter by, such that p_new = p_old * (1 +/- perturbation_scale)

use_subset_of_force_field
If true, the reduced potential will be estimated using an OpenMM system which only contains the parameter of interest.

effective_sample_indices
NOTE - this is currently a placeholder input ONLY, and currently is not used for anything.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (`ProtocolReplicator`) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (*ProtocolPath*) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

CentralDifferenceGradient

class propertyestimator.protocols.gradients.**CentralDifferenceGradient**(*protocol_id*)
A protocol which employs the central diference method to estimate the gradient of an observable A, such that

grad = (A(x-h) - A(x+h)) / (2h)

Notes

The *values* input must either be a list of unit.Quantity, a ProtocolPath to a list of unit.Quantity, or a list of ProtocolPath which each point to a unit.Quantity.

__init__(*protocol_id*)
Constructs a new CentralDifferenceGradient object.

Methods

`__init__`(protocol_id)	Constructs a new CentralDifferenceGradient object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.

Continued on next page

Table 143 – continued from previous page

dependencies	A list of pointers to the protocols which this protocol takes input from.
forward_observable_value	The value of A(x+h).
forward_parameter_value	The value of x+h.
gradient	The estimated gradient.
id	The unique id of this protocol.
parameter_key	The key that describes which parameters this gradient was estimated for.
reverse_observable_value	The value of A(x-h).
reverse_parameter_value	The value of x-h.
schema	A serializable schema for this object.

parameter_key
The key that describes which parameters this gradient was estimated for.

reverse_observable_value
The value of A(x-h).

forward_observable_value
The value of A(x+h).

reverse_parameter_value
The value of x-h.

forward_parameter_value
The value of x+h.

gradient
The estimated gradient.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (*ComputeResources*) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (*ProtocolReplicator*) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

This parameter is mutually exclusive with *template_index* and *template_value*

- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`

The unique id of this protocol.

Type str

`merge`(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`replace_protocol`(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

DivideGradientByScalar

class propertyestimator.protocols.gradients.**DivideGradientByScalar**(*protocol_id*)

A protocol which divides a gradient by a specified scalar

Notes

Once a more robust type system is built-in, this will be deprecated by *DivideValue*.

__init__(*protocol_id*)

Constructs a new DivideValue object.

Methods

`__init__`(protocol_id)	Constructs a new DivideValue object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`divisor`	The scalar to divide by.
`id`	The unique id of this protocol.
`result`	The result of the division.
`schema`	A serializable schema for this object.
`value`	The value to divide.

value

The value to divide.

divisor

The scalar to divide by.

`result`
The result of the division.

`execute`(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

`allow_merging`
If true, this protocol is allowed to merge with other identical protocols.

Type bool

`apply_replicator`(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (*`list of Any`*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **`template_value`** (*`Any, optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id

The unique id of this protocol.

Type str

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

MultiplyGradientByScalar

class propertyestimator.protocols.gradients.**MultiplyGradientByScalar**(*protocol_id*)

A protocol which multiplies a gradient by a specified scalar

Notes

Once a more robust type system is built-in, this will be deprecated by *MultiplyValue*.

__init__(*protocol_id*)

Constructs a new DivideValue object.

Methods

__init__(protocol_id)	Constructs a new DivideValue object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.

Continued on next page

Table 146 – continued from previous page

get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
result	The result of the division.
scalar	The scalar to multiply by.
schema	A serializable schema for this object.
value	The value to divide.

value
The value to divide.

scalar
The scalar to multiply by.

result
The result of the division.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (*`list of Any`*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **`template_value`** (*`Any, optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)

Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`property dependencies`

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`get_attribute_type`(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id

The unique id of this protocol.

Type str

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

AddGradients

class propertyestimator.protocols.gradients.**AddGradients**(*protocol_id*)

A temporary protocol to add together multiple gradients.

Notes

Once a more robust type system is built-in, this will be deprecated by *AddValues*.

__init__(*protocol_id*)

Constructs a new AddGradients object.

Methods

__init__(protocol_id)	Constructs a new AddGradients object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
result	The sum of the values.
schema	A serializable schema for this object.
values	The gradients to add together.

values

The gradients to add together.

result
The sum of the values.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

SubtractGradients

class propertyestimator.protocols.gradients.**SubtractGradients**(*protocol_id*)

A temporary protocol to add together two gradients.

Notes

Once a more robust type system is built-in, this will be deprecated by *SubtractValues*.

__init__(*protocol_id*)

Constructs a new AddValues object.

Methods

__init__(protocol_id)	Constructs a new AddValues object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.

Continued on next page

Table 150 – continued from previous page

get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
result	The sum of the values.
schema	A serializable schema for this object.
value_a	*value_a* in the formula *result* = *value_b* - *value_a*
value_b	*value_b* in the formula *result* = *value_b* - *value_a*

value_a
value_a in the formula *result* = *value_b* - *value_a*

value_b
value_b in the formula *result* = *value_b* - *value_a*

result
The sum of the values.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (*`list of Any`*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **`template_value`** (*`Any, optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)

Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`property dependencies`

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`get_attribute_type`(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

`get_value_references`(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`
The unique id of this protocol.

Type str

`merge`(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`replace_protocol`(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

Groups

ProtocolGroup	A collection of protocols to be executed in one batch.
ConditionalGroup	A collection of protocols which are to execute until a given condition is met.

ProtocolGroup

class propertyestimator.protocols.groups.**ProtocolGroup**(*protocol_id*)

A collection of protocols to be executed in one batch.

This may be used for example to cluster together multiple protocols that will execute in a linear chain so that multiple scheduler execution calls are reduced into a single one.

Additionally, a group may provide enhanced behaviour, for example running all protocols within the group self consistently until a given condition is met (e.g run a simulation until a given observable has converged).

__init__(*protocol_id*)

Constructs a new ProtocolGroup.

Methods

__init__(protocol_id)	Constructs a new ProtocolGroup.
add_protocols(*protocols)	
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol group can be merged with another.
execute(directory, available_resources)	Executes the protocols within this groups
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols parameters / inputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another ProtocolGroup with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols parameters / inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.

Continued on next page

Table 154 – continued from previous page

dependants_graph	A dictionary of which stores which grouped protocols are dependant on other grouped protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
execution_order	The ids of the protocols in the group, in the order in which they will be internally executed.
id	The unique id of this protocol.
protocols	A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value the protocol itself.
root_protocols	The ids of the protocols in the group which do not take input from the other grouped protocols.
schema	A serializable schema for this object.

property root_protocols
The ids of the protocols in the group which do not take input from the other grouped protocols.

Type List[str]

property execution_order
The ids of the protocols in the group, in the order in which they will be internally executed.

Type List[str]

property dependants_graph
A dictionary of which stores which grouped protocols are dependant on other grouped protocols. Each key in the dictionary is the id of a grouped protocol, and each value is the id of a protocol which depends on the protocol by the key.

Type Dict[str, str]

property protocols
A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value the protocol itself.

Type Dict[str, *BaseProtocol*]

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a different one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

execute(*directory*, *available_resources*)
Executes the protocols within this groups

Parameters

- **directory** (*str*) – The root directory in which to run the protocols

- **available_resources** (ComputeResources) – The resources available to execute on.

Returns True if all the protocols execute correctly.

Return type bool

can_merge(*other*)

Determines whether this protocol group can be merged with another.

Parameters **other** (ProtocolGroup) – The protocol group to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

merge(*other*)

Merges another ProtocolGroup with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocol groups are compatible to be merged together.

Parameters **other** (ProtocolGroup) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)

Returns the value of one of this protocols parameters / inputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input

Return type object

set_value(*reference_path*, *value*)

Sets the value of one of this protocols parameters / inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

This parameter is mutually exclusive with *template_index* and *template_value*

- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list / dict* which contains at least one `ProtocolPath`.

Parameters input_path (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

property schema
A serializable schema for this object.

Type *ProtocolSchema*

ConditionalGroup

class `propertyestimator.protocols.groups.`**`ConditionalGroup`**(*protocol_id*)

A collection of protocols which are to execute until a given condition is met.

`__init__`(*protocol_id*)

Constructs a new ConditionalGroup

Methods

`__init__`(protocol_id)	Constructs a new ConditionalGroup
`add_condition`(condition_to_add)	Adds a condition to this groups list of conditions if it not already in the condition list.
`add_protocols`(*protocols)	
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol group can be merged with another.
`execute`(directory, available_resources)	Executes the protocols within this groups
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols parameters / inputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another ProtocolGroup with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols parameters / inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`conditions`	
`dependants_graph`	A dictionary of which stores which grouped protocols are dependant on other grouped protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`execution_order`	The ids of the protocols in the group, in the order in which they will be internally executed.
`id`	The unique id of this protocol.
`max_iterations`	The maximum number of iterations to run for to try and satisfy the groups conditions.

Continued on next page

Table 156 – continued from previous page

protocols	A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value the protocol itself.
root_protocols	The ids of the protocols in the group which do not take input from the other grouped protocols.
schema	A serializable schema for this object.

class ConditionType
The acceptable conditions to place on the group

max_iterations
The maximum number of iterations to run for to try and satisfy the groups conditions.

execute(*directory*, *available_resources*)
Executes the protocols within this groups

Parameters

- **directory** (*str*) – The root directory in which to run the protocols
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns True if all the protocols execute correctly.

Return type bool

can_merge(*other*)
Determines whether this protocol group can be merged with another.

Parameters other (ProtocolGroup) – The protocol group to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

merge(*other*)
Merges another ProtocolGroup with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocol groups are compatible to be merged together.

Parameters other (ConditionalGroup) – The protocol to merge into this one.

add_condition(*condition_to_add*)
Adds a condition to this groups list of conditions if it not already in the condition list.

Parameters condition_to_add (ConditionalGroup.Condition) – The condition to add.

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters value (*str*) – The uuid of the parent calculation.

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a different one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.

- **new_id** (*str*) – The id of the new input protocol.

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols parameters / inputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input

Return type object

set_value(*reference_path*, *value*)
Sets the value of one of this protocols parameters / inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*

- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **`template_value`** (*`Any, optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`property dependants_graph`
A dictionary of which stores which grouped protocols are dependant on other grouped protocols. Each key in the dictionary is the id of a grouped protocol, and each value is the id of a protocol which depends on the protocol by the key.

Type Dict[str, str]

`property dependencies`
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`property execution_order`
The ids of the protocols in the group, in the order in which they will be internally executed.

Type List[str]

`property id`
The unique id of this protocol.

Type str

`property protocols`
A dictionary of the protocols in this groups, where the dictionary key is the protocol id, and the value the protocol itself.

Type Dict[str, *BaseProtocol*]

`property root_protocols`
The ids of the protocols in the group which do not take input from the other grouped protocols.

Type List[str]

`property schema`
A serializable schema for this object.

Type *ProtocolSchema*

Storage

UnpackStoredDataCollection	Loads a *StoredDataCollection* object from disk, and makes its inner data objects easily accessible to other protocols.
UnpackStoredSimulationData	Loads a *StoredSimulationData* object from disk, and makes its attributes easily accessible to other protocols.

UnpackStoredDataCollection

class propertyestimator.protocols.storage.**UnpackStoredDataCollection**(*protocol_id*)

Loads a *StoredDataCollection* object from disk, and makes its inner data objects easily accessible to other protocols.

__init__(*protocol_id*)

Constructs a new UnpackStoredDataCollection object.

Methods

__init__(protocol_id)	Constructs a new UnpackStoredDataCollection object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
collection_data_paths	A dictionary of data object path, data directory path and force field path tuples partitioned by the unique collection keys.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.

Continued on next page

Table 159 – continued from previous page

input_data_path	A tuple which contains both the path to the simulation data object, it's ancillary data directory, and the force field which was used to generate the stored data.
schema	A serializable schema for this object.

input_data_path

A tuple which contains both the path to the simulation data object, it's ancillary data directory, and the force field which was used to generate the stored data.

collection_data_paths

A dictionary of data object path, data directory path and force field path tuples partitioned by the unique collection keys.

execute (*directory*, *available_resources*)

Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging

If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)

Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`property dependencies`

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`get_attribute_type`(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

`get_value_references`(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`

The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (`ProtocolPath`) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

UnpackStoredSimulationData

class `propertyestimator.protocols.storage.`**UnpackStoredSimulationData**(*protocol_id*)
Loads a *StoredSimulationData* object from disk, and makes its attributes easily accessible to other protocols.

__init__(*protocol_id*)
Constructs a new UnpackStoredSimulationData object.

Methods

`__init__`(protocol_id)	Constructs a new UnpackStoredSimulationData object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`coordinate_file_path`	A path to the stored simulation trajectory.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`force_field_path`	A path to the force field parameters used to generate the stored data.
`id`	The unique id of this protocol.
`schema`	A serializable schema for this object.
`simulation_data_path`	A tuple which contains both the path to the simulation data object, it's ancillary data directory, and the force field which was used to generate the stored data.
`statistical_inefficiency`	The statistical inefficiency of the stored data.
`statistics_file_path`	A path to the stored simulation statistics array.
`substance`	The substance which was stored.
`thermodynamic_state`	The thermodynamic state which was stored.
`total_number_of_molecules`	The total number of molecules in the stored system.
`trajectory_file_path`	A path to the stored simulation trajectory.

`simulation_data_path`
A tuple which contains both the path to the simulation data object, it's ancillary data directory, and the force field which was used to generate the stored data.

`substance`
The substance which was stored.

`total_number_of_molecules`
The total number of molecules in the stored system.

`thermodynamic_state`

The thermodynamic state which was stored.

statistical_inefficiency
The statistical inefficiency of the stored data.

coordinate_file_path
A path to the stored simulation trajectory.

trajectory_file_path
A path to the stored simulation trajectory.

statistics_file_path
A path to the stored simulation statistics array.

force_field_path
A path to the force field parameters used to generate the stored data.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*propertyestimator.workflow.utils.ProtocolPath*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (BaseProtocol) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema

A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

Miscellaneous

AddValues	A protocol to add together a list of values.
SubtractValues	A protocol to subtract one value from another such that:
MultiplyValue	A protocol which multiplies a value by a specified scalar
DivideValue	A protocol which divides a value by a specified scalar
FilterSubstanceByRole	A protocol which takes a substance as input, and returns a substance which only contains components whose role match a given criteria.

AddValues

class propertyestimator.protocols.miscellaneous.**AddValues**(*protocol_id*)

A protocol to add together a list of values.

Notes

The *values* input must either be a list of unit.Quantity, a ProtocolPath to a list of unit.Quantity, or a list of ProtocolPath which each point to a unit.Quantity.

__init__(*protocol_id*)

Constructs a new AddValues object.

Methods

__init__(protocol_id)	Constructs a new AddValues object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
result	The sum of the values.
schema	A serializable schema for this object.
values	The values to add together.

values

The values to add together.

result

The sum of the values.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

SubtractValues

class propertyestimator.protocols.miscellaneous.**SubtractValues**(*protocol_id*)
A protocol to subtract one value from another such that:

result = value_b - value_a

__init__(*protocol_id*)
Constructs a new AddValues object.

Methods

__init__(protocol_id)	Constructs a new AddValues object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol

Continued on next page

Table 165 – continued from previous page

set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
id	The unique id of this protocol.
result	The sum of the values.
schema	A serializable schema for this object.
value_a	*value_a* in the formula *result* = *value_b* - *value_a*
value_b	*value_b* in the formula *result* = *value_b* - *value_a*

value_a
value_a in the formula *result* = *value_b* - *value_a*

value_b
value_b in the formula *result* = *value_b* - *value_a*

result
The sum of the values.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (str) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*

- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)
Determines whether this protocol can be merged with another.

Parameters **other** (BaseProtocol) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (ProtocolPath) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`

The unique id of this protocol.

Type str

`merge`(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`replace_protocol`(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path*, *value*)

Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

MultiplyValue

class propertyestimator.protocols.miscellaneous.**MultiplyValue**(*protocol_id*)
A protocol which multiplies a value by a specified scalar

__init__(*protocol_id*)
Constructs a new MultiplyValue object.

Methods

`__init__`(protocol_id)	Constructs a new MultiplyValue object.
`apply_replicator`(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
`can_merge`(other)	Determines whether this protocol can be merged with another.
`execute`(directory, available_resources)	Execute the protocol.
`get_attribute_type`(reference_path)	Returns the type of one of the protocol input/output attributes.
`get_value`(reference_path)	Returns the value of one of this protocols inputs / outputs.
`get_value_references`(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
`merge`(other)	Merges another BaseProtocol with this one.
`replace_protocol`(old_id, new_id)	Finds each input which came from a given protocol
`set_uuid`(value)	Store the uuid of the calculation this protocol belongs to
`set_value`(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

`allow_merging`	If true, this protocol is allowed to merge with other identical protocols.
`dependencies`	A list of pointers to the protocols which this protocol takes input from.
`id`	The unique id of this protocol.
`multiplier`	The scalar to multiply by.
`result`	The result of the multiplication.
`schema`	A serializable schema for this object.
`value`	The value to multiply.

value
The value to multiply.

multiplier
The scalar to multiply by.

result
The result of the multiplication.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **`directory`** (*`str`*) – The directory to store output data in.
- **`available_resources`** (`ComputeResources`) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

`allow_merging`

If true, this protocol is allowed to merge with other identical protocols.

Type bool

`apply_replicator`(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)

Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **`replicator`** (`ProtocolReplicator`) – The replicator to apply.
- **`template_values`** (*`list of Any`*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **`template_index`** (*`int, optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **`template_value`** (*`Any, optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)

Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies
A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)
Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)
Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)
Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id
The unique id of this protocol.

Type str

merge(*other*)
Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path, value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

DivideValue

class propertyestimator.protocols.miscellaneous.**DivideValue**(*protocol_id*)
A protocol which divides a value by a specified scalar

__init__(*protocol_id*)
Constructs a new DivideValue object.

Methods

__init__(protocol_id)	Constructs a new DivideValue object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to

Continued on next page

Table 169 – continued from previous page

set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
dependencies	A list of pointers to the protocols which this protocol takes input from.
divisor	The scalar to divide by.
id	The unique id of this protocol.
result	The result of the division.
schema	A serializable schema for this object.
value	The value to divide.

value
The value to divide.

divisor
The scalar to divide by.

result
The result of the division.

execute (*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator (*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*

- **`template_index`** (*`int`, `optional`*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.

- **`template_value`** (*`Any`, `optional`*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.

- **`update_input_references`** (*`bool`*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

`can_merge`(*other*)

Determines whether this protocol can be merged with another.

Parameters **`other`** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

`property dependencies`

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

`get_attribute_type`(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

`get_value`(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

`get_value_references`(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **`input_path`** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

`property id`

The unique id of this protocol.

Type str

`merge`(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **`other`** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

`replace_protocol`(*old_id, new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **`old_id`** (*`str`*) – The id of the old input protocol.
- **`new_id`** (*`str`*) – The id of the new input protocol.

`property schema`

A serializable schema for this object.

Type *ProtocolSchema*

`set_uuid`(*value*)

Store the uuid of the calculation this protocol belongs to

Parameters **`value`** (*`str`*) – The uuid of the parent calculation.

`set_value`(*reference_path, value*)

Sets the value of one of this protocols inputs.

Parameters

- **`reference_path`** (`ProtocolPath`) – The path pointing to the value to return.
- **`value`** (*`Any`*) – The value to set.

FilterSubstanceByRole

class propertyestimator.protocols.miscellaneous.**FilterSubstanceByRole**(*protocol_id*)

A protocol which takes a substance as input, and returns a substance which only contains components whose role match a given criteria.

__init__(*protocol_id*)

Constructs a new AddValues object.

Methods

__init__(protocol_id)	Constructs a new AddValues object.
apply_replicator(replicator, template_values)	Applies a *ProtocolReplicator* to this protocol.
can_merge(other)	Determines whether this protocol can be merged with another.
execute(directory, available_resources)	Execute the protocol.
get_attribute_type(reference_path)	Returns the type of one of the protocol input/output attributes.
get_value(reference_path)	Returns the value of one of this protocols inputs / outputs.
get_value_references(input_path)	Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.
merge(other)	Merges another BaseProtocol with this one.
replace_protocol(old_id, new_id)	Finds each input which came from a given protocol
set_uuid(value)	Store the uuid of the calculation this protocol belongs to
set_value(reference_path, value)	Sets the value of one of this protocols inputs.

Attributes

allow_merging	If true, this protocol is allowed to merge with other identical protocols.
component_role	The role to filter substance components against.
dependencies	A list of pointers to the protocols which this protocol takes input from.
expected_components	The number of components expected to remain after filtering.
filtered_substance	The filtered substance.
id	The unique id of this protocol.
input_substance	The substance to filter.
schema	A serializable schema for this object.

input_substance

The substance to filter.

component_role

The role to filter substance components against.

expected_components

The number of components expected to remain after filtering. An exception is raised if this number is not matched. Setting this value to -1 will disable this check.

filtered_substance
The filtered substance.

execute(*directory*, *available_resources*)
Execute the protocol.

Protocols may be chained together by passing the output of previous protocols as input to the current one.

Parameters

- **directory** (*str*) – The directory to store output data in.
- **available_resources** (ComputeResources) – The resources available to execute on.

Returns The output of the execution.

Return type Dict[str, Any]

allow_merging
If true, this protocol is allowed to merge with other identical protocols.

Type bool

apply_replicator(*replicator*, *template_values*, *template_index=-1*, *template_value=None*, *update_input_references=False*)
Applies a *ProtocolReplicator* to this protocol. This method should clone any protocols whose id contains the id of the replicator (in the format *$(replicator.id)*).

Parameters

- **replicator** (ProtocolReplicator) – The replicator to apply.
- **template_values** (*list of Any*) – A list of the values which will be inserted into the newly replicated protocols.

  This parameter is mutually exclusive with *template_index* and *template_value*
- **template_index** (*int, optional*) – A specific value which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_value*.
- **template_value** (*Any, optional*) – A specific index which should be used for any protocols flagged as to be replicated by the replicator. This option is mainly used when replicating children of an already replicated protocol.

  This parameter is mutually exclusive with *template_values* and must be set along with a *template_index*.
- **update_input_references** (*bool*) – If true, any protocols which take their input from a protocol which was flagged for replication will be updated to take input from the actually replicated protocol. This should only be set to true if this protocol is not nested within a workflow or a protocol group.

  This option cannot be used when a specific *template_index* or *template_value* is providied.

Returns A dictionary of references to all of the protocols which have been replicated, with keys of original protocol ids. Each value is comprised of a list of the replicated protocol ids, and their index into the *template_values* array.

Return type dict of ProtocolPath and list of tuple of ProtocolPath and int

can_merge(*other*)

Determines whether this protocol can be merged with another.

Parameters **other** (`BaseProtocol`) – The protocol to compare against.

Returns True if the two protocols are safe to merge.

Return type bool

property dependencies

A list of pointers to the protocols which this protocol takes input from.

Type list of ProtocolPath

get_attribute_type(*reference_path*)

Returns the type of one of the protocol input/output attributes.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value whose type to return.

Returns The type of the attribute.

Return type type

get_value(*reference_path*)

Returns the value of one of this protocols inputs / outputs.

Parameters **reference_path** (`ProtocolPath`) – The path pointing to the value to return.

Returns The value of the input / output

Return type Any

get_value_references(*input_path*)

Returns a dictionary of references to the protocols which one of this protocols inputs (specified by *input_path*) takes its value from.

Notes

Currently this method only functions correctly for an input value which is either currently a `ProtocolPath`, or a *list* / *dict* which contains at least one `ProtocolPath`.

Parameters **input_path** (*`propertyestimator.workflow.utils.ProtocolPath`*) – The input value to check.

Returns A dictionary of the protocol paths that the input targeted by *input_path* depends upon.

Return type dict of ProtocolPath and ProtocolPath

property id

The unique id of this protocol.

Type str

merge(*other*)

Merges another BaseProtocol with this one. The id of this protocol will remain unchanged.

It is assumed that can_merge has already returned that these protocols are compatible to be merged together.

Parameters **other** (`BaseProtocol`) – The protocol to merge into this one.

Returns A map between any original protocol ids and their new merged values.

Return type Dict[str, str]

replace_protocol(*old_id*, *new_id*)

Finds each input which came from a given protocol and redirects it to instead take input from a new one.

Notes

This method is mainly intended to be used only when merging multiple protocols into one.

Parameters

- **old_id** (*str*) – The id of the old input protocol.
- **new_id** (*str*) – The id of the new input protocol.

property schema
A serializable schema for this object.

Type *ProtocolSchema*

set_uuid(*value*)
Store the uuid of the calculation this protocol belongs to

Parameters **value** (*str*) – The uuid of the parent calculation.

set_value(*reference_path*, *value*)
Sets the value of one of this protocols inputs.

Parameters

- **reference_path** (ProtocolPath) – The path pointing to the value to return.
- **value** (*Any*) – The value to set.

1.5.10 Workflow Construction Utilities

BaseReweightingProtocols	
BaseSimulationProtocols	
generate_base_reweighting_protocols	Constructs a set of protocols which, when combined in a workflow schema, may be executed to reweight a set of existing data to estimate a particular property.
generate_base_simulation_protocols	Constructs a set of protocols which, when combined in a workflow schema, may be executed to run a single simulation to estimate a particular property.
generate_gradient_protocol_group	Constructs a set of protocols which, when combined in a workflow schema, may be executed to reweight a set of existing data to estimate a particular property.

BaseReweightingProtocols

class propertyestimator.protocols.utils.**BaseReweightingProtocols**(*unpack_stored_data*, *analysis_protocol*, *decorrelate_statistics*, *decorrelate_trajectory*, *concatenate_trajectories*, *concatenate_statistics*, *build_reference_system*, *reduced_reference_potential*, *build_target_system*, *reduced_target_potential*, *mbar_protocol*)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

analysis_protocol	Alias for field number 1
build_reference_system	Alias for field number 6
build_target_system	Alias for field number 8
concatenate_statistics	Alias for field number 5
concatenate_trajectories	Alias for field number 4
decorrelate_statistics	Alias for field number 2
decorrelate_trajectory	Alias for field number 3
mbar_protocol	Alias for field number 10
reduced_reference_potential	Alias for field number 7
reduced_target_potential	Alias for field number 9
unpack_stored_data	Alias for field number 0

property analysis_protocol

Alias for field number 1

property build_reference_system

Alias for field number 6

property build_target_system

Alias for field number 8

property concatenate_statistics
Alias for field number 5

property concatenate_trajectories
Alias for field number 4

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

property decorrelate_statistics
Alias for field number 2

property decorrelate_trajectory
Alias for field number 3

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

property mbar_protocol
Alias for field number 10

property reduced_reference_potential
Alias for field number 7

property reduced_target_potential
Alias for field number 9

property unpack_stored_data
Alias for field number 0

BaseSimulationProtocols

class propertyestimator.protocols.utils.**BaseSimulationProtocols** (*build_coordinates*, *assign_parameters*, *energy_minimisation*, *equilibration_simulation*, *production_simulation*, *analysis_protocol*, *converge_uncertainty*, *extract_uncorrelated_trajectory*, *extract_uncorrelated_statistics*)

__init__ ()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.

Continued on next page

Table 176 – continued from previous page

`count`(value)	
`index`(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

`analysis_protocol`	Alias for field number 5
`assign_parameters`	Alias for field number 1
`build_coordinates`	Alias for field number 0
`converge_uncertainty`	Alias for field number 6
`energy_minimisation`	Alias for field number 2
`equilibration_simulation`	Alias for field number 3
`extract_uncorrelated_statistics`	Alias for field number 8
`extract_uncorrelated_trajectory`	Alias for field number 7
`production_simulation`	Alias for field number 4

property analysis_protocol
Alias for field number 5

property assign_parameters
Alias for field number 1

property build_coordinates
Alias for field number 0

property converge_uncertainty
Alias for field number 6

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

property energy_minimisation
Alias for field number 2

property equilibration_simulation
Alias for field number 3

property extract_uncorrelated_statistics
Alias for field number 8

property extract_uncorrelated_trajectory
Alias for field number 7

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

property production_simulation
Alias for field number 4

propertyestimator.protocols.utils.generate_base_reweighting_protocols

`propertyestimator.protocols.utils.`**`generate_base_reweighting_protocols`**(*analysis_protocol*, *mbar_protocol*, *workflow_options*, *replicator_id='data_repl'*, *id_suffix=''*)

Constructs a set of protocols which, when combined in a workflow schema, may be executed to reweight a set of existing data to estimate a particular property. The reweighted observable of interest will be calculated by following the passed in *analysis_protocol*.

Parameters

- **analysis_protocol** (`AveragePropertyProtocol`) – The protocol which will take input from the stored data, and generate a set of observables to reweight.
- **mbar_protocol** (*BaseReweightingProtocol*) – A template mbar reweighting protocol, which has it's reference observables already set. This method will automatically set the reduced potentials on this object.
- **workflow_options** (`WorkflowOptions`) – The options being used to generate a workflow.
- **replicator_id** (*str*) – The id to use for the data replicator.
- **id_suffix** (*str*) – A string suffix to append to each of the protocol ids.

Returns

- *BaseReweightingProtocols* – A named tuple of the protocol which should form the bulk of a property estimation workflow.
- *ProtocolReplicator* – A replicator which will clone the workflow for each piece of stored data.

propertyestimator.protocols.utils.generate_base_simulation_protocols

`propertyestimator.protocols.utils.`**`generate_base_simulation_protocols`**(*analysis_protocol*, *workflow_options*, *id_suffix=''*, *conditional_group=None*)

Constructs a set of protocols which, when combined in a workflow schema, may be executed to run a single simulation to estimate a particular property. The observable of interest to extract from the simulation is determined by the passed in *analysis_protocol*.

The protocols returned will:

1) Build a set of liquid coordinates for the property substance using packmol.
2) Assign a set of smirnoff force field parameters to the system.
3) Perform an energy minimisation on the system.
4) Run a short NPT equilibration simulation for 100000 steps using a timestep of 2fs.
5) Within a conditional group (up to a maximum of 100 times):

5a) Run a longer NPT production simulation for 1000000 steps using a timestep of 2fs

5b) Extract the average value of an observable and it's uncertainty.

5c) If a convergence mode is set by the options, check if the target uncertainty has been met.
If not, repeat steps 5a), 5b) and 5c).

6) Extract uncorrelated configurations from a generated production simulation.
7) Extract uncorrelated statistics from a generated production simulation.

Parameters

- **`analysis_protocol`** (`AveragePropertyProtocol`) – The protocol which will extract the observable of interest from the generated simulation data.
- **`workflow_options`** (`WorkflowOptions`) – The options being used to generate a workflow.
- **`id_suffix`** (*`str`*) – A string suffix to append to each of the protocol ids.
- **`conditional_group`** (`ProtocolGroup`, *`optional`*) – A custom group to wrap the main simulation / extraction protocols within. It is up to the caller of this method to manually add the convergence conditions to this group. If *None*, a default group with uncertainty convergence conditions is automatically constructed.

Returns

- *BaseSimulationProtocols* – A named tuple of the generated protocols.
- *ProtocolPath* – A reference to the final value of the estimated observable and its uncertainty (an *EstimatedQuantity*).
- *WorkflowSimulationDataToStore* – An object which describes the default data from a simulation to store, such as the uncorrelated statistics and configurations.

propertyestimator.protocols.utils.generate_gradient_protocol_group

`propertyestimator.protocols.utils.`**`generate_gradient_protocol_group`**(*template_reweighting_protocol*, *reference_force_field_paths*, *target_force_field_path*, *coordinate_file_path*, *trajectory_file_path*, *statistics_file_path=''*, *replicator_id='repl'*, *perturbation_scale=0.0001*, *substance_source=None*, *id_suffix=''*, *enable_pbc=True*, *use_subset_of_force_field=True*, *effective_sample_indices=None*)

Constructs a set of protocols which, when combined in a workflow schema, may be executed to reweight a set of existing data to estimate a particular property. The reweighted observable of interest will be calculated by following the passed in *analysis_protocol*.

Parameters

- **`template_reweighting_protocol`** (`BaseMBARProtocol`) – A template protocol which will be used to reweight the observable of interest to small perturbations to the parameter of interest. These will then be used to calculate the finite difference gradient. This utility takes care of setting the target and reference reduced potentials.

  In the case that the template is of type *ReweightStatistics* and the observable is an energy, the statistics path will automatically be pointed to the energies evaluated using the perturbed parameter as opposed to the energy measured during the reference simulation.

- **`reference_force_field_paths`** (`ProtocolPath` *`or list of ProtocolPath`*) – The paths to the force field parameters which were used to generate the trajectories from which the observables of interest were calculated.

- **`target_force_field_path`** (`ProtocolPath`) –

  The path to the force field parameters which the observables are being estimated at
  (this is mainly only useful when estimating the gradients of reweighted observables).

- **`coordinate_file_path`** (`ProtocolPath`) – A path to the initial coordinates of the simulation trajectory which was used to estimate the observable of interest.

- **`trajectory_file_path`** (`ProtocolPath`) – A path to the simulation trajectory which was used to estimate the observable of interest.

- **`statistics_file_path`** (`ProtocolPath`, *`optional`*) – A path to the statistics where were generated from the trajectory passed to the *trajectory_file_path* parameter. This is optional in cases where multiple reference force fields are passed to this method.

- **replicator_id** (*str*) – A unique id which will be used for the protocol replicator which will replicate this group for every parameter of interest.
- **perturbation_scale** (*float*) – The default amount to perturb parameters by.
- **substance_source** (PlaceholderInput, *optional*) – An optional protocol path to the substance whose gradient is being estimated. If None, the global property substance is used.
- **id_suffix** (*str*) – An optional string to append to the end of each of the protocol ids.
- **enable_pbc** (*bool*) – If true, periodic boundary conditions are employed when recalculating the reduced potentials.
- **use_subset_of_force_field** (*bool*) – If True, any reduced potentials will only be calculated from a subset of the force field which depends on the parameter of interest.
- **effective_sample_indices** (ProtocolPath, *optional*) – A placeholder variable which can be used to make the gradient protocols dependant on an MBAR protcol to ensure gradients aren't calcuated when the MBAR protocol failed due to insufficient samples.

Returns

- *ProtocolGroup* – The protocol group which will estimate the gradient of an observable with respect to one parameter.
- *ProtocolReplicator* – The replicator which will copy the gradient group for every parameter of interest.
- *ProtocolPath* – A protocol path which points to the final gradient value.

1.6 Release History

Releases will eventually follow the `major.minor.micro` scheme recommended by PEP440, where

- `major` increments denote a change that may break API compatibility with previous `major` releases
- `minor` increments add features but do not break API compatibility
- `micro` increments represent bugfix releases or improvements in documentation

All early releases however will simply recieve a `micro` version bump regardless of how major the changes may be.

1.6.1 0.0.1 - Initial Release

The initial pre-alpha release of the framework.

See our installation instructions.

Please report bugs, request features, or ask questions through our issue tracker.

Please note that there may still be some changes to the API prior to a stable 1.0.0 release.

1.7 Release Process

This document aims to outline the steps needed to release the `propertyestimator` on `omnia`. This should only be done with the approval of the core maintainers.

1.7.1 1. Update the Release History

If no PR has been submitted, create a new one to keep track of changes to the release notes *only*. Only the `releasehistory.rst` file may be edited in this PR.

Ensure that the release history file is up to date, and conforms to the below template:

```
# X.Y.Z

This release ...

A richer version of these release notes with live links to API documentation is␣
↪available
on [our ReadTheDocs page](https://property-estimator.readthedocs.io/en/latest/
↪releasehistory.html)

See our [installation instructions](https://property-estimator.readthedocs.io/en/
↪latest/installation.html).

Please report bugs, request features, or ask questions through our
[issue tracker](https://github.com/openforcefield/propertyestimator/issues).

**Please note that this is a pre-alpha release and there will still be major changes␣
↪to the API
prior to a stable 1.0.0 release.**

### Bugfixes

* PR #1: Summary of PR #1
* PR #2: Summary of PR #2

### Breaking Change

* A list of those changes which break either the API, or the fundamental science␣
↪performed by the
  framework.

### Migration Guide

This release contained several major public API breaking changes. For the most part,␣
↪these can be
remedied by the follow steps:

* `method_x` is now exposed as `method_y`
```

1.7.2 2: Cut the Release on GitHub

To cut a new release on GitHub:

1) Go to the `Releases` tab on the front page of the repo and choose `Create a new release`.
2) Set the release tag using the form: `X.Y.Z`
3) Added a descriptive title using the form: `X.Y.Z [Descriptive Title]`
4) Ensure the `This is a pre-release` checkbox is ticked.
5) Copy the release notes from part one into the description box.

Note - You do not need to upload any files. The source code will automatically be added as a tar.gz.

1.7.3 3: Trigger a New Build on Omnia

To trigger the build in `omnia`:

1) Create branch or fork of omnia-md/conda-recipes with the following changes to propertyestimator in **‘meta.yaml<https://github.com/omnia-md/conda-recipes/blob/master/propertyestimator/meta.yaml>‘_**:

a) Set `git_tag` to match the git release tag

b) Update the `version` to match the release (this will go into the conda package name)

c) Set `build` to 0

d) Update any dependencies in the `requirements` section

e) If we want to push to special `rc` label use `extra.upload`

2) Open PR to merge branch or fork into omnia-md master:

a) The PR title should have the format `[propertyestimator] X.Y.Z (label:  rc)`

b) No PR body text is needed

c) Travis will run on this PR (~30 minutes) and attempt to build the package. Under no conditions will the package be uploaded before the PR is merged. This step is just to ensure that building doesn’t crash.

d) If the build is successful the PR should be reviewed and merged by the `omnia` maintainers

e) **Once merged into master** the package is built again on travis, and pushed to the channel set in meta.yaml (`main`, `beta`, or `rc`)

3) Test the `omnia` package:

a) `conda install -c omnia/label/rc propertyestimator`

Note: Omnia builds take about 30 minutes to run. When you open a PR the build will run, and you can check the bottom of the travis logs for “package failed to build” listings. Some packages always fail (protons, assaytools), but propertyestimator shouldn’t be there. Ctrl-F for ‘‘propertyestimator‘‘ to ensure that it did build at all though.

1.7.4 4: Update the ReadTheDocs Build Versions

To ensure that the read the docs pages are updated:

1) Trigger a RTD build of `latest`.

2) Under the `Versions` tab add the new release version to the list of built versions and **save**.

3) Verify the new version docs have been built and pushed correctly

4) Under `Admin` | `Advanced Settings`: Set the new release version as Default version to display and **save**.

INDEX

Symbols